M.Teresa Masot Ibars

Protecció del copyright per a imatges

Introduir i recuperar les marques en imatges en format JPEG

Autor: M.Teresa Masot Ibars
Consultor: Francesc Sebé Feixas
Universitat Oberta de Catalunya.
Data: 21/06/2002

1 Índex

2 INTRODUCCIÓ

2.1 El Projecte

El projecte consisteix en la implementació d'un esquema de protecció del **"copyright"** per a imatges.

Això implica:

1) la realització del software que permetrà introduir marques imperceptibles en una imatge.

2) el mateix software ha de permetre posteriorment extreure les marques que s'han introduït.

2.1.1 Marc de desenvolupament

Aquest projecte és desenvolupa com a Treball Fi de Carrera de l'Enginyeria Tècnica en Informàtica de Sistemes a la Universitat Oberta de Catalunya, durant el semestre comprés entre Març i Juny del 2002.

2.1.2 Coneixements previs

Per a la realització d'aquest treball es requereixen coneixements de programació en llenguatge JAVA que permet manipular imatges en format JPEG de forma còmoda.

2.1.3 Maquinari i programari

Per a desenvolupar el projecte és necessari:

- Un PC estàndard.
- Microsoft WORD, com a processador de text per redactar els documents pertinents al projecte
- Microsoft Excel per a la planificació i seguiment del projecte.
- Java com a llenguatge de programació, per tant és necessària la utilització d'un compilador de JAVA i les llibreries pertinents per a la manipulació d'imatges JPEG, opcionalment emprant l'entorn JBuilder4 com a eina de desenvolupament.

2.2 Imatges digitals

Les imatges en un ordinador difereixen dels altres tipus d'imatges - com poden ser les fotografies i els dibuixos - en el fet que són digitals, guardades en forma d'una seqüència de bits i que poden ser modificades en diferents aspectes, com la mida i el color. Com es desa aquesta informació també serà diferent segons el nombre de colors que formin la imatge (profunditat de color).
Els paràmetres bàsics que defineixen la imatge són la resolució i la profunditat de color.

- La **resolució**, estrictament parlant, depèn del nombre de píxels (punts) que formen la imatge.
- La quantitat de bits que necessita cada píxel per guardar informació de la imatge determina la **profunditat de color**. Així:
  - 1 bit, blanc (0) i negre (1);
  - 2 bits, 4 colors;
  - 4 bits, 16 colors;
  - 8 bits, 256 colors (o escala de grisos);
  - 16 bits, 65536;
  - 24 bits, 16 milions de colors (True color); o el que és el mateix, 3 bytes, 8 bits per cada canal de color RGB "Red, Green, Blue" (RVA: vermell, verd, blau)

Els formats d'arxiu condicionen la profunditat de color, la resolució i, lògicament, el software que caldrà per llegir-los. Cal indicar que hi ha formats que suporten 24 bits, però que en canvi no accepten el blanc i negre. Els formats més emprats a Internet són el **GIF i el JPEG**.

Una imatge es pot codificar digitalment mitjançant un mapa de bits, uns vectors que formen objectes o amb fractals. Parlarem dels dos primers tipus.

2.2.1 Mapa de bits

El podem associar a una matriu de punts (píxels) que formen la imatge. Una imatge d'alta qualitat pot estar formada per 786.432 píxels (resolució de 1024 d'ample X 768 de llarg). Quan es digitalitza un document imprès mitjançant un escàner s'obté una imatge en mapa de bits.
Els mapes de bits, a diferència dels gràfics vectorials, tenen una resolució fixa. Per tant, obtenim resultats de visualització o impressió òptims quan la respectem. L'ampliació o la reducció implica una distorsió de la imatge, ja que li afegim o li esborrem punts.

GIF i JPEG (o JPG) són els formats d'imatges més usats per posar a les pàgines web. GIF són les sigles de *Graphics Interchange Format* i es va desenvolupar específicament per a gràfics *on line*. JPEG correspon a *Joint Photographic Experts Group* i es va desenvolupar com un mitjà per comprimir fotografies digitals.

2.2.1.1 Format GIF

- Utilitza un algorisme de compressió sense cap mena de pèrdua.
- Els gràfics estan limitats a 256 colors.
- Admeten transparència. Això vol dir que tenen la possibilitat de convertir en transparent o invisible un sol color, de manera que el fons que tingui aquest color sigui invisible.
- Permeten fer animació amb una tècnica de posar moltes imatges en el mateix arxiu GIF.

2.2.1.2 Format JPEG (JPG)

- L'algorisme que utilitza descarta alguns blocs de les dades de les imatges quan les comprimeix. Per aquesta raó és millor només desar una vegada una imatge amb compressió JPEG. Cada vegada que s'utilitzi aquest algorisme, s'eliminen més dades.
- Està optimitzat per a fotografies i imatges escannejades amb 16 milions de colors o, el que és el mateix, de 24 bits de profunditat de color.

2.2.2 La codificació vectorial

Es basa en descriure la imatge mitjançant objectes (equacions algebraiques) que són independents els uns dels altres i que, per tant, es poden manipular per separat. Poden incloure informació en bitmap. Quan es defineix una imatge pel seu contorn s'estalvia memòria.
L'avantatge principal dels gràfics vectorials és que cada peça de la imatge pot ser manipulada separadament. És possible moure objectes individuals al voltant de la pantalla, i allargar-los, rotar-los, duplicar-los o introduir una distorsió. També cal tenir en compte que al modificar la mida de les imatges aquestes no perden definició (en canvi, els bitmaps sí), i a més, els fitxers tenen una mida compacta.
El principal desavantatge d'aquest format és que les imatges, com més complicades són, l'ordinador triga més temps a calcular-les i, per tant, a representar-les.

2.2.3 Compressió

Durant la compressió, les dades que són duplicades o que no tenen cap valor són eliminades o salvades d'una forma més curta, reduint la grandària d'un arxiu. Quan la imatge és corregida o mostrada, el procés de compressió és invertit.
Hi ha dos formes de compressió, sense pèrdues i amb perdudes:

2.2.3.1 Compressió sense pèrdues

La compressió sense pèrdues descomprimeix una imatge tant que la seva qualitat coincideix amb la de la font original. Encara que la compressió sense pèrdues sembli ideal, no produeix molta compressió.

2.2.3.2 Compressió amb pèrdua

Encara que sigui possible comprimir imatges sense perdre qualitat, això no és pràctic en molts casos.
Encara que la compressió amb pèrdua no descomprimeixi imatges a la mateixa qualitat que la imatge original de la font, la imatge roman visualment sense pèrdues i pareix normal si no s'amplia massa.
La idea consisteix a treure les dades que no són obvies per a l'espectador. Per exemple, si les àrees grans del cel tenen el mateix blau, només el valor per a un píxel blau ha de ser salvat amb les posicions d'altres píxels idèntics en la imatge.
L'esquema de compressió és JPEG (Joint Photographic Experts Group) usat en arxius JFIF (JPEG File Interchange Format). Aquest esquema li permet seleccionar el grau de compressió. Les relacions de compressió entre 10:1 i 40:1 són comuns.

3 ESTUDI PREVI

3.1 Fonaments de protecció del copyright

El fort creixement que està experimentant el comerç electrònic obre noves perspectives per al desenvolupament de noves aplicacions informàtiques que serveixin de suport a aquest nou mitjà de relació. Des del seu vessant informàtic, els reptes que planteja el comerç electrònic són molts i molt variats: implementació de sistemes de pagament electrònic, desenvolupament de mètodes de protecció del copyright electrònic, estudi de sistemes d'agents intel·ligents per a la gestió de la informació, etc.

L'aparició del comerç electrònic pur fa que s'hagin de desenvolupar noves aplicacions per tal que la seguretat del comerç electrònic sigui adequada tan pel que fa al comprador com al venedor. El comerç electrònic pur es pot definir com el conjunt de transaccions econòmiques que es poden realitzar totalment a través de la xarxa informàtica, des de la comanda i el pagament del producte fins a la recepció de la mercaderia. Aquest fet només és possible quan el producte venut es pot obtenir en suport electrònic. El handicap que presenten els productes en suport electrònic és que les còpies són molt barates i totalment exactes. Aquest fet redueix els costos de producció per al venedor però al mateix temps afegeix el risc de la pirateria: el comprador redistribueix la còpia de forma il·legal. Sorgeix doncs el problema de la protecció de la propietat intel·lectual, el que es coneix com a copyright electrònic.

