REPRESENTACIÓ NO-LINEAL DE LES IMATGES PER A CODIFICACIÓ PERCEPTIVA

Memòria del Projecte Fi de Carrera
d'Enginyeria en Telecomuncicació

realitzat per
Xènia Albà Cantero
i dirigit per
Rosa Maria Figueras i Ventura

Bellaterra, 28 de gener de 2009

Universitat Autònoma de Barcelona

etse
Escola Tècnica Superior d'Enginyeria

La sotasignant, **ROSA MARIA FIGUERAS i VENTURA**, professora del Departament d'Enginyeria de la Informació i de les Comunicacions de l'Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA:

Que la present memòria ha estat realitzada sota la seva direcció per l'alumna **XÈNIA ALBÀ CANTERO.**

I per que així consti firma la present.

Bellaterra, 28 de gener de 2009

Agraïments

Primer de tot, li voldria agrair a la Rosa la seva dedicació i paciència a l'hora d'explicar-me tots els dubtes que m'han anat sorgint al llarg del projecte. Gràcies per ajudar-me a seguir endavant i ensenyar-me tot el que he après fent aquest projecte.

Gràcies a la gent del GICI per haver-me permès utilitzar el BOI. Moltes gràcies a la gent del VISTA per cedir-me una documentació que ha estat clau en el projecte.

A tota la gent que ha participat en el test visual, gràcies per dedicar-me una mica del vostre temps. Als meus amics, us agraeixo les vostres paraules de suport i que m'hagueu fet la vida més fàcil. A la meva família, sobretot la meva mare, gràcies per aguantar, un cop més, el meus maldecaps i per cuidar-me tan i tan bé. Gràcies al Xavi pels seus ànims i consells, per ajudar-me i motivar-me cada dia.

En resum, gràcies a tota aquella gent que d'una manera o una altra m'han ajudat, potser sense saber-ho, a realitzar aquest projecte.

Índex

1 Introducció **7**

1.1 Objectius 7

1.2 Continguts de la memòria 8

2 Conceptes previs **10**

2.1 Imatges digitals 10

2.2 Compressió amb i sense pèrdues 11

2.3 Distorsió 11

2.4 Mesures de compressió 12

2.5 Elements d'un sistema de compressió 12

2.5.1 Transformada 13

2.5.2 Quantificació 13

2.5.3 Codificació 14

3 JPEG2000 **15**

3.1 Introducció 15

3.2 Funcionament 16

3.2.1 Transformada wavelet 17

3.2.2 Quantificació 23

3.2.3 Codificació de bloc 23

3.2.4 Codificació per bit-planes 27

3.2.5 Escaneig fraccional dels bit-planes 27

3.2.6 Resum 28

3.3 BOI 29

3.3.1 Disseny i estructura 29
3.3.2 Ús i paràmetres 31
3.3.3 Classes 32
3.3.4 Estructura de les dades 33

4 SSIM **36**
4.1 Definició 36
4.2 Integració de la mesura MSSIM a BOI 40

5 Normalització divisiva **43**
5.1 Descripció 43
5.2 Paràmetres de la normalització 44
5.2.1 Guany freqüencial depenent de l'escala i l'orientació 44
5.2.2 Matriu d'interacció 45
5.2.3 Paràmetres a optimitzar 45

6 Integració de la normalització divisiva al JPEG2000 **46**
6.1 Implementació 46
6.2 Optimització dels paràmetres 50
6.3 Resultats 51

7 Conclusions **67**

A ImageCompareMSSIM **72**

B BOICode: Inici normalització divisiva **78**

C BOICode: Funcions de la normalització divisiva **81**

D ConvexHull: Modificació Convex Hull **89**

Índex de figures

2.1 Imatge digital amb un detall de l'ull on s'observen els píxels amb claredat . . 10

2.2 Elements d'un sistema de compressió 13

2.3 Exemple de senyal quantificada 14

3.1 Diagrama de blocs de JPEG2000 [1]. a) Codificador b) Descodificador 17

3.2 Arbre de descomposició de la wavelet de tres nivells 20

3.3 Banc de filtres pel càlcul de la DWT de dos dimensions [4] 20

3.4 Resultat de l'aplicació d'una primera etapa de la DWT 21

3.5 DWT de 3 nivells 22

3.6 Descomposició de la DWT amb D=3 nivells [5] 22

3.7 Llindars del quantificador amb dead-zone 23

3.8 Divisió de les subbandes en code-blocks [5] 25

3.9 Paquet simple [5] 26

3.10 Convex hull dels parells rate-distortion per el bloc i. Els punts sòlids corresponen als candidats a punts de truncament. [5] 29

3.11 Comparació entre l'esquema de l'estàndard JPEG2000 i la implementació de BOI [10] 30

3.12 Disseny de les classes de BOI en el codificador [10] 31

3.13 Tipus de dades utilitzades per BOI [10] 33

4.1 Diagrama del sistema de la mesura de similitud estructural [2] 37

4.2 MSE i SSIM per una imatge amb diferents distorsions [14] 39

4.3 Esquema de funcionament de la classe ImageCompareMSSIM 40

4.4 Esquema de funcionament del mètode compMSSIM 41

6.1 Esquema de la integració de la normalització divisiva al BOI 47

6.2 Esquema de la implementació de la normalització divisiva 48

6.3 Exemples per (a) contorns marcats (b) contorns no marcats o molt junts . . 49

6.4 Resultats per un bit-rate de 0.75bps (a) Amb normalització PSNR=31.673708 dB, SSIM= 0.9195955747008607 (b)Sense normalització PSNR=34.229836 dB, SSIM= 0.9291282424585365 53

6.5 (a) Mapa SSIM de la imatge original amb la imatge obtinguda amb la normalització (b) Mapa SSIM de la imatge original amb la imatge obtinguda sense la normalització, on blanc significa SSIM=1 (exactament la mateixa imatge visual) i negre significa SSIM=0 (poca similitud visual) (c) Error de la imatge original amb la imatge obtinguda amb la normalització (d) Error de la imatge original amb la imatge obtinguda sense la normalització, on (negre, error 0 i blanc, error molt gran) 54

6.6 Detall de les imatges (a) imatge original (b) imatge amb normalització (c) imatge sense normalització 55

6.7 Resultats de la imatge ChurchTower per un bitrate de 0.75bps (a) Amb normalització PSNR=37.784336, SSIM=0.9586933982042634 (b) Sense normalització PSNR=42.147003, SSIM=0.9687704113082091 56

6.8 (a) Mapa SSIM de la imatge original amb la imatge obtinguda amb la normalització (b) Mapa SSIM de la imatge original amb la imatge obtinguda sense la normalització, on blanc significa SSIM=1 (exactament la mateixa imatge visual) i negre significa SSIM=0 (poca similitud visual) (c) Error de la imatge original amb la imatge obtinguda amb la normalització (d) Error de la imatge original amb la imatge obtinguda sense la normalització, on (negre, error 0, blanc, error molt gran) 57

6.9 Resultats per la imatge Goldhill amb un bitrate de 0.75bps (a) Amb normalització PSNR= 34.445007, SSIM= 0.9017559452515643 (b) Sense normalització PSNR= 35.013535, SSIM= 0.9097004536957387 58

6.10 Resultats obtinguts per la imatge Leaf amb un bitrate de 0.75bps. (a) Amb normalització PSNR=29.68547, SSIM=0.8362241739734876 (b) sense normalització PSNR=31.692026, SSIM=0.8695787469662211 59

6.11 (a) Mapa SSIM de la imatge original amb la imatge obtinguda amb la normalització (b) Mapa SSIM de la imatge original amb la imatge obtinguda sense la normalització, on blanc significa SSIM=1 (exactament la mateixa imatge visual) i negre significa SSIM=0 (poca similitud visual) (c) Error de la imatge original amb la imatge obtinguda amb la normalització (d) Error de la imatge original amb la imatge obtinguda sense la normalització, on (negre, error 0, blanc, error molt gran) 60

6.12 Resultats per un bitrate de 0.75bps (a)Lena amb normalització PSNR=35.124565 dB, SSIM= 0.9220759949136988 (b)Lena sense normalització PSNR= 37.725376 dB, SSIM= 0.9294548149441602 (c)Peppers amb normalització PSNR=36.336433 dB, SSIM= 0.9003507228944935 (d)Peppers sense normalització PSNR=37.10341 dB, SSIM= 0.9083685938701637 (e)Posies amb normalització PSNR=36.629505 dB, SSIM= 0.9738359413680001 (f)Posies sense normalització PSNR=38.806812 dB, SSIM= 0.9765227569688442 62

6.13 Resultats per un bitrate de 0.75bps (a) Sidewalk amb normalització PSNR=32.11766 dB, SSIM= 0.9215481722338017 (b) Sidewalk sense normalització PSNR=33.87218 dB, SSIM= 0.9292907043031611 (c) Tables amb normalització PSNR=39.11412 SSIM= 0.9769535067437072 (d) Tables sense normalització PSNR=40.508606, SSIM= 0.9768509282196824 (e) Waverly amb normalització PSNR=31.932108, SSIM= 0.9268495667560196 (f) Waverly sense normalització PSNR=33.258915, SSIM= 0.9320208401411235 63

6.14 Gràfic amb els resultats del test psico-visual 64

6.15 Gràfics del PSNR (a) i l'MSSIM (b) en funció del bit-rate per imatges de 128x128 píxels 65

6.16 Gràfics del PSNR (a) i l'SSIM (b) en funció del bit-rate per imatges de 256x256 píxels 65

6.17 Gràfics del PSNR (a) i l'SSIM (b) en funció del bit-rate per imatges de 512x512 píxels 66

Capítol 1

Introducció

Avui en dia la compressió d'imatges és molt important en sistemes i aplicacions multimèdia, ja que és important reduir la mida que les imatges ocupen al ser emmagatzemades i també l'ample de banda necessari per transmetre-les. Els estàndards de compressió anteriors es basaven en la transformada discreta del cosinus (DCT), com és el cas de JPEG, però s'està introduint la transformada discreta wavelet (DWT) que es considera millor i més eficient en aquest camp i proporciona una major flexibilitat en termes d'escalabilitat, gràcies als diferents nivells de resolució de la transformada.

En aquest projecte treballarem amb JPEG2000, estàndard creat conjuntament per ISO i ITU, que utilitza la transformada wavelet discreta. Tenint en compte que les tendències actuals en les tecnologies de compressió d'imatge són optimitzar aspectes com l'eficiència, l'escalabilitat i la interoperativitat en nous entorns multimèdia, JPEG2000 representa un gran pas endavant, respecte el clàssic JPEG, molt utilitzat per a la transmissió i emmagatzemament d'imatges.

En els sistemes de compressió d'informació visual, generalment el que més interessa no és que el sistema no tingui pèrdues sinó que aquestes pèrdues que s'introdueixen siguin el menys visibles possible. Serà necessari doncs estudiar i analitzar què és més perceptible per l'ull humà per poder aplicar tècniques que tenen en compte el seu comportament i poder eliminar aquella informació que pel sistema visual no és tant important i deixar intacta la informació rellevant.

1.1 Objectius

La compressió d'imatges s'implementa normalment utilitzant una transformada, posteriorment la quantificació i seguidament la codificació entròpica dels coeficients quantificats. JPEG2000 utilitza la transformada wavelet com a transformada i posteriorment una quantificació uniforme dels coeficients amb dead-zone. Els coeficients wavelet, a més a més, presenten certes dependències tant estadístiques com visuals. Les dependències estadístiques es tenen en compte a l'esquema JPEG2000 en la codificació aritmètica dels coeficients amb plans de bits

i contextos que depenen del valors dels coeficients wavelets veïns. Les dependències visuals, no obstant, no estan preses en consideració. Les característiques específiques del sistema visual humà, fan que el que resulta òptim en termes de distorsió matemàtica no sigui òptim visualment. Aquest projecte té com a objectiu final trobar una representació més adaptada al sistema visual que la que proporciona JPEG2000 directament.

Per dur a terme aquest objectiu s'utilitzarà el que s'anomena la normalització divisiva dels coeficients. Idealment, el que es voldria fer és reconvertir els coeficients a un espai de valors en els quals un valor més elevat dels coeficients impliqui un valor més elevat d'aportació visual. No obstant això, per no sortir del que marca l'estàndard JPEG2000 s'haurà d'adaptar la tècnica de normalització divisiva. En lloc de modificar directament els coeficients wavelets i canviar tot l'esquema de compressió, es reordenaran els coeficients JPEG2000 segons els errors obtinguts amb la normalització però s'enviaran els coeficients originals. Això, tal i com veurem al llarg d'aquest treball, farà que el sistema no sigui tan òptim com es podia esperar, , ja que s'hauran de fer moltes aproximacions per evitar sortir del marc de JPEG2000 part 1.

Un objectiu paral·lel serà el d'integrar aquestes modificacions del JPEG2000 a la implementació BOI, desenvolupada per part del Grup de Compressió Interactiva d'Imatges (GICI) del departament d'Enginyeria de la Informació i les Comunicacions (DEIC), integració que permetrà l'obtenció d'imatges amb escalabilitat visual.

En termes generals el objectius d'aquest treball són:

- Estudi i anàlisis de l'estàndard JPEG2000
- Estudi de tècniques de mesura de la qualitat de la imatge
- Estudi de la estructura i funcionament de la implementació BOI del GICI.
- Integració de l'algoritme de normalització divisiva dels coeficients wavelets al JPEG2000
- Proves experimentals sobre la plataforma BOI i obtenció dels resultats
- Anàlisis dels resultats mitjançant mesures de qualitat de la imatge i tests psico-visuals.
- Obtenció de les conclusions i redacció de la documentació

1.2 Continguts de la memòria

A la present memòria s'hi trobarà en el Capítol 2 un repàs d'alguns conceptes bàsics que sortiran a la resta de capítols i que serviran com a introducció al projecte. Seguidament, al Capítol 3, s'explicarà en què consisteix l'estàndard JPEG2000, entrant més en detall en aquelles parts que s'han de conèixer per entendre la resta del projecte. En el Capítol 4 es parlarà de la nova mesura SSIM, que serà utilitzada posteriorment en la part de normalització per ser optimitzada, que serà explicada en el Capítol 5. En l'últim capítol es mostrarà la

implementació que s'ha fet i els resultats obtinguts. Finalment es descriuran les conclusions a les que s'ha arribat al llarg del projecte. Les funcions més important implementades en el projecte es poden trobar en els àpendixs. Tot el codi font utilitzat en el projecte, així com les imatges utilitzades, es poden trobar al CD adjunt.

Capítol 2

Conceptes previs

En aquest capítol es presentaran conceptes bàsics de la compressió d'imatges. Aquests conceptes serviran com a introducció i ajudaran a entendre la resta de projecte.

2.1 Imatges digitals

Pels nostres propòsits una imatge és una seqüència de dues dimensions amb els valors de les mostres, tal que

$$x[n_1, n_2],\ 0 \leq n_1 < N_1,\ 0 \leq n_2 < N_2,$$

on N_1 i N_2 són extrems finits en la direcció vertical i horitzontal, respectivament. La primera coordenada, n_1, és l'index de les files, mentre que la segona coordenada, n_2, representa l'índex de la columna de la mostra o del píxel. El valor de la mostra, $x[n_1, n_2]$, representa la intensitat (luminància) de la imatge a la posició $[n_1, n_2]$, seran normalment enters amb signe o sense signe.

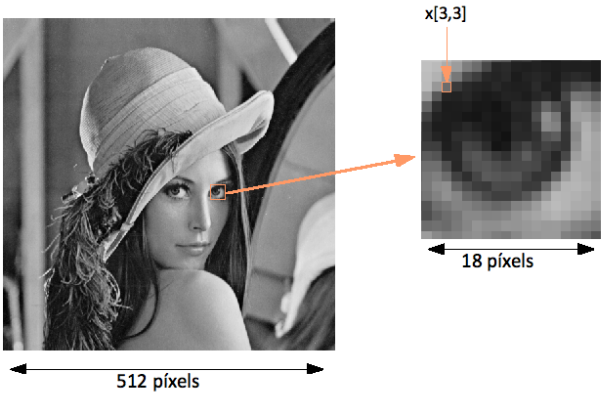


Figura 2.1: Imatge digital amb un detall de l'ull on s'observen els píxels amb claredat

2.2 Compressió amb i sense pèrdues

L'objectiu de la compressió és reduir la mida en nombre de bits de la representació de diferents tipus de senyal. La compressió és necessària per economitzar i ser més eficient en l'emmagatzematge i la transmissió d'informació. Els procediments de representació i processat dels senyals digitals poden variar considerablement segons l'aplicació que es requereixi. Algunes aplicacions d'ús molt estès porten a l'establiment d'estàndards de codificació i compressió.

Si considerem un senyal digital d'una determinada mida, es poden considerar diferents representacions d'aquest senyal i aquestes tindran mides diferents. Cal distingir primer entre dos tipus de compressió:

- **Compressió sense pèrdues**. El senyal comprimit ha de permetre reconstruir exactament el senyal original. Això limita la quantitat de compressió que es pot fer però és necessari segons l'objecte que tractem.
- **Compressió amb pèrdues.** Admeten la pèrdua d'informació que es traduirà en distorsió en el senyal reconstruït. L'objectiu d'aquesta compressió és arribar a un equilibri entre el grau de compressió i el nivell de distorsió.

Per compressió d'imatges, la pèrdua d'informació s'accepta habitualment degut a tres raons:

- Una pèrdua significativa pot ser tolerada pel sistema visual humà sense interferir en la percepció del contingut de l'escena.
- En molts casos, l'entrada digital del sistema de compressió és en sí mateixa un representació imperfecta d'una escena del món real.
- La compressió sense pèrdues és, normalment, incapaç d'aconseguir els requeriments d'una bona compressió en la majoria d'aplicacions d'emmagatzematge i distribució.

Tot i així, a vegades la compressió sense pèrdues és necessària en algunes aplicacions, com per exemple en aplicacions mèdiques on la introducció d'errors és inacceptable.

2.3 Distorsió

En el cas de la compressió amb pèrdues, al permetre la introducció de petits errors, és natural esperar que es pugui representar la imatge d'una manera aproximada utilitzant un menor nombre de bits que amb la compressió sense pèrdues. Com més distorsió es permeti, menor pot ser la mida de la representació de la imatge comprimida. El principal objectiu de la

compressió amb pèrdues és el de minimitzar el nombre de bits necessaris per un determinat nivell de distorsió. Aquest nivell de distorsió ha de ser degudament mesurat. Definim, $D(\mathbf{x}, \hat{\mathbf{x}})$ com la distorsió entre la imatge original, $\mathbf{x} \equiv x\,[n_1, n_2]$, i la imatge reconstruïda, $\hat{\mathbf{x}} \equiv \hat{x}\,[n_1, n_2]$, i tenint en compte que $N_1 \cdot N_2$ és la mida de la imatge.

La mesura que més s'utilitza per mesurar la distorsió és l'error en mitjana quadràtica, **MSE** (*Mean Square Error*), definit com

$$MSE \triangleq \frac{1}{N_1 N_2} \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} (x[n_1, n_2] - \hat{x}[n_{1,}, n_2])^2$$

Per compressió d'imatges, l'MSE s'expressa utilitzant la mesura de relació de pic de senyal a soroll, **PSNR** (*Peak Signal to Noise Ratio*), definida com

$$PSNR \triangleq 10 \cdot log_{10} \frac{(2^B - 1)^2}{MSE}$$

on B indica els bits per píxel. El PSNR s'expressa en dB (decibels). Un valor bo d'imatge reconstruïda ronda els 30dB o més.

2.4 Mesures de compressió

El propòsit de la compressió en imatges és el de representar la imatge amb una cadena de dígits binaris o "bits", anomenat el *bit-stream* comprimit, al que denotarem com $\mathbf{c}$. L'objectiu doncs és mantenir la longitud, $\|\mathbf{c}\|$, el més petita possible. En absència de compressió, es requereixen $N_1 N_2 B$ bits per representar els valors de les mostres de la imatge. Es defineix la relació de compressió com

$$\text{relació de compressió} \triangleq \frac{N_1 N_2 B}{\|\mathbf{c}\|},$$

on $N_1 \cdot N_2$ és la mida de la imatge i B són els bits per mostra de la imatge original.

Equivalentment, definim el *bit-rate* comprimit, expressat en bps (*bits per sample*), com

$$\text{bit-rate (bps)} \triangleq \frac{\|\mathbf{c}\|}{N_1 N_2}$$

aquesta mesura és sovint la més significativa per indicar el grau de compressió de les imatges.

2.5 Elements d'un sistema de compressió

En un sistema de compressió d'imatges existeix una estructura fixa, amb unes parts que apareixen en qualsevol dels sistemes, tal i com es mostra a la Figura 2.2 . El primer pas consisteix

en aplicar una transformada de les mostres originals en un nou conjunt de mostres que seran més manejables per a la compressió. La segona fase representa les mostres transformades d'una forma aproximada utilitzant una seqüència d'índex de quantificació. Finalment aquests índex es codifiquen per formar el *bit-stream* final.

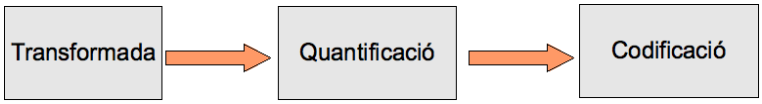


Figura 2.2: Elements d'un sistema de compressió

2.5.1 Transformada

La transformada és la responsable d'adaptar les mostres originals de la imatge en una forma que comparativament faci més simple la quantificació i la codificació. Per una banda, la transformada ha de capturar l'essència de les dependències estadístiques d'entre les mostres originals de manera que les mostres transformades, i en conseqüència les quantificades, tinguin unes dependències molt locals, idealment haurien de ser estadísticament independents. Per una altra banda, la transformada ha de separar la informació irrellevant de la informació rellevant.

Afortunadament, és possible trobar transformades que aconsegueixin, com a mínim parcialment, els dos objectius simultàniament. Com veurem en JPEG2000 s'utilitza la **transformada wavelet discreta** (DWT), a diferència de JPEG que utilitza la **transformada discreta del cosinus** (DCT).

2.5.2 Quantificació

La quantificació és en gran part la responsable de la introducció de distorsió, per tant, per la compressió sense pèrdues no n'hi hauria d'haver. El cas més simple de quantificació és el de mapejar cada mostra transformada, $y[k_1, k_2]$ independentment al corresponent índex de quantificació, $q[k_1, k_2]$. Aquest sistema és el que es coneix com a **quantificació escalar**, i és el més simple i més utilitzat. La quantificació escalar associa cada índex de quantificació amb un interval de la recta real d'acord a

$$q[k_1, k_2] = i \text{ if } y[k_1, k_2] \in I_i,$$

on els intervals, I_i, són disjunts i cobreixen la recta real.

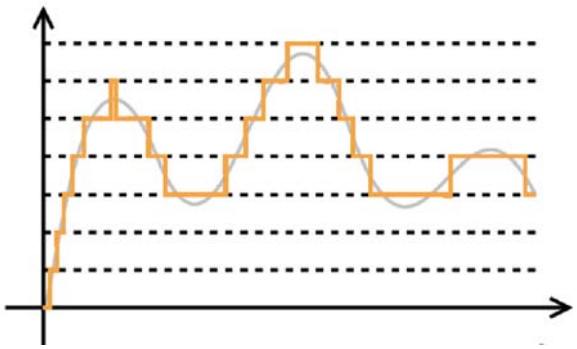


Figura 2.3: Exemple de senyal quantificada

2.5.3 Codificació

El propòsit de la codificació és el d'aprofitar la redundància estadística dels índex de la quantificació. Els elements de la quantificació i la transformació estan especialment dissenyats per assegurar que aquesta redundància sigui localitzable. Idealment els índex són independents, de manera que els índexs poden ser codificats independentment i la única forma de redundància estadística que s'ha de tenir en compte és aquella associada amb qualsevol no uniformitat en les seves distribucions de probabilitat.

No obstant, no és possible assegurar que els índex de la quantificació siguin estadísticament independents. Tot i així, com que les interaccions estadístiques es troben bàsicament amb els veïns més propers d'una mostra donada, sovint és possible trobar esquemes de codificació eficients d'una complexitat relativa.

En aquest sentit, és l'algoritme EBCOT el que realitza aquesta codificació a JPEG2000. Més endavant explicarem en detall en què es basa aquest algoritme.

Capítol 3

JPEG2000

En aquest capítol es presenten les característiques bàsiques de l'estàndard JPEG2000. Es farà especial èmfasi en les parts afectades a l'hora de desenvolupar el projecte, sense entrar en excessiu detall en l'operació interna de cadascuna d'elles. Primer farem una introducció a l'estàndard explicant les característiques principals i després descriurem el seu funcionament. En l'últim apartat presentarem la implementació de JPEG2000, desenvolupada pel Grup de Codificació Interactiva d'Imatges, anomenada BOI.

3.1 Introducció

El desenvolupament d'estàndards per a la compressió d'imatges per part de diferents organitzacions ha donat com a resultats molts productes aplicables a diferents camps de treball.

Des de mitjans dels 80, membres de la Organització internacional d'estandardització (ISO) i els de la Unió Internacional de telecomunicacions (ITU) van estar treballant conjuntament per trobar un estàndard per a la compressió d'imatges. Aquests esforços van donar lloc a JPEG (Joint Photographic Experts Group) que es va convertir en un estàndard internacional (IS, International Standard) l'any 1992, [4].

