
Datavetenskap

---

**Hannes Persson**

# Redovisning av JPEG2000

# -med fokus på robusthet

---

Magisteruppsats

2001:05

# Redovisning av JPEG2000

# -med fokus på Robusthet

**Hannes Persson**

© 2001 Hannes Persson och Karlstads universitet

Denna rapport är skriven som en del av det arbete som krävs för att erhålla en magisterexamen i datavetenskap. Allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Hannes Persson

Godkänd, 2001-06-21

Handledare: Anna Brunström

Examinator: Donald Ross

# Sammanfattning

Detta magisterarbete redovisar den nya bildkodningsstandarden JPEG2000. Grundläggande egenskaper i standarden som exempelvis förlustfri och förlustlös kompression och kodning av intresseregioner förklaras. I JPEG2000 används waveletkodning för att uppnå en hög komprimeringskvot. Principen för waveletkodning av signaler redovisas och kodningsförfarandet i den nya standarden beskrivs mer grundligt. Fokus i detta arbete ligger på robusthetsmekanismerna i den nya standarden. Mekanismerna utvärderas genom tester med olika feldistributioner och med olika konfigurationer på en befintlig implementation av JPEG2000-standarden. Testerna visar att kodarens mekanismer klarar av att upptäcka bit- och skurfel. Dessa tester visar vidare att på grund av den wavelettransformering som utförs i JPEG2000-kodaren, och hur datan sedan placeras i bitströmmen, kan dataförluster tillåtas, men på en bekostnad av den visuella bildkvaliteten.

# Description of JPEG2000

# -with focus on Robustness

## Abstract

This master thesis presents the new image compression standard JPEG2000. Basic attributes in the standard like lossless and lossy compression and region of interest coding are briefly described. JPEG2000 uses wavelet coding to achieve a high compression ratio. A description of the principles of basic wavelet compression of signals and a more thorough description of the coding steps in JPEG2000 are provided. The focus of this thesis work is the error-robustness of the new standard. The robustness of an existing implementation of the JPEG2000 standard is evaluated through tests with different kinds of error-distributions and coder configurations. The tests show that the coder can detect bit and burst errors. The tests also show that, because of the wavelet decomposition and depending on how the data is positioned in the file, a data loss can be accepted but with loss in image quality.

# Tack

Jag vill tacka min handledare Anna Brunström för en konstruktiv och professionell handledning.

# Innehållsförteckning

# Figurförteckning

# Tabellförteckning

# 1 Introduktion

Dagens användning av multimediateknologier ställer högre krav på de ingående komponenternas tekniska lösningar och funktioner. Den digitala bilden är en viktig komponent i multimediaupplevelsen. Hanteringen av digitala bilder förväntas ge en högre kompressionsförmåga och samtidigt hålla en hög bildkvalitet. För att möta dessa krav på stillbildskodning har det utvecklats en ny kodningsstandard kallad JPEG2000. JPEG2000-standarden tillhandahåller egenskaper som endast delvis funnits i tidigare standarder. Waveletkodning, aritmetisk kompressionskodning, möjligheten att fokusera på ett avgränsat område av bilden, progressiv visualisering med avseende på kvalitet och upplösning, förlustfri och förlustkompression och tolerans mot bitfel är några av de intressanta egenskaper som återfinns i JPEG2000. Dessa egenskaper förklaras i detta arbete. Från [30] framgår det att den nya standarden, som planeras vara klar i slutet av år 2001, ska vända sig till områden som andra standarder inte har lyckats tillgodose. Tillsammans med ovanstående tekniker och en öppen arkitektur ska JPEG2000 enligt [20] tjäna en bredare marknad och fler applikationsområden än sin föregångare JPEG. Internet, medicinska bilder, mobila trådlösa applikationer, digitala bildbibliotek, kameror, faxar och e-handel utgör några exempelområden.

Syftet med detta arbete är att ge en översikt av den nya kodningsstandarden JPEG2000 och dess inre struktur. Arbetets senare del behandlar framförallt JPEG2000-kodarens robusthet gällande olika typer av fel i bildströmmen. Arbetet riktar sig främst till datavetare, som önskar få en bredare förståelse för den nya JPEG2000-standarden, men även till utvecklare av system där bildhantering är av central betydelse.