Per tal de prevenir la distribució il·legal, cal incloure nous elements de seguretat en la venda de productes en suport electrònic. Els primers intents de prevenció de la pirateria han anat encaminats a la protecció dels productes per tal d'evitar la seva còpia. Però com s'està veient, últimament (el cas fallit d'intent de protecció dels DVD n'és un exemple), intentar evitar la còpia d'una informació, que en el fons no és més que una cadena de bits, és molt difícil, sinó impossible. Per aquest motiu la recerca realitzada en aquest àmbit es concentra en el concepte de *"detectar"* i abandona la idea d'*evitar"*. No es tracta d'evitar que un usuari faci una còpia il·legal del producte sinó d'assegurar-nos que si la fa més tard podrà ser identificat. La idea general és incloure certes marques identificatives del comprador dins del producte de manera que la còpia del producte impliqui també la còpia de les marques i per tant la còpia de la identitat de l'infractor.
Ja veurem quines propietats han de tenir aquestes marques per tal que no puguin ser ni detectades ni esborrades per un pirata.

3.1.1 Introducció

Com ja hem comentat anteriorment, en el comerç electrònic pur, on l'article a vendre està en suport electrònic, se'ns presenta el problema de la distribució il·legal: la pirateria.

Per tal de resoldre aquest problema estan apareixent en els últims anys diferents tècniques destinades a marcar els productes, dels quals podem anomenar entre altres:

Producte	Empresa	Descripció

Digital Right Manager	Microsoft	Sistema per a la distribució via Internet de música, vídeos i altres formats de forma protegida.
AutoKey	Cognicity	Sistema de marcatge de copyright per a fitxers d'àudio.
Cryptolope	IBM	Sistema de gestió de copyrights electrònics.

Com ja hem apuntat anteriorment, la idea és realitzar una sèrie de marques imperceptibles en el producte a vendre de manera que la duplicació del producte implicarà la duplicació de les marques i per tant de la identitat del comprador, que d'aquesta manera podrà ser acusat de redistribució il·legal. Aquestes tècniques es coneixen amb el nom de Fingerprinting.

Si bé la base teòrica de la protecció del copyright és comuna a qualsevol article en format electrònic, de cara a centrar-nos en el nostre treball, només tractarem imatges.

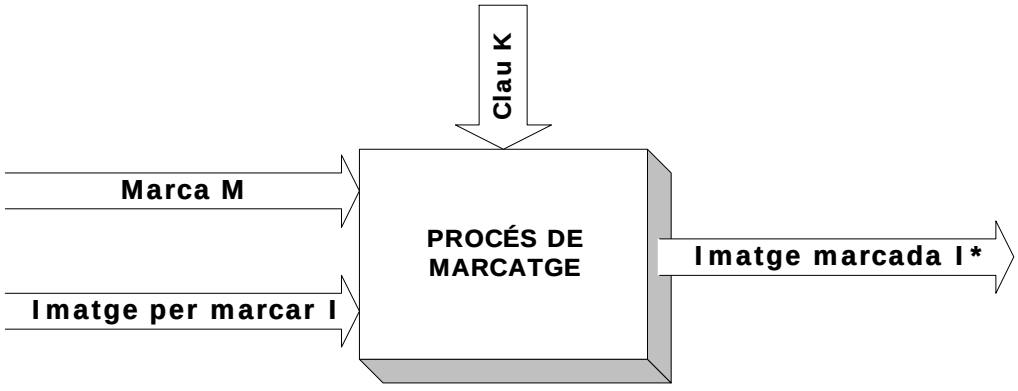
Podem distingir entre dos tipus de marques:

- **Watermarking** o "marques d'aigua": Insereix un missatge de copyright en la imatge, anàleg al símbol . que apareix en les publicacions impreses en paper.
  La marca inclosa en la imatge és la mateixa per tots els usuaris i permet provar l'autoria de la imatge.

- **Fingerprinting** o "empremptatge": Introdueix una mena de número de sèrie, quelcom que identifica al comprador d'aquella imatge concreta, la marca depèn de la identitat del comprador.
  Permet identificar el redistribuidor il·legal d'una imatge, cal però que resisteixi atacs de confabulació de diferents usuaris.

3.1.2 Processos de marcatge i recuperació de la marca

3.1.2.1 Procés de marcatge

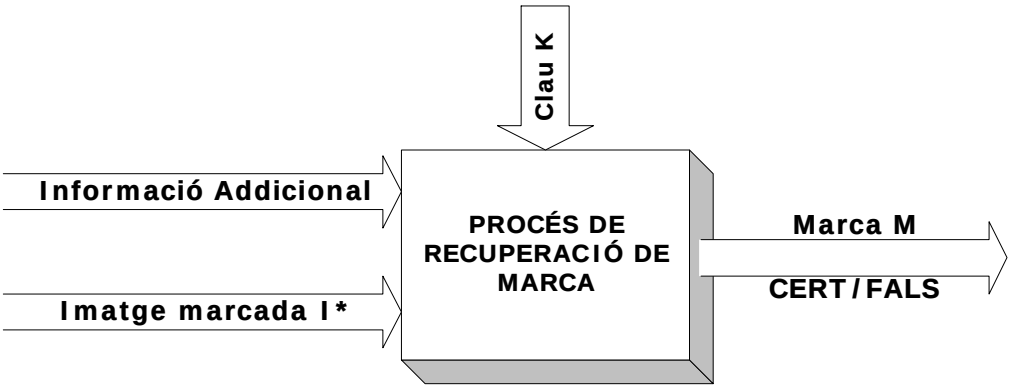


En el procés de marcatge l'algorisme té com a entrades:

- una clau secreta *K* (que serveix per distribuir secretament els bits de la marca dins de la imatge),
- la marca que es vol inserir *M*,
- la imatge que es vol marcar *I*.

Com a sortida, l'algorisme donarà la imatge marcada *I**.

3.1.2.2 Procés de recuperació de la marca



En el procés de recuperació de la marca *M*, les entrades de l'algorisme seran:

- la imatge marcada
- certa informació addicional (que dependrà del tipus d'esquema que fem servir).

Com a sortida obtindrem:

- la marca *M* que hi havia en l'objecte marcat o bé (també depenent del tipus d'esquema que fem servir) un valor binari que ens indicarà si la comprovació que fa el procés de recuperació és certa o falsa.

3.1.3 Propietats dels esquemes de protecció del copyright

Les propietats que podem demanar a un esquema de protecció del copyright són les següents:

3.1.3.1 Imperceptibilitat

La marca que realitzem en la imatge ha de ser imperceptible per al comprador. És a dir, la qualitat de la imatge no pot disminuir a causa de la inclusió de la marca.

3.1.3.2 Robustesa

Aquesta propietat ens mesura la resistència de la marca contra manipulacions o atacs que pot sofrir la imatge. Aquesta propietat és potser la més important ja que l'objectiu final d'un esquema de protecció del copyright és el de mantenir la marca dins la imatge a menys que la imatge resulti tan manipulada que perdi el seu valor com a tal.

D'atacs se'n poden distingir de dos tipus:
- atacs d'un sol atacant
- atacs de confabulacions de diversos atacants

Els atacs d'un sol atacant són els que pot realitzar un comprador amb la seva còpia de la imatge marcada.
Aquest tipus d'atacs es basen en distorsions imperceptibles de la imatge.

Podem citar com a exemples d'atacs:

- L'atac Jitter
- StirMark
- L'atac Mosaic
- Atac en "Echo Hiding"

Els atacs de confabulació són aquells que es realitzen en base al coneixement de més d'una còpia de la imatge. Aquest tipus d'atacs es poden realitzar sobre esquemes de Fingerprinting però no tenen sentit en esquemes de Watermarking ja que aquests últims marquen totes les imatges amb la mateixa informació i per tant el coneixement de diferents còpies no aporta cap informació extra. En el cas de Fingerprinting, com que les marques que s'introdueixen per cada comprador són diferents, comparant diferents còpies es podria revelar la posició d'alguns bits de la marca.