El JPEG ha estat llargament utilitzat per a la compressió i l'emmagatzematge de fotografies i material gràfic estàtic. Tot i així, la gran expansió de les aplicacions multimèdia i d'Internet va provocar l'aparició de noves necessitats i el creixement dels requeriments de les tecnologies utilitzades. Per aquest motiu, el març de 1997, va començar el desenvolupament d'un nou estàndard, JPEG2000. Aquest nou estàndard de compressió intenta crear un nou sistema de codificació per a diferents tipus d'imatges, amb diferents característiques i permetent diferents models, preferiblement en un mateix sistema. JPEG2000 representa un avanç en les tecnologies de la compressió d'imatges on el sistema de codificació s'ha d'optimitzar no només pel que fa a la seva eficiència sinó també en escalabilitat i interoperativitat en entorns mòbils i de xarxes.

El JPEG2000 proporciona un conjunt de característiques que són important per noves aplicacions aprofitant-se de les noves tecnologies. Aquestes aplicacions o mercats són Internet, la impressió, l'escaneig, la fotografia digital, la telefonia mòbil, les imatges mèdiques, els arxius i biblioteques digitals, etc... Cadascuna d'aquestes àrees d'aplicació imposa alguns dels requeriments de què l'estàndard ha de disposar. Algunes de les característiques més importants del JPEG2000 són:

Millor rendiment a taxes de bit baixes. S'ha d'aconseguir aquesta característica sense sacrificar l'espectre de la distorsió. Útil, per exemple, per a millorar la transmissió d'imatges a través de xarxes.

Compressió d'imatges binàries i de to contínua. És bo tenir un estàndard que sigui capaç de codificar aquests dos tipus d'imatges, si fos possible, utilitzant recursos similars.

Compressió amb i sense pèrdues. Es desitja per algunes aplicacions que l'estàndard pugui donar una compressió sense pèrdues, evidentment al llarg de la descodificació.

Transmissió progressiva en resolució i precisió de píxel. Una transmissió progressiva que permeti que les imatges es reconstrueixin incrementant la seva precisió de píxel o la seva resolució espacial.

Codificació amb regió d'interès (ROI). Sovint en les imatges hi ha parts que tenen més importància que d'altres. Aquesta característica de l'estàndard permet a l'usuari definir una ROI en la imatge per a que sigui codificada i transmesa amb una millor qualitat i menys distorsió que la resta de la imatge.

Arquitectura oberta. És desitjable que hi hagi una arquitectura oberta de manera que es pugui optimitzar el sistema per a diferents tipus d'imatges i d'aplicacions.

Robustesa als errors de bit. Ha de proporcionar sistemes de correcció a errors que poden afectar la descodificació.

Seguretat en la protecció de dades. Hi ha diferents maneres de protegir una imatge, JPEG2000 ha de ser capaç de proporcionar eines per dur-ho a terme.

Algunes d'aquestes característiques les aconseguirem, tal i com explicarem més endavant, gràcies a la utilització d'una transformada multi-escala i un model estadístic acurat del comportament dels coeficients de la transformada.

3.2 Funcionament

El JPEG2000 segueix un sistema de blocs, amb les mateixes etapes que s'han descrit en la secció 2.5, tal i com es mostra a la Figura 3.1.

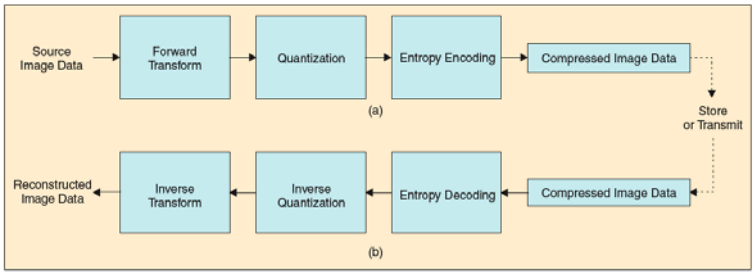


Figura 3.1: Diagrama de blocs de JPEG2000 [1]. a) Codificador b) Descodificador

Un petit resum del seu funcionament és:

- La imatge es descompon en components, si és una imatge en colors. En el cas de les imatges en blanc i negre només hi haurà una component.
- Les components es poden, de manera opcional, descompondre en tires rectangulars (*tiles*).
- La transformada wavelet s'aplica a cada tira, que es descompon en diferents nivells de resolució. JPEG2000 utilitza la transformada wavelet discreta (DWT), a diferència de JPEG que utilitza la transformada de cosinus discreta (DCT). Aquesta diferència és un dels punts claus de JPEG2000, més endavant s'explicaran les característiques de la transformada wavelet en la secció 3.2.1.
- La descomposició per nivells està feta per a subbandes de coeficients que descriuen les característiques espai-freqüencials i d'escala.
- Les subbandes de coeficients es quantifiquen i es col·loquen en matrius de *code-blocks*.
- Els *bit-planes* dels coeficients en un *code-block* es codifiquen entròpicament.
- La codificació es pot fer de tal manera que certes regions d'interès puguin ser codificades amb una qualitat major que la resta.
- S'afegeixen marcadors al *bit-stream* per permetre la "*error resilience*".
- El *code-stream* té una capçalera principal al principi que descriu la imatge original i els estils de codificació i descomposició que s'han utilitzat per localitzar, extreure, descodificar i reconstruir la imatge amb la resolució, fidelitat, regió d'interès i altres característiques desitjades.

3.2.1 Transformada wavelet

Les wavelets són funcions matemàtiques que divideixen les dades en diferents components freqüencials i estudien cada component amb una resolució corresponent a la seva escala.

Les wavelets són funcions que satisfan certs requeriments matemàtics i que s'utilitzen en la representació de dades i d'altres funcions. L'aproximació utilitzant la superposició de funcions no és nova, ja existeix des de principis de 1800, quan Fourier va descobrir que podia superposar sinus i cosinus per a representar altres funcions. No obstant, en l'anàlisi wavelet, l'escala que utilitzem per mirar les dades juga un paper molt important. Els algoritmes de wavelets processen les dades a diferents escales o resolucions. El procediment en l'anàlisi per wavelets és agafar una funció prototip wavelet, anomenada wavelet analítica o wavelet mare. El conjunt de wavelets que s'utilitzarà per representar la funció, s'obté a partir de la wavelet mare, mitjançant escalats i translacions. El nombre de wavelets existent és enorme, des de wavelets molt simples com la Haar [22] fins a wavelets molt més complexes com la Daubechies 9/7 [23].

Transformada wavelet contínua

L'inici de la teoria wavelet va tenir lloc per la seva versió contínua, el que s'anomena la transformada wavelet contínua (CWT) i que s'expressa formalment de la manera següent:

$$\gamma(s,\tau) = \int f(t)\Psi^*_{s,\tau}(t)dt, \tag{3.1}$$

on * és el conjugat complexe. Aquesta equació mostra com una funció $f(t)$ es descompon en un conjunt de funcions base $\Psi_{s,\tau}(t)$, les anomenades wavelet mitjançant un producte escalar. Les variables s i τ, escala i translació, són les noves dimensions després d'aplicar la transformada wavelet.

Les wavelets es generen a partir d'una única wavelet base $\Psi(t)$, l'anomenada wavelet mare, a partir de l'escalat i la translació:

$$\Psi_{s,\tau}(t) = \frac{1}{\sqrt{s}}\Psi\left(\frac{t-\tau}{s}\right) \tag{3.2}$$

on s és el factor de l'escala, τ és el factor de la translació i el factor $1/\sqrt{s}$ serveix per normalitzar l'energia a les diferents escales.

El terme translació s'utilitza en relació a la col·locació de la finestra, ja que la finestra es mou a través del senyal. Aquest terme, òbviament, correspon a la informació temporal en el domini de la transformada. A diferència d'altres transformades, no tenim el paràmetre de la freqüència, en el seu lloc tenim l'escala. L'escala en l'anàlisi wavelet és similar a l'escala utilitzada en els mapes, on escales grans representen la visió global i escales petites els detalls. En termes de freqüència, baixes freqüències (escales grans) corresponen a una informació global del senyal, mentre que altres freqüències (escales baixes) correspon a la informació detallada en un patró amagat en el senyal. Escalar, com a operació matemàtica, correspon tan a dilatar com a comprimir el senyal. Així doncs, les escales grans corresponen a dilatar el senyal mare i les escales petites corresponen a comprimir-lo.

Cal destacar que en cap de les expressions anteriors s'ha especificat quina funció s'utilitzarà com a wavelet mare. Aquesta és una de les grans diferències que existeixen entre la transformada wavelet i la transformada de Fourier. La teoria de la transformada wavelet només especifica les propietats que han de complir les funcions, no quines han de ser.

Transformada wavelet discreta

La idea principal és la mateixa que per la CWT. El senyal transformat és una representació en escala de temps d'un senyal digital utilitzant tècniques de filtrat. La transformada wavelet contínua es calcula canviant l'escala de la finestra d'anàlisis, desplaçant la finestra en el temps, multiplicant el senyal i integrant al llarg de tot el temps. En el cas discret, s'utilitzaran filtres amb diferents freqüències de tall per analitzar el senyal a diferents escales. El senyal passarà a través d'una sèrie de filtres passa-alts per analitzar les altes freqüències i, també, passarà a través d'una sèrie de filtres passa-baixos per analitzar les baixes freqüències.

La resolució del senyal, que és una mesura de la quantitat d'informació dels detalls en un senyal, variarà a partir de les operacions de filtrat, i l'escala variarà a partir de les operacions de delmat i interpolació. Delmar un senyal correspon a eliminar algunes mostres del senyal i interpolar consisteix en afegir noves mostres al senyal.

El procediment consistirà en passar el senyal discret $x[n]$ a través d'uns filtres digital amb una resposta impulsional $h[n]$. Filtrar un senyal correspon a l'operació matemàtica de convolució del senyal amb la resposta impulsional del filtre, tal i com es defineix a continuació:

$$x[n] \star h[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[n-k]. \tag{3.3}$$

Així doncs la DWT analitza els senyal a diferents bandes de freqüències amb diferents resolucions a partir de la descomposició del senyal en una aproximació i en detalls. La DWT utilitza dos conjunts de funcions, anomenades funcions d'escalat i funcions wavelet, que estan associades als filtres passa-baixos i els filtres passa-alts, respectivament. Com ja s'ha comentat, la transformada wavelet discreta és calcularà mitjançant una serie de filtres passa-baixos i passa-alts per on passarà el senyal discret en el domini temporal, tal i com es mostra a la Figura 3.2.

El senyal original $x[n]$ passa primer per un filtre passa-alt de mitja banda H i un filtre passa-baix G. Una vegada s'ha filtrat el senyal, la meitat de les mostres es poden eliminar d'acord amb el criteri de Nyquist. El criteri de Nyquist ens diu que si el senyal original té com a freqüència més alta ω, es requereix una freqüència de mostreig de 2ω radiants. Donat que el filtre té freqüència de tall $\omega/2$, es pot mostrejar a una freqüència de ω radiants i eliminar la meitat de les mostres sense perdre informació. L'operador del delmat $\downarrow k$ (*"down-sampling"*), converteix una seqüència d'entrada. $x[n]$, en una de sortida tal que $d[n] = x[kn]$, les sortides delmades s'anomenaran subbandes. En aquest cas, el delmat per dos divideix la resolució

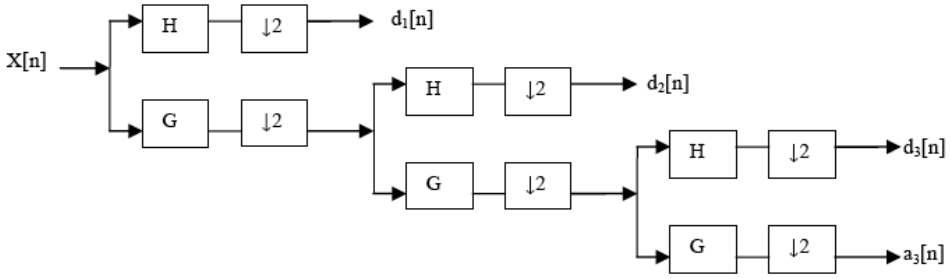


Figura 3.2: Arbre de descomposició de la wavelet de tres nivells

temporal ja que el senyal sencer es representarà a partir d'ara amb només la meitat de les seves mostres. De la mateixa manera que el filtre passa-baix de mitja banda elimina la meitat de les freqüències i divideix la resolució a la meitat, la delmació per dos dobla l'escala.

Transformada wavelet discreta de dos dimensions

En el cas de les imatges, que són senyals bidimensionals, s'haurà d'aplicar la transformada wavelet discreta bidimensional. És aquesta transformada la que utilitza JPEG2000 en el seu esquema.

Consisteix en aplicar la DWT al senyal original, $x[n, m]$, separadament a les columnes i després a les files, produint, en un primer nivell, quatre subbandes, de la $y_{0,0}[n, m]$ a la $y_{1,1}[n, m]$. La subbanda $y_{0,0}[n, m]$ prové de l'aplicació del nucli d'anàlisis passa-baix tant en la direcció horitzontal com vertical. D'aquesta manera ens referim a $y_{0,0}[n, m]$ com la subbanda LL. La subbanda $y_{0,1}[n, m]$ implica l'aplicació del nucli d'anàlisis passa-baix en la direcció vertical i del nucli d'anàlisis passa-alt en la direcció horitzontal, en referim a aquesta subbanda com a la subbanda HL (passa-alt horitzontalment). Les altres subbandes, $y_{1,0}[n, m]$ i $y_{1,1}[n, m]$, s'identifiquen com LH i HH, respectivament. A la Figura 3.3 es mostra un banc de filtres per una transformada de dos dimensions, del que s'obtenen les quatre subbandes que hem anomenat.

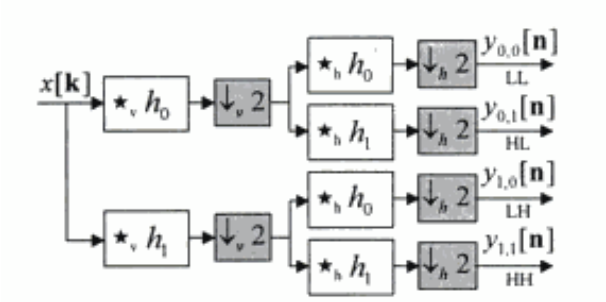


Figura 3.3: Banc de filtres pel càlcul de la DWT de dos dimensions [4]

El resultat de la transformada és una imatge descomposta en 4 sub-imatges de mida 4 vegades inferior cadascuna, tal i com es veu a la Figura 3.4.

- La subbanda LL (corresponent a la imatge superior esquerra) és una versió de menys resolució de la imatge original. Si es volen fer varis nivells de descomposició wavelet, és la que s'utilitzaria, en la segona etapa de filtrat, com a senyal d'entrada $x[k]$ de l'esquema que presenta la figura 3.3.
- La subbanda LH (imatge superior dreta) representa l'alta freqüència continguda en la direcció vertical i la baixa freqüència continguda en la direcció horitzontal . Respon més fortament a les arestes i segments horitzontals.
- La subbanda HL (imatge inferior esquerra) representa l'alta freqüència continguda en la direcció horitzontal i la baixa freqüència continguda en la direcció vertical. Aquesta subbanda respon a les arestes i segments verticals de les imatges.
- La subbanda HH (imatge inferior dreta) representa l'alta freqüència continguda tan en les direccions horitzontal com vertical. Respon principalment a les característiques orientades en diagonal.

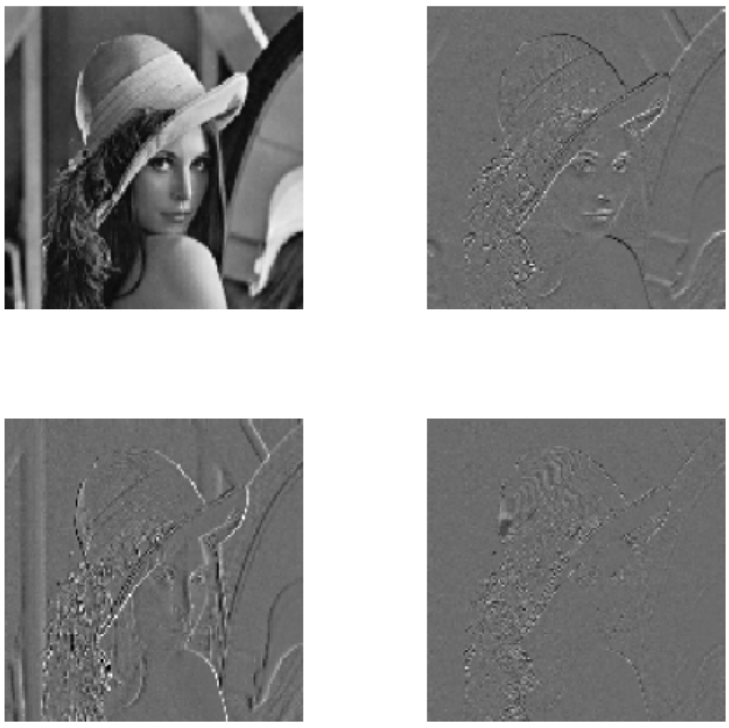


Figura 3.4: Resultat de l'aplicació d'una primera etapa de la DWT

Per una descomposició de D nivells, procedirem, com ja hem dit, en un primer pas de la DWT en dividir la imatge en quatre subbandes, anomenades LL_1, HL_1 (passa-alt horitzontal), LH_1(passa-alt vertical) i HH_1. En el següent pas de la DWT es descompon la subbanda LL_1, en quatre subbandes més, LL_2, HL_2, LH_2 i HH_2. El procés continua de manera similar per un determinat nombre de passos, D, produint un total de 3D+1 subbandes que representen la imatge original. A les Figures 3.5 i 3.6 es mostra la descomposició de 3 nivells.

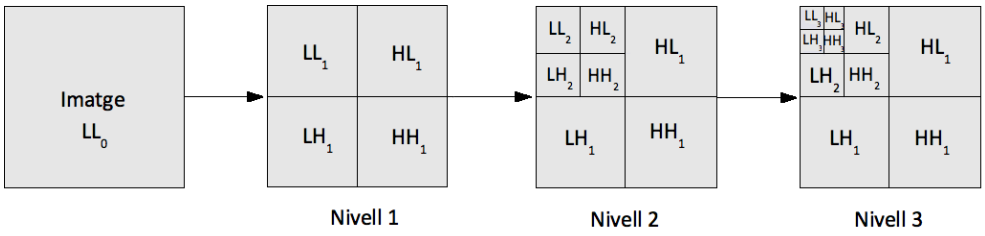


Figura 3.5: DWT de 3 nivells

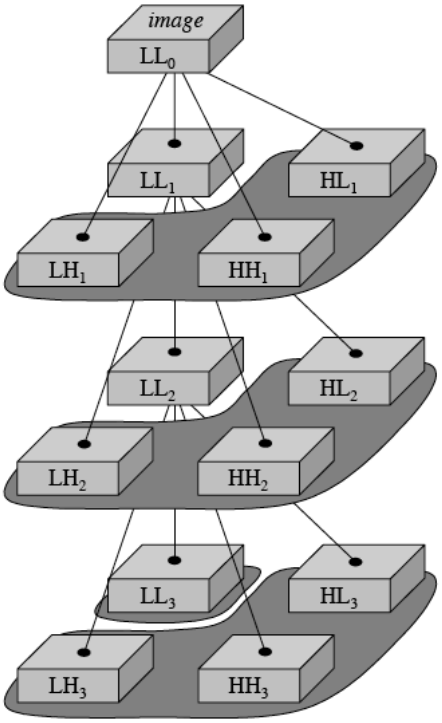


Figura 3.6: Descomposició de la DWT amb D=3 nivells [5]

La banda LL d'una descomposició uniforme és una versió de baixa resolució de la imatge. LL_d (amb $0 < d \leq D$) representa una família de versions de baixa resolució de la imatge original. La densitat de mostreig per la banda LL_d és 2^{-d} vegades que la de la imatge original en cada direcció, on $d = 1, 2, \ldots, D$. Clar que totes excepte la última d'aquestes imatges de baixa resolució seran resultats intermedis; només LL_D és realment una de les subbandes de la transformada en forma d'arbre. No obstant, cada una de les imatges en aquesta família de multi-resolució podrà ser recuperada mitjançant l'aplicació parcial del sistema de síntesis, d'aquesta manera LL_{D-1} es pot sintetitzar a partir de les subbandes LL_D, HL_D, LH_Di HH_D.

Aquesta propietat de multi-resolució és particularment interessant en les aplicacions de la compressió d'imatges, tema que ens ocupa en el present treball, ja que proporciona un mecanisme mitjançant el qual un *bit-stream* comprimit pot ser parcialment descomprimit per obtenir, successivament, versions de més resolució de la imatge original.

Aquesta propietat de multi-resolució, provinent de les característiques de la transformada wavelet discreta, dóna a JPEG2000 la característica de l'escalabilitat a nivell de resolució. Tal i com s'ha comentat, la subbanda LL_d és una versió de baixa resolució de la LL_{d-1}, amb la meitat d'amplada i llargada. D'aquesta manera tenim diferents versions amb diferents resolucions de la mateixa imatge, de la que la LL_0 és la versió de major resolució i la LL_D la de menor.

3.2.2 Quantificació

En JPEG2000, després de la transformació, tots els coeficients wavelet passen per un procés de quantificació escalar uniforme amb dead-zone. La quantificació és el procés pel qual els coeficients es redueixen en precisió. Aquesta operació té pèrdues, a menys que el pas de quantificació sigui 1 i els coeficients siguin enters. Cada coeficient $a_b(u,v)$ de la subbanda b és quantificat al valor $q_v(u,v)$ d'acord amb l'expressió:

$$q_b(u,v) = sign(a_b(u,v)) \left\lfloor \frac{|a_b(u,v)|}{\Delta_b} \right\rfloor \tag{3.4}$$

Per tant, la quantificació s'aconsegueix dividint la magnitud de cada coeficient pel pas de quantificació (Δ_b), és a dir, per la mida dels intervals, arrodonint a la baixa i mantenint el signe del coeficient. El fet que la quantificació tingui dead-zone significa que l'interval de quantificació al voltant del zero és simètric i més gran que els altres passos de quantificació, que estan repartits uniformement entre el llindar de la dead-zone i el rang superior del quantificador, com es mostra a la Figura 3.7.

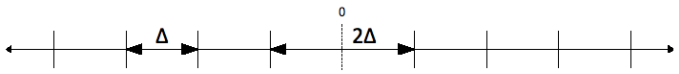


Figura 3.7: Llindars del quantificador amb dead-zone

El valor del pas de quantificació és relatiu al rang dinàmic la subbanda en que es trobi, en altres paraules, l'estàndard JPEG2000 permet una mida diferent del pas de quantificació per cada subbanda. No obstant, només es permet un valor del pas de quantificació per subbanda. El rang dinàmic depèn del nombre de bits utilitzats per representar la tira de la imatge original i de la transformada wavelet escollida.

3.2.3 Codificació de bloc

Els mecanismes de codificació entròpica de JPEG2000 són més eficients i tenen una major flexibilitat a l'hora d'obtenir diferents representacions de la imatge que els de JPEG. El seu algoritme té una bona compressió sense pèrdues, també una bona compressió de les imatges binàries i de imatges amb una baixa profunditat de bit, i uns *bit-streams* on hi ha incrustats bones versions amb pèrdues de la imatge juntament amb la versió sense pèrdues.

Una part molt important de JPEG2000 és l'escalabilitat, per tal de poder donar resposta a la gran varietat d'aplicacions. Aquesta propietat està relacionada amb l'ordenació de la informació en el *bit-stream*. De totes formes, la codificació entròpica juga un paper important en l'escalabilitat, ja que les dependències creades en aquest procés poden eliminar un o més graus d'escalabilitat.

Com ja s'ha descrit al punt 3.2.1, la propietat de multi-resolució ve de la transformada wavelet discreta. Un *bit-stream* escalable en termes de resolució és aquell que es pot obtenir descartant porcions de les dades comprimides que no desitgem, corresponents a la informació per crear la imatge d'alta resolució, quedant-nos només amb la informació necessària per a crear la imatge a més baixa resolució. La representació de baixa resolució ha de ser idèntica a la que obtindríem si haguéssim comprimit directament una versió de la imatge de baixa resolució.

Un altre tipus de escalabilitat s'obté de descartar elements del *bit-stream*, per tal d'obtenir versions d'una qualitat menor i, per tant, distorsió major, és per aquest motiu que anomenem aquest tipus d'escalabilitat, escalabilitat de distorsió. Per dur-ho a terme s'ha de produir una codificació per bit-planes, de manera que es codifiquen els bits d'un a un de més significants a menys. Així que es poden descartar o bé certs bit-planes de grups de coeficients, o bé grups sencers de coeficients, depenent del que aporta més reducció de la distorsió.

Malauradament, degut a les possibles dependències introduïdes en el procés de codificació, la combinació de transformada wavelet amb codificació per bit-planes no garanteix que s'obtinguin bit-streams que siguin a la vegada escalables per resolució i per distorsió, ja que si tenim el bit-stream ordenat per tal de descartar els bits que redueixen menys la distorsió, no tindrem en compte l'ordenació per resolució, i si posem tots els bits corresponents a components d'alta resolució al final del bit-stream per a poder-los descartar i obtenir escalabilitat per resolució, no podrem fer escalabilitat per distorsió.

Com que és impossible tenir el bit-stream ordenat al mateix temps per a que truncar-lo doni escalabilitat per resolució òptima i que també doni escalabilitat per a distorsió òptima, el JPEG2000 intenta solucionar el problema dividint les subbandes en petits blocs que seran codificats de manera independent i introduint capes de qualitat, que donaran referències per a reordenar el bit-stream segons l'escalabilitat desitjada per a cada aplicació, sense necessitat de descodificar i re-codificar.