Kapitel 2 beskriver översiktligt den nya JPEG2000-standarden med dess möjligheter och begränsningar. Syftet med kapitlet är även att kontrastera den befintliga JPEG-standarden med den nya standarden. En redogörelse för ett enklare fall av wavelettransform tillsammans med centrala komponenter i bildkodning ges i kapitel 3, för att öka förståelsen av denna typ av komprimering. Kapitel 4 ger en förklaring av den nya standardens kodningsförfarande. Kapitlet syftar till att öka förståelsen för kapitel 5 som redogör för robusthetsmekanismerna som används i JPEG2000. Kapitel 6 redovisar genomförda tester som utförts med en befintlig JPEG2000-kodare [16]. Testerna är utförda med olika typer av felmönster och tillgängliga robusthetsmekanismer. I kapitlet presenteras även beräknade värden på kvaliteten och typiska

visuella resultat för respektive testfall. Uppsatsen avslutas med kapitel 7 som ger en sammanfattande bild av den kommande bildkodningsstandarden. Kapitlet tar även upp förslag på vidare utökningar av arbetet.

# 2 Grundläggande egenskaper i JPEG2000

I detta kapitel ges en översikt över de mest grundläggande egenskaperna i den nya bildkodningsstandarden (i bilaga A.1 återfinns en kort orientering om hur en bild kan representeras i en dator). JPEG2000-standarden består av flera delar och en kort redogörelse för varje del äger rum nedan. En jämförelse angående bildkvaliteten mellan den tidigare JPEG-standarden och den nya utförs. Egenskaper som förlustfri respektive förlustkompression samt kodning av intresseområden presenteras också.

## 2.1 Historik

Redan 1994 började tanken på en ny bildkomprimeringsmetod. Det var en person vid namn Ahmad Zandi, som försökte utveckla ett förlustfritt kompressionssystem för medicinska bilder. Arbetet innebar att han använde sig av en wavelettransformation (se kapitel 3) för att uppnå en effektiv kompression. Utvecklare vid Ricoh [28] i Silicon Valley fick upp ögonen för Zandis arbete. Då hans arbete endast innefattade ett system, som klarade både förlustfri och förlustkompression, började de att titta på ytterligare delar som skulle kunna innefattas i samma system. Bilder med olika karakteristik undersöktes, till exempel bilder som innehåller text och bilder med låg och hög upplösning för Internet och i fotografier. Att kunna använda samma metod för helt olika typer av bilder var något som inte var möjligt vid denna tidpunkt.

Med en lång lista av applikationer, som det nya kompressionssystemet skulle täcka, blev målet att hitta ett system som kunde ta ut den mest relevanta informationen ur en godtycklig dataström. En applikation skulle till exempel kunna specificera ett avgränsat område i bilden. Området kodades sedan med en högre bitrepresentation. Applikationen skulle även ges möjlighet att visa en delmängd av bilden, och inte för den skull behöva applicera en komplicerad komprimeringsalgoritm utan istället tolka kodströmmen genom ett enklare förfarande.

Med hjälp av en waveletalgoritm som kunde inverteras, utvecklade Ricoh ett system som gav möjlighet att nå en bredare sfär av användningsområden, och som passade deras applikationer. Utvecklare vid Ricoh tog kontakt med JPEG kommittén för att föreslå dem att en ny bildkompressionsmetod var möjlig att utveckla, så att i princip alla typer av bilder skulle kunna kodas i ett och samma format. JPEG kommittén, som var på jakt efter en

förlustlös version av JPEG formatet, JPEG-LS, mottog förslaget och ett samarbete tog fart. JPEG kommittén insåg också behovet av en standard som når ut till många fler applikationer än vad de befintliga kodarna gör. De inledde arbetet med att utveckla en ny standard som gick under namnet JPEG2000.

Initiala förslag till standardens komprimeringsalgoritmer presenterades i Sapporo i Japan sommaren 1997. Redan samma höst i Sydney utvärderades ytterligare mer än 20 olika förslag. Kommittén inriktade sig på ett waveletsystem som kom att fungera som en referens att testa senare system emot. I mars 1998 kom den första versionen av den så kallad verifikationsmodellen (VM) [24]. Syftet med VM var, att den skulle fungera som en testbänk för standarden. VM är en mjukvara som följer specifikationen av standarden, samt innehåller ett antal algoritmer för att utföra kompressionen. Modellen används således för att testa och utvärdera olika algoritmer. Visar algoritmerna ett tillfredsställande resultat i form av effektivitet, gällande bland annat tid och kompression, finns det möjlighet för dem att ingå i standarden. Exempel på teknologier som finns i VM är robusthet (se kapitel 5) och kodning för att uppnå visuella kvalitetsförbättringar av utvalda bildregioner [1]. Vid de möten som ägt rum under utvecklingen av standarden har modellen genomgått förbättringar allteftersom experimenten fortlöpt. Den senaste versionen av VM innehåller inte alla metoder som ska ingå men huvudstrukturen är dock klar. Arbetet med det nya bildkomprimeringssystemet, som inleddes för sex år sedan, beräknas resultera i en komplett internationell standard (IS) år 2001.