3.1.3.3 Taxa d'informació

Aquesta propietat mesura la quantitat d'informació que es pot introduir en una imatge sense que afecti la seva qualitat. Es pot mesurar com la longitud en bits de la marca inclosa. De cara a augmentar la robustesa de l'esquema, normalment la informació que s'insereix en cada imatge s'inclou de forma redundant. Aquest fet fa que el valor net de taxa d'informació de l'esquema es redueixi. Caldrà doncs, obtenir un equilibri entre taxa d'informació i robustesa.

3.1.3.4 Seguretat per al comprador

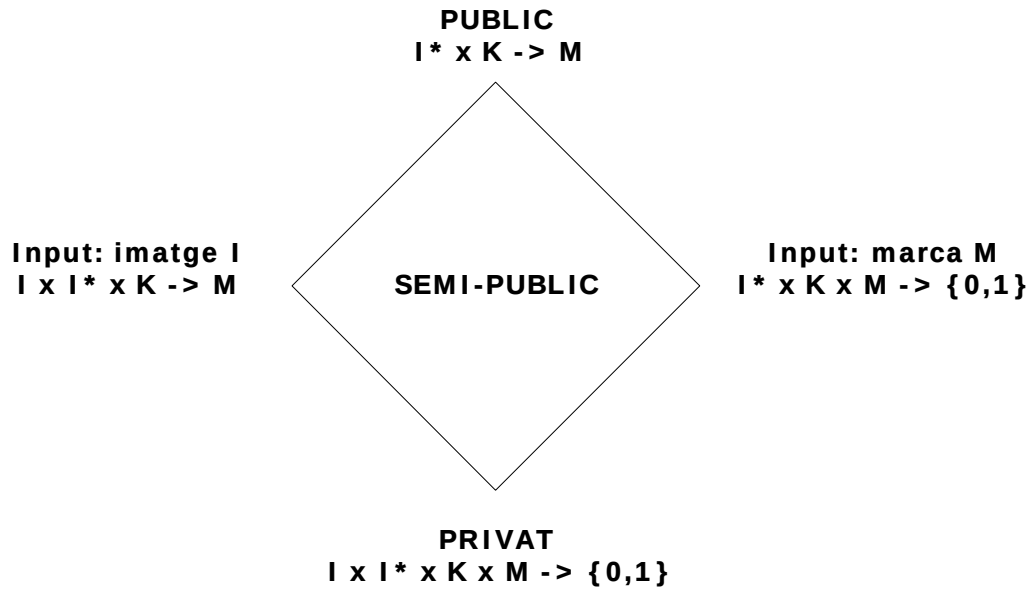
El comprador ha de poder estar segur que no serà acusat falsament de redistribució il·legal.

3.1.3.5 Anonimat

El comprador ha de poder preservar el seu anonimat.

3.1.4 Classificació dels esquemes de protecció del copyright per Watermarking

Els esquemes de Watermarking admeten la següent classificació (representada en la següent figura) depenent de la informació necessària en el procés de recuperació de la marca:



3.1.4.1 **Watermarking privat**

Són aquells que necessiten més informació per la recuperació de la marca. En concret, es necessita la imatge original *I*, la imatge marcada *I**, la clau utilitzada per marcar *K* i la marca inclosa *M*. Com a sortida s'obté cert/fals que determina si és cert que la marca donada com a entrada està continguda en la imatge marcada o si no hi és present.

3.1.4.2 **Watermarking semi-public**

no necessiten tanta informació com els privats. En trobem de dos tipus, els que necessiten la imatge original per recuperar la marca, però no necessiten saber quina marca hi havia inclosa i els que no necessiten la imatge inicial però si la marca. Aquests últims només poden donar com a sortida un argument que determina si la marca donada està inclosa en la imatge marcada o no.

3.1.4.3 Watermarking públic

són els que en tenen prou amb la imatge marcada i la clau per tal d'extreure el valor de la marca inclosa. Tot i ser els millors esquemes que pot haver-hi, ja que necessiten el mínim d'informació per recuperar la marca, hi ha poques propostes d'aquest tipus. A més, aquest tipus d'esquemes no poden resistir atacs de retalls en la imatge ja que al no disposar de la imatge original no poden sincronitzar el tros de la imatge marcada per tal de trobar-ne la marca.

3.1.5 Classificació dels esquemes de protecció del copyright per Fingerprinting

3.1.5.1 **Fingerprinting simètric**

És aquell que no incorpora la propietat de ser segur per al comprador (veure propietats dels esquemes en l'apartat anterior). Aquest fet es deu a que la còpia marcada la coneixen tant el comprador com el venedor (d'aquí la denominació de simètric). Això fa que un venedor deshonest pugui redistribuir la imatge que ha venut a un determinat comprador i després acusar-lo injustament. És més, aquest argument podria ser esgrimit davant d'un jutge per un comprador culpable acusat de redistribució.

3.1.5.2 Fingerprinting asimètric [Pfit96]

Va néixer per tal d'oferir la propietat de seguretat per al comprador. La principal diferència amb els esquemes simètrics és que l'algorisme de marcatge (que abans l'executava el venedor) passa a ser un protocol en el que hi intervenen tant el comprador com el venedor. Aquests esquemes tenen la peculiaritat que el venedor no veu la còpia marcada que al final obté el comprador. D'aquesta manera, com que el comprador és l'únic que la coneix, la posterior localització i identificació de la còpia per part del venedor acusa el comprador de redistribució de manera inequívoca. Aquesta propietat, que podria semblar estranya, s'aconsegueix utilitzant diferents tècniques: el càlcul segur a dues bandes [Pfit96,Domi98a] (on comprador i venedor donen les entrades de la funció i només el venedor obté la sortida: la còpia marcada), la transferència inconscient combinada amb proves de coneixement nul [Domi98c] o l'eina criptogràfica de transferència inconscient compromesa [Domi98b].

3.1.5.3 Fingerprinting anònim [Pfit97]

Es pot veure com una analogia als esquemes de pagament anònim, on el comprador no ha de revelar la seva identitat. En el cas de Fingerprinting, a diferència dels esquemes de pagaments anònims no revocables, la identitat del comprador ha de ser emmagatzemada d'alguna manera per tal de poder obtenir-la en cas que es demostri que és culpable de redistribució. Així doncs, la majoria d'esquemes anònims existents és basen en la utilització de pseudònims controlats per una autoritat de registre que és la que controlaria la correlació de les identitats dels compradors i els seus pseudònims.

3.2 Fonaments de la compressió JPEG

El mètode de compressió JPEG (acrònim de Joint Photographic Experts Group) s'utilitza per a la compressió d'imatges tant en nivells de grisos com en colors (no va bé per imatges en blanc i negre –un bit per píxel-).
Aquest estàndard només comprimeix imatges fixes, el corresponent estàndard per a vídeo és l'MPEG.
El sistema de compressió JPEG és un sistema de compressió amb pèrdues la qual cosa vol dir que sempre perdrem alguna quantitat d'informació quan comprimim utilitzant aquesta tècnica. Per tant, el valor original dels píxels no el podem recuperar i així no podem tornar a tenir la imatge original amb la mateixa qualitat.
Malgrat això, el gran avantatge és que es tracta d'un mètode de compressió molt efectiu i per tant molt útil alhora de transferir imatges sobre les xarxes que tenen molt trànsit o per canals de poc ample de banda com la xarxa telefònica convencional.

3.2.1 Algorisme de compressió

Hi ha quatre passes claus en l'algorisme de compressió JPEG:

1. Extreure un bloc de píxels 8x8 de la imatge.

2. Calcular la transformació discreta del cosinus per cada element del bloc.

3. Arrodonir els coeficients de la transformació discreta del cosinus d'acord amb la **qualitat** especificada per a la imatge.

4. Comprimir els coeficients emprant un esquema de codificació com ara la Huffman o l'aritmètica. El codi final comprimit es desa al fitxer de sortida.