Com que no és possible separar el fet de codificar i d'ordenar la informació, ja que una codificació eficient introdueix forçosament dependències, es pot intentar solucionar el problema dividint les subbandes en petits blocs que seran codificats de manera independent. La mida dels blocs determinarà el grau de sacrifici de l'eficiència que es tindrà a canvi de la flexibilitat per ordenar la informació en el bit-stream final. És aquest el sistema que utilitzarà JPEG2000 basat en el concepte de *Codificació de bloc incrustada amb truncament òptim* (Embedded Block Coding with Optimal Truncation, EBCOT).

Algoritme EBCOT

Segons el EBCOT adoptat per JPEG2000, cada subbanda es divideix en "petits" blocs que anomenarem *code-blocks* (tal i com es mostra a la Figura 3.8), que seran codificats de forma independent produint un bit-stream elemental. Cada bloc genera bit-streams independents que s'agrupen en capes de qualitat (*quality layers*). Per tal de generar aquestes capes de qualitat, aquests bit-streams independents són dividits en un gran nombre de fragments. Llavors el compressor és lliure de moure aquests fragments, sempre que respecti els seu ordre dins dels blocs.

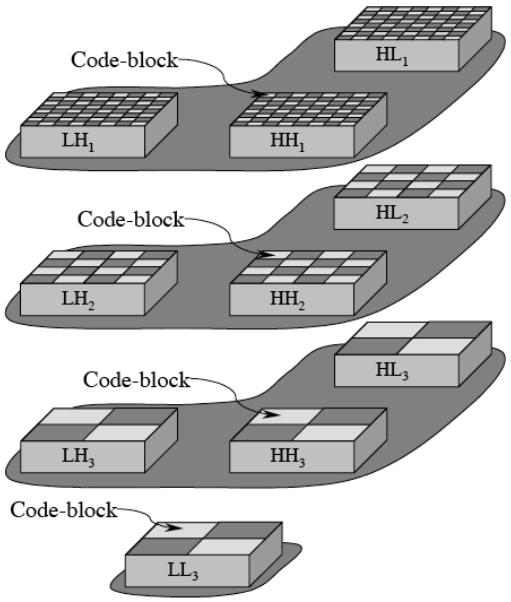


Figura 3.8: Divisió de les subbandes en code-blocks [5]

Els bit-streams independents poden truncar-se al final de cada fragment, aquests punts són els anomenats *punts de truncament*. La selecció d'aquests punts de truncament porta a un problema a l'hora d'ordenar-los, ja que afecta les propietats de la taxa de distorsió de la imatge global. En un esquema de codificació per bit-planes, els punts finals dels bit-planes, són punts de truncament naturals. No obstant, el fet de poder trobar molts més punts de truncament útils és la clau de l'èxit del paradigma EBCOT, que utilitza tres passes de codificació (*significance pass*, *magnitude refinement pass* i *clean-up pass*) per a cada bit-plane. El fnal de cada passa de quantificació és un candidat a punt de truncament del bit-strem JPEG2000.

Per aconseguir una màxima reducció de les dependències entre coeficients, i per tant més eficàcia en la compressió, la seqüència en la que es codifiquen els bits de diferents mostres ha de ser depenent de les dades. Aquesta seqüència tendeix a codificar la informació més important (en el sentit de reduir la distorsió) el més aviat possible.

Com ja hem dit, cada subbanda es divideix en code-blocks (per exemple de 64x64 o de 32x32 mostres). Cada code-block, B_i, es codifica independentment produint un bit-stream, c_i. Restringirem l'atenció a un nombre finit de punts de truncament, $Z_i + 1$, per code-block B_i, amb unes longituds, $L_i^{(z)}$, tals que

$$0 = L_i^{(0)} \leqslant L_i^{(1)} \leqslant \cdots \leqslant L_i^{(Z_i)}.$$

Se suposa que la distorsió de la imatge reconstruïda es pot representar com la suma de les contribucions de les distorsions, $D_i^{(z)}$ que provenen de cada bloc B_i, si el seu bit-stream elemental és trunca a la longitud $L_i^{(z)}$.

Com que cada code-block està comprimit de manera independent, som lliures d'escollir quina política utilitzarem per truncar els bit-streams, clar que la tria més atractiva és aquella que minimitzi la distorsió global. La selecció dels punts de truncament s'ha de realitzar després que tots els code-blocks siguin comprimits, moment en el qual es coneixen les longituds disponibles i les seves distorsions. Per aquest motiu ens referim a l'estratègia del truncament òptim com una optimització post-compressió de la taxa de distorsió.

El bit-stream comprimit total es construeix amb l'empaquetament de contribucions de varis bit-streams dels code-blocks. El paquet més simple és aquell que té els bit-streams dels blocs truncats concatenats amb una etiqueta de les longituds per identificar la contribució de cada code-block, com s'il·lustra a la Figura 3.9 . Aquest paquet simple és escalable en resolució ja que cada nivell de resolució està format per una col·lecció de code-blocks ben definits i identificats amb la mitja de les longituds. També té un grau de escalabilitat espacial, ja que cada code-block influeix només en una regió finita de la imatge reconstruïda.

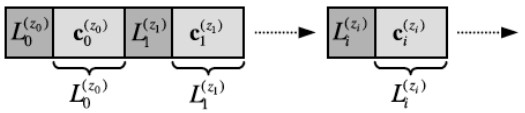


Figura 3.9: Paquet simple [5]

El problema és que el paquet simple no és escalable en distorsió, ja que no proporciona informació per construir de manera òptima un paquet més petit amb contribucions de code-blocks. Com ja hem comentat, per resoldre aquest problema l'algoritme EBCOT introdueix els nivells de qualitat. Cada nivell de qualitat conté contribucions de code-blocks optimitzades que minimitzen la distorsió, tenint en compte la restricció de la longitud. Encara que conceptualment cada nivell de qualitat conté contribucions de cada code-block, s'ha de fer notar que pot ser que algunes, o fins i tot totes, estiguin buides. Un paquet escalable en distorsió ha de construir-se incloent suficient informació per identificar la contribució feta per cada code-block.

3.2.4 Codificació per bit-planes

La codificació dels code-blocks en JPEG2000 es fa per bit-planes. Un bit-stream es pot formar de la següent manera. Primer de tot, codificant el bit-plane de magnitud més significatiu juntament amb el signe de qualsevol mostra no nul·la. Llavors codificant el següent bit-plane de magnitud més significatiu incloent el signe de qualsevol nova mostra no nul·la. Procedim d'aquesta manera per cada bit-plane de magnitud incloent el signe d'aquelles mostres no nul·les.

Hi ha gran varietat de tècniques que poden ser utilitzades per codificar els bits de magnitud i signe. Tot i així, un codificador eficient hauria d'utilitzar la redundància que existeix generalment entre els successius bit-planes. Aquest objectiu es pot aconseguir utilitzant codificació aritmètica condicional, que requereix la definició de l'ordre de l'exploració i del model de contextos.

Codificació per entropia

La codificació per entropia s'aconsegueix mitjançant un sistema de codificació aritmètic que comprimeix els símbols binaris relatius a un model de probabilitat adaptativa associat amb els diferents contextos. Concretament, JPEG2000 utilitza el codificador MQ. El codificador aritmètic MQ és un codificador en el que es representa la codificació com un valor dins d'un interval. Partint d'un rang de valors donat, es divideix aquest rang en intervals proporcionals a la probabilitat d'aparició d'un símbol i es selecciona com a nou rang l'interval del símbol aparegut. Aquest interval es va refinant de forma successiva a mesura que van apareixent nous símbols. Un cop processats tots els símbols es tria un valor de l'interval resultant i es diu que aquest valor és la codificació aritmètica de l'entrada.

El codificador MQ treballa amb els contextos que s'han anat generant a la fase anterior per tal d'estimar millor els rangs de probabilitats. JPEG2000 utilitza un nombre restringit de contextos per un determinant tipus de bit. Això proporciona una ràpida adaptació de la probabilitat i fa decréixer el cost de segment codificats independentment. Els models de context es reinicien al començament de cada code-block i el codificador aritmètic acaba al final de cada bloc.

3.2.5 Escaneig fraccional dels bit-planes

En aquesta secció s'explicarà en quin ordre es visiten les mostres quan s'explora un determinat bit-plane, aquest ordre és depenent de les dades. Per cada bit-plane, la codificació es realitza en un seguit de passes diferents, que anomenem "fractional bit-planes". Cada passa de codificació implica l'exploració de les mostres del code-block en tires. Les mostres només es miren en una única passa, de manera que la mostra ja codificada a la primera passa s'esquiva en les altres dues passes. Els "fractional bit-planes" són unitats indivisibles i, per tant, determinen els candidats a punts de truncament pel code-stream.

Significance propagation pass Es diu que una posició d'un bit-plane és significant si és 1 per primera vegada en aquella posició (és a dir, un coeficient passa a ser significant en un determinat bit-plane si no havia passat a ser significant encara i en aquesta passa, és 1). En la primera passa en cada bit-plane s'inclou qualsevol mostra que sigui insignificant però que té com a mínim un dels seus 8 veïns directes significants.

Magnitude refinement pass En aquesta fase es codifiquen la magnitud dels bits de qualsevol mostra que ja hagi estat significant en el bit-plane anterior.

Clean-up pass Aquesta passa final inclou totes les mostres que no han estat codificades en el bit-plane actual.

Propietats de la rate-distortion

Com s'ha comentat abans, els punts de truncament disponibles seran els corresponents al final de cada passa de codificació. Cadascuna de les tres passes aporten guanys a l'hora d'optimitzar el parell rate-distortion. Just després de fer la codificació per bit-planes és el moment en el que es decideix separar la informació rellevant que cal preservar i descartar la resta. Això es pot fer ja que es disposa de tots els bit-streams degudament etiquetats amb els valors d'energia que aporten i la quantitat d'informació que requereixen per fer-ho (Rate/Distortion).

Els punts de truncament candidats per un determinat bit-stream són aquells que pertanyen al convex hull de la corba rate-distortion descrita per tots els punts de truncament disponibles. Cada coding pass de cada bit-plane produeix doncs un punt de truncament candidat, però de tots aquest punts candidats cal escollir només els que es troben exactament a la corba del convex hull , com el que es mostra a la Figura 3.10. Els punts que es troben per damunt de la corba del convex hull no haurien de ser seleccionats per un algoritme òptim. Per tal de trobar els punts de truncament candidats de cada bloc només necessitem doncs guardar els rates i els pendents rate-distortion.

3.2.6 Resum

En aquesta secció s'han presentat les parts bàsiques de JPEG2000, sobretot aquelles que ens serà útil conèixer per entendre parts posteriors del projecte.

Hem vist com una gran avantatge d'aquest estàndard és el fet de les diferents escalabilitats que permet, gràcies al ús de la transformada wavelet i el mètode de codificació per nivells de qualitat i blocs. Aquesta escalabilitat es dóna a nivell de qualitat, resolució, components i espacial, això permet una transmissió progressiva de la imatge a aquests nivells.

Dóna també una millor compressió a baixes qualitats a diferència d'altres estàndards, com JPEG, en els que es patien deficiències molt notables que es notaven en la presència d'artefactes visuals.

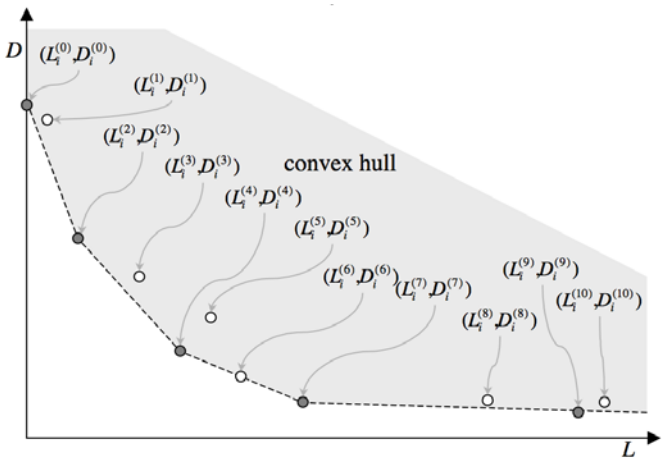


Figura 3.10: Convex hull dels parells rate-distortion per el bloc i. Els punts sòlids corresponen als candidats a punts de truncament. [5]

3.3 BOI

BOI és una implementació de l'estàndard JPEG2000 desenvolupada pel GICI (Grup de Codificació Interactiva d'Imatges) del Departament d'enginyeria de la informació i comunicacions (DEIC) de la Universitat Autònoma de Barcelona, [9]. BOI està dissenyat i programat amb l'objectiu de proporcionar una bona base per al test i el desenvolupament de noves idees dins del sistema de codificació de JPEG2000.

Com ja s'ha comentat BOI no és només una nova implementació de l'estàndard, sinó que el seu objectiu és proporcionar un marc pel test de nous mètodes per comprimir i manipular imatges. Les operacions principals de codificació del BOI estan basades en el nucli dels sistema de codificació de JPEG2000: la divisió de la imatge per blocs, la codificació per bit planes, la codificació aritmètica, etc... per tant, BOI pot proporcionar un arxiu JPEG2000. No obstant, el disseny permet la fàcil incorporació de nous algoritmes i tècniques de compressió.

3.3.1 Disseny i estructura

La motivació en el disseny i el desenvolupament de BOI va ser la de generar un esquema de mòduls que funcionessin de manera independent. Per tal d'entendre-ho millor, tots els mòduls segueixen el mateix esquema i només s'utilitzen eines del llenguatge de programació bàsiques.

BOI s'estructura en quatre paquets principals o fases de compressió:

1. **Transformació de les dades**: operacions de preprocessament, transformació wavelet i quantificació dels coeficients.
2. **Codificació de les dades**: codificació dels coeficients quantificats que generen petits *code-streams*.

3. **Reorganització del codestream**: selecció dels millors codestreams que seran classificats i ordenats en capes de qualitat utilitzant el paradigma EBCOT.

4. **Generació de l'arxiu:** generació de les capçaleres principals, de les capçaleres dels paquets i l'escriptura de l'arxiu utilitzant un ordre progressiu especific i introduint-ne les capçaleres.

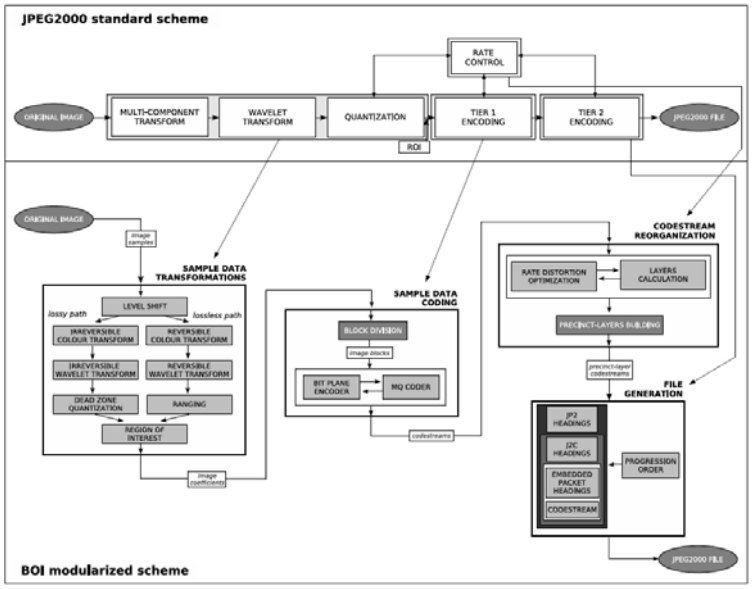


Figura 3.11: Comparació entre l'esquema de l'estàndard JPEG2000 i la implementació de BOI [10]

La Figura 3.11 mostra les relacions entre les fases clàssiques de JPEG2000 i les utilitzades en el BOI. Com es pot veure a l'esquema, les quatre fases contenen classes que duen a terme les operacions bàsiques de l'esquema de compressió JPEG2000. En el primer paquet es realitzen les operacions de transformació de multi-components, la transformada wavelet i la quantificació. En el segon es realitza la codificació que genera els codestream codificats. En l'estàndard JPEG2000, el control de la taxa gestiona la quantificació, però en el BOI es realitza en el tercer paquet d'operacions. Finalment l'última fase conté les operacions necessàries per a generar l'arxiu i incrustar les capçaleres en el codestream final.

Com s'explicarà més endavant, en aquest projecte de les quatre fases enumerades anteriorment,ens centrarem en el punt 3, corresponent a la reorganització, i introduirem alguns canvis en el punt 2, sobre la codificació de les dades.

En la Figura 3.12 es mostra les classes del BOI. En una primer pas, es carrega la imatge i es passen al codificador (Coder) els paràmetres. El codificador és la classe principal de l'aplicació. Primer es comproven tots els paràmetres i després es fa la compressió cridant les classes corresponents.

Totes les fases de compressió es divideixen en mòduls simples que interaccionen entre ells. Cada mòdul està programat de manera independent i té els seus propis paràmetres. La

interacció entre classes es duu a terme mitjançant estructures que es passes entre les classes, com es veu a l'esquerra de la figura 3.12. El codificador gestiona aquestes estructures i controla la seva destrucció.

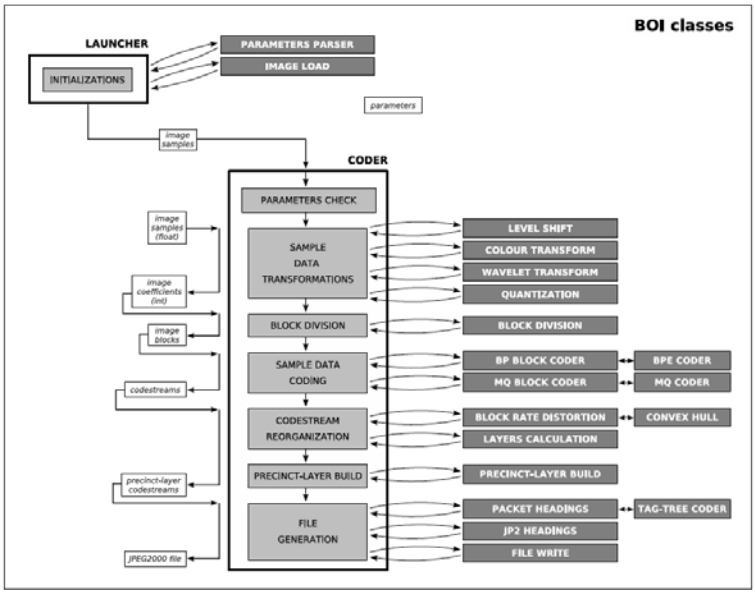


Figura 3.12: Disseny de les classes de BOI en el codificador [10]

Pel que fa al descodificador, aproximadament existeixen les mateix classes i paquets que en el codificador, a excepció del paquet BOI Codestream Reorganization, que reorganitza el codestream comprimit. Totes les classes del descodificador són rèpliques del codificador però amb els algoritmes inversos.

3.3.2 Ús i paràmetres

BOI està dividit en 2 aplicacions principals: el codificador i el descodificador. El seu ús és molt simple, per a comprimir una imatge només caldrà fer ús del codificador especificant alguns paràmetres. Una vegada tenim el codestream comprimit, es pot utilitzar el descodificador per recuperar la imatge original i desar-la amb altres formats.

En general, la utilització típica de l'aplicació seria:

```
# BOIcode -i lena.pgm -o lenaCompressed
# BOIdecode -i lenaCompressed.jpc -o lenaRecovered.pgm
```

BOI incorpora diferents paràmetres que poden utilitzar-se en el codificador i en descodificador, per tal d'utilitzar les diferents funcionalitats de l'aplicació. La documentació completa dels paràmetres està disponible en l'ajuda de la implementació.

A continuació s'anomenen els paràmetres que s'han utilitzant en aquest projecte.

Paràmetres del BOIcode

–**inputImage -i** Imatge d'entrada. Els formats vàlids són: pgm, ppm, pbm, jpg, tiff, png, bmp, gif, fpx.

–**outputfile -o** Nom d'arxiu de sortida (sense extensió)

–**layerCalculationType -lt** Tipus de càlcul per a construir els nivells de qualitat per a una assignació de velocitat. Podem donar tres valors a aquest paràmetre: 0, 1 i 2. 0-Genèric: l'assignació de la taxa i els nivells es construeixen genèricament utilitzant mesures de la taxa de distorsió (definit per EBCOT, explicat en el punt 3.2.3). Tots els nivells tenen aproximadament la mateixa mida. 1- *BitPlane*: l'assignació de la taxa i els nivells es construeixen des del *bit-plane* MSB al LSB sense utilitzar les mesures de distorsió del BPE. 2- *CodingPasses*: en aquest cas es construeixen a partir de les *coding passes.*

–**targetBytes -tb** Nombre de bytes pel bitstream comprimit.

–**cpi -cpi** Utilització o no de l'algoritme de codificació CPI. És un paràmetre booleà on 0 significa que no s'utilitzarà i 1 que sí que s'utilitzarà. Per defecte el paràmetre pren el valor 1.

Per l'execució de les nostres proves posarem el valors dels paràmetres –layerCalculationType i –cpi a 0.

3.3.3 Classes

A continuació es descriurà molt breument algunes de les classes i paquets de la implementació BOI. Només parlarem d'aquelles classes que s'han utilitzat en aquest projecte, les que s'han modificat i les que són importants per entendre parts del que s'explicarà en altres punts de la memòria.

BOICode És la classe principal de l'aplicació de codificació. S'encarrega d'agafar els arguments, carregar la imatge i executar el codificador.

Coder És la classe principal de BOICoder. Rep tots els paràmetres, els valida i executar la codificació.

BlockCode Aquesta classe rep els blocs de la imatge i hi aplica la codificació de bloc definida en l'estàndard JPEG2000. La classe que codifica cada bloc és BPECoder.

BPECoder Aquesta classe rep les mostres d'un bloc i les codifica produint un bytestream JPEG2000.

BlockConvexHull Aquesta classe rep els errors i les longituds de la codificació dels blocs i calcula els seus *convex hulls.*

CH Aquesta classe realitza el càlcul incremental del convex hull i de les pendents per tal de trobar els punts de truncament.

GiciAnalysis És el paquet que conté algunes de les classes que permeten comparar dues imatges. Dins d'aquest paquet, s'ha utilitza la següent classe:

ImageCompare Aquesta classe, pertanyent al paquet GiciAnalysis, rep dues imatges i calcula el valor de les mesures MSE i PSNR.

Concretament, en el projecte s'ha modificat la classe BlockCode, com s'explicarà més endavant. També s'ha creat una nova classe, tenint com a base la classe ImageCompare.

3.3.4 Estructura de les dades

Les diferents classes del BOI interaccionen mitjançant estructures, la majoria de les quals són matrius multi-dimensionals. Cada índex de les matrius indica la component, el nivell de resolució, la subbanda, el bloc, etc... amb petites variacions entre les estructures. Aquestes matrius multi-dimensionals s'utilitzen per definir les mostres de la imatge, els blocs, els codestream, etc...

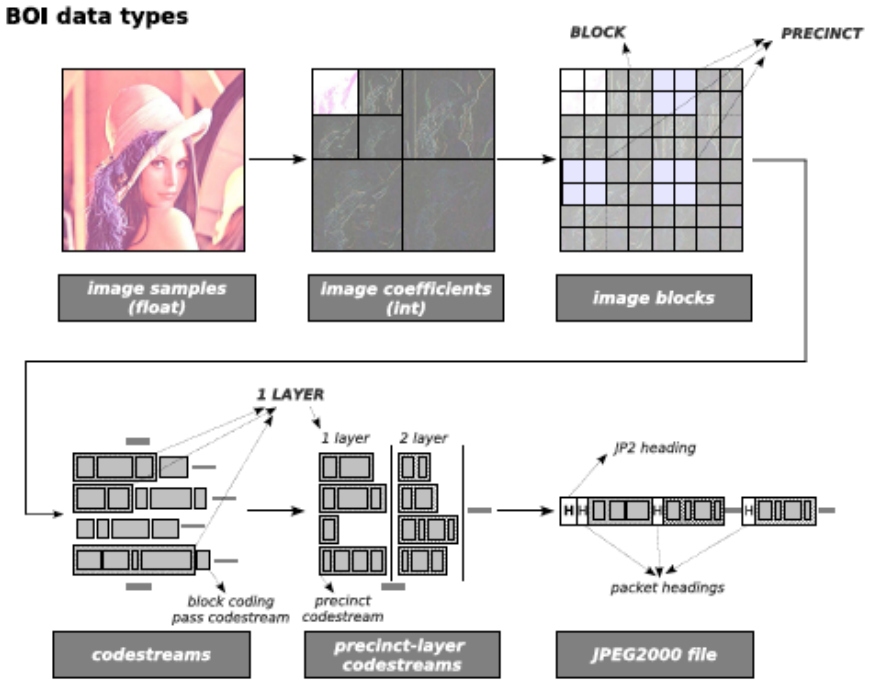


Figura 3.13: Tipus de dades utilitzades per BOI [10]

A l'esquerra de la figura 3.12 es mostraven les dades que utilitzen les classes per interaccionar, a la Figura 3.13 es mostren aquestes estructures amb més detall. La primera de les sis estructures és *image sample*, que s'utilitza per emmagatzemar la imatge original. La transformada wavelet reorganitza aquestes mostres en diferents nivells de resolució i subbandes. Llavors la

quantificació els converteix en enters (*image coefficients*) perquè les següents fases utilitzen aquest tipus de dades.

Abans de la codificació, la divisió per blocs separa els coeficients en petits conjunts anomenats *image blocks*. Cada bloc es codifica independentment i produeix petits codestreams amb una distorsió associada. Aquesta mesura s'utilitza per calcular els nivells de qualitat i produir els *precinct-layer codestreams,* construïts seguint un determinat ordre i produint un arxiu JPEG2000.

Pel que fa al descodificador, els noms de les variables és mantenen per tal de facilitar la compressió de l'usuari.