## 2.2 De olika delarna i standarden

Den nya bildkodningsstandarden är uppdelad i sju dokument (I - VII) [17]. Varje dokument tar upp olika områden som standarden innehåller, och adresserar olika nivåer av komplexitet i de steg som kodaren gör. Värt att notera är att standarden endast uttrycker krav på avkodaren och bitströmmens syntax [2]. Dokumenten som beskriver standarden uttrycker dock vissa funktioner som kodaren ska uppfylla, men det finns inga krav på hur kodaren ska utföra kompressionen. Detta lämnar utrymme för nya förslag på hur kodaren ska implementera bildkomprimeringen.

Del I beskriver det minsta antalet komponenter som ingår i kodaren. Syftet med den nya standarden är att hålla nere antalet variationer, så att så många olika typer av apparaturer som möjligt ska kunna implementera en avkodare. Exempelvis ska webbläsare, handdatorer, kameror, skrivare och faxmaskiner kunna implementera avkodaren och därmed visa en

komprimerad bild oavsett vilken kodare bilden komprimerats med [1]. I detta arbete är det del I som i huvudsak berörs.

Del II tar upp andra teknologier som tillfredsställer mer specifika applikationer. Bilder komprimerade med kodare från del II kommer således inte alltid att kunna avkodas med en avkodare från del I.

Del III standardiserar den "rörliga" JPEG2000. Ett användningsområde är bland annat högkvalitetsvideo. En motsvarande standard för JPEG har inte existerat tidigare, men har som metod ändå använts för att redigera videosekvenser.

Enligt [7] planeras del IV innehålla tester av olika implementationer av standarden och deras anpassning till varandra.

Deldokument V tar upp den mjukvara som följer specifikationen för hur en bild ska kodas genom den nya standarden. Det finns för närvarande två implementationer av standarden. Den första kodaren, JJ2000, är en JAVA™ implementation och återfinns vid [16]. JJ2000 är framtagen genom ett samarbete mellan Ericsson, Canon (Frankrike) och EPFL (Ecole polytechnique fédérale de Lausanne). Den andra kodaren, Jasper, som återfinns vid [15] är utvecklad av ImagePower och University of British Columbia. Jasper är implementerad i programspråket C. Avsikten med del V är att uppmuntra utvecklingen av fri mjukvara av standarden.

På grund av ett stort intresse för utvecklingen av standarden har ytterligare två delar, VI och VII, planerats [7]. Del VI kommer att behandla filformatet för bilder kodade med en blandning av JPEG2000 och andra kodningsförfaranden. Del VII kommer att vara en teknisk guide över del I.

Ovanstående dokument som redogör för JPEG2000 kommer att bli internationella standarder. Del I och II planeras bli färdiga internationella standarder i december år 2001. Datum för när delarna III - VII blir färdiga har ej fastställts.

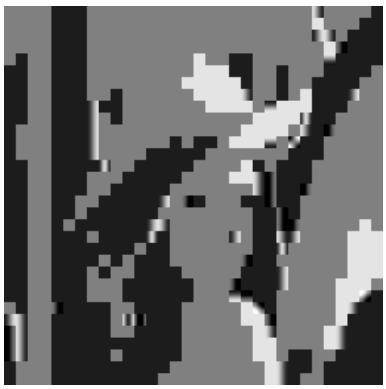
En avkodare är endast tvingad till att implementera kärnan av komponenterna i standarden (del I) och till att kunna tolka kodströmmen från en kodare. JPEG2000 kommer att ha en öppen arkitektur. Ett tänkt scenario är att om en avkodare inte har tillgång till ett verktyg, exempelvis förmågan att kunna använda en specifik waveletkodare, som bilden är kodad med, ska avkodaren kunna begära att få det från kodningskällan för att sedan avkoda bilden. Det ställer då krav på den applikation som används. Vidare ska en kodare kunna komprimera en bild med olika upplösningar och kvaliteter samtidigt. Om en bild är kodad på detta sätt, ska avkodare av olika komplexitet kunna avkoda samma dataström. En enklare avkodare avkodar

bilden med grundläggande kvalitet, medan en mer avancerad avkodare avkodar dataströmmen med en resulterande bild av högre kvalitet och upplösning än den föregående [30].