3.2.1.1 Entrada inicial

Les mostres de la imatge inicial s'agrupen en una matriu 8x8 (bloc). La imatge inicial es converteix de l'espai de colors RGB a un espai de brillantor/color com el YUV.

L'ull humà és més sensible a la brillantor que al color, per tant es pot descartar més informació referent al color que a la brillantor. No és necessari l'ús de l'esquema YUV en imatges en escala de grisos.

Un cop s'han manipulat les dades en la matriu 8x8, aquesta matriu serà entrada a l'algorisme de Transformació Discreta del Cosinus.

3.2.1.2 Transformació Discreta del Cosinus (Discrete Cosine Transform (DCT))

Emprant formules matemàtiques és calculen els coeficients DCT per a cada element de la matriu.

Cada element de la matriu 8x8 s'introdueix en l'algorisme DTC i es tradueix. Després de DTC la matriu 8x8 conté 64 coeficients DCT on el primer (coeficient DC) és la mitja dels restants 63 (coeficients AC).

3.2.1.3 Quantificació

En aquest punt ja s'ha produït certa compressió.

L'estàndard JPEG defineix dues taules de constants quantitatives, una per a la brillantor i l'altra per al color. Aquestes contats es calculen basades en la **qualitat de la imatge JPEG** normalment triada per l'usuari.

El número emprat per a calcular les constants s'emmagatzema en la capçalera de la imatge JPEG, fent possible la descodificació dels coeficients.

Un cop construïdes les taules, s'empren les constants de les dues taules per a quantificar els coeficients DCT. Cada coeficient es divideix per la seva constant corresponent i s'arrodoneix a l'enter més proper, obtenint coeficients més petits.

3.2.1.4 Codificació

Per acabar cal codificar les dades emprant un esquema de codificació. Abans de la codificació la matriu s'arregla en un determinat ordre. Els elements que representen freqüències baixes es passen al principi de la matriu i els de freqüències altes al final.

Per a la compressió final es poden emprar tant els algorismes Huffman o esquemes de codificació aritmètics.

3.3 Manipulació d'imatges JPEG en Java

La llibreria emprada per al desenvolupament del projecte correspon al paquet **com.sun.image.codec.jpeg** del llenguatge Java (Sun Microsystems).

Cal notar que les classes del paquet **com.sun.image.codec.jpeg** no són part de les API bàsiques de Java. Formen part de les distribucions JDK i JRE de Sun. No és garantitza la seva disponibilitat en altres implementacions de Java que o siguin de Sun.

Tasques a identificar dins el programari:

1) **Obtenir una imatge JPEG de disc.**
   La classe **JPEGCodec** ens proporciona un descodificador d'imatges JPEG des de disc **JPEGImageDecoder** amb la que obtindrem les imatges a tractar. Les imatges tractades les emprarem amb el format **java.awt.image.Raster** que permet la manipulació dels seus píxels.

2) **Desar una imatge a disc.**
   La classe **JPEGCodec** ens proporciona un codificador d'imatges JPEG cap a disc **JPEGImageEncoder** amb la que generarem les imatges i les desarem a disc.

3) **Identificar si la imatge JPEG tractada és en color o en escala de grisos.**
   La classe **java.awt.image.Raster** permet la identificació del nombre de bandes emprades en la imatge

4) **Comprimir una imatge un determinat tant per cent.**
   Amb el codificador d'imatges JPEG cap a disc **JPEGImageEncoder** generarem les imatges marcades. Permet la inicialització de paràmetres abans de guardar a disc, entre elles el factor de qualitat de la compressió.

5) **Comparar els píxels entre dues imatges.**
   La classe **java.awt.image.Raster** representa els píxels de la imatge, per tant podem accedir al seu valor per banda, x e y i comparar-lo amb les d'una altra imatge.

6) **Canviar els píxels d'una imatge.**
   Amb la classe **java.awt.image.WritableRaster** que permet modificar els píxels de la imatge, podem canviar el valor per banda, x e y d'un píxel.

4 ESPECIFICACIONS DEL TREBALL

4.1 Especificacions funcionals

4.1.1 Procés de marcatge

La funcionalitat de marcatge necessitarà com a entrades:

- la imatge original sense marcar
- la imatge marcada destí
- q percentatge de compressió
- la clau secreta
- llavor aleatòria per a desxifrar la marca

Per implementar la funcionalitat de marcatge, el programa ha de rebre com a entrada la imatge que volem marcar, la marca que hi volem introduir i una clau secreta. Com a sortida ens donarà la imatge marcada.

El procés de marcatge es realitza modificant sensiblement la imatge, és a dir certs píxels de la mateixa. Donat que la marca que inclourem en la imatge és una cadena de bits (0's i 1's), per tal de codificar-los en la imatge prendrem la següent convenció:

- si el píxel de la imatge marcada és superior al píxel de la imatge original voldrà dir que hi ha un 1 codificat.

- si el píxel de la imatge marcada és inferior al píxel de la imatge original voldrà dir que hi ha un 0 codificat.

Per tal de determinar quins píxels de la imatge s'han de modificar i quant s'han de modificar, utilitzarem la compressió JPEG. El procés de marcatge haurà de demanar com a entrada un valor *q* que representa un percentatge de qualitat de compressió usant l'algorisme JPEG. Obtingut aquest valor, l'algorisme comprimirà al *q* % la imatge original a marcar *I*, obtenint-ne *I'*.

La diferència entre aquestes dues imatges *I* i *I'* determinarà els píxels a modificar de la següent manera:

- Només es marcaran els píxels que hagin estat modificats per la compressió JPEG, és a dir, si definim $d_j = |i_j - i'_j|$ només marcarem els píxels i_j tals que $d_i \neq 0$, (on i_j són els píxels de *I* i i'_j són els d' *I'*)

- L'increment o decrement dels píxels que marquem és exactament el valor d_i definit anteriorment.

Si ens hi fixem, la imatge comprimida *I'* només serveix per a veure quins píxels s'han de modificar i quant, ja que les modificacions per obtenir la imatge marcada es fan a la imatge original.

Utilitzarem una clau *K* per xifrar la informació que incloem en la imatge. Per tal que la clau sigui segura haurà de ser un valor aleatori. La longitud d'aquesta clau serà la mateixa que la de la informació que incloem *E*.

Així la marca final *M* que s'inserirà en la imatge serà la suma XOR bit a bit de la informació que volem incloure *E* i la clau *K*, és a dir:

$m_j = e_j \oplus k_j$ on m_j representen els bits d'*M*
e_j representen els bits d'*E*
k_j representen els bits d'*K*

Alhora d'inserir la marca dins la imatge, per tal que l'esquema tingui un elevat grau de robustesa, replicarem la marca tantes vegades com sigui possible. És a dir, si *M* és la marca que volem inserir en la imatge, l'anirem repetint fins a obtenir una cadena *M'* amb tants bits com píxels diferents hi ha entre les dues imatges, és a dir, tants bits com $d_i \neq 0$.

4.1.2 Procés de recuperació de la marca

La funcionalitat d'extracció de la marca necessitarà com a entrades:

- la imatge marcada
- la imatge original sense marcar
- la longitud de la clau secreta
- llavor aleatòria per a desxifrar la marca

és a dir, es tracta d'un esquema semi-public. Com a sortida el procés ha de donar la marca que ha trobat en la imatge o bé un codi d'error en cas que no s'hagi trobat cap marca.

El procediment és el següent:

Primer que tot determinarem en quins píxels de la imatge hi ha bit de la marca enterrats. Això es farà igual que en el procés de marcatge, és a dir, l'algorisme comprimirà al *q* % la imatge original *I*, obtenint *I'*. La diferencià entre aquestes dues imatges *I* i *I'* determina els píxels on hi ha bits de la marca (aquells on $d_j = |i_j - i'_j| \neq 0$).

En aquests píxels, determinarem si el bit inclòs és un 0 o un 1 mirant si el valor del píxel de la imatge marcada és menor o major que el de la imatge original respectivament. Així, recuperem *M'* i amb el valor de la clau *K* podrem extreure la informació *E*.