La variable que més utilitzarem al projecte és la de *imageBlocks,* és per aquest motiu que la descriurem en més detall.

int[][][][][][][][][][][] imageBlocks
És una matriu multi-dimensional que conté els coeficients de la imatge dividida per blocks. Els índexs són *imageBlocks[z][rLevel][subband][yBlock][xBlock]* i el seu significat:

z: component de la imatge

rLevel: nivell de resolució. 0 és la subbanda LL, i 1, 2, ... representa el següent començant amb el més petit.

subband: representa la subbanda on 0 - HL, 1 - LH, 2 - HH (si *rLevel* == 0 –> 0 - LL)

yBlock: la fila del bloc en la subbanda

xBlock: la columna del bloc en la subbanda

y: la fila de la mostra

x: la columna de la mostra

Una altra de les variables, que no s'ha esmentat encara, que tindrà una especial importància al llarg del projecte serà la corresponent als *BCErrors*.

long[][][][][][][][][][][] BCErrors
És una matriu multi-dimensional que conté l'error de cada coding pass dels blocs.
El significat dels seus índex és *BCErrors [z] [rLevel] [subband] [yBlock] [xBlock] [bitPlane] [codingPass]* i el seu significat:

z: component de la imatge

rLevel: nivell de resolució. 0 és la subbanda LL, i 1, 2, ... representa el següent començant amb el més petit.

subband: representa la subbanda on 0 - HL, 1 - LH, 2 - HH (si $rLevel == 0 \rightarrow$ 0 - LL)

yBlock: la fila del bloc en la subbanda

xBlock: la columna del bloc en la subbanda

bitPlane: bit-plane de l'algoritme de codificació de blocs (0 és el bit-plane més significatiu)

codingPass: corresponent a les passes de codificació on 0 és la passa Significance Propagation, 1 és la passa Magnitude Refinement i 2 és la passa de Cleanup.

Serà en aquesta variable on guardarem els errors associats a cada coding pass i obtinguts amb els coeficients normalitzats de que parlarem en el capítol 5.

Per tal de facilitar la integració de nous algoritmes dins del BOI i optimitzar la seva funcionalitat es faran servir classes i variables ja definides. Per la implementació de noves classes serà de molta utilitat fer servir com a base estructural, classes ja implementades de funcionament similar. També s'utilitzaran variables del programa com a base per crear-ne de noves. Aquest fet ens permetrà reduir el temps de programació, afectarà poc a l'estructura del programa inicial i facilitarà la integració.

Capítol 4

SSIM

Els senyals d'imatges naturals estan altament estructurats, és a dir, els seus píxels presenten una dependència important, especialment aquells que són pròxims. Aquestes dependències aporten informació molt important sobre l'estructura dels objectes que apareixen. Sembla ser que els sistema visual humà està adaptat per extreure la informació estructural del seu camp de visió. D'això se'n dedueix que una mesura basada en l'estructura de les imatges pot donar una bona aproximació a l'hora de percebre la distorsió de les imatges.

La mesura més utilitzada per tal de mesurar mètricament la qualitat d'una imatge ha estat l'error quadràtic mig (*Mean Squared Error*, MSE), així com la relació pic de senyal a soroll (*Peak Signal-to-Noise Ratio*, PSNR). Aquestes mesures són fàcils de mesurar i tenen un sentit físic però, tal i com es comenta a molts articles [11, 12, 13] , aquesta mesura no té massa a veure amb la qualitat visual de les imatges.

L'índex SSIM és una mesura sobre la semblança estructural (Structural SIMilarity) que existeix entre dues imatges. Aquesta mesura ha estat desenvolupada per Wang, Bovik, Sheikh and Simoncelli a [2]. L'SSIM compara la intensitat dels píxels normalitzada per la luminància i el contrast.

Existeixen altres mesures, a part de l'SSIM, que estudien la qualitatde les imatges tenint en compte la percepció del sistema visual humà, com la proposada per Winckler a [17] o les estudiades a [18].

4.1 Definició

En [2], es proposa una nova mesura de qualitat. L'índex SSIM és un combinació de tres comparacions: luminància, contrast i estructura.

$$SSIM(x,y) = [l(x,y)^{\alpha} \cdot [c(x,y)]^{\beta} \cdot [s(x,y)]^{\gamma}. \tag{4.1}$$

La luminància de la superfície d'un objecte que està sent observat és el producte de la il·luminació i la reflectància, però les estructures dels objectes en una escena són independents de la il·luminació. En conseqüència, per tal d'estudiar la informació estructural en una imatge, s'haurà separar la influencia de la il·luminació. Es defineix la informació estructural d'una imatge com aquells atributs que representen l'estructura dels objectes en una escena, de manera independent a la mitjana de la luminància i el contrast. També es tindrà en compte que aquestes dues mesures poden variar durant l'escena, per tant s'utilitza una mesura local de la luminància i el contrast en la seva definició.

A la Figura 4.1 es mostra un diagrama de la mesura proposada, on $\mathbf{x}$ i $\mathbf{y}$ són dues senyals de imatges no negatives, que s'han alineat entre elles.

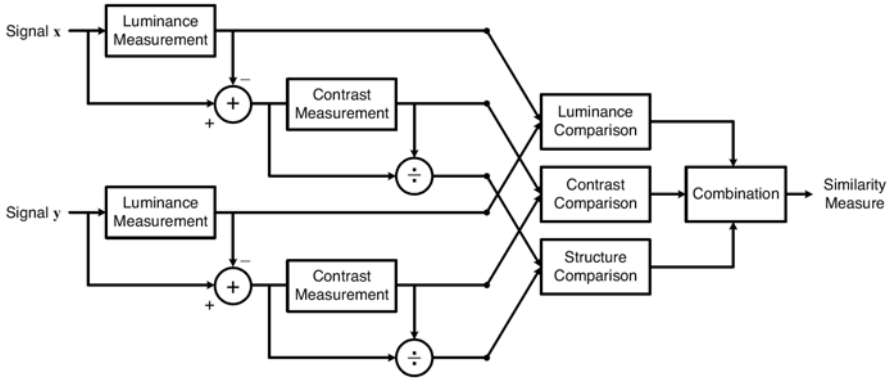


Figura 4.1: Diagrama del sistema de la mesura de similitud estructural [2]

La comparació de la luminància està definida com

$$l(x,y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{4.2}$$

on μ_x i μ_y són la mitja de la luminància de cadascuna de les imatges i la constant C_1 s'ha inclòs per evitar problemes d'estabilitat. S'escull com a valor $C_1 = (K_1 L)^2$, on L és el rang dinàmic dels valors dels píxels (255 per a imatges de 8 bits en escala de grisos) , i on $K_1 \ll 1$ és una petita constant.

La funció de la comparació del contrast pren una forma similar

$$c(x,y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \tag{4.3}$$

on σ_x i σ_y són la desviació estàndard de cada senyal i de manera equivalent, $C_2 = (K_2 L)^2$, i $K_2 \ll 1$.

Finalment la comparació estructural es defineix tal i com segueix.

$$s(x,y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3} \tag{4.4}$$

on, de la mateixa manera que en l'expressió anterior, σ_x i σ_y són la desviació estàndard de cada senyal i σ_{xy} és la correlació; i C3 és una constant que, igual que C2, depèn del quadrat del valor màxim de les mostres de la imatge.

En totes les expressions definides anteriorment, la mitja, la desviació estàndard i la correlació poden ser estimades en la forma discreta següent:

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{4.5}$$

$$\sigma_x = \left(\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)^2\right)^{\frac{1}{2}} \tag{4.6}$$

$$\sigma_{xy} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y). \tag{4.7}$$

Es substitueixen les expressions 4.2 ,4.3 i 4.4 a l'expressió general 4.1. Per tal de simplificar-les es defineixen els paràmetres $\alpha = \beta = \gamma = 1$ i $C_3 = C_2/2$. D'aquesta manera, l'expressió final de l'índex SSIM és

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}. \tag{4.8}$$

Per incorporar una millora en la mesura de la qualitat de la imatge, és millor aplicar SSIM en local en comptes de fer-ho globalment. Per aquest motiu s'enfinestrarà la imatge i es calcularà l'SSIM local per a cada imatge enfinestrada. Posteriorment es farà la mitja de tots els SSIM.

La finestra que s'utilitza és una funció gaussiana de 11 x 11 amb $W = \{w_i | i = 1, 2, \cdots, N\}$, amb una desviació estàndard de 1.5 mostres i normalitzada ($\sum_{i=1}^{N} w_i$=1). L'estimació de les variables estadístiques comentades anteriorment. μ_x,σ_x i σ_{xy} es veurà modificada de la següent forma:

$$\mu_x = \sum_{i=1}^{N} w_i x_i \tag{4.9}$$

$$\sigma_x = \left(\sum_{i=1}^{N} w_i (x_i - \mu_x)^2\right)^{\frac{1}{2}} \tag{4.10}$$

$$\sigma_{xy} = \sum_{i=1}^{N} w_i (x_i - \mu_x)(y_i - \mu_y) \tag{4.11}$$

Normalment es necessita una mesura global de tota la imatge, s'utilitza llavors la mitja dels SSIM obtinguts localment,

$$MSSIM(X,Y) = \frac{1}{M}\sum_{j=1}^{M} SSIM(x_j, y_j) \tag{4.12}$$

on X i Y són la imatge de referència i la distorsionada, M és el nombre de finestres locals que s'han utilitzat, x_j i y_j són els valors de la imatge en la finestra j.

Per entendre visualment el significat d'aquesta mesura es pot veure la Figura 4.2 extreta de la pàgina web de SSIM [14]. Fixem-nos com totes les imatges distorsionades tenen pràcticament el mateix MSE respecte l'original però diferent qualitat. Veiem doncs com la mesura SSIM dóna una indicació millor sobre la qualitat de la imatge.

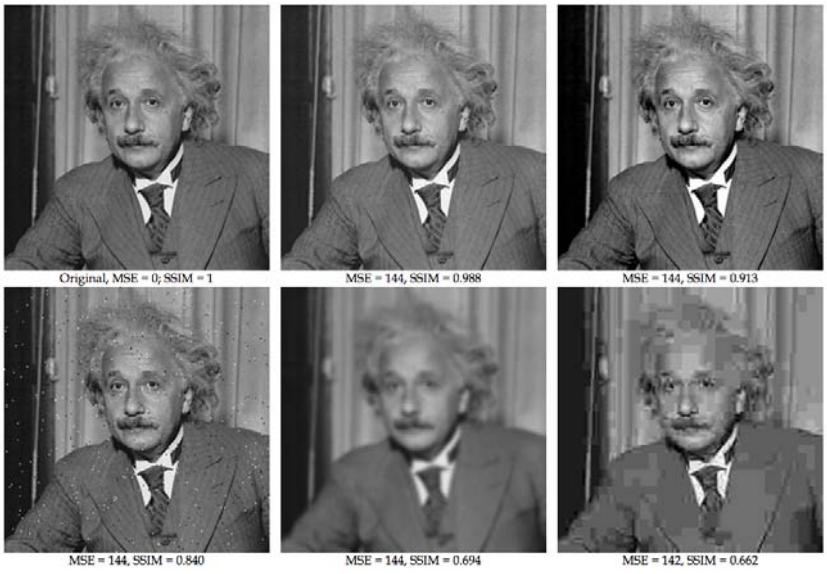


Figura 4.2: MSE i SSIM per una imatge amb diferents distorsions [14]

Tot i així, la mesura SSIM no és perfecta. És important fer notar que SSIM no detecta cares, estructura molt important en la mesura de la qualitat visual de les imatges. Un altre dels punts negatius és que SSIM no té massa en compte els errors de les textures, ja que les considera poc importants pel sistema visual humà, no obstant, algunes textures (com per exemple les de ratlles) és necessari que siguin ben definides perquè l'ull doni per bona la imatge. A més SSIM considera que la posició dels contorns és important, tot i una petita traslació no fa variar la percepció de qualitat de la imatge si la traslació és consistent per a tot el contorn. SSIM també considera que un petit canvi de luminància no afecta a la percepció. Igual que passa amb la traslació, si aquest canvi de luminància és coherent per a tot l'objecte o per a tota la imatge, és cert, però si no és consistent pot causar una gran distorsió visual.

A part de l'SSIM existeixen altres mesures que estudien la qualitat perceptiva de les imatges, com la proposada per Winckler a [17] o les estudiades a [18].

4.2 Integració de la mesura MSSIM a BOI

S'ha implementat una versió en Java de l'algoritme descrit anteriorment per tal d'incorporar-ho a la implementació BOI, descrita en el punt 3.3. La classe s'ha creat dins del paquet *GiciAnalysis*, com hem dit al descriure'l es troben classes que comparen dues imatges. Per implementar-la s'ha seguit la mateixa estructura de les classes que té BOI i que es defineix en la seva documentació [10].

S'ha creat la classe anomenada ImageCompareMSSIM, el diagrama del seu funcionament es mostra a la Figura 4.3. Aquesta classe rep dues imatges, que han de ser en escala de grisos i de la mateixa mida, i calcula el MSSIM entre elles. Els punts que s'han seguit en la implementació són:

- Definició de constructor de la classe
- Definició dels paràmetres, s'han creat diferents tipus de mètodes: un que deixa per defecte uns valors dels paràmetres (els definits en l'article [2]) i uns altres que permeten que aquests siguin definits per l'usuari.
- Classe principal, en aquest cas, càlcul del MSSIM (l'esquema del seu funcionament es mostra a la Figura 4.4)
- Funció *get* que retorna el valor

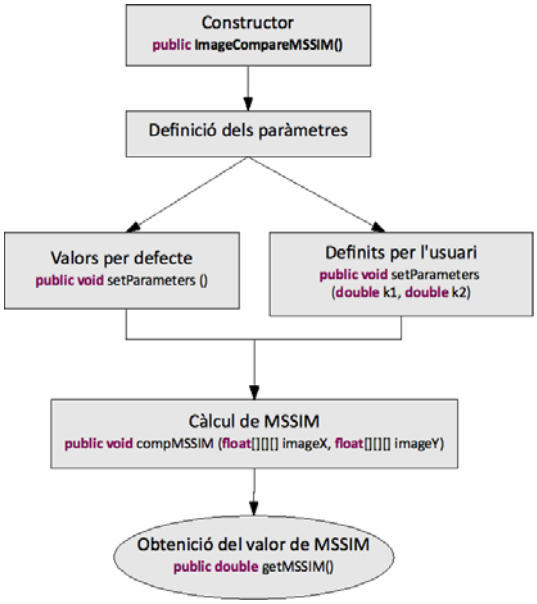


Figura 4.3: Esquema de funcionament de la classe ImageCompareMSSIM

Veiem com a la figura 4.4 el primer que s'ha de fer és comprovar que els paràmetres estiguin definits, seguidament es comprovarà que les imatges tinguin la mateixa mida i que sigui en escala de grisos. S'utilitzarà una finestra gaussiana, que es mourà píxel per píxel, per recòrrer

tota la imatge. En cada pas es calcularà les variables estadístiques i el valor del SSIM local. Quan es tinguin tots els SSIM calculats, es farà la mitja dels valors que donarà el MSSIM.

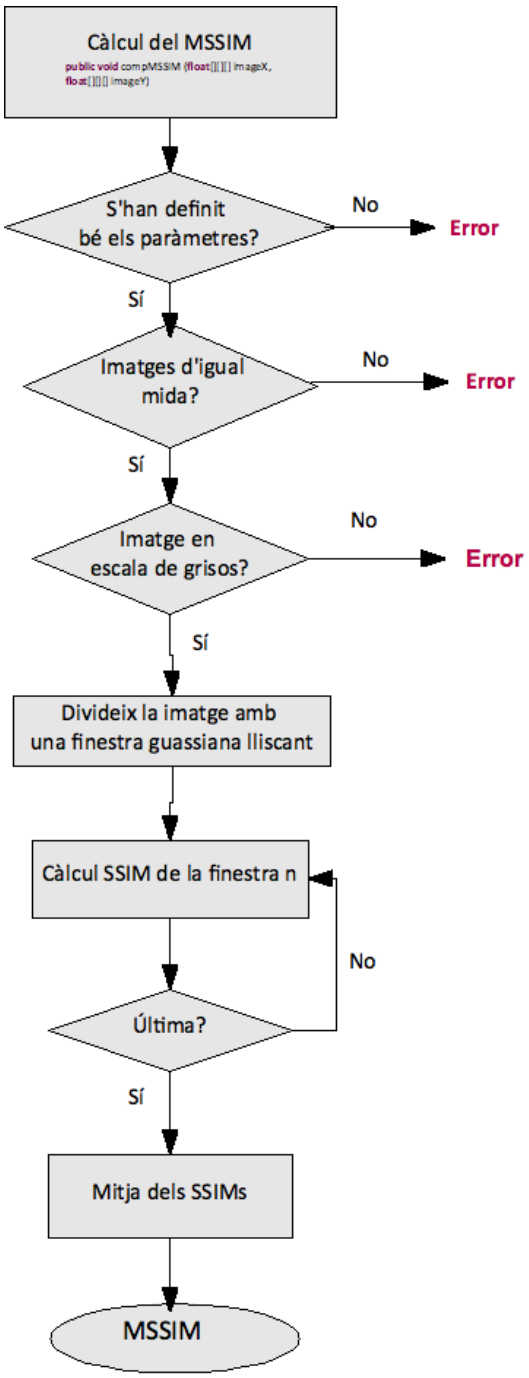


Figura 4.4: Esquema de funcionament del mètode compMSSIM

A part dels punts principals descrits, dins la classe també s'han creat mètodes que realitzen els càlculs intermedis de la mitja, la desviació estàndard i la correlació expressades en 4.5,4.6 i 4.7, que s'han deixat públics per si volen utilitzar-se en altres parts del projecte. També s'ha

implementat un mètode per crear una funció Gaussiana amb les característiques que s'han explicat anteriorment, que s'utilitzarà com a finestra.

Mitjançant la implementació del MSSIM presentada en aquest capítol disposarem d'una mesura de la qualitat de la imatge integrada dins del BOI, que podrà ser utilitzada paral·lelament amb altres mesures ja implementades. Aquestes mesures s'utilitzaran en l'avaluació dels resultats obtinguts en el projecte.

Serà també aquesta mesura la que s'utilitzarà també per a ser optimitzada en la cerca dels paràmetres òptims de la normalització divisiva del capítol següent.

Capítol 5

Normalització divisiva

En aquest capítol es parlarà de la normalització divisiva que posteriorment s'introduirà a la implementació de JPEG2000, BOI, per normalitzar els coeficients de la wavelet i utilitzar-los en el càlcul dels errors que serviran per la reordenació i creació posterior del byte-stream.

Els sistemes de compressió d'imatge operen transformant el senyal d'entrada amb una nova representació els elements de la qual són quantificats independentment. L'èxit d'aquests sistemes depèn de dues propietats de la representació. Per una banda, la taxa de codificació es minimitzarà només si els elements de la representació són estadísticament independents, cosa que s'ha comentat al capítol de conceptes previs. Per l'altra, la distorsió de codificació percebuda es minimitzarà només si els errors en la imatge reconstruïda, que provenen de la quantificació dels diferents elements de la representació, són independents perceptivament parlant.

A l'article [1] es discuteix sobre si les transformacions lineals poden aconseguir aquests objectius, es proposa la representació d'imatges no-lineal adaptativa i es demostra com aquesta operació redueix considerablement la redundància tan l'estadística com perceptiva.

5.1 Descripció

A l'article [1] es defineix una normalització divisiva dividida en dos fases de transformació en cascada

$$\{c_i\} \xrightarrow{T} \{a_i\} \xrightarrow{R} \{r_i\} \tag{5.1}$$

on primer els píxels de la imatge, $\{c_i\}$ s'analitzen utilitzant una transformació lineal i seguidament s'aplica una transformació no-lineal als coeficients, $\{a_i\}$, per obtenir els coeficients normalitzats, $\{r_i\}$.

La transformació lineal ha de ser una representació freqüencial tal i com s'acostuma a utilitzar en codificació, en el nostre cas serà la transformada wavelet. La fase de normalització divisiva

descriu els mecanismes de control del guany normalitzant l'energia de cada coeficient lineal a partir de la combinació dels seus veïns en espai, orientació i temps.

L'expressió de la normalització divisiva, descrita a [15], és :

$$r_i = sign(a_i)\frac{|s_i \cdot a_i|^\gamma}{1 + \sum_{j=1}^{N} H_{ij} \cdot |s_j a_j|^\gamma} \tag{5.2}$$

on N és el nombre de coeficients wavelet tret del residu de baixa freqüència, a_i són els coeficients wavelet (excepte, també, del residu de baixa freqüència) normalitzats pel seu valor absolut màxim. Cada coeficient, a_i, del domini wavelet té un significat espai-freqüencial, i, de 4 dimensions $i = (e, o, x, y)$ on $e = 1, ..., E$ és l'escala (que va de més resolució -alta freqüència- (1) a menys resolució (E) -baixa freqüència-); $o = 1, 2, 3$ és l'orientació (1=horitzontal, 2=vertical i 3=diagonal); x i y són la posició en files i columnes dels coeficients en cada subbanda. Dins de l'expressió també trobem s_i que representa el guany freqüencial i H que és la matriu d'interacció, a la secció 5.2 es troben les seves expressions.

Així doncs en l'equació 5.2 cada coeficient es rectifica i s'eleva, els valors resultant es divideixen per una suma ponderada de la resta de coeficients, on la matriu H_{ij} és el conjunt de pesos que especifiquen les interaccions entre tots els coeficients a_j, on $j = 1 \dots N$, i el coeficient utilitzat a_i. El signe de cada coeficient normalitzat serà el del corresponent coeficient.

5.2 Paràmetres de la normalització

Com hem vist anteriorment dins de l'expressió de la normalització divisiva, 5.2, trobem dues funcions, s i H, que tenen unes expressió amb uns paràmetres que seran optimitzats més endavant.

5.2.1 Guany freqüencial depenent de l'escala i l'orientació

L'expressió del guany diferencial depèn de l'escala i l'orientació segons la següent expressió:

$$s_i = s_{e.o.x.y} = A_o \cdot exp\left(-\frac{(E-e)^\theta}{\sigma_s^\theta}\right) \tag{5.3}$$

on A_o és el paràmetre que controla el guany, que se suposa constant per a totes les orientacions; $\sigma_s = [\sigma_{hv}, \sigma_d]$ són els paràmetres que controlen l'amplada de l'exponencial segons l'orientació, σ_d per a diagonal i σ_{hv} per a vertical i horitzontal, que es consideren iguals; i θ és un paràmetre que ens permet controlar la forma de la funció. Els paràmentres que haurem d'optimitzar en l'expressió de s són doncs θ, σ_s i A_0.

5.2.2 Matriu d'interacció

La matriu d'Interacció respon a l'expressió següent:

$$H_{ij} = H_{(x,y),(x',y')} = K \cdot exp\left(-\left(\frac{(x-x')^2}{\sigma_x^2} + \frac{(y-y')^2}{\sigma_y^2}\right)\right) \tag{5.4}$$

on el paràmetre per gaussiana el considerem igual, és a dir, $\sigma_x = \sigma_y = \sigma_{xy}$; i K és tal que $\sum_j H_{ij} = 1$. Per qüestions d'implementació pràctiques, truncarem els valors molt petits de la Gaussiana sense que això afecti significativament als resultats de la nostra normalització. En l'expressió de H, els valors que otimitzarem són $\sigma_x = \sigma_y = \sigma_{xy}$.

5.2.3 Paràmetres a optimitzar

Per tant, analitzant les expressions anteriors veiem que els paràmetres que s'hauran d'optimitzar són:

$$[\gamma, A_o, \sigma_d, \sigma_{hv}, \theta, \sigma_{xy}]$$

Els valors orientatius per a variar els paràmetres són:

1. $\gamma \in [0.5, \ldots, 3]$
2. $A_o \in [10, \ldots, 80]$
3. $\sigma_{hv}, \sigma_d \in [1.5, \ldots, 3.5]$
4. $\theta \in [1.5, \ldots, 10]$
5. $\sigma_{xy} = 0.125\frac{f_s}{2^e} \cdot [0.25, \ldots, 3]$

on en els càlculs s'ha pres una freqüència de mostreig f_s=128 cycles/degree (cpd), valors extrets de [15] . D'aquesta manera la freqüència màxima en el senyal és de 64 cpd, on la sensibilitat freqüencial és quasi zero.

El següent capítol, on s'explica com s'ha introduït la normalització divisiva al BOI, també explicarà el procés que s'ha seguit per a trobar els valors òptims d'aquests paràmetres.

Capítol 6

Integració de la normalització divisiva al JPEG2000

En aquest capítol es presentarà la integració de la normalització divisiva al JPEG2000 i, més concretament, utilitzarem la implementació BOI (descrita en la secció **3.3**) per integrar-la i poder procedir a les proves D'aquesta manera, com s'ha comentat als objectius, s'haurà d'ajustar la tècnica de normalització divisiva al que marca l'estàndard JPEG2000, es normalitzaran els coeficients wavelet per utilitzar-los en el càlcul d'errors però s'enviaran els coeficients no normalitzats i seran també aquests coeficients els que s'utilitzaran en el càlcul dels bit-planes.

6.1 Implementació

Comencem per implementar la normalització divisiva descrita en el capítol 5 dins de la classe **Coder**, que hem comentat al punt 3.3.3. A la Figura 6.1 es mostra un esquema de la integració de la normalització al BOI.

Com es veu a la figura 6.1, hem introduït en el programa una variable de control (*NIREPCUse*) com a paràmetre d'entrada del BoiCode que permetrà a l'usuari escollir si desitja dur a terme la normalització o no dels coeficients. Aquest paràmetre s'utilitzarà de manera similar a com s'han descrit altres paràmetres d'entrada del BoiCode al punt 3.3.2. La seva definició és la següent:

-nirepc –nirepcUse Utilització o no de l'algoritme de normalització divisiva dels coeficients. És un paràmetre booleà on 0 significa que no s'utilitzarà i 1 que sí que s'utilitzarà. Per defecte el paràmetre pren el valor 0.