## 2.3 Kort jämförelse mellan JPEG och JPEG2000

JPEG2000-standarden har som mål att bli en enhetlig standard för olika typer av digitala bilder. Standarden ska täcka bilder med olika karakteristik som naturbilder, medicinska bilder och bilder som innehåller text. För att uppnå denna bredd har helt andra kodningsförfaranden integrerats i kodaren för JPEG2000 än i föregångaren JPEG (för JPEG-standarden se [19]). I JPEG används Diskret Cosinus Transform (DCT)- och huffmankodning för att komprimera bilden. I den nya standarden används istället waveletteknik och aritmetisk kodning (se kapitel 3). Genom att waveletkoda bilderna kan man uppnå en avsevärt högre kompressionsnivå jämfört med tidigare bildkodare. Antalet bitar för att representera en pixel kan bli mycket lågt, något som inte har varit möjligt tidigare [20]. Nivåer på 0,25, 0,125 och 0,0625 bitar per pixel (bpp) kan uppnås med JPEG2000-standarden på vissa gråskaliga bilder, samtidigt som bilden kan bibehålla en godtagbar kvalitet [1]. Kvaliteten för en bild kan uttryckas genom Peak Signal-to-Noise Ratio (PSNR, se bilaga B). Figur 2.1 visar den visuella skillnaden när kodning sker med JPEG2000 respektive JPEG av en gråskalig bild vid 0,25 och 0,125 bitar per pixel. För kodning av JPEG2000-bilder har kodaren vid [16] använts. En standardkodare har använts för att koda JPEG-bilderna och återfinns vid [6]. Den angivna kvalitetsfaktorn (KF) är vald efter önskad bpp-nivå. Ingen optimering har gjorts för kodarna och kodningen har utförts med standardinställningarna. Originalbilderna återfinns i bilaga D.

**A.** JPEG2000 kodad bild 0.25 bpp, PSNR=28,13 dB. **B.** JPEG kodad bild 0.25 bpp, PSNR=24,87 dB, KF 5.



**C.** JPEG2000 kodad bild 0.125 bpp, PSNR=25,23 dB. **D.** JPEG kodad bild 0.125 bpp, PSNR=17,15 dB, KF 0.

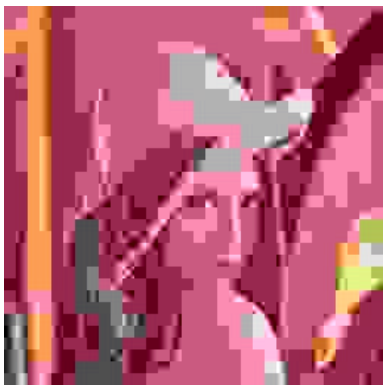
Figur 2.1. Kodning vid lågt antal bitrepresenterade pixlar - bilder i gråskala.

Att koda en bild med ett lågt antal bitar per pixel går även att applicera på färgbilder, men där blir de negativa visuella effekterna mer markanta vid 0,25 bpp. Figur 2.2 visar skillnaden mellan en JPEG2000- och en JPEG-kodad färgbild med pixlar lagrade med 0,5 respektive 0,25 bitar. En klar visuell försämring äger rum vid kodning med JPEG med ett lågt antal bitar per pixel (se figur 2.2 D).

**A.** JPEG2000 kodad bild 0.5 bpp, PSNR=27,49 dB.

**B.** JPEG kodad bild 0.5 bpp, PSNR=26,86 dB, KF 12.

**C.** JPEG2000 kodad bild 0.25 bpp, PSNR=25,17 dB.



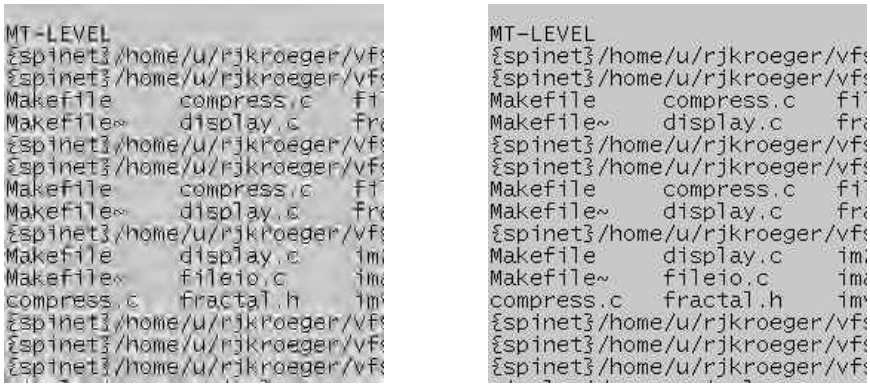
**D.** JPEG kodad bild 0.25 bpp, PSNR=18,37 dB, KF 2.