4.2 Especificacions tècniques

Des del punt de vista tècnic, és important tenir en compte els següents punts:

1) El format de la imatge d'entrada serà JPEG.

2) Abans dels processos de marcatge i extracció caldrà esbrinar si la imatge a manipular és en color o bé en nivells de grisos ja que això determinarà si hi ha una sola matriu de píxels o bé les tres corresponents a les components RGB.

3) En cas que la imatge a marcar sigui en color, el procés de marcatge és realitzarà en totes les tres components RGB de la imatge.

4) Cal anar en compte quan incrementem o decrementem un píxel per tal que no ens passem del límit, és a dir si, per exemple en una imatge en grisos, tenim un píxel que pren el valor 254 i li sumem 3 ens donarà 1 (en mòdul 256). Això s'ha d'evitar perquè voldria dir que canviem un píxel negre per un de blanc i per tant el bit de la marca no compliria els requisits d'imperceptibilitat.

5) El valor de la clau secreta que necessitem en el procés de marcatge el podem obtenir amb una funció rand del llenguatge de programació que utilitzem. Aquest valor s'ha de donar com a sortida en el procés de marcatge per tal que l'usuari que marca l'emmagatzemi per a la posterior utilització en el procés d'extracció.

6) Les imatges que s'utilitzaran per fer les proves les podeu obtenir de la següent adreça: http://www.cl.cam.ac.uk/~fapp2/Watermarking/benchmark/image_database.html

5 IMPLEMENTACIÓ

El programa s'ha implementat en una classe Java amb la següent signatura:
Per a més detalls veure annex A: CODI FONT DE L'APLICATIU

5.1 Constructor

5.1.1 marquesJPG

```
public marquesJPG(java.lang.String psFileIn,
          java.lang.String psFileOut)
```

Constructor
Paràmetres:
psFileIn - Fitxer entrada
psFileOut - Fitxer Sortida

5.2 Mètodes

5.2.1 marcaImatge

```
public void marcaImatge(java.lang.String psMarca,
             long plSeed,
             int piCompress)
```

Marca la imatge
Paràmetres:
psMarca - marca
plSeed - llavor aleatòria
piCompress - percentatge de compressió

5.2.2 marcaImatge

```
public void marcaImatge(java.lang.String pMarca)
```

Marca la imatge
Paràmetres:
pMarca - bits de la marca

5.2.3 recuperaMarca

```
public void recuperaMarca(int piLongMarca,
         long plSeed)
```

Recupera la marca
Paràmetres:
piLongMarca - longitud marca
plSeed - llavor aleatòria

5.2.4 getMarca

```
public java.lang.String getMarca(java.util.BitSet bits)
```

Obté la marca d'una llista de bits
Paràmetres:
cadena - de bits
Retorna:
String de 0s i 1s

5.2.5 recuperaMarca

`public java.lang.String` **`recuperaMarca`**`(int piBitsMarca)`

Recupera la Marca de la imatge

Paràmetres:

`piBitsMarca` - # de bits de la marca

Retorna:

marca recuperada

5.2.6 getMarcaXifrada

`public java.util.BitSet` **`getMarcaXifrada`**`(`
`java.lang.String psMarca, long plSeed)`

Obté la marca xifrada
Utilitzarem una clau per xifrar la informació que incloem en la imatge. Per tal que la clau sigui segura haurà de ser un valor aleatori. La longitud d'aquesta clau serà la mateixa que la de la informació que incloem E.
Així la marca final M que s'inserirà en la imatge serà la suma XOR bit a bit de la informació que volem incloure E i la clau K, es a dir:
mj = ej xor kj
on mj representen els bits d'M
ej representen els bits d'E
kj representen els bits d'K

Paràmetres:

`psMarca` - marca E

`plSeed` - llavor aleatòria

Retorna:

El valor de marcaXifrada

5.2.7 desxifraMarca

`public java.util.BitSet` **`desxifraMarca`**`(`
`java.lang.String psMarcaXifrada, long plSeed)`

Desxifrem la marca. Per tal que la clau sigui segura haurà de ser un valor aleatori.
La longitud d'aquesta clau serà la mateixa que la de la informació que volem recuperar E.

Paràmetres:

`plSeed` - llavor aleatòria

`psMarcaXifrada` - El valor de marcaXifrada

Retorna:

El valor de marca

5.2.8 isGrayScale

`protected boolean` **`isGrayScale`**`(java.awt.image.Raster pImage)`

Identifica si la imatge JPEG tractada és en color o en escala de grisos

Paràmetres:

`pImage` - Imatge a comprovar

Retorna:

cert si es imatge en escala de grisos

5.2.9 compressImage

```
protected void compressImage()
```

Comprimeix la imatge

5.2.10 calculateChangeMatrix

```
protected void calculateChangeMatrix(
        java.awt.image.Raster image1,
        java.awt.image.Raster image2)
```

Calcula la matriu de diferencies entre dues imatges
Calcular dij = Iij - I'ij
(on Iij son els píxels de I i I'ij son els de I')

Paràmetres:

image1 - image2 - Imatges a comparar

5.2.11 getCompressedFile

```
public java.lang.String getCompressedFile(
        java.lang.String psFile)
```

Construeix el nom de fitxer on guardarem la imatge comprimida. Afegeix "Compressed" al nom de la imatge

Paràmetres:

psFile - Fitxer imatge original

Retorna:

el nom del nou fitxer

5.2.12 main

```
public static void main(java.lang.String[] args)
```

Programa principal
Format dels paràmetres:
java tfc.marquesJPG -m imatgeIn imatgeOut llavor marca q
java tfc.marquesJPG -r imatgeIn imatgeOut llavor longMarca

Paràmetres:

args - Arguments del main

6 JOCS DE PROVES

Adjunto alguns del jocs de proves realitzats:

6.1 brandyrose.jpg

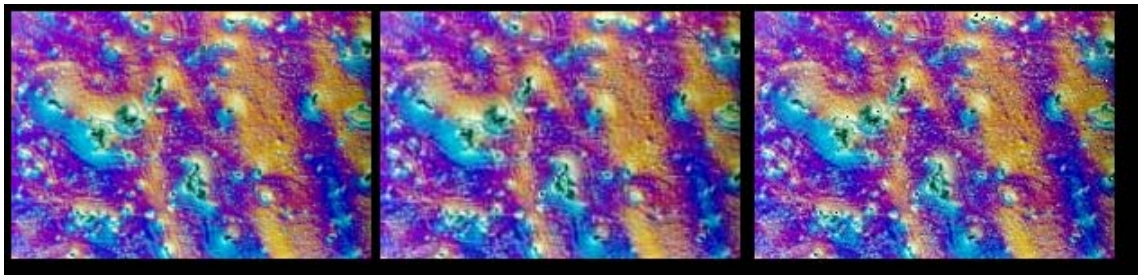
Llavor=10
Marca = 000000000000
Qualitat Compressió = 50

Original **Comprimida** **Marcada**

Marca Recuperada = 000000000000

6.2 alu.jpg

Llavor=10
Marca = 010101010101
Qualitat Compressió = 50



Original **Comprimida** **Marcada**

Marca Recuperada = 010101010101

6.3 arctic_hare.jpg

Llavor=10
Marca = 000000111111
Qualitat Compressió = 50

Original **Comprimida** **Marcada**

Marca Recuperada = 000000111111

6.4 baboon.jpg

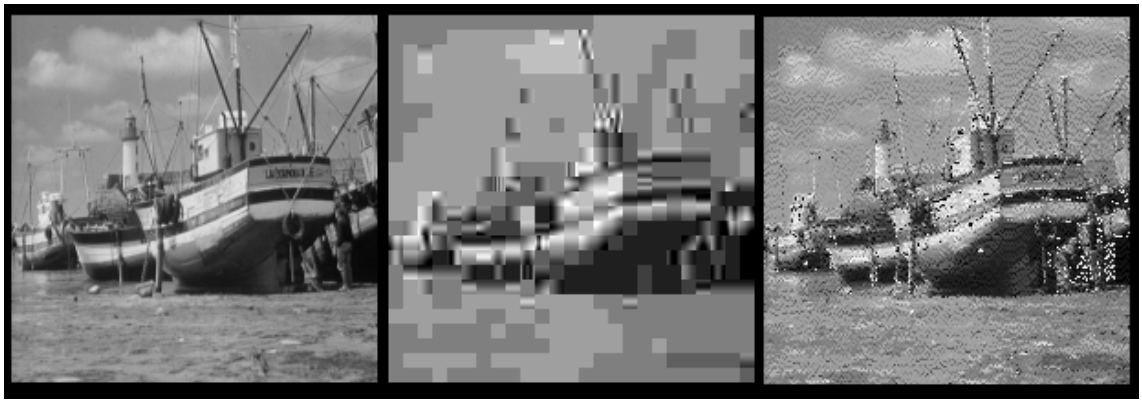
Llavor=10
Marca = 111111111111
Qualitat Compressió = 100