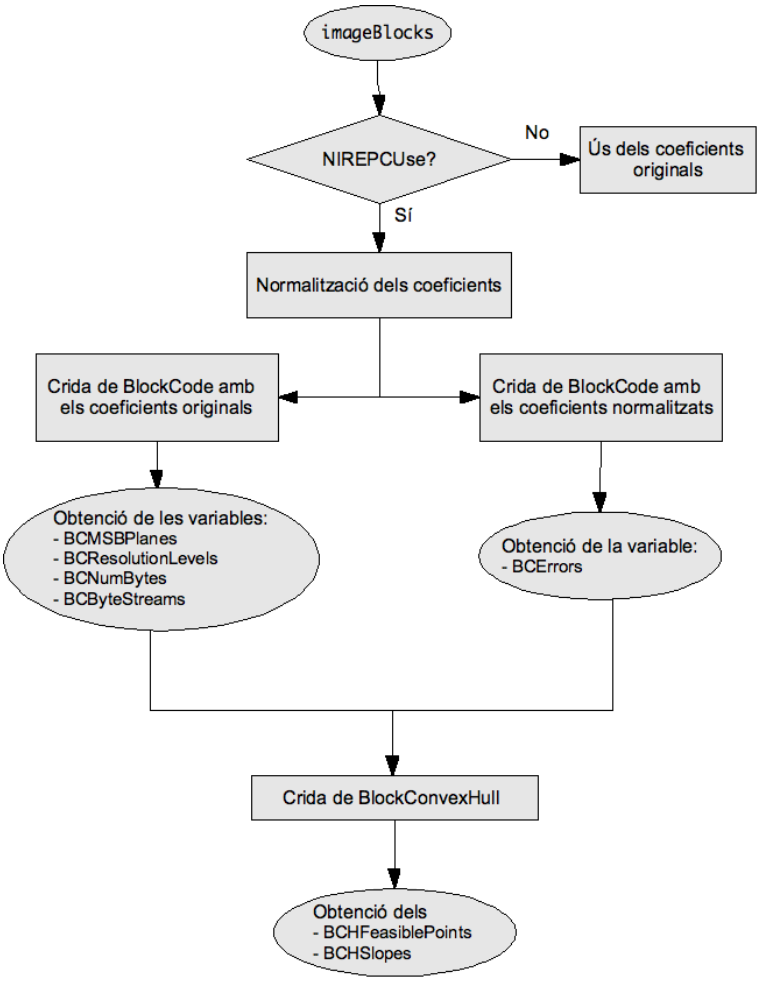


Figura 6.1: Esquema de la integració de la normalització divisiva al BOI

Si es vol dur a terme la normalització, es cridarà un mètode que la realitza seguint els passos descrits en el capítol 5 . L'esquema del bloc de la normalització dels coeficients es detalla a la Figura 6.2.

Podem veure com el primer pas és el discriminar entre els coeficients de baixa freqüència i la resta, ja que els de baixa freqüència no han de ser inclosos en la normalització divisiva i, per tant, mantindran el seu valor original. La resta de coeficients seran normalitzats.

Primer de tot, els dividirem pel màxim per tals de tenir-los normalitzats pel seu valor absolut i poder-los introduir a l'expressió de la normalització divisiva, abans però caldrà calcular la matriu d'interacció i el guany freqüencial. Finalment només caldrà aplicar l'equació 5.2 i guardar el seu valor en una nova variable que hem creat, semblant a *imageBlocks* (descrita a 3.3.4), i que anomenarem *imageBlocksNorm*.

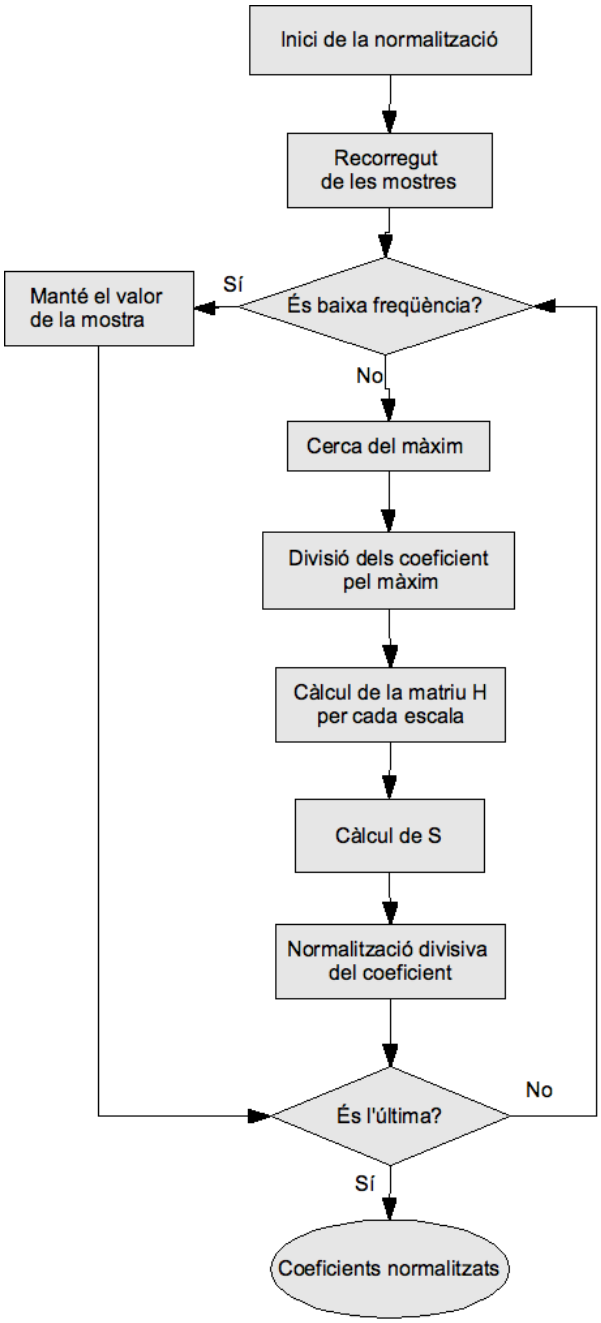


Figura 6.2: Esquema de la implementació de la normalització divisiva

Per poder incorporar les expressions 5.2, 5.3 i 5.4 al sistema de BOI s'han hagut de realitzar unes adaptacions en aquestes. Per exemple, la variable utilitzada com a índex *rLevel* de la classe Coder corresponent als nivells de resolució, que hem vist al punt 3.3.4, no correspon automàticament a l'*escala, e,* de les expressions del punt 5.1. La relació que existeix entre elles és $e = E - rlevel$, sabent que $e = 1, ..., E$ que va de més resolució -alta freqüència- (1) a menys resolució (E) -baixa freqüència.

També s'han hagut de reescalar els coeficients per tal que el nombre de bit-planes dels coeficients normalitzats sigui el més semblant possible al dels coeficients no normalitzats. Així,

es podrà aproximar l'aportació visual d'una passa de codificació dels coeficients normalitzats, que se suposa que són els visualment rellevants, amb una passa de codificació dels coeficients no normalitzats. Tot i l'aproximació, l'ordre dels coeficients no necessàriament es manté, només triem quina passa agafem. No obstant, això no garanteix que els coeficients no normalitzats escollits corresponguin als coeficients normalitzats, només ens garanteix que agafarem informació del codeblock que els coeficients normalitzats han triat.

Un exemple del problema plantejat es mostra a la Figura 6.3. En la primera figura (6.3a) es mostra un exemple d'un conjunt de coeficients en un contorns marcat. Podem veure com abans i després de la normalització els coeficients mantenen les relacions, és a dir, el més gran continua sent el més gran i així amb la resta. Per tant, en aquest cas segurament els coeficients no normalitzats escollits en cada passa correspondran als coeficients normalitzats. En canvi, en la segona figura (6.3b) es mostra un exemple de contorns poc definits o molt junts, en aquest cas el coeficient normalitzat central perd la seva relació amb els demés passant a ser inferior. Aquest coeficient no normalitzat pertanyerà a una coding pass diferent a la obtinguda amb el coeficient normalitzat.

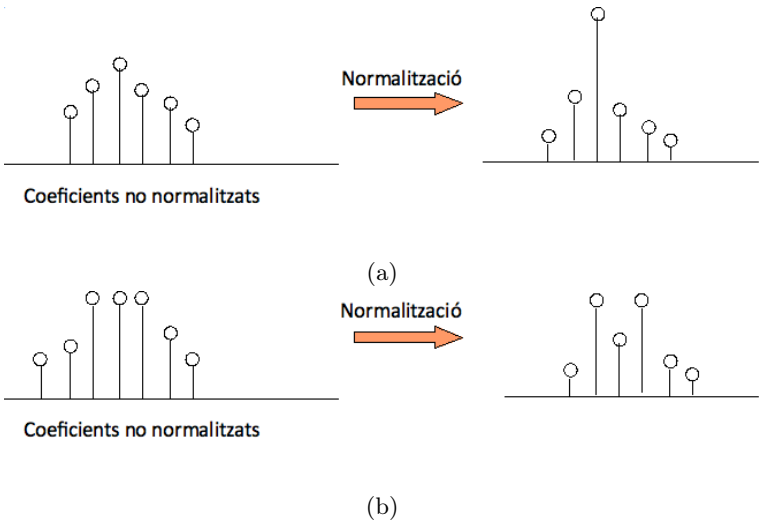


Figura 6.3: Exemples per (a) contorns marcats (b) contorns no marcats o molt junts

L'aproximació evidentment no és exacta, i farà que els resultats finals no siguin òptims, però el fet de limitar-nos a no sortir de l'estàndard és el que ens obliga a fer aquestes aproximacions. Aquest re-escalat s'ha realitzat multiplicant cada coeficient normalitzat per una constant, K, el valor de la qual es calcula mitjançant l'expressió

$$K = \frac{max(imageBlocks)}{max(imageBlocksNorm)}$$

on es divideix el màxim de les mostres originals pel màxim de les mostres normalitzades (excepte el residu de baixa freqüència). Aquest re-escalat no canviarà el resultat ja que es multipliquen tots coeficients.

Com es veu a la figura 6.1, un cop obtinguts els coeficients normalitzats s'utilitzen per obtenir

els errors utilitzant la classe **BlockCode**. Per una altra banda, s'utilitzarà aquesta mateixa classe però amb els coeficients no normalitzats per aconseguir la resta de variables que proporciona. Seran els errors obtinguts amb la classe **BlockCode** i els coeficients normalitzats i guardats en la variable *BCErrors*, descrita al punt 3.3.4, els que s'enviaran al **BlockConvexhull** per calcular els *FeasiblePoints* i les *Slopes* que s'utilitzaran per crear els nivells de qualitat.

El fet de no enviar, com ja em dit, els coeficients normalitzats ha fet que hàgim hagut d'introduir uns canvis dins de la funció **BlockConvexhull**. El motiu és que els errors obtinguts amb els coeficients normalitzats i els coeficients no normalitzats poden tenir diferent nombre de bit-planes . Per tal d'evitar aquest desajust, s'ha incorporat al codi un control pels diferents casos:

- Si el nombre de bit planes dels no normalitzats és igual al nombre de bit-planes dels normalitzats, no hi ha problema i es deixa treballar amb aquest valor.
- Si el nombre de bit planes dels no normalitzats és menor al nombre de bit-planes dels normalitzats, s'ignoren els coeficients normalitzats "sobrants".
- Si el nombre de bit planes dels normalitzats és major al nombre de bit-planes dels normalitzats, és manté el mínim error de l'últim bit-plane dels normalitzats per la resta de bit-planes dels no normalitzats.

Això és una aproximació del que s'obtindria si s'enviessin els coeficients normalitzats i es desnormalitzessin al receptor, tot i que, com hem explicat abans, molt inexacta. Aquesta inexactitud pot generar la suboptimalitat dels resultats.

Les funcions més important implementades en el projecte es poden trobar en els àpendixs. El codi font generat en aquest punt es pot trobar al CD adjunt.

6.2 Optimització dels paràmetres

Com s'ha comentat a l'apartat 5.2.3 hi ha 6 paràmetres que caldrà optimitzar per tal d'obtenir la combinació que ens doni, en aquest cas, un millor SSIM, mesura descrita en el capítol 4. La optimització dels paràmetres es farà iterativament i per etapes: optimitzarem un paràmetre cada vegada, i inicialitzarem la següent etapa d'optimització amb els millors valors assolits per als paràmetres ja optimitzats. Això ho farem iterativament fins que els valors dels paràmetres ja no presentin variacions.

Per tal de dur a terme l'optimització s'han preparat una sèrie d'scripts, que aniran variant el valor d'un dels paràmetres, segons el rang descrit, per a un conjunt d'imatges diferents. Aquestes proves serviran per generar un document de text on hi guardarem el valor del SSIM i el PSNR de les imatges obtingudes en comparació amb la imatge original. Cada combinació

de valors i imatge generarà, per tant, un valor d'SSIM diferent. Al acabar la iteració, es farà la mitja dels diferents SSIM de cada imatge i s'escollirà el valor del paràmetre que en mitja sigui millor. Per a realitzar les proves següents s'utilitzaran els paràmetres d'entrada del BOI que s'han descrit al punt 3.3.2.

Una vegada fixat un dels paràmetres es seguirà amb el següent i anirem fent de la mateixa manera fins que obtinguem uns valors estables. Les proves de l'optimització s'han fet per imatges de 128 x 128, per tal de minimitzar el temps d'execució, però s'ha comprovat que optimitzar amb imatges de mides més grans no fa variar significativament els valors.

Per començar l'optimització cal definir el valor inicial dels paràmetres. Es van realitzar diferents inicialitzacions dels paràmetres, i es va comprovar que els valors obtinguts variaven molt depenent de la inicialització. Es va observar que inicialitzant els paràmetres al seu valor mitjà donava un millor mínim local. Tot i així, al no fer una cerca exhaustiva dels valors òptims dels paràmetres, no podem estar segurs d'haver trobat el màxim global, és molt probable que tinguem un màxim local. Això és degut a la no convexitat de l'espai a optimitzar.

Així doncs mostrem a la taula 6.1 els resultats obtinguts. Com ja s'ha comentat hem escollit com a valor d'inici, aproximadament, la meitat del rang de valors i l'ordre seguit a l'hora de variar els valors va ser el següent: $\sigma_{xy} \rightarrow \theta \rightarrow \sigma_d \rightarrow \sigma_{hv} \rightarrow A_o \rightarrow \gamma$. D'aquesta manera les iteracions d'optimització han quedat de la següent manera[1]:

	Gamma γ	***Guany*** A_0	***Sigma HV*** σ_{hv}	***Sigma D*** σ_d	***Theta*** θ	***Sigma XY*** σ_{xy}
1a Iteració	*1.2*	*40*	*3.2*	*3.2*	*6*	-
2a Iteració	*1*	*12*	*3.2*	*2.8*	*2.1*	*0.2*
3a Iteració	*1*	*12*	*3.2*	*2.6*	*2*	*0.3*
4a Iteració	*1*	*12*	*3.2*	*2.6*	*2*	*0.3*
Resultats estables	***1***	***12***	***3.2***	***2.6***	***2***	***0.3***

Taula 6.1: Resultats de les diferents iteracions d'optimització dels paràmetres.

Un vegada s'ha acabat la quarta iteració observem que els valors no han variat i, per tant, s'agafen els valors dels paràmetres obtinguts com a òptims.

6.3 Resultats

Una vegada s'han trobat els valors dels paràmetres que optimitzen l'SSIM passarem a realitzar les proves experimentals amb imatges de diferents dimensions. Es realitza la comparació de les imatges obtingudes amb les que s'obtenen sense utilitzar la normalització divisiva. A la Taula 6.2 es mostren els valors en mitja de l'SSIM i del PSNR per imatges de diferents mides.

[1] *Cal fer notar que el valor de sigmaXY que es variarà és el factor que multiplica l'expressió* $0.125\frac{f_s}{2^e}$ *on e és l'escala i* f_s *és la freqüència de mostreig de valor 128 cpd.*

	Amb normalització	**Sense normalització**
imatges 128x128	29.440339	30.177357909
imatges 256x256	31.76219	33.482793272
imatges 5125x512	34.48429	36.4357699

(a)

	Amb normalització	**Sense normalització**
imatges 128x128	0.887088130	0.892821010
imatges 256x256	0.912459474	0.920986873
imatges 512x512	0.923788299	0.932968649

(b)

Taula 6.2: Valors del PSNR (a) i l'MSSIM (b) per les 10 imatges de prova amb un bit-rate de 0.75 bps

A la taula es pot observar com els resultats obtinguts amb la normalització, en global, no són millors que els obtinguts sense la normalització. A l'hora d'analitzar els resultats, hem de recordar que, com hem explicat anteriorment, en realitat, no treballem amb els bit-planes dels coeficients normalitzats, sinó amb els dels coeficients sense normalitzar. Això ens penalitza, ja que no es manté el ràtio entre coeficients, ni s'agafen exactament els mateixos coeficients per a cada coding pass, degut a l'ordre adaptatiu amb el que es fan les coding passes.

També cal tenir en compte la modificació que hem introduït en el càlcul del convex hull, descartant coeficients o mantenint el valor de l'error quan teníem diferent nombre de bit-planes entre els coeficients normalitzats i els no normalitzats. A l'agafar els punts de tall que donen els coeficients normalitzats com a punts òptims per al convex hull, és possible que escollim punts de tall que no siguin òptims per als coeficients no normalitzats, que són els que codifiquem. Per exemple podria ser que suposant que en la figura 3.10 es mostra un convex hull dels coeficients no normalitzats, s'escollis en algun moment un dels punts blancs en comptes dels sòlids, com a candidat a punt de truncament.

Una altre problema, que ja hem comentat abans, és que l'espai de cerca dels paràmetres no és convex, i té molts mínims locals. Per tant, segurament, la cerca iterativa que s'ha fet dóna un mínim local, però no el global.

Els resultats obtinguts per a 10 imatges diferents de dimensions 512 x 512 es mostra a continuació, juntament amb el valor del PSNR i de l'MSSIM. En alguns casos es mostra també els mapes d'error, en que negre representa error 0, i de qualitat SSIM, on més blanc significa millor qualitat.

Barbara

(a)

(b)

Figura 6.4: Resultats per un bit-rate de 0.75bps (a) Amb normalització PSNR=31.673708 dB, SSIM= 0.9195955747008607 (b)Sense normalització PSNR=34.229836 dB, SSIM= 0.9291282424585365

Podem observar com a la Figura 6.4 la imatge obtinguda mitjançant la normalització dels coeficients, la Fig. 6.4a, els contorns queden més ben definits i com la textura del terra també és força més homogènia.

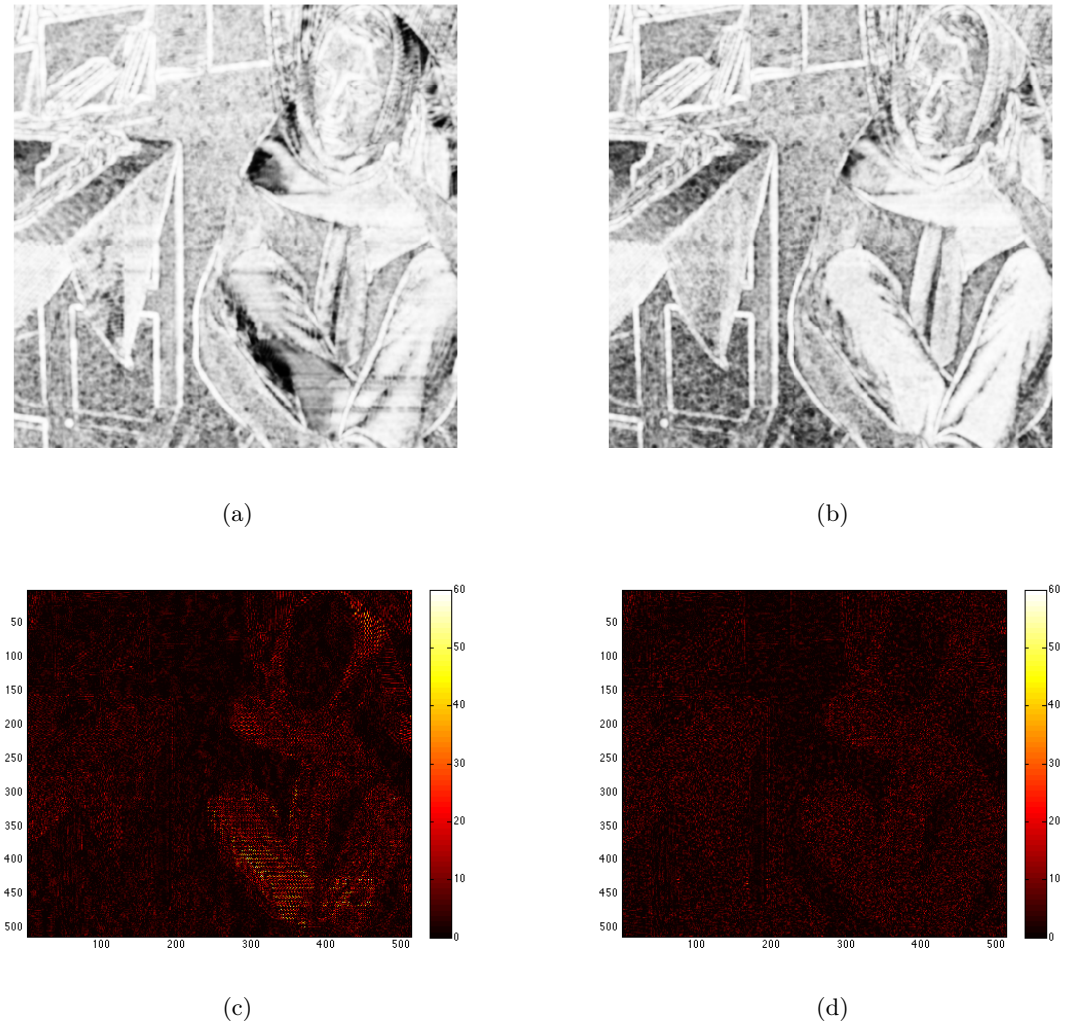


Figura 6.5: (a) Mapa SSIM de la imatge original amb la imatge obtinguda amb la normalització (b) Mapa SSIM de la imatge original amb la imatge obtinguda sense la normalització, on blanc significa SSIM=1 (exactament la mateixa imatge visual) i negre significa SSIM=0 (poca similitud visual) (c) Error de la imatge original amb la imatge obtinguda amb la normalització (d) Error de la imatge original amb la imatge obtinguda sense la normalització, on (negre, error 0 i blanc, error molt gran)

A la Figura 6.5a i 6.5b hi ha representat el mapa dels SSIMs de les imatges en comparació amb l'original. Podem veure com en general la imatge que utilitza la normalització és més brillant, és a dir, en general la qualitat visual està més homogèniament repartida. Observem com hi ha unes zones, la part dels pantalons i del mocador, és més negre cosa que provoca

que el MSSIM total doni més baix que amb la no normalització. Però si ens fixem amb els contorns de la taula o de la prestatgeria dels llibres aquestes zones són més blanques que en l'altre mapa i, per tant, presenten una millor qualitat.

Això últim també ho podem veure en la representació de l'error de les figures 6.5c i 6.5d, fixem-nos com en la corresponen a la no normalització es veu l'error en el contorn de les potes de la taula. El detall de les potes es mostra en la Figura 6.6 perquè es vegi més clarament la millora de què parlem.

La conclusió que se'n pot treure és que la normalització divisiva redueix l'error de les zones on és visualment més rellevant (contorns i zones llises) i el desplaça a zones on visualment no és tant rellevant, com zones amb textures (per exemple, els pantalons o el mocador de la Barbara). L'error en les textures pot ser degut al fet que les *coding passes* dels coeficients no normalitzats contenen grups de coeficients que són diferents als grups de coeficients que contenen les *coding passes* dels coeficients normalitzats. Aquest error a les textures provoca l'error global de la imatge pugi. Això es podria evitar si en lloc de voler complir estrictament JPEG2000 part 1, es permetés un re-escalat dels coeficients previ a l'enviament, i que el descodificador desfés aquest mateix escalat.

(a) (b) (c)

Figura 6.6: Detall de les imatges (a) imatge original (b) imatge amb normalització (c) imatge sense normalització

ChurchTower

(a)

(b)

Figura 6.7: Resultats de la imatge ChurchTower per un bitrate de 0.75bps (a) Amb normalització PSNR=37.784336, SSIM=0.9586933982042634 (b) Sense normalització PSNR=42.147003, SSIM=0.9687704113082091

En aquest cas, la imatge obtinguda amb la normalització és força pitjor que la obtinguda sense ella. Sobretot a la part de la cúpula principal on, en el cas de la normalització, es veu un

difuminat en el contorn i, en general, en tota la zona. Ho podem veure clarament tant en la representació del mapa del SSIM com de l'error en la Figura 6.8. Això és segurament degut al fet que, a causa de la lleugera textura que hi ha a la cúpula, les *coding passes* dels coeficients normalitzats contenen grups de coeficients que són molt diferents dels grups de coeficients que contenen les *coding passes* dels coeficients sense normalitzar. Aquest cas correspondria al descrit en la figura 6.3b.