Figur 2.2. Kodning vid lågt antal bitrepresenterad pixlar - bilder i färg.

Jämfört med JPEG är JPEG2000-kodare mer komplexa. Befintliga implementationer av JPEG2000 är ibland upp till tre gånger långsammare än optimerade JPEG-kodare. Även om hastigheten på nya kodare ökar genom nya implementationer, kommer de inte att nå samma hastigheten som JPEG [24]. Generellt så kräver JPEG2000 även mer minne för att koda bitströmmen, men detta är något som kan optimeras bort [24]. Det finns möjlighet att sätta en högsta nivå på hur mycket minne som får användas. Det gör det möjligt för bland annat skrivare och scanners att implementera standarden.

Den befintliga JPEG-standarden är inte anpassad för alla typer av bilder. Den bör inte användas för att koda svartvita bilder eller bilder som innehåller text [13]. Däremot är den väl anpassad för 24-bitars färgbilder och gråskaliga bilder. JPEG2000 klarar av att koda i stort sett alla typer av bilder, allt från naturbilder, fotografier till röntgenbilder och datorgenererade bilder [20]. Bilder innehållande text som kodats med en befintlig JPEG2000-kodare visar dock att en av de två wavelettransformer som används som standard, inte är anpassade för dessa typer av bilder. Arbeten på andra typer av wavelettransformer har dock visat på klart bättre PSNR-resultat [12]. Den öppna arkitekturen i JPEG2000 tillåter att en godtycklig wavelettransform används. Utförda kodningstester av en bild innehållande text med en

nuvarande JPEG2000-kodare och en standard JPEG-kodare (dessa kodare återfinns vid [16] respektive [6]), visar att JPEG-kodaren ger en klart bättre visuell upplevelse (se figur 2.3).



**A**. JPEG2000, PSNR 18,58 dB, 2 bpp. **B**. JPEG, PSNR 21,98 dB, 2 bpp.

Figur 2.3. Kodning av bilder innehållande text.

## 2.4 Ytterligare egenskaper

Följande delkapitel redogör för de egenskaper som finns i standarden JPEG2000.

### 2.4.1 Förlustfri eller förlustkompression

JPEG2000 kan koda en bild både genom förlustfri och förlustkompression (för exempel se figur 2.4). Förlustfri kompression sker när den avkodade datan är exakt lika med originalet. Förlustkompression sker när den avkoda datan är en replik av originalet, men nödvändigtvis inte exakt lika. Förlustfri kompression är aktuell för medicinska bilder där förluster oftast inte är acceptabla. I bildarkiv är nödvändigtvis inte varje presentation av en bild tvungen till att vara förlustfri. Däremot ska själva bevarandet av bilden vara förlustfritt.

**A.** PSNR=34,10 dB. **B.** Samma kvalitet som originalet.

Figur 2.4. JPEG2000 komprimering (A) med och (B) utan förluster.

Även vid förlustkompression av en bild, där bildinformation faller bort kan bilden ändå visuellt vara av mycket god kvalitet (se figur 2.5, se även bilaga D för en jämförelse). Filstorleken på bilden är mindre jämfört med motsvarande version som komprimerades utan förluster.

2 bpp, PSNR=38,10 dB, 16121 bytes.

Figur 2.5. JPEG2000 förlustkomprimerad bild.

### 2.4.2 Kodning av intresseregioner

I många applikationer där digitala bilder förekommer (e-handel, medicinska sammanhang) är det intressant att fokusera på en viss region av bilden (se figur 2.6). Det existerar två typer av bildregionskodning i JPEG2000. Den första kallas dynamisk områdesavkodning [8]. Dynamisk avkodning av bildregioner innebär att mottagaren kan avkoda valda delar av bilden med avseende på olika bildkvaliteter. Dynamisk områdesavkodning är inte beroende av hur bilden kodats, utan sker helt och hållet på mottagarsidan. Krav är dock att bilden tas emot i

progressivt läge med