Original **Comprimida** **Marcada**

Marca Recuperada = 111111111111

6.5 fishingboat.jpg

Llavor=10
Marca = 111111111111
Qualitat Compressió = 100

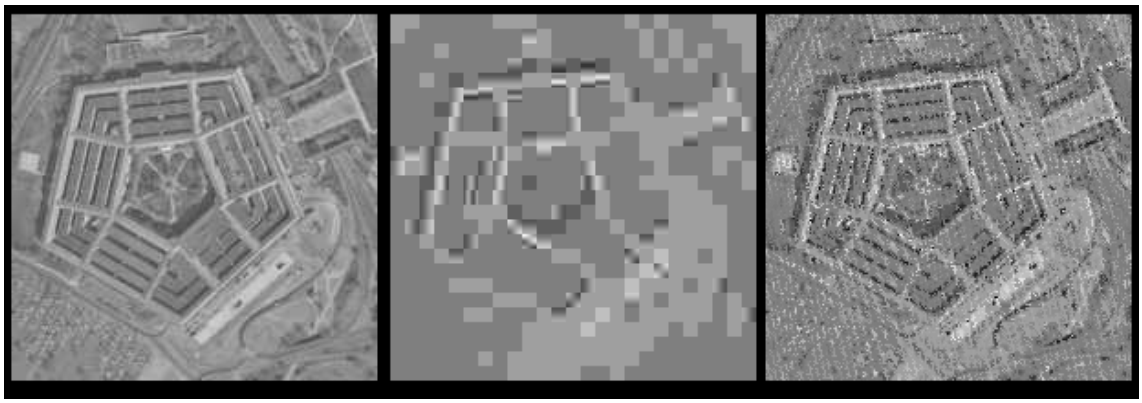


Original **Comprimida** **Marcada**

Marca Recuperada = 000000111111

6.6 pentagon.jpg

Llavor=10
Marca = 11111000001111100000
Qualitat Compressió = 100



Original **Comprimida** **Marcada**

Marca Recuperada = 11111000001111100000

7 PLANIFICACIÓ DEL PROJECTE

7.1 Tasques a realitzar

Detallem a continuació les tasques a realitzar que òbviament van fortament lligades amb les especificacions fetes en la secció 3 d'aquest document.

7.1.1 ESTUDI PREVI

Estudi general dels conceptes bàsics de tractament d'imatges (mètodes l'emmagatzemament (formats), manipulacions estàndards, etc.), tant en format de nivells de grisos com en color.

7.1.2 Estudi JPEG

Estudi detallat del mètode de compressió JPEG per a imatges.

7.1.3 Estudi copyrigth

Estudi dels conceptes bàsics dels esquemes de protecció del copyright per a imatges.

7.1.4 Estudi Tècnic

Estudi tècnic de la manipulació d'imatges amb el llenguatge de programació escollit. Es proposa el C com a llenguatge de programació ja que existeixen diverses llibreries de lliure distribució que permeten treballar en format JPEG de forma còmoda.

7.1.5 Implementació marcatge

Implementació de la part de marcatge de les imatges.

7.1.6 Implementació recuperació

Implementació de la part de recuperació de les marques.

7.1.7 Proves

Determinació del joc de proves i realització de les mateixes.

7.1.8 Memòria

Realització de la memòria del treball.

7.2 Estimació del calendari

			Març			Abril					Maig				Juny			
Tasca	**Inici**	**Fi**	**11**	**18**	**25**	**1**	**8**	**15**	**22**	**29**	**6**	**13**	**20**	**27**	**3**	**10**	**17**	**24**
ESTUDI PREVI	11/03	24/03																
ESTUDI JPEG	18/03	31/03																
ESTUDI COPYRIGHT	25/03	07/04																
ESTUDI TECNIC	01/04	14/04																
IMPLEM. MARCATGE	08/04	26/05																
IMPLEM. RECUPERACIÓ	29/04	16/06																
PROVES	27/05	16/06																
MEMORIA	11/03	21/06																

8 APENDIXS

8.1 APENDIX A: CODI FONT DE L'APLICATIU

```
/*
 *  Copyright (c) 2002 M.Teresa Masot-Ibars
 *  All Rights Reserved.
 */
package tfc;

import com.sun.image.codec.jpeg.JPEGCodec;
import com.sun.image.codec.jpeg.JPEGImageDecoder;
import com.sun.image.codec.jpeg.JPEGImageEncoder;
import com.sun.image.codec.jpeg.JPEGEncodeParam;
import com.sun.image.codec.jpeg.JPEGDecodeParam;
import java.awt.image.Raster;
import java.awt.image.WritableRaster;
import java.util.Random;
import java.util.BitSet;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.awt.Point;

/**
 * Clase per a la introduccio i recuperacio de marques en una imatge JPEG
 *
 * @since     1.2
 * @author    M.Teresa Masot-Ibars
 * @version   1.0 (Juny/2002) M.Teresa Masot-Ibars
 */
public class marquesJPG
{
    /**
     * Prefix afegit al nom del fitxer per a la imatge comprimida
     */
    static String COMPRESSED_SUFIX = "Compressed";
    /**
     * Fitxer a marcar
     */
    String fileInJPEG = "";
    /**
     * Fitxer resultat
     */
    String fileOutJPEG = "";
    /**
     * si es en escala de grisos
     */
    boolean isGrayScale = false;
    /**
     * Imatge llegida
     */
    Raster imageJPEG = null;
    /**
     * Imatge comprimida
     */
    Raster compressedImageJPEG = null;
    /**
     * Imatge marcada
     */
    Raster markedImageJPEG = null;
    /**
     * Percentatge compressio
     */
    int compressPercent = 0;
    /**
     * Amplada en pixels
     */
    int pixelsWidth = 0;
    /**
     * Alcada en pixels
     */
    int pixelsHeight = 0;
```

```
    /**
     * Nombre de bandes en la imatge (GRISOS = 1) (RGB = 3)
     */
    int numBands = 0;
    /**
     * Diferencies entre la imatge original i la comprimida
     */
    double changes[][][] = null;
    /**
     * Nombre de Diferencies entre la imatge original i la comprimida
     */
    int changesSize = 0;
    /**
     * Longitud de la marca
     */
    int marcaSize = 0;


    /**
     * Constructor
     *
     * @param psFileIn Fitxer entrada
     * @param psFileOut Fitxer Sortida
     */
    public marquesJPG(String psFileIn, String psFileOut)
    {
        fileInJPEG = psFileIn;
        fileOutJPEG = psFileOut;
        System.out.println("FITXERS ["+fileInJPEG+"]=>["+fileOutJPEG+"]");
    }


    /**
     * Marca la imatge
     *
     * @param psMarca     marca
     * @param plSeed      llavor aleatoria
     * @param piCompress  percentatge de compressio
     */
    public void marcaImatge(String psMarca, long plSeed, int piCompress)
    {
        try
        {
            compressPercent = piCompress;
            System.out.println("MARCA [" + fileInJPEG + "]=>[" +
                    fileOutJPEG + "] PERCENTATGE [" +
                    compressPercent + "] MARCA [" +
                    psMarca + "]");

            marcaSize = psMarca.length();

            //LLegim la imatge original
            JPEGImageDecoder decoder =
                JPEGCodec.createJPEGDecoder(new FileInputStream(fileInJPEG));
            imageJPEG = decoder.decodeAsRaster();

            pixelsWidth = imageJPEG.getWidth();
            pixelsHeight = imageJPEG.getHeight();

            System.out.println("IMATGE [" + fileInJPEG + "] "+
                "MIDA [" + pixelsWidth + "," + pixelsHeight + "]");

            //Mirem si es en COLOR o en GRISOS
            isGrayScale = isGrayScale(imageJPEG);
            if (isGrayScale)
            {
                System.out.println("FITXER EN ESCALA DE GRISOS");
            }
            else
            {
                System.out.println("FITXER EN COLOR");
            }

            System.out.println("COMPRIMIR FITXER [" + fileInJPEG + "]"+
                " AL " + compressPercent + "% => "+
                "[" + getCompressedFile(fileInJPEG) + "]");
```

```
            //Comprimim la imatge
            compressImage();

            //LLegim la imatge comprimida
            decoder = JPEGCodec.createJPEGDecoder(
                new FileInputStream(getCompressedFile(fileInJPEG)));
            compressedImageJPEG = decoder.decodeAsRaster();

//            decoder = JPEGCodec.createJPEGDecoder(new FileInputStream(fileInJPEG));
//            imageJPEG = decoder.decodeAsRaster();

            System.out.println("CALCULAR DIFERENCIES ENTRE [" + fileInJPEG + "] i "+
                "[" + getCompressedFile(fileInJPEG) + "]");

            //Calculem les diferencies entre les dues imatges
            calculateChangeMatrix(imageJPEG, compressedImageJPEG);

            //Xifrem la marca amb un String aleatori de bits
            String marcaXifrada = getMarca(getMarcaXifrada(psMarca, plSeed));

            //Marquem la imatge
            marcaImatge(marcaXifrada);
        }
        catch (Exception ex)
        {
            ex.printStackTrace();
        }
    }


    /**
     * Marca la imatge
     *
     * @param pMarca  bits de la marca
     */
    public void marcaImatge(String pMarca)
    {
        String lsCurrentPixel = "";
        try
        {
            System.out.println("MARCA [" + pMarca + "] [" + changes.length + "] [" +
                    changes[0].length + "] [" + changes[0][0].length + "]");
            int bitMarcat = 0;
            for (int b = 0; b < numBands; b++)
            {
                for (int x = 0; x < pixelsWidth; x++)
                {
                    for (int y = 0; y < pixelsHeight; y++)
                    {
                        lsCurrentPixel = "[" + b + "][" + x + "][" + y + "]";
                        double ldValorActual = imageJPEG.getSampleDouble(x, y, b);
                        double ldValorNou = ldValorActual;
                        if (changes[b][x][y] != 0)
                        {
                            if (pMarca.charAt(bitMarcat % marcaSize)=='1')
                            {
                                //Tenim un 1 a la marca incrementem el pixel
                                ldValorNou = ldValorActual +
                                    Math.abs(changes[b][x][y]) + 1;
                                ((WritableRaster)imageJPEG).setSample(x, y, b, ldValorNou);
                            }
                            else
                            {
                                //Tenim un 0 a la marca decrementem el pixel
                                ldValorNou = ldValorActual -
                                    Math.abs(changes[b][x][y]) - 1;
                                ((WritableRaster)imageJPEG).setSample(x, y, b, ldValorNou);
                            }
                            bitMarcat++;
                        }
                        else
                        {
                            ((WritableRaster)imageJPEG).setSample(x, y, b, ldValorNou);
                        }
                    }
```

```
                }
            }
            //Desem la imatge marcada
            JPEGImageEncoder encoder =
                JPEGCodec.createJPEGEncoder(new FileOutputStream(fileOutJPEG));
            JPEGEncodeParam param =
                JPEGCodec.getDefaultJPEGEncodeParam(imageJPEG,imageJPEG.getNumBands());
            float lfQuality = 1;
            param.setQuality(lfQuality, false);
            encoder.encode(imageJPEG, param);

        }
        catch (Exception ex)
        {
            System.out.println("EXCEPCIO marcaImatge() PIXEL " + lsCurrentPixel);
            ex.printStackTrace();
        }
    }


    /**
     * Recupera la marca
     *
     * @param piLongMarca  longuitud marca
     * @param plSeed       llavor aleatoria
     */
    public void recuperaMarca(int piLongMarca, long plSeed)
    {
        try
        {
            System.out.println("RECUPERA [" + fileInJPEG + "] DE [" +
                    fileOutJPEG + "] LONG MARCA [" +
                    piLongMarca + "]");

            marcaSize = piLongMarca;

            //LLegim la imatge original
            JPEGImageDecoder decoder = JPEGCodec.createJPEGDecoder(
                    new FileInputStream(fileInJPEG));
            imageJPEG = decoder.decodeAsRaster();

            pixelsWidth = imageJPEG.getWidth();
            pixelsHeight = imageJPEG.getHeight();
            System.out.println("IMATGE [" + fileInJPEG + "] MIDA [" + pixelsWidth + "," + pixelsHeight + "]");

            //LLegim la imatge marcada
            decoder = JPEGCodec.createJPEGDecoder(
                    new FileInputStream(fileOutJPEG));
            markedImageJPEG = decoder.decodeAsRaster();
            System.out.println("IMATGE MARCADA [" + fileOutJPEG + "]");

            System.out.println("CALCULAR DIFERENCIES ENTRE [" + fileInJPEG +
                    "] i [" + fileOutJPEG + "]");
            //Calculem les diferencies entre la imatge original i la marcada
            calculateChangeMatrix(imageJPEG, markedImageJPEG);

            //Recuperem la marca xifrada
            String lsMarcaXifrada = recuperaMarca(piLongMarca);
            System.out.println("MARCA XIFRADA = " + lsMarcaXifrada);
            //Desxifrem la marca
            String lsMarca = getMarca(desxifraMarca(lsMarcaXifrada, plSeed));
            System.out.println("MARCA RECUPERADA = " + lsMarca);
        }
        catch (Exception ex)
        {
            ex.printStackTrace();
        }
    }

    /**
     * Obte la marca d'una llista de bits
     *
     * @param cadena de bits
     * @return String de 0s i 1s
     */
    public String getMarca(BitSet bits)
```

```
    {
        String lsMarca = "";
        for (int i = 0; i < marcaSize; i++)
        {
            if (bits.get(i))
            {
                //Tenim un 1
                lsMarca += "1";
            }
            else
            {
                //Tenim un 0
                lsMarca += "0";
            }
        }
        System.out.println("getMarca("+bits+")="+lsMarca);
        return lsMarca;
    }

    /**
     * Recupera la Marca de la imatge
     *
     * @param piBitsMarca  # de bits de la marca
     * @return marca recuperada
     */
    public String recuperaMarca(int piBitsMarca)
    {
        String lsMarca = "";
        String lsCurrentPixel = "";
        try
        {
            System.out.println("RECUPERA MARCA [" + piBitsMarca + "] [" + changes.length + "] [" +
                    changes[0].length + "] [" + changes[0][0].length + "]");
            int bitMarcat = 0;
            for (int b = 0; b < numBands && bitMarcat < piBitsMarca; b++)
            {
                for (int x = 0; x < pixelsWidth && bitMarcat < piBitsMarca; x++)
                {
                    for (int y = 0; y < pixelsHeight && bitMarcat < piBitsMarca; y++)
                    {
                        lsCurrentPixel = "[" + b + "][" + x + "][" + y + "]";
                        if (Math.abs(changes[b][x][y]) > 1)
                        {
                            if (changes[b][x][y] > 0)
                            {
                                //Tenim un 0
                                lsMarca = lsMarca + "0";
                            }
                            else
                            {
                                //Tenim un 1
                                lsMarca = lsMarca + "1";
                            }
                            bitMarcat++;
                        }
                    }
                }
            }
        }
        catch (Exception ex)
        {
            System.out.println("EXCEPCIO recuperaMarca() PIXEL " + lsCurrentPixel);
            ex.printStackTrace();
        }
        return lsMarca;
    }


    /**
     * Obte la marca xifrada<br>
     * Utilitzarem una clau per xifrar la informacio que incloem en la imatge.
     * <br>
     * Per tal que la clau sigui segura haura de ser un valor aleatori.<br>
     * La longitud d'aquesta clau sera la mateixa que la de la informacio que
     * incloem E.<br>
     * Aixi la marca final M que s'inserira en la imatge sera la suma XOR bit a
```

```
     * bit de la informacio que volem incloure E i la clau K, es a dir:<br>
     * mj = ej xor kj<br>
     * on mj representen els bits d'M<br>
     * ej representen els bits d'E<br>
     * kj representen els bits d'K
     *
     * @param psMarca  marca E
     * @param plSeed   llavor aleatoria
     * @return         El valor de marcaXifrada
     */
    public BitSet getMarcaXifrada(String psMarca, long plSeed)
    {
        BitSet bitsMarca = new BitSet();
        for (int i = 0; i < marcaSize; i++)
        {
            if (psMarca.charAt(i) == '1')
            {
                bitsMarca.set(i);
            }
        }
        BitSet bitsClau = new BitSet(marcaSize);
        Random random = new Random(plSeed);
        for (int i = 0; i < marcaSize; i++)
        {
            if (random.nextBoolean())
            {
                bitsClau.set(i);
            }
        }
        bitsMarca.xor(bitsClau);
        return bitsMarca;
    }


    /**
     * Desxifrem la marca. Per tal que la clau sigui segura haura de ser un
     * valor aleatori.<br>
     * La longitud d'aquesta clau sera la mateixa que la de la informacio que
     * volem recuperar E.<br>
     *
     *
     * @param plSeed          llavor aleatoria
     * @param psMarcaXifrada  El valor de marcaXifrada
     * @return                El valor de marca
     */
    public BitSet desxifraMarca(String psMarcaXifrada, long plSeed)
    {
        BitSet bitsClau = new BitSet(marcaSize);
        Random random = new Random(plSeed);
        for (int i = 0; i < marcaSize; i++)
        {
            if (random.nextBoolean())
            {
                bitsClau.set(i);
            }
        }
        BitSet bitsMarca = new BitSet(marcaSize);
        for (int i = 0; i < marcaSize; i++)
        {
            if (psMarcaXifrada.charAt(i) == '1')
            {
                bitsMarca.set(i);
            }
        }
        bitsMarca.xor(bitsClau);
        return bitsMarca;
    }


    /**
     * Identifica si la imatge JPEG tractada és en color o en escala de grisos
     *
     * @param pImage
     * @return        cert si es imatge en escala de grisos
     */
    protected boolean isGrayScale(Raster pImage)
```

```
    {
        boolean lbGray = false;
        try
        {
            int liBands = pImage.getNumBands();
            if(liBands == 1)
            {
                lbGray = true;
            }
        }
        catch (Exception ex)
        {
            ex.printStackTrace();
        }
        return lbGray;
    }


    /**
     * Comprimeix la imatge
     */
    protected void compressImage()
    {
        try
        {
            JPEGImageEncoder                encoder                =                JPEGCodec.createJPEGEncoder(new
FileOutputStream(getCompressedFile(fileInJPEG)));
            JPEGEncodeParam param = JPEGCodec.getDefaultJPEGEncodeParam(imageJPEG,imageJPEG.getNumBands());
            float lfQuality = 1 - ((float)compressPercent / 100);
            param.setQuality(lfQuality, false);
            encoder.encode(imageJPEG, param);
        }
        catch (Exception ex)
        {
            ex.printStackTrace();
        }
    }


    /**
     * Calcula la matriu de diferencies entre dues imatges<br>
     * Calcular dij = |Iij - I'ij|<br>
     * (on Iij son els píxels de I i I'ij son els de I')
     *
     * @param image1
     * @param image2
     */
    protected void calculateChangeMatrix(Raster image1, Raster image2)
    {
        changesSize = 0;
        try
        {
            numBands = image1.getNumBands();
            changes = new double[numBands][pixelsWidth][pixelsHeight];
            for (int b = 0; b < numBands; b++)
            {
                for (int x = 0; x < pixelsWidth; x++)
                {
                    for (int y = 0; y < pixelsHeight; y++)
                    {
                        double ldVal1 = image1.getSampleDouble(x, y, b);
                        double ldVal2 = image2.getSampleDouble(x, y, b);
                        changes[b][x][y] = ldVal1 - ldVal2;
                        if (changes[b][x][y] != 0)
                        {
                            changesSize++;
                        }
                    }
                }
            }
            System.out.println("# BITS DIFERENTS = [" + changesSize + "] DE [" +
                    numBands * pixelsWidth * pixelsHeight + "] BITS TOTALS");
        }
        catch (Exception ex)
        {
            ex.printStackTrace();
```

```
        }
    }
    /**
     * Construeix el nom de fitxer on guardarem la imatge comprimida.
     * Afegeix "Compressed" al nom de la imatge
     *
     * @param psFile Fitxer imatge original
     * @return el nom del nou fitxer
     */
    public String getCompressedFile(String psFile)
    {
        String lsCompressedFile = "";
        lsCompressedFile = psFile.substring(0,psFile.indexOf(".jpg"))+COMPRESSED_SUFIX+".jpg";
        return lsCompressedFile;
    }
    /**
     * Programa principal<br>
     * Format dels paramatres:<br>
     * java tfc.marquesJPG -m imatgeIn imatgeOut llavor marca q<br>
     * java tfc.marquesJPG -r imatgeIn imatgeOut llavor longMarca
     *
     * @param args  Argumentos del main
     */
    public static void main(String[] args)
    {
        if ((args.length != 5) && (args.length != 6))
        {
            System.out.println("Usage: java tfc.marquesJPG -m imatgeIn imatgeOut llavor marca q");
            System.out.println("or     java tfc.marquesJPG -r imatgeIn imatgeOut llavor longMarca");
        }
        else
        {
            String lsOp = args[0];
            String lsFileIn = args[1];
            String lsFileOut = args[2];
            String lsSeed = args[3];
            String lsMark = args[4];
            long llSeed = Long.parseLong(lsSeed);
            marquesJPG lmarquesJPG = new marquesJPG(lsFileIn, lsFileOut);
            if (lsOp.equals("-m"))
            {
                if (args.length == 6)
                {
                    String lsCompress = args[5];
                    int liCompress = Integer.parseInt(lsCompress);
                    if (liCompress > 100 || liCompress < 15)
                    {
                        System.err.println("Percentatge de compressio erroni cal que sigui [15,100]");
                    }
                    else
                    {
                        lmarquesJPG.marcaImatge(lsMark, llSeed, liCompress);
                    }
                }
                else
                {
                    System.out.println("Usage: java tfc.marquesJPG -m imatgeIn imatgeOut llavor marca q");
                    System.out.println("or     java tfc.marquesJPG -r imatgeIn imatgeOut llavor longMarca");
                }
            }
            else if (lsOp.equals("-r"))
            {
                int liLongMark = Integer.parseInt(lsMark);
                lmarquesJPG.recuperaMarca(liLongMark, llSeed);
            }
            else
            {
                System.err.println("Operació incorrecta [-m] marca [-r] recupera");
            }
        }
    }
}
```

8.2 APENDIX B: BIBLIOGRAFÍA i WEBGRAFIA

http://www.cl.cam.ac.uk/~fapp2/steganography/products.html
Programari i Companyies que es dediquen a marcar productes de distribució online.

http://www.cl.cam.ac.uk/~fapp2/Watermarking/stirmark/
Informació sobre possibles atacs per a imatges i programari que els realitza.

http://www.faqs.org/faqs/compression-faq/part2/section-6.html
Complerta introducció al format JPEG

ftp://ftp.uu.net/graphics/jpeg/wallace.ps.gz
Article original de G. K. Wallace [Wall91] en que es proposa la compressió JPEG

http://www.w3.org/Graphics/JPEG/jfif3.pdf
El format concret dels fitxers JPEG

http://vision.arc.nasa.gov/publications/mathjournal94.pdf
Una visió més matemàtica de la compressió JPEG o més concretament de la compressió que utilitza la transformada discreta del cosinus (DCT)