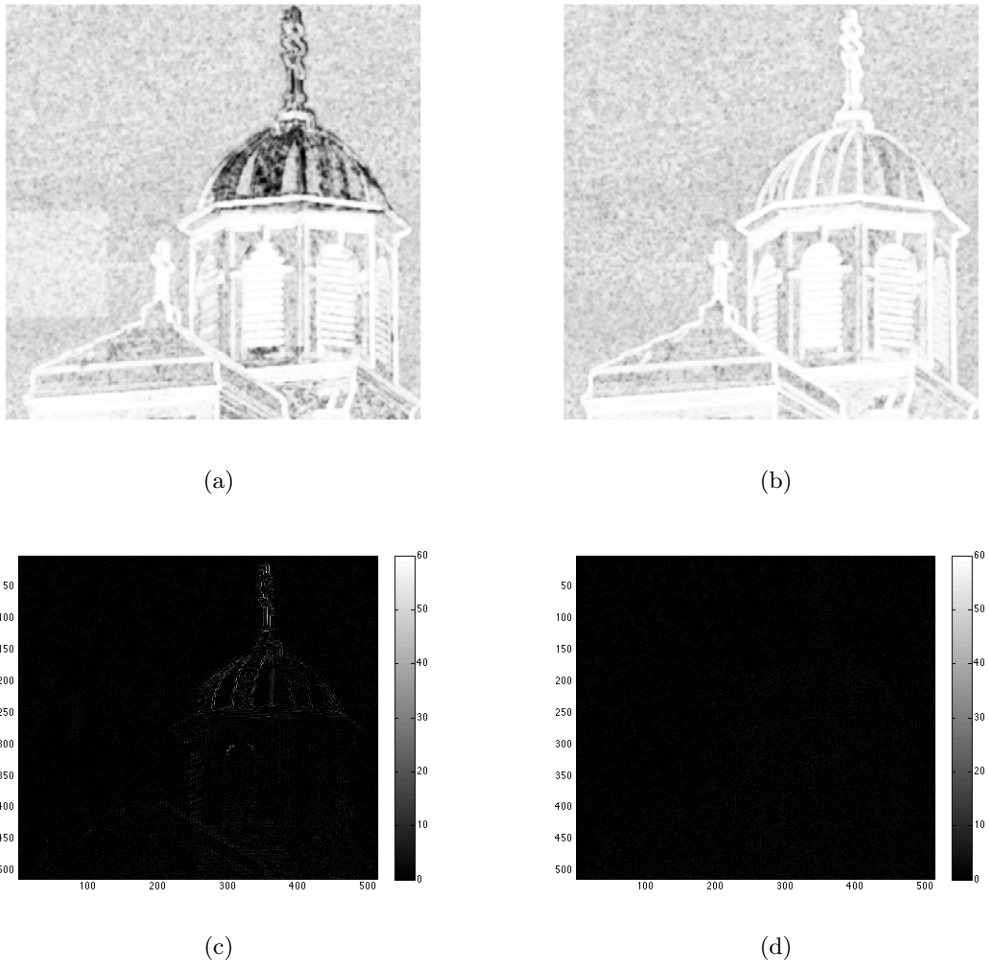


Figura 6.8: (a) Mapa SSIM de la imatge original amb la imatge obtinguda amb la normalització (b) Mapa SSIM de la imatge original amb la imatge obtinguda sense la normalització, on blanc significa SSIM=1 (exactament la mateixa imatge visual) i negre significa SSIM=0 (poca similitud visual) (c) Error de la imatge original amb la imatge obtinguda amb la normalització (d) Error de la imatge original amb la imatge obtinguda sense la normalització, on (negre, error 0, blanc, error molt gran)

Aquesta imatge és un clar exemple de les limitacions que té el fet de cenyir-nos estrictament a complir les especificacions de JPEG2000 part 1. Si en aquesta imatge codifiquéssim els coeficients normalitzats i permetéssim que el descodificador desfés la normalització abans de fer la transformada wavelet inversa, probablement la cúpula no tindria aquesta pèrdua de qualitat tant accentuada.

Goldhill

(a)

(b)

Figura 6.9: Resultats per la imatge Goldhill amb un bitrate de 0.75bps (a) Amb normalització PSNR= 34.445007, SSIM= 0.9017559452515643 (b) Sense normalització PSNR= 35.013535, SSIM= 0.9097004536957387

Aquesta imatge és tan complexa i té tanta textura que el fet de normalitzar o no, no fa que els errors siguin més o menys visibles. Per aquest motiu la diferència entre una i altra és imperceptible.

Leaf

(a)

(b)

Figura 6.10: Resultats obtinguts per la imatge Leaf amb un bitrate de 0.75bps. (a) Amb normalització PSNR=29.68547, SSIM=0.8362241739734876 (b) sense normalització PSNR=31.692026, SSIM=0.8695787469662211

En el cas d'aquesta imatge podem veure com la obtinguda mitjançant la normalització presenta un millor resultat en la zona de la fulla però pitjor qualitat en la part de la textura de la taula. Tot el contrari del cas de la imatge obtinguda mitjançant la no normalització. Això té la mateixa explicació que en el cas de la Barbara, és a dir, que la normalització divisiva redueix l'error de les zones amb contorns i zones llises, que és el cas de la fulla i la zona de l'escrit, i el desplaça a zones on visualment no és tant rellevant, com zones amb textures, que és el cas de la taula.

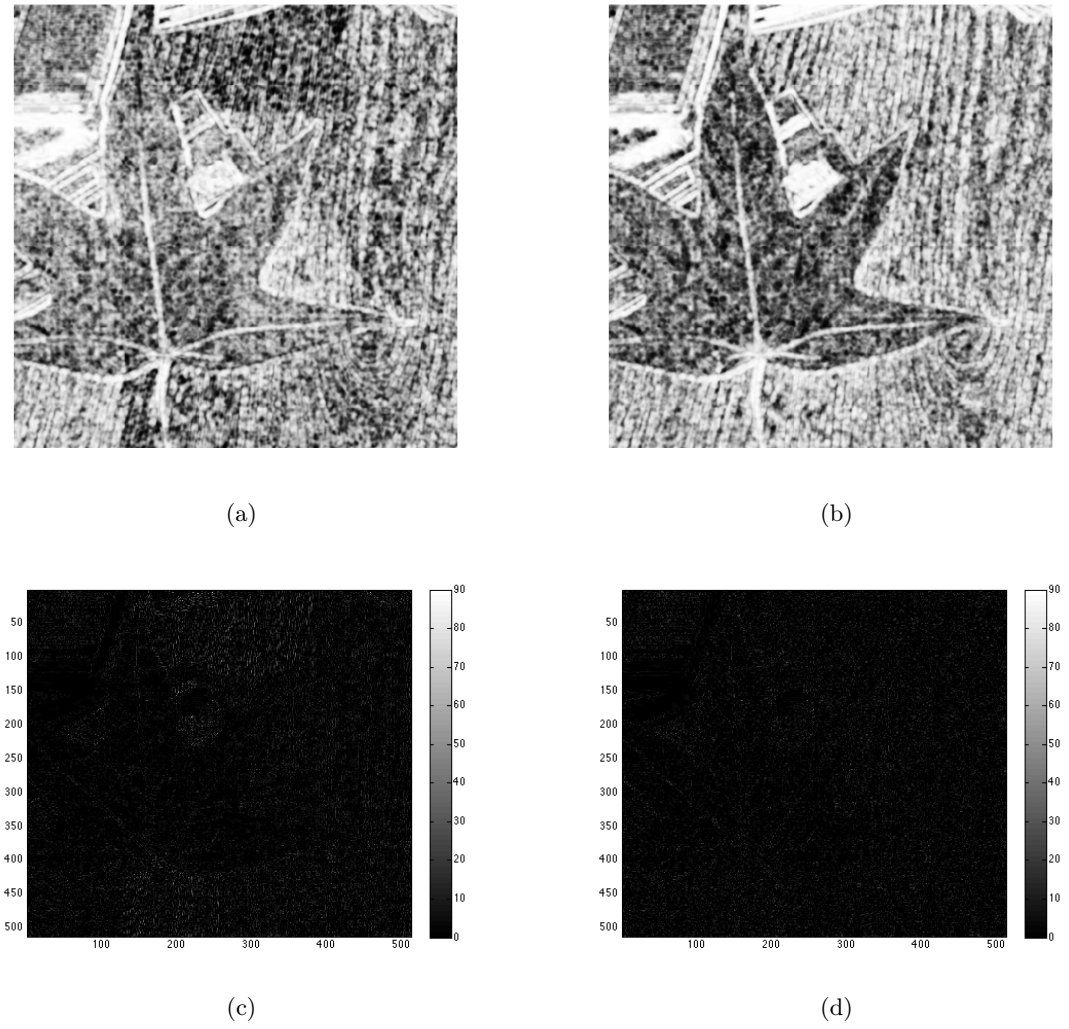
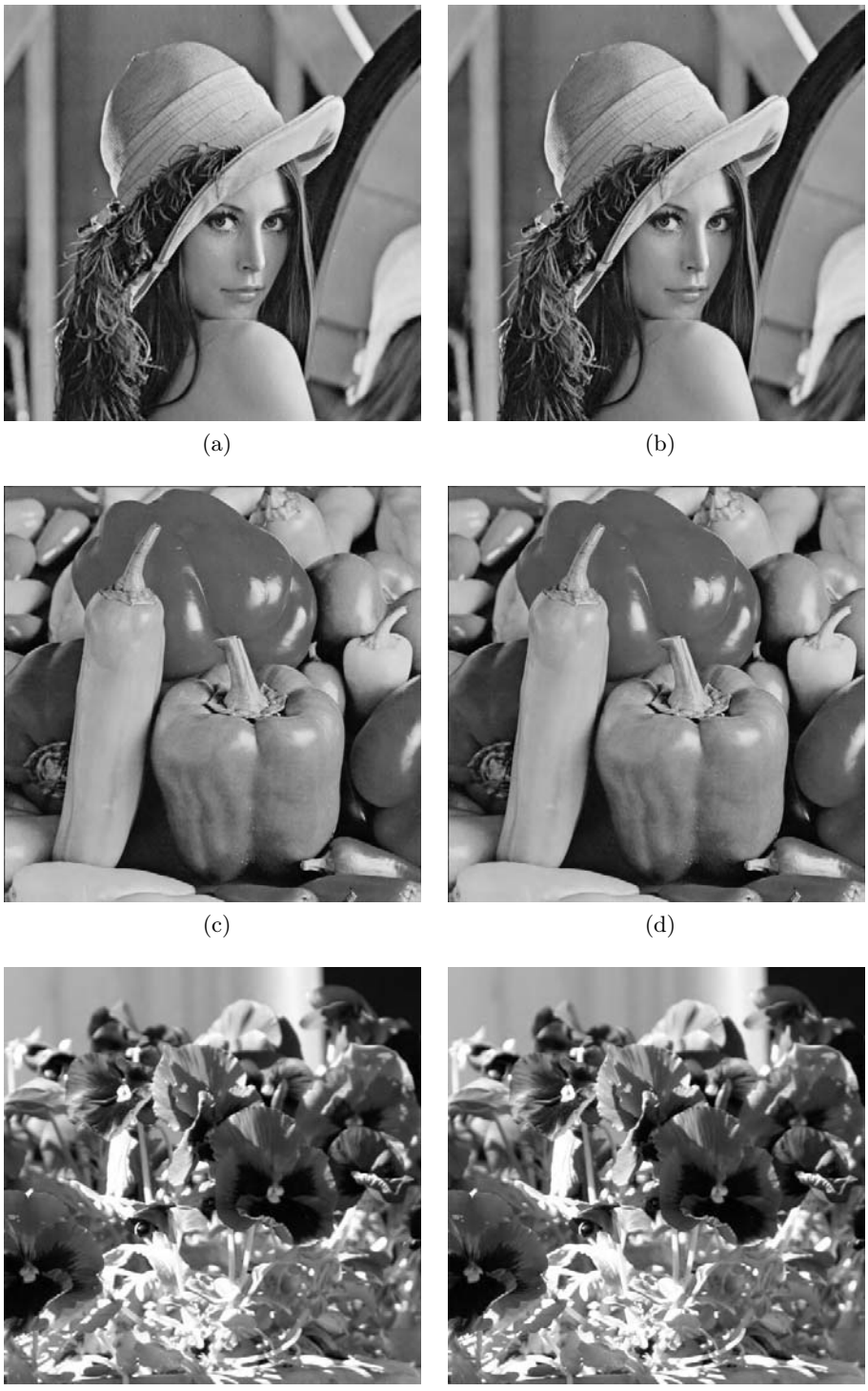


Figura 6.11: (a) Mapa SSIM de la imatge original amb la imatge obtinguda amb la normalització (b) Mapa SSIM de la imatge original amb la imatge obtinguda sense la normalització, on blanc significa SSIM=1 (exactament la mateixa imatge visual) i negre significa SSIM=0 (poca similitud visual) (c) Error de la imatge original amb la imatge obtinguda amb la normalització (d) Error de la imatge original amb la imatge obtinguda sense la normalització, on (negre, error 0, blanc, error molt gran)

A més, en les zones amb contorns marcats vorejades de textures llises, el fet de normalitzar no fa canviar quins són els primers coeficients a esdevenir significatius en el codificador per plans de bit (Bit-plane Encoder, BPE), tal i com hem comentat a l'exemple de la figura 6.3a. Per contra, si hi ha una zona amb molta textura (cas de la taula), és possible que la normalització divisiva canviï quins són els primers coeficients a esdevenir significatius al BPE, i que, per tant, les coding passes dels coeficients normalitzats i dels sense normalitzar no tinguin massa coeficients en comú, exemple que hem mostrat a la figura 6.3b.

Les Figures 6.12 i 6.13 mostren més resultats visuals, que tenen resultats semblants a les imatges mostrades fins ara, i que per tant no es comentaran una per una.

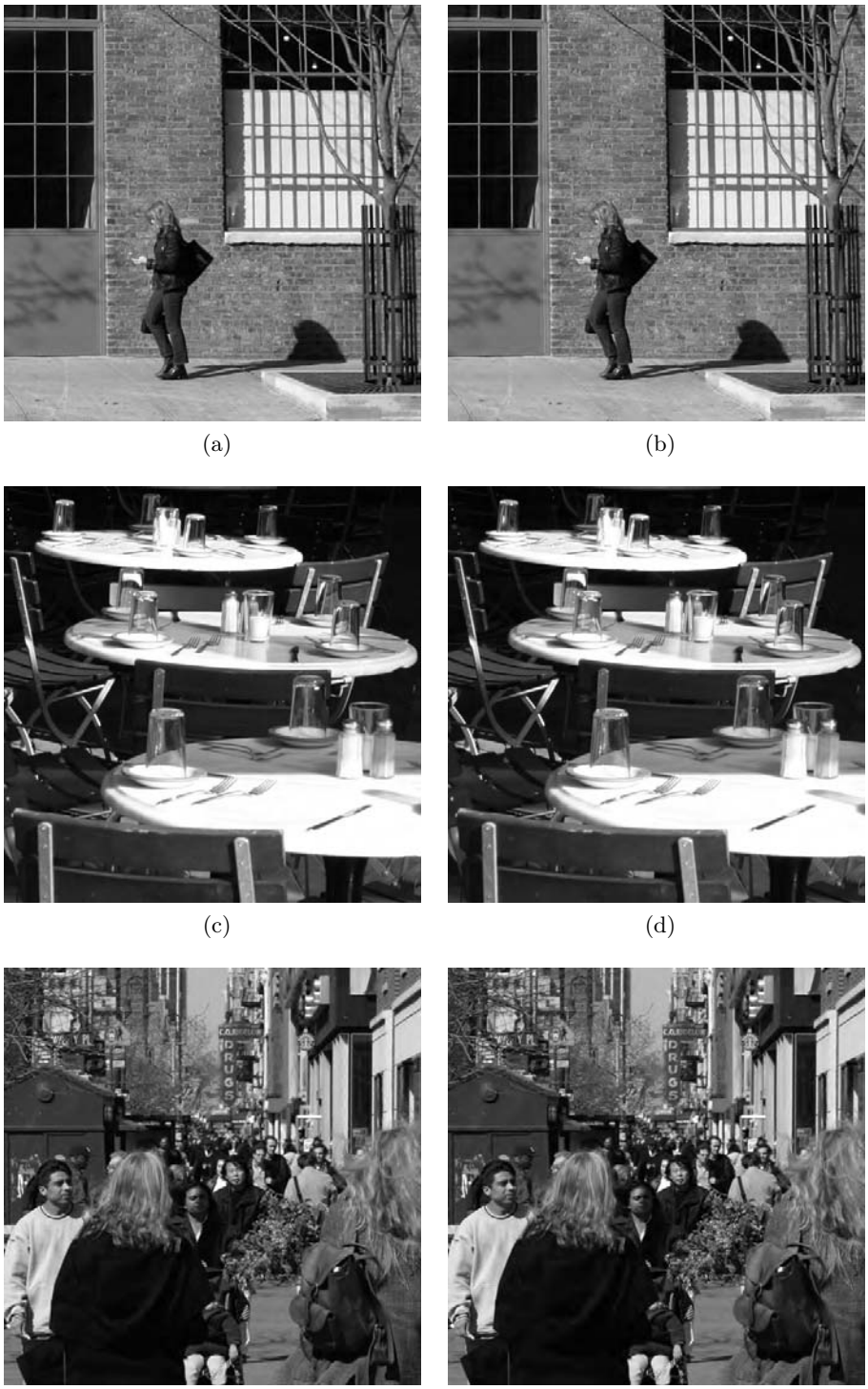


(a) (b)

(c) (d)

(e) (f)

Figura 6.12: Resultats per un bitrate de 0.75bps (a)Lena amb normalització PSNR=35.124565 dB, SSIM= 0.9220759949136988 (b)Lena sense normalització PSNR= 37.725376 dB, SSIM= 0.9294548149441602 (c)Peppers amb normalització PSNR=36.336433 dB, SSIM= 0.9003507228944935 (d)Peppers sense normalització PSNR=37.10341 dB, SSIM= 0.9083685938701637 (e)Posies amb normalització PSNR=36.629505 dB, SSIM= 0.9738359413680001 (f)Posies sense normalització PSNR=38.806812 dB, SSIM= 0.9765227569688442



(a) (b)

(c) (d)

(e) (f)

Figura 6.13: Resultats per un bitrate de 0.75bps (a) Sidewalk amb normalització PSNR=32.11766 dB, SSIM= 0.9215481722338017 (b) Sidewalk sense normalització PSNR=33.87218 dB, SSIM= 0.9292907043031611 (c) Tables amb normalització PSNR=39.11412 SSIM= 0.9769535067437072 (d) Tables sense normalització PSNR=40.508606, SSIM= 0.9768509282196824 (e) Waverly amb normalització PSNR=31.932108, SSIM= 0.9268495667560196 (f) Waverly sense normalització PSNR=33.258915, SSIM= 0.9320208401411235

Amb les imatges anteriors s'ha realitzat uns tests psico-visuals a 31 persones per estudiar quina de les imatges consideraven millors, les que s'han obtingut utilitzant la normalització o les obtingudes sense. Al gràfic següent, corresponent a la Figura 6.14, es mostra el nombre de persones que ha escollit la imatge obtinguda utilitzant normalització i el nombre de persones que han triat el mètode que ja hi havia, és a dir, sense normalitzar els coeficients.

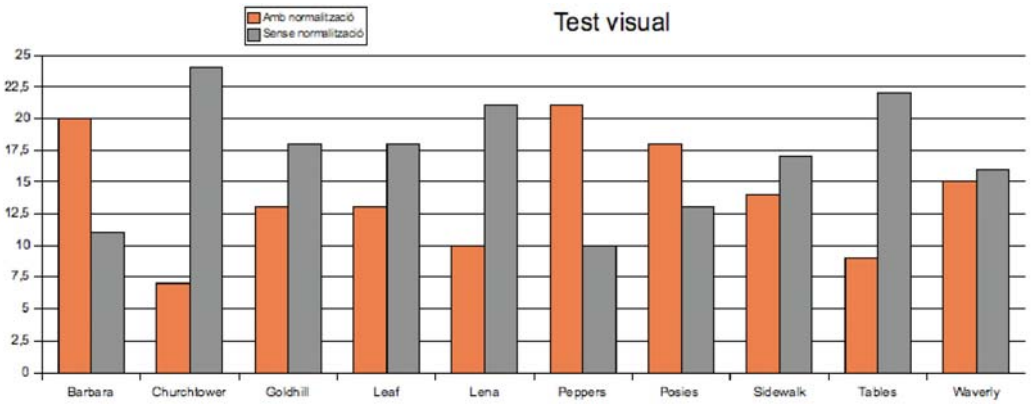


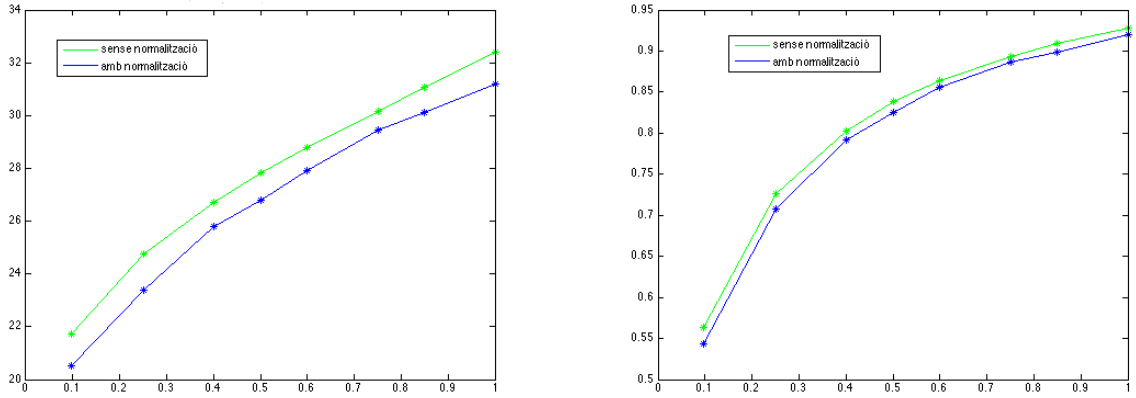
Figura 6.14: Gràfic amb els resultats del test psico-visual

Podem observar al gràfic com els resultats estan molt igualats per a imatges amb normalització i imatges sense. Per a les imatges on els contorns són molt marcats i amb molt de contrast, com és el cas de Barbara i Peppers, els individus del test prefereixen les imatges obtingudes amb normalització.

Seguidament, hi ha un seguit d'imatges que presenten més o menys els mateixos resultats per a la normalització i sense la normalització (seria el cas de Waverly, Sidewalk, Goldhill), aquestes imatges es caracteritzen per tenir molta textura que provoca que l'ús o no de la normalització no produeixi que els errors siguin més o menys perceptibles.

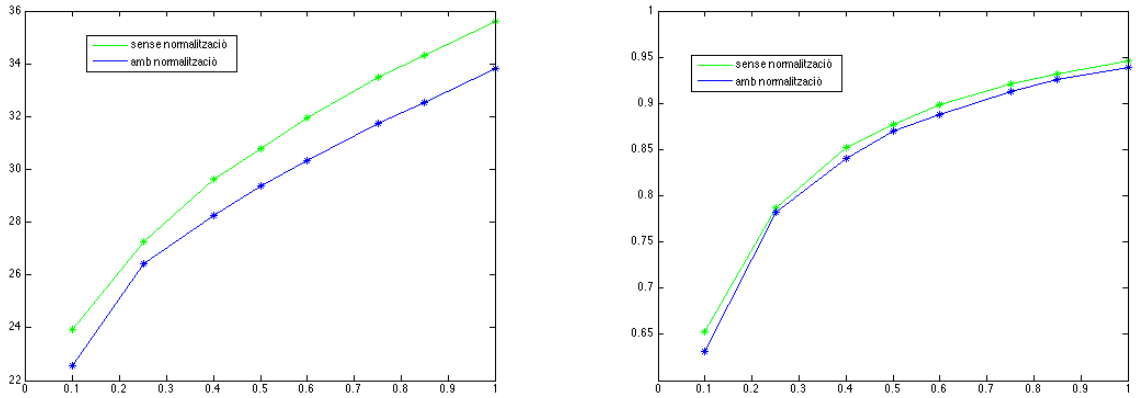
Finalment hi ha el cas de Churchtower i Tables, on els testos psicovisuals indiquen clarament que la imatge sense normalització obté millors resultats visuals. Com ja hem comentat en l'anàlisi de la imatge Churchtower, aquestes imatges corresponen a imatges amb molts contorns poc marcats, on és probable que el problema de tenir conjunts de coeficients molt diferents per a les coding passes dels coeficients normalitzats i dels coeficients sense normalitzar sigui habitual per a molts dels codeblocks.

A continuació farem proves variant el bit-rate per al conjunt d'imatges que hem utilitzat per formar la taula 6.2.



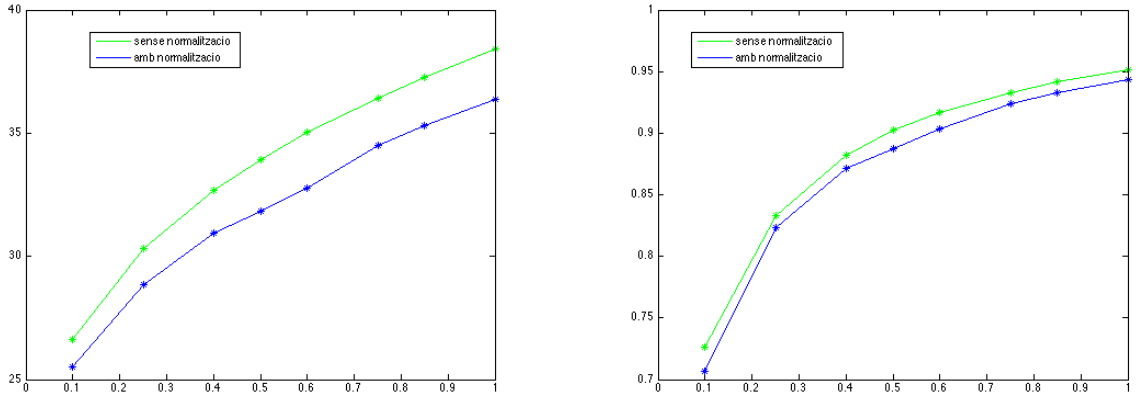
(a) (b)

Figura 6.15: Gràfics del PSNR (a) i l'MSSIM (b) en funció del bit-rate per imatges de 128x128 píxels



(a) (b)

Figura 6.16: Gràfics del PSNR (a) i l'SSIM (b) en funció del bit-rate per imatges de 256x256 píxels



(a) (b)

Figura 6.17: Gràfics del PSNR (a) i l'SSIM (b) en funció del bit-rate per imatges de 512x512 píxels

En les gràfiques de variació del bit-rate podem veure com el valor de PSNR i del MSSIM de les imatge que utilitzen la normalització està sempre per sobre de les que no l'utilitzen. Tot i això, la diferència entre els valors del MSSIM es manté constant pels diferents bit-rates, cosa que indica coherència al llarg de tot el bit-rate (les no-optimalitats d'utilitzar uns coeficients per a càlcul d'errors i enviar-ne uns altres no s'agreugen a mesura que augmentem el nombre de coding passes enviades). També veiem un comportament similar per a les diferents mides de les imatges.

Pel que fa a les gràfiques del PSNR aquesta diferència no es manté constant, cal recordar que en la part d'optimització dels paràmetres ho fèiem en base al valor del SSIM i no del PSNR, amb el que era d'esperar que aquest decreixés. Tot i que, la gràfica de la normalització quedi per sota de l'altra, ja hem vist, en l'anàlisi dels resultats, com les imatges on creiem que la no-optimalitat deguda a mantenir l'estàndard intacte afecta menys, presenten una millora.

Capítol 7

Conclusions

Aquest projecte ha modificat JPEG2000 per a tenir una codificació escalable en qualitat visual, i no només en PSNR. Per fer-ho, ha calgut eliminar les dependències estadístiques i visuals dels coeficients obtinguts en la transformada del sistema codificació. Com s'ha vist, hem treballat amb l'estàndard de compressió JPEG2000, que ens proporciona escalabilitat a nivell de distorsió. A més, l'esquema del JPEG2000 elimina bona part de les dependències estadístiques dels coeficients wavelet obtinguts. Tot i així aquest estàndard no elimina completament les dependències visuals. Al llarg de la memòria hem vist com JPEG2000 proporciona unes molt bones prestacions en qualitat, fins i tot, a bit-rate baixos, i en escalabilitat. Aquest fet el converteix en un dels sistemes de compressió d'imatges amb més bones prestacions que existeix actualment, i per tant en un bon candidat com a futur estàndard de transmissió d'imatges. En aquest escenari és interessant explotar les possibilitats que ofereixen les noves tecnologies basades en la percepció del sistema visual humà, ja que són aquestes les que permeten un grau de compressió més elevat.

En el contexte de modificar l'estàndard JPEG2000 per a que la informació sigui transmesa per order d'importància visual, la normalització divisiva es presenta com una solució per reduir les dependències tan estadístiques,així com per proporcionar una transformada amb on cada coeficient és proporcional al seu valor perceptiu[1]. En aquest projecte, gràcies a la seva utilització s'ha aconseguit aproximar el JPEG2000 a una representació més pròxima al sistema visual humà. Tot i així, degut a la inexactitud de la mesura visual i les aproximacions que s'han hagut de fer per no sortir de l'estàndard, els resultats no han estat numèricament tant bons com podrien ser.

Com s'ha explicat, la normalització divisiva depèn d'una combinació de cinc paràmetres que poden prendre valors dins d'un ampli rang.Per tal de trobar la combinació òptima d'aquests paràmetres s'ha realitzat una procés d'optimització basat en iteracions individuals. Degut a la no convexitat del sistema que intentem minimitzar (ni la normalització divisiva ni la norma SSIM no són mesures convexes) aquest procés d'optimització no ens garanteix trobar un màxim global, sinó només un de local. L'estudi de mètodes d'optimització més eficaços i adients per trobar la combinació òptima dels paràmetres de la normalització divisiva, queda

obert i es presenta com una de les futures línies a seguir per poder extreure resultats definitius i concloents sobre la viabilitat del mètode proposat en aquest projecte. Per exemple, es podrien utilitzar tècniques com la programació lineal [19], optimització convexa [20] o algoritmes de gradient conjugat [21].

Un altre dels factors que han afectat els resultats finals és el fet d'adaptar la definició de normalització divisiva per tal de complir les restriccions que marca l'estàndard JPEG2000. En aquest procés d'adaptació s'ha hagut d'obviar els passos de normalització i hem hagut de treballar amb els coeficients no normalitzats per crear els bit-planes i ser enviats. Com a futures línies de treball no s'hauria de tenir en compte les limitacions de l'estàndard, i s'haurien d'enviar els coeficients normalitzats. Això milloraria els resultats, però aleshores no estaríem dins del marc de l'estàndard part 1, tot i que s'hauria d'analitzar si és possible mantenir-nos dins de l'estàndard part 2 a pesar d'enviar els coeficients normalitzats.

Per tal de poder avaluar i comparar els diferents sistemes de compressió és necessari utilitzar mètodes de mesura de la qualitat. A més de la mesura PSNR, àmpliament utilitzada en la comparació d'imatges, SSIM ens proporciona una mesura sobre la qualitat perceptiva de les imatges. Les limitacions del PSNR fan del SSIM un paràmetre més adequat per a l'avaluació de la qualitat visual de les imatges. Cal dir però que SSIM, tot i ser una de les millors mesures de distorsió visual que existeixen, no és perfecte: encara hi ha característiques visuals que no estan capturades per aquesta mesura. És per això que també s'han realitzar testos psico-visual amb persones de la qualitat de les imatges. Tot i les limitacions del SSIM, és una de les mesures perceptives més utilitzades actualment, ja que és la que proporciona millors resultats. Ha estat aquesta mesura la utilitzada per optimitzar els paràmetres de la normalització divisiva. Com hem dit, SSIM no és ideal, per tant, una de les línies a seguir a partir d'aquest projecte seria la d'utilitzar una altra mesura perceptiva per realitzar aquesta optimització i, també, per analitzar els resultats obtinguts. Per exemple, una bona prova seria utilitzar la nova mesura que estan desenvolupant el grup VISTA de la Universitat de València [16] i que es basa en la normalització divisiva.

Totes les proves que s'han dut a terme en aquest projecte, s'han efectuat utilitzant i modificant la implementació BOI, que és una implementació de JPEG2000 dissenyada i programada pel grup GICI amb l'objectiu de proporcionar una bona base pel desenvolupament i avaluació de noves idees entorn de l'estàndard. El fet de disposar d'una versió del codi font de BOI ha estat de gran ajuda per poder implementar les millores que s'han proposat en el projecte i per poder avaluar els resultats.

Així doncs com a conclusió final podem dir que la idea d'aplicar tècniques de codificació perceptiva pot ser de gran ajuda en sistemes de compressió, però les restriccions marcades per l'estàndard amb què ens hem trobat en la implementació al llarg de projecte, han dificultat la tasca d'obtenir resultats totalment òptims.

En resum, durant aquest projecte s'han assolit els següents objectius:

- S'ha estudiat i analitzat l'estàndard JPEG2000 així com les tècniques basades en la

percepció visual humana

- S'ha integrat dins de la implementació BOI una nova mesura de la qualitat de la imatge
- S'ha implementat l'algoritme de normalització divisiva dels coeficients wavelets al JPEG2000 i s'han optimitzat els seus paràmetres
- S'han obtingut resultats experimentals i han estat avaluats mitjançant mesures de qualitat de la imatge i tests psico-visuals.

En resum es pot veure que s'han assolit els objectius plantejats a l'inici del projecte i que les conclusions obtingudes obre noves portes per a la recerca.

Bibliografia

[1] J. Malo, I. Epifanio, R. Navarro and E. Simondelli, "Nonlinear Image Representation for Efficient Perceptual Coding". *IEEE Trans. Image Process*, Vol. 15, No.1 Jan. 2006.

[2] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Perceptual image quality assessment: From error visibility to structural similarity," *IEEE Trans. Image Process.*, vol. 13, no. 4, pp. 600–612, Apr., 2004.

[3] Z. Wang and A. C. Bovik, "A universal image quality index," *IEEE Signal Processing Letters*, vol. 9, pp. 81–84, Mar. 2002.

[4] D. Taubman and M. Marcellin, *JPEG2000: Image Compression Fundamentals, Standards and Practice*. Boston, MA: Kluwer, 2001.

[5] D. Taubman, E. Ordentlich, M.Weinberger and G. Seroussi, "Embedded block coding in JPEG2000". *Signal Processing: Image Communication* 17 (2002) 49-72.

[6] A. Skodras, C. Christopoulos and T. Ebrahimi, "The JPEG2000 Still Image Compression Standard". *IEEE Signal Processing Magazine*. Sept., 2001

[7] D. Taubman. "High Performance Scalable Image Compression with EBCOT". *IEEE Trans. Image Proc. IEEE Trans. on image processing*, vol. 9, n. 7, Jul. 2000

[8] M. Marcellin, M. Gormish, A. Bilgin, M. Boliek. "An Overview of JPEG-2000". *Proc. of IEEE Data Compression Conference*, pp. 523-541, 2000.

[9] Programari BOI i documentació a http://www.gici.uab.cat/BOI/

[10] GICI group Department of Information and Communications Engineering, "BOI development manual (version 1.0)". Universitat Autònoma Barcelona. Sept. 2005

[11] B. Girod, "What's wrong with mean-squared error," in *Digital Images and Human Vision*, A. B. Watson, Ed. Cambridge, MA: MIT Press, 1993, pp. 207–220.

[12] P. C. Teo and D. J. Heeger, "Perceptual image distortion," in *Proc. SPIE*, vol. 2179, 1994, pp. 127–141.

[13] A. M. Eskicioglu and P. S. Fisher, "Image quality measures and their per- formance," *IEEE Trans. Commun.*, vol. 43, pp. 2959–2965, Des. 1995.

[14] Pàgina web del "The SSIM Index for Image Quality Assessment" per Z. Wang, http://www.ece.uwaterloo.ca/~z70wang/research/ssim

[15] J. Malo, "Formes funcionals dels paràmetres de la normalització divisiva i rangs dels valors", VISTA, Març 12, 2008, Treball no publicat.

[16] J. Malo, J. Gutiérrez, J. Muñoz and M. Simón. "VistaQualityTools: an image quality assessment toolbox for Matlab", http://www.uv.es/vista/vistavalencia/software/software.html

[17] S. Winkler, "A perceptual distortion metric for digital color video" *Proc. SPIE*, vol. 3644, 1999, pp. 175–184.

[18] T. N. Pappas and R. J. Safranek, "Perceptual criteria for image quality evaluation," in *Handbook of Image and Video Proc.*, A. Bovik, Ed. New York: Academic, 2000.

[19] V. Chandru and M.R.Rao, Chapter 31 "Linear Programming", in *Algorithms and Theory of Computation Handbook*, edited by M.J.Atallah, CRC Press 1999.

[20] S. Boyd and L. Vandenberghe. "Convex Optimization". Cambridge University Press, 2004

[21] R. Pytlak, *Conjugate gradient algorithms in nonconvex optimization*, Editorial Springer, Nov., 2008 # ISBN: 978-3-540-85633-7

[22] A. Haar, "Zur Theorie der orthogonalen Funktionensysteme", *Mathematische Annalen*, 69, pp 331-371, 1910.

[23] I. Daubechies, *Ten Lectures on Wavelets*, SIAM 1992

Apèndix A

ImageCompareMSSIM

```java
package GiciAnalysis;
import GiciException.*;


/**
 * This class receives two images and calculates their SSIM. Usage
 * &amp;nbsp; construct<br>
 * &amp;nbsp; setParameters<br> (tree different methods)
 * &amp;nbsp; compSSIM <br>
 * &amp;nbsp; getMSSIM<br>
 *
 * @author Xènia Albà Cantero
 * @version 1.0
 *
 */
public class ImageCompareMSSIM{

   /**
    * Structural similarity index of the image. Compares local patterns of pixel
intensities that have been normalized for luminance and contrast.
    * <p>
    * Positive values allowed. (From 0 to 1)
    */
   double MSSIM =0;

   //Constants in the SSIM index formula.
   double C1,C2 = 0;

   /**
    * To know if parameters are set.
    * <p>
    * True indicates that they are set otherwise false.
    */
   boolean parametersSet = false;


   /**
    * Constructor that compute SSIM to compare images.
    *
    * @param image1 a 3D float array of image samples (index are [z][y][x])
    * @param image2 a 3D float array of image samples (index are [z][y][x])
    * @throws ErrorException when image sizes are not the same
    * @throws ErrorException when images are not gray-scale
    *
    */
   public ImageCompareMSSIM(){

   }


   /*Default values for L, K1, K2 */
   public void setParameters (){
      //constants
      int L=255; //L: dynamic range of the images. default: L = 255
      double K1=0.01;//K: constants in the SSIM index formula default value: K = [0.01
0.03]
      double K2=0.03;

      C1=Math.pow(K1*L, 2);
      C2=Math.pow(K2*L, 2);
      parametersSet = true;
   }

   /* K1 and K2 set by the user */
   public void setParameters (double k1, double k2){
```

```
      //constants
      int L=255; //L: dynamic range of the images. default: L = 255
      double K1=k1;
      double K2=k2;
      if(K1<0 || K2<0){
         new ErrorException ("K1 and K2 must be positive");
      }
      C1=Math.pow(K1*L, 2);
      C2=Math.pow(K2*L, 2);
      parametersSet = true;

}
   /* L, K1 and K2 set by the user */
   public void setParameters (double k1, double k2, int l){
      int L=l; //L: dynamic range of the images. default: L = 255
      double K1=k1;
      double K2=k2;

      if(K1<0 || K2<0){
         new ErrorException ("K1 and K2 must be positive");
      }
      C1=Math.pow(K1*L, 2);
      C2=Math.pow(K2*L, 2);
      parametersSet = true;
   }


   public void compMSSIM (float[][][] imageX, float[][][] imageY) throws
ErrorException{

      if(!parametersSet){
         throw new ErrorException("Parameters not set.");
      }

      //Size set
      int zSizeX = imageX.length; //number of image components
      int ySizeX = imageX[0].length; //image height
      int xSizeX = imageX[0][0].length; //image width

      int zSizeY = imageY.length; //number of image components
      int ySizeY = imageY[0].length; //image height
      int xSizeY = imageY[0][0].length; //image width

      //Check if images have same sizes
      if((zSizeX != zSizeY) || (ySizeX != ySizeY) || (xSizeX != xSizeY)){
         throw new ErrorException("Image sizes must be the same to perform
comparisons.");
      }
      //Check if there is only one component, gray level image.
      if(zSizeX != 1){
         throw new ErrorException ("The image must be gray level to perform comparisons
with SSIM.");
      }


      int HW=11;
      float[][] block1=new float[HW][HW];
      float[][] block2=new float[HW][HW];
      double [] SSIM = new double[(ySizeX-HW+1)*(xSizeX-HW+1)];

      int index=0;


      float[][] window = gaussian (HW, 1.5);
```

```
      for (int i=0; i<ySizeX-HW+1; i++){
         for (int j=0; j<xSizeX-HW+1; j++){
            for(int s=0; s<HW; s++){
               for(int l=0; l<HW; l++){
                  block1[l][s] = imageX [0][j+l][i+s];
                  block2[l][s] = imageY [0][j+l][i+s];
               }
            }
            SSIM [index]=compSSIM(block1, block2, HW*HW, window);
            index++;
         }
      }

      MSSIM = meanSSIM(SSIM);
   }


   /*Compute SSIM index. L,K1 and K2 defined by the user */
   public double compSSIM (float[][] blockX, float[][] blockY, int tSize, float[][]
window){

      //Memory allocation
      float[] signalX = new float[tSize];
      float[] signalY = new float[tSize];
      float[] signalW = new float[tSize];

      //Convert blocks to signal
      signalX = block2signal(blockX);
      signalY = block2signal(blockY);
      signalW = block2signal(window);


      //Mean intensity
      double muX = mean(signalX, signalW);
      double muY = mean(signalY, signalW);


      //Standard deviation (square root of variance)
      float sigmaX= (float) deviation(signalX, muX, signalW);
      float sigmaY = (float) deviation(signalY, muY, signalW );

      //Correlation
      float sigmaXY= (float) correlation(signalX,signalY, muX, muY, signalW);
      double numerator=(2 * muX * muY+ C1)*(2 * sigmaXY+ C2);
      double denominator=(muX*muX + muY*muY + C1) * ( sigmaX + sigmaY + C2);

      return numerator/denominator;
   }

   /*Convert squared blocks to signals*/

   public float[] block2signal (float[][] block){

      int ySizeX = block.length; //image height
      int xSizeX = block[0].length; //image width

      int tSize =ySizeX*xSizeX; //signal size
      //Memory allocation
      float[] signalX = new float[tSize];
      int i=0;
      for(int y = 0; y < ySizeX; y++){
         for(int x = 0; x < xSizeX; x++){
            signalX[i] = block [y][x];
            i++;
         }
```

```
      }
      return signalX;
   }


   public double meanSSIM (double[] x){
      double SumX = 0;
      for (int k=0; k<x.length; k++){
         SumX +=x[k];
      }
      return (SumX / x.length);
   }

   /*Compute deviation*/
   public double deviation(float[] x, double mu, float [] w){

      double sigma = 0;
      for (int k=0; k<x.length; k++){
         sigma += w[k] * (Math.pow(x[k] - mu, 2));
         //SumX = SumX + x[k]*x[k];
      }
      return sigma;
      //return ((SumX /x.length)-mu*mu);
   }

   /*Compute the mean*/
   public double mean (float[] x, float[] w){

      double mu = 0;
      for (int k=0; k<x.length; k++){
         mu +=w[k]*x[k];
      }

      return mu;
   }

   /*Compute the correlation*/
   public double correlation(float[] x, float[] y, double muX, double muY, float[]
w){

      double tempXY= 0;

      for (int k=0; k<x.length; k++){
         tempXY = tempXY + w[k]*(x[k]-muX)*(y[k]-muY);
      }

      return (tempXY);
   }



   /*Method for computing the gaussian window*/
   public float [][] gaussian (int mida, double sigma)
   {

      float [][] x=new float[mida][mida];
      float[][] y=new float[mida][mida];
      float [][] f=new float[mida][mida];

      int siz= (mida-1)/2;
      int n=-siz;

      for (int k=0; k<x.length; k++){
         for (int j=0; j<x.length; j++){
               x[j][k]=n;
```

```
                y[j][k]=n;

            }
            n=n+1;
        }

        for (int k=0; k<x.length; k++){
            for (int j=0; j<x.length; j++){
                f[j][k]= (float) ((-(x[0][k]*x[0][k] + y[0][j]*y[0][j]))/(2*sigma*sigma));
                f[j][k]=(float) Math.exp(f[j][k]);
            }
        }

        double sum = 0;
        for (int k=0; k<x.length; k++){
            for (int j=0; j<x.length; j++){
                sum= sum + f[j][k];
            }
        }

        double maxim=max(f);
        double eps= 2.2204e-16;

        for (int k=0; k<x.length; k++){
            for (int j=0; j<x.length; j++){
                if(f[j][k]<eps*maxim){
                    f[j][k]=0;
                }

            }
        }
        for (int k=0; k<x.length; k++){
            for (int j=0; j<x.length; j++){
                f[j][k]= (float) (f[j][k]/sum);
            }
        }

        return f;
    }


    public double max(float[][] f) {
        float maximum = f[0][0];   // start with the first value
         for (int i=0; i<f.length; i++) {
            for (int j=0; j<f[0].length; j++) {
                if (f[i][j] > maximum) {
                    maximum = f[i][j];   // new maximum
                }
               }
         }
         return maximum;
    }//end method max


    /*
     * @return SSIM definition in this class
     */
    public double getMSSIM(){
        return( MSSIM) ;
    }

}
```

Apèndix B

BOICode:
Inici normalització divisiva

```
if (NIREPCUse){ //XAC
      //BIT PLANE BLOCK CODE
      System.out.println("Using NIREPC ");

      BlockCode bc = new BlockCode(imageBlocks, NIREPCUse);
      bc.setParameters(BCResolutionLevels, BCLCRateDistortionAdjustment);
      bc.run();
      BCMSBPlanes = bc.getMSBPlanes();
      BCResolutionLevels = bc.getResolutionLevels();
      //BCErrors = bc.getErrors(); we don't need these errors
      BCNumBytes = bc.getNumBytes();
      BCByteStreams = bc.getByteStreams();
      //Free unused memory
      bc = null;

      //XAC - normalize
      int [][][][][][][] imageBlocksNorm = normalization (imageBlocks);

      //BLOCK CODE using normalized samples to get the errors
      bc = new BlockCode(imageBlocksNorm,NIREPCUse);
      bc.setParameters(BCResolutionLevels, BCLCRateDistortionAdjustment);
      bc.run();
      BCErrors = bc.getErrors();
      System.out.println("NIREPC finished ");
      //Free unused memory
      bc = null;

   }
      else {
      //BIT PLANE BLOCK CODE
      System.out.println("Not using NIREPC "); //END - XAC
      BlockCode bc = new BlockCode(imageBlocks, NIREPCUse);
      bc.setParameters(BCResolutionLevels, BCLCRateDistortionAdjustment);
      bc.run();
      BCMSBPlanes = bc.getMSBPlanes();
      BCResolutionLevels = bc.getResolutionLevels();
      BCErrors = bc.getErrors();
      BCNumBytes = bc.getNumBytes();
      BCByteStreams = bc.getByteStreams();
      //Free unused memory
      bc = null;
      }

   imageBlocks = null;
   System.gc();
   //Show statistics
   showTimeMemory("bit plane block coding + mq coding");
   showBPEStatisticsLong("bit plane coding + mq coding", BCNumBytes);
   showBPEStatisticsShort("BPEMQ", BCNumBytes);

   //LAYERS CALCULATION
   LayerCalculation lc = null;

   //BLOCK CONVEX HULL CALCULATION
   BlockConvexHull bch = null;
   bch = new BlockConvexHull(BCErrors, BCNumBytes);
   bch.setParameters(BCResolutionLevels);
   bch.run();
   BCHFeasiblePoints = bch.getFeasiblePoints();
   BCHSlopes = bch.getSlopes();
   //Free unused memory
   bch = null;
   BCErrors = null;
   System.gc();
   //Show statistics
```

```
   showTimeMemory("rate distortion");
   showCPStatistics("bit plane coding + mq coding", BCHFeasiblePoints, BCMSBPlanes);
   lc = new LayerCalculation(BCNumBytes, BCHFeasiblePoints, BCHSlopes,
BCResolutionLevels);
   lc.setParameters(LCType, LCTargetNumBytes, LCTargetNumLayers, LCForceSingleLayer,
BCLCRateDistortionAdjustment);
   lc.run();
   LCAchievedNumBytes = lc.getAchievedNumBytes();
   LCAchievedNumLayers = lc.getAchievedNumLayers();
   LCLayers = lc.getLayers();

   //Free unused memory
   lc = null;
   BCNumBytes = null;
   BCHFeasiblePoints = null;
   BCHSlopes = null;
   System.gc();
   //Show statistics
   showTimeMemory("layer calculation");
}
```

Apèndix C

BOICode: Funcions de la normalització divisiva

```
////////////////////////////////////////////////////////////
   //FUNCTIONS DIVISIVE NORMALIZATION TO THE SAMPLES - XAC /
   ////////////////////////////////////////////////////////////

/** XAC
 * Apply the divisive normalization to the samples
 *
 * @return normalized samples
 */

private int [][][][][][][] normalization( int [][][][][][][] imageBlocks ){

   double [][][][][][][] imageBlocksNorm =null;
   int [][][][][][][] imageBlocksNormInt =null;
   float [][][][][][][] imageBlocksNormAbs =null;

   System.out.println("Starting first normalization... ");

   //find the absolute maximum sample, without low frequency samples
   int maxSam0= maxSample(imageBlocks);

   //Normalizes by the maximum absolute value
   imageBlocksNormAbs = normalizationAbs(imageBlocks, maxSam0);

   //Memory allocation
   int zSize=imageBlocks.length;
   imageBlocksNorm=new double[zSize][][][][][][]; //num of component
   imageBlocksNormInt=new int[zSize][][][][][][]; //num of component

   System.out.println("Starting divisive normalization ");
   //Divisive Normalization

   double[][] hMatrix=null;

   System.out.println("Using: ");
   System.out.println("Gamma is "+ gamma/10 );
   System.out.println("Gain is "+ gainA );
   System.out.println("sigmaHV is "+ sigmaHV/10 );
   System.out.println("sigmaD is "+ sigD/10 );
   System.out.println("theta is "+ theta/10 );
   System.out.println("sigmaXY is "+ sigXY/10 );

   for(int z = 0; z < zSize; z++){

      //Memory allocation
      int numRLevel = imageBlocks[z].length;
      imageBlocksNorm[z] = new double[numRLevel][][][][][];
      imageBlocksNormInt[z] = new int[numRLevel][][][][][];



      for(int rLevel = 0; rLevel < numRLevel; rLevel++){

         //Memory allocation
         int numSubbands = imageBlocks[z][rLevel].length; //num of subbands
         imageBlocksNorm [z][rLevel] = new double[numSubbands][][][][];
         imageBlocksNormInt [z][rLevel] = new int[numSubbands][][][][];


         //H matrix
         double sigmaXY=(sigXY/10)*0.125*(128/Math.pow(2, numRLevel-rLevel));
         int n=(int) Math.pow (2, numRLevel-rLevel);
         hMatrix = new double [ySize/n] [xSize/n];
         hMatrix = getHMatrix( ySize/n,xSize/n, sigmaXY);
```

```
         for(int subband = 0; subband < numSubbands; subband++){

            //Memory allocation
            int numYBlocks = imageBlocks[z][rLevel][subband].length;
            imageBlocksNorm [z][rLevel][subband] = new double[numYBlocks][][][];
            imageBlocksNormInt [z][rLevel][subband] = new int [numYBlocks][][][];

            for(int yBlock = 0; yBlock < numYBlocks; yBlock++){

               //Memory allocation
               int numXBlocks =imageBlocks[z][rLevel][subband][yBlock].length;
               imageBlocksNorm [z][rLevel][subband][yBlock] = new
double[numXBlocks][][];
               imageBlocksNormInt [z][rLevel][subband][yBlock] = new int
[numXBlocks][][];

               for(int xBlock = 0; xBlock < numXBlocks; xBlock++){

                  //Memory allocation
                  int ySize =imageBlocks[z][rLevel][subband][yBlock][xBlock].length;
                  imageBlocksNorm [z][rLevel][subband][yBlock][xBlock] = new
double[ySize][];
                  imageBlocksNormInt [z][rLevel][subband][yBlock][xBlock] = new
int[ySize][];

                  for(int y = 0; y <ySize; y++){

                     //Memory allocation
                     int xSize
=imageBlocks[z][rLevel][subband][yBlock][xBlock][y].length;
                     imageBlocksNorm [z][rLevel][subband][yBlock][xBlock][y] = new
double[xSize];
                     imageBlocksNormInt [z][rLevel][subband][yBlock][xBlock][y] = new
int[xSize];

                     for(int x = 0; x < xSize; x++){
                        if (rLevel == 0){ //not low freq, the divisive normalization is
not applied to rLevel=0
                           imageBlocksNorm [z][rLevel][subband][yBlock][xBlock][y][x]=
(int) imageBlocksNormAbs [z][rLevel][subband][yBlock][xBlock][y][x];
                           }
                           else{
                              //normalize the sample
                              imageBlocksNorm [z][rLevel][subband][yBlock][xBlock][y][x]
= normSample (imageBlocksNormAbs, hMatrix, z, rLevel, subband, numRLevel,yBlock, xBlock,
y, x);
                        }
                     }
                  }
               }
            }
         }
      }
   }

    double maxSamNorm = maxSample (imageBlocksNorm);

    double K = maxSam0/maxSamNorm;

      for(int z = 0; z < imageBlocksNorm.length; z++){
         for(int rLevel = 0; rLevel < imageBlocksNorm[z].length; rLevel++){
            for(int subband = 0; subband < imageBlocksNorm [z][rLevel].length;
subband++){
```

```
              for(int yBlock = 0; yBlock < imageBlocksNorm
[z][rLevel][subband].length; yBlock++){
                  for(int xBlock = 0; xBlock < imageBlocksNorm
[z][rLevel][subband][yBlock].length; xBlock++){
                     for(int y = 0; y <imageBlocksNorm
[z][rLevel][subband][yBlock][xBlock].length; y++){
                        for(int x = 0; x < imageBlocksNorm
[z][rLevel][subband][yBlock][xBlock][y].length; x++){
                           if (rLevel == 0){ //no low freq
                              imageBlocksNormInt
[z][rLevel][subband][yBlock][xBlock][y][x]= (int) imageBlocksNorm
[z][rLevel][subband][yBlock][xBlock][y][x];
                              }
                              else{
                                 imageBlocksNormInt
[z][rLevel][subband][yBlock][xBlock][y][x]=(int) (K*imageBlocksNorm
[z][rLevel][subband][yBlock][xBlock][y][x]);
                              }
                        }
                     }
                  }
               }
            }
         }

   return imageBlocksNormInt;

}



/** XAC
 * Normalizes the sample by the maximum absolute value of the samples
 *
 * @return normalized samples
 */
private float [][][][][][][] normalizationAbs( int [][][][][][][] imageBlocks, int
maxSam ){

   float [][][][][][][] imageBlocksNormAbs =null;

   //Memory allocation
   int zSize=imageBlocks.length;

   imageBlocksNormAbs=new float[zSize][][][][][][];

   for(int z = 0; z < zSize; z++){

      int numRLevel = imageBlocks[z].length;
      imageBlocksNormAbs[z] = new float[numRLevel][][][][][];

      for(int rLevel = 0; rLevel < numRLevel; rLevel++){
         int numSubbands = imageBlocks[z][rLevel].length; //num of subbands
         imageBlocksNormAbs [z][rLevel] = new float [numSubbands][][][][];

         for(int subband = 0; subband < numSubbands; subband++){
            int numYBlocks = imageBlocks[z][rLevel][subband].length;
            imageBlocksNormAbs [z][rLevel][subband] = new float[numYBlocks][][][];

            for(int yBlock = 0; yBlock < numYBlocks; yBlock++){
               int numXBlocks =imageBlocks[z][rLevel][subband][yBlock].length;
               imageBlocksNormAbs [z][rLevel][subband][yBlock] = new
float[numXBlocks][][];
```

```
               for(int xBlock = 0; xBlock < numXBlocks; xBlock++){
                  int ySize =imageBlocks[z][rLevel][subband][yBlock][xBlock].length;
                  imageBlocksNormAbs [z][rLevel][subband][yBlock][xBlock] = new float
[ySize][];

                  for(int y = 0; y <ySize; y++){

                     int xSize
=imageBlocks[z][rLevel][subband][yBlock][xBlock][y].length;
                     imageBlocksNormAbs [z][rLevel][subband][yBlock][xBlock][y] = new
float[xSize];

                     for(int x = 0; x < xSize; x++){

                        if (rLevel == 0){ //not low freq
                        imageBlocksNormAbs [z][rLevel][subband][yBlock][xBlock][y][x]=
imageBlocks [z][rLevel][subband][yBlock][xBlock][y][x];
                        }
                        else{
                           //normalized with their maximum
                           int sample = imageBlocks
[z][rLevel][subband][yBlock][xBlock][y][x];
                           imageBlocksNormAbs
[z][rLevel][subband][yBlock][xBlock][y][x]= (float) sample /maxSam;
                        }//end if
                     }//end x
                  }//end y
               }//end xblock
            }//end yBlock
         }//end subband
      } //end rLevel
   }//end z
   return imageBlocksNormAbs;
}



/** XAC
 * Calculate the maximum value of the samples
 *
 * @return a double representing the maximum sample
 */
public  double maxSample (double [][][][][][][] blockSam) {
   double maximum = Math.abs(blockSam[0][1][0][0][0][0][0]);   // start with the first
value

   for(int z = 0; z < blockSam.length; z++){
      for(int rLevel = 1; rLevel < blockSam[z].length; rLevel++){ //no low freq
         for(int subband = 0; subband < blockSam [z][rLevel].length; subband++){
            for(int yBlock = 0; yBlock < blockSam [z][rLevel][subband].length;
yBlock++){
               for(int xBlock = 0; xBlock < blockSam
[z][rLevel][subband][yBlock].length; xBlock++){
                  for(int y = 0; y <blockSam
[z][rLevel][subband][yBlock][xBlock].length; y++){
                     for(int x = 0; x < blockSam
[z][rLevel][subband][yBlock][xBlock][y].length; x++){
                     if (Math.abs(blockSam[z][rLevel][subband][yBlock][xBlock][y][x])
> maximum) {
                        maximum =
Math.abs(blockSam[z][rLevel][subband][yBlock][xBlock][y][x]);   // new maximum
                     }
                  }
               }
```

```
                }
            }
        }
    }
   }
    return maximum;
}//end method maxSample - XAC



/** XAC
 * Calculate the maximum value of the samples
 *
 * @return a int representing the maximum sample
 */
public int maxSample(int[][][][][][][] blockSam) {
   int maximum = Math.abs(blockSam[0][1][0][0][0][0][0]);   // start with the first
value

   for(int z = 0; z < blockSam.length; z++){
      for(int rLevel = 1; rLevel < blockSam[z].length; rLevel++){ //excepte el residu
de baixa freqüència, rLevel=0
         for(int subband = 0; subband < blockSam [z][rLevel].length; subband++){
            for(int yBlock = 0; yBlock < blockSam [z][rLevel][subband].length;
yBlock++){
               for(int xBlock = 0; xBlock < blockSam
[z][rLevel][subband][yBlock].length; xBlock++){
                  for(int y = 0; y <blockSam
[z][rLevel][subband][yBlock][xBlock].length; y++){
                     for(int x = 0; x < blockSam
[z][rLevel][subband][yBlock][xBlock][y].length; x++){
                        if (Math.abs(blockSam[z][rLevel][subband][yBlock][xBlock][y][x])
> maximum) {
                           maximum =
Math.abs(blockSam[z][rLevel][subband][yBlock][xBlock][y][x]);   // new maximum
                        }
                     }
                  }
               }
            }
         }
      }
   }
    return maximum;
}//end method maxSample - XAC



/** XAC
 * Apply the divisive normalization to a sample
 *
 * @return normalized sample
 */
private double normSample(float [][][][][][][] imageBlocksNormAbs, double[][]
hMatrix, int z, int rLevel, int subband, int numRLevel, int yBlock, int xBlock, int
y, int x ){


   float SampleNormAbs = imageBlocksNormAbs [z][rLevel][subband][yBlock][yBlock][y][x];

   double gamma_bis=gamma/10; //we entered gamma multiply by 10

   int E =numRLevel;
```

```
   //compute the S
   double S= getS (rLevel,subband, E);
   double Sj= S;  // Because we only use the samples in the same rLevel and subband.
Sj= getS (rLevelj,subbandj, E);

   int ySize= imageBlocksNormAbs[z][rLevel][subband][yBlock][xBlock].length;
   int xSize= imageBlocksNormAbs[z][rLevel][subband][yBlock][xBlock][y].length;

   //x and y index values
   int inY= yBlock*ySize+y;
   int inX= xBlock*xSize+x;

   //  computation of the  summation of the denominator
   double sum=0;
   int numYBlocks= imageBlocksNormAbs[z][rLevel][subband].length;

   for(int yBlocki = 0; yBlocki < numYBlocks; yBlocki++){
      int numXBlocks= imageBlocksNormAbs[z][rLevel][subband][yBlocki].length;

      for(int xBlocki = 0; xBlocki < numXBlocks; xBlocki++){

         int ySizeB =imageBlocksNormAbs[z][rLevel][subband][yBlocki][xBlocki].length;

         for(int yi = 0; yi <ySizeB; yi++){
            int coefY= yBlock*ySizeB+yi;
            int xSizeB=
imageBlocksNormAbs[z][rLevel][subband][yBlocki][xBlocki][yi].length;
            int difY = Math.abs(inY-coefY);
            for(int xi = 0; xi < xSizeB; xi++){
               int coefX= xBlock*xSizeB+xi;
               int difX = Math.abs(inX-coefX);
               sum = (sum + hMatrix [difY][difX]*Math.pow(
Math.abs(Sj*imageBlocksNormAbs[z][rLevel][subband][yBlocki][xBlocki][yi][xi]),gamma_bis)
);


            }
         }
      }
   }

   //compute the rest of the divisive normalization formula
   double sampleNormDouble =
Math.signum(SampleNormAbs)*(Math.pow(Math.abs(S*SampleNormAbs),gamma_bis)) / ( 1 + sum);

   return sampleNormDouble;
}



/**
 * XAC
 * Compute the frequency gain, S, formula
 * @return a double representing the frequency gain
 */
private double getS (int rLevel, int subband, int E){

   //parameters, we enter the multiply by 10
   double sigmaHV_bis = sigmaHV/10;
   double sigmaD_bis = sigD/10;
   double theta_bis = theta/10;
   double sigmaS;

   if((rLevel!=0)&&(subband==2)){ //HH
      sigmaS=sigmaD_bis;
```

```
      }else{ //HL and LH
         sigmaS=sigmaHV_bis;
      }

      //compute the S
      double num = Math.pow((E-(E-rLevel)), theta_bis);
      double den = Math.pow(sigmaS, theta_bis);

      double S = gainA*Math.exp(-num /den);

      return S;
   }

   /**
    * XAC
    * Compute H matrix
    *
    * @return a 2-D double array representing the interaction matrix
    */
   private double [][] getHMatrix( int sizeY, int sizeX, double sigmaXY){

      //memory allocation
      double [][] hMatrix=new double[sizeY][sizeX];

      double sum3=0;
      double K=0;


      for(int i = 0; i < sizeY; i++){

         double sum1 = Math.pow((i/sigmaXY), 2);

         for(int j = 0; j < sizeX; j++){

            double sum2 = Math.pow((j/sigmaXY), 2);
            hMatrix[i][j]= Math.exp(-(sum1+sum2));

            if (hMatrix[i][j] < 0.0000001){ //truncate the small values
               hMatrix[i][j] = 0;
            }

            sum3= sum3 + hMatrix[i][j];

         }//end of j
      }//end i

      K=1/sum3;

      //Multiply by K to normalize the matrix

      for(int i = 0; i < hMatrix.length; i++){
         for(int j = 0; j < hMatrix[i].length; j++){

         hMatrix[i][j] = K*hMatrix[i][j];

         }//end of j
      }//end i

      return hMatrix;
   }

//////////////////////////////////////////////
   //END - XAC
//////////////////////////////////////////////
```

Apèndix D

ConvexHull:
Modificació Convex Hull

```
package BOI.BOICoder.RateDistortion;
import GiciException.*;


/**
 * This class receives errors and lengths of image block coding and calculates their
convex hulls. Usage example:<br>
 *   construct<br>
 *   setParameters<br>
 *   run<br>
 *   get functions<br>
 *
 * @author Group on Interactive Coding of Images (GICI)
 * @version 1.0
 */
public class BlockConvexHull{

   /**
    * Definition in {@link BOI.BOICoder.Code.BlockCode#BCErrors}
    */
   long[][][][][][][] BCErrors = null;

   /**
    * Definition in {@link BOI.BOICoder.Code.BlockCode#BCNumBytes}
    */
   long[][][][][][][] NumBytes = null;

   /**
    * Definition in {@link BOI.BOICoder.Code.BlockCode#BCResolutionLevels}
    */
   int[] BCResolutionLevels = null;

   /**
    * Feasible points of each block coding (index meaning
[z][resolutionLevel][subband][yBlock][xBlock][feasiblePoint]):<br>
    *   z: image component<br>
    *   resolutionLevel: 0 is the LL subband, and 1, 2, ... represents next
starting with the little one<br>
    *   subband: 0 - HL, 1 - LH, 2 - HH (if resolutionLevel == 0 --> 0 - LL)<br>
    *   yBlock: block row in the subband<br>
    *   xBlock: block column in the subband<br>
    *   feasiblePoint: feasible point
    * <p>
    * The values represents a point in the convex hull.
    */
   int[][][][][][] BCHFeasiblePoints = null;

   /**
    * Slopes of feasible points of each block coding (index meaning
[z][resolutionLevel][subband][yBlock][xBlock][feasiblePoint]):<br>
    *   z: image component<br>
    *   resolutionLevel: 0 is the LL subband, and 1, 2, ... represents next
starting with the little one<br>
    *   subband: 0 - HL, 1 - LH, 2 - HH (if resolutionLevel == 0 --> 0 - LL)<br>
    *   yBlock: block row in the subband<br>
    *   xBlock: block column in the subband<br>
    *   feasiblePoint: feasible point
    * <p>
    * All values allowed.
    */
   float[][][][][][] BCHSlopes = null;

   //INTERNAL VARIABLES

   /**
```

```
    * To know if parameters are set.
    * <p>
    * True indicates that they are set otherwise false.
    */
   boolean parametersSet = false;


    /**
    * Constructor that receives block lengths and error saved.
    *
    * @param BCErrors definition in {@link BOI.BOICoder.Code.BlockCode#BCErrors}
    * @param NumBytes definition in {@link BOI.BOICoder.Code.BlockCode#BCNumBytes}
    */
   public BlockConvexHull(long[][][][][][] BCErrors, long[][][][][][] NumBytes){
      //Data copy
      this.BCErrors = BCErrors;
      this.NumBytes = NumBytes;
   }

   /**
    * Set the parameters to do block convex hull calculation.
    *
    * @param BCResolutionLevels definition in {@link
BOI.BOICoder.Code.BlockCode#BCResolutionLevels}
    */
   public void setParameters(int[] BCResolutionLevels){
      parametersSet = true;

      //Parameters copy
      this.BCResolutionLevels = BCResolutionLevels;
   }

   /**
    * Performs block convex hull calculation.
    *
    * @throws ErrorException when parameters not set
    */
   public void run() throws ErrorException{
      //If parameters are not set run cannot be executed
      if(!parametersSet){
         throw new ErrorException("Parameters not set.");
      }

      //Performs CH calculation
      BCHFeasiblePoints = new int[BCErrors.length][][][][][];
      BCHSlopes = new float[BCErrors.length][][][][][];

      for(int z = 0; z < BCErrors.length; z++){
         BCHFeasiblePoints[z] = new int[BCErrors[z].length][][][][];
         BCHSlopes[z] = new float[BCErrors[z].length][][][][];

         for(int rLevel = 0; rLevel < BCErrors[z].length; rLevel++){

            BCHFeasiblePoints[z][rLevel] = new int[BCErrors[z][rLevel].length][][][];
            BCHSlopes[z][rLevel] = new float[BCErrors[z][rLevel].length][][][];

            for(int subband = 0; subband < BCErrors[z][rLevel].length; subband++){
               BCHFeasiblePoints[z][rLevel][subband] = new
int[BCErrors[z][rLevel][subband].length][][];
               BCHSlopes[z][rLevel][subband] = new
float[BCErrors[z][rLevel][subband].length][][];


               for(int yBlock = 0; yBlock < BCErrors[z][rLevel][subband].length;
yBlock++){
```

```
                BCHFeasiblePoints[z][rLevel][subband][yBlock] = new
int[BCErrors[z][rLevel][subband][yBlock].length][];
                BCHSlopes[z][rLevel][subband][yBlock] = new
float[BCErrors[z][rLevel][subband][yBlock].length][];

                for(int xBlock = 0; xBlock <
BCErrors[z][rLevel][subband][yBlock].length; xBlock++){

                   if(rLevel <= BCResolutionLevels[z]){

                      int numBitPlanes =
BCErrors[z][rLevel][subband][yBlock][xBlock].length;
                      int numBitPlanesBytes =
NumBytes[z][rLevel][subband][yBlock][xBlock].length;//XAC

                      // Allocate memory for input arguments to CH
                      long[] errors = new long[numBitPlanesBytes * 3];
                      long[] bytes = new long[numBitPlanesBytes * 3];

                      //XAC
                      if(numBitPlanes==numBitPlanesBytes){
                         for(int bitPlane = 0; bitPlane < numBitPlanes; bitPlane++){
                               for(int codingPass = 0; codingPass < 3; codingPass++){
                                  /* Organize errors and lengths in a 1-D vector */
                                  errors[bitPlane*3 + codingPass] =
BCErrors[z][rLevel][subband][yBlock][xBlock][bitPlane][codingPass];
                                  bytes[bitPlane*3 + codingPass] =
NumBytes[z][rLevel][subband][yBlock][xBlock][bitPlane][codingPass];
                               }
                         }
                      }else if(numBitPlanes>numBitPlanesBytes){//less bit planes from
bytes than from errors

                         for(int bitPlane = 0; bitPlane < numBitPlanesBytes;
bitPlane++){
                         //for(int bitPlane = 0; bitPlane < numBitPlanes; bitPlane++){
                            for(int codingPass = 0; codingPass < 3; codingPass++){

                               /* Organize errors and lengths in a 1-D vector */
                               errors[bitPlane*3 + codingPass] =
BCErrors[z][rLevel][subband][yBlock][xBlock][bitPlane][codingPass];
                               bytes[bitPlane*3 + codingPass] =
NumBytes[z][rLevel][subband][yBlock][xBlock][bitPlane][codingPass];
                            }
                         }
                      }else{
                         long errorMin=
minError(BCErrors[z][rLevel][subband][yBlock][xBlock][numBitPlanes-1]);
                         for(int bitPlane = 0; bitPlane < numBitPlanesBytes;
bitPlane++){
                               for(int codingPass = 0; codingPass < 3; codingPass++){

                                  if(bitPlane<numBitPlanes){
                                  /* Organize errors and lengths in a 1-D vector */
                                  errors[bitPlane*3 + codingPass] =
BCErrors[z][rLevel][subband][yBlock][xBlock][bitPlane][codingPass];
                                  bytes[bitPlane*3 + codingPass] =
NumBytes[z][rLevel][subband][yBlock][xBlock][bitPlane][codingPass];
                                  }else{

                                     errors[bitPlane*3 + codingPass] = errorMin;
                                     bytes[bitPlane*3 + codingPass] =
NumBytes[z][rLevel][subband][yBlock][xBlock][bitPlane][codingPass];
                                  }
                               }
```

```
                        }
                    }

                    // Run convex hull algorithm on the two vector 'errors' and
'bytes'
                    CH ch = new CH(bytes, errors);
                    ch.run();

                    // Copy results to output arrays
                    BCHFeasiblePoints[z][rLevel][subband][yBlock][xBlock] =
ch.getFeasiblePoints();
                    BCHSlopes[z][rLevel][subband][yBlock][xBlock] = ch.getSlopes();

                }
            }
        }
    }
  }
}



  /** XAC
   * Calculate the minimum value of the errors in a bitplane
   *
   * @return a double representing the maximum sample
   */
  public  long minError (long [] errors) {
     long min = errors[0];   // start with the first value
     for(int codingPass = 0; codingPass < 3; codingPass++){
        if (errors [codingPass]< min) {
           min =errors [codingPass] ;   // new min
        }

     }
      return min;
  }//end method maxSample - XAC


  /**
   * @return BCHFeasiblePoints definition in {@link #BCHFeasiblePoints}
   */
  public int[][][][][][] getFeasiblePoints(){
     return(BCHFeasiblePoints);
  }

  /**
   * @return BCHSlopes definition in {@link #BCHSlopes}
   */
  public float[][][][][][] getSlopes(){
     return(BCHSlopes);
  }


}
```

Firmat: Xènia Albà Cantero

Bellaterra, gener de 2009

Resum

JPEG2000 és un estàndard de compressió d'imatges que utilitza la transformada wavelet i, posteriorment, una quantificació uniforme dels coeficients amb dead-zone. Els coeficients wavelet presenten certes dependències tant estadístiques com visuals. Les dependències estadístiques es tenen en compte a l'esquema JPEG2000, no obstant, no passa el mateix amb les dependències visuals. En aquest treball, es pretén trobar una representació més adaptada al sistema visual que la que proporciona JPEG2000 directament. Per trobar-la utilitzarem la normalització divisiva dels coeficients, tècnica que ja ha demostrat resultats tant en decorrelació estadística de coeficients com perceptiva [1]. Idealment, el que es voldria fer és reconvertir els coeficients a un espai de valors en els quals un valor més elevat dels coeficients impliqui un valor més elevat d'aportació visual, i utilitzar aquest espai de valors per a codificar. A la pràctica, però, volem que el nostre sistema de codificació estigui integrat a un estàndard. És per això que utilitzarem JPEG2000, estàndard de la ITU que permet una elecció de les distorsions en la codificació, i utilitzarem la distorsió en el domini de coeficients normalitzats com a mesura de distorsió per a escollir quines dades s'envien abans.

Resumen

JPEG2000 es un estándar de compresión de imágenes que utiliza la transformada wavelet y, posteriormente, una cuantificación uniforme de los coeficientes con dead-zone. Los coeficientes wavelets presenta ciertas dependencias tanto estadísticas como visuales. Las dependencias estadísticas se tienen en cuenta en el esquema de JPEG2000, no obstante, no ocurre lo mismo en el caso de las visuales. En este trabajo se pretende encontrar una representación más adaptada al sistema visual humano que la que proporciona JPEG2000 directamente. Para hallarla utilizaremos la normalización divisiva de los coeficientes, técnica que ya ha demostrado resultados tanto en des-correlación estadística de coeficientes como perceptiva, [1]. Idealmente, se quiere reconvertir los coeficientes a un espacio de valores en los cuales un valor elevado de los coeficientes implique un valor más elevado de aportación visual, y utilizar este espacio de valores para codificar. A la práctica, no obstante, queremos que nuestro sistema de codificación este integrado en un estándar. Es por eso que utilizaremos JPEG2000, estándar de la ITU que permite una elección de las distorsiones en la codificación, y utilizaremos la distorsión en el dominio de los coeficientes normalizados como medida de distorsión para escoger que datos se envían antes.

Abstract

JPEG2000 is a wavelet-based image compression standard. After the wavelet transform, the coefficients are scalar-quantized using a dead-zone quantizer. Wavelet coefficients present both statistical and perceptual dependencies. JPEG2000 takes into account coefficient statistical dependencies in its entropy coding scheme, but not the visual ones. In this work, we aim at finding a representation that is more adapted to visual perception than that of the JPEG200 standard. Given the great statistical and perceptual redundancy reduction rates shown by divisive normalization [1] , We propose to introduce the use of divisive coefficient normalization into the JPEG2000 encoding scheme. Ideally, we would like to reconvert the coefficients in a space of values in which higher value of the coefficients implies higher value of visual contribution, and use this space of values to encode. In practice, we want our coding system to be integrated into a standard, so we will use JPEG2000, an ITU standard that allows a choice of the distortions in the coding, and we will use the distortion in the normalized domain as a measure to choose which data have to be sent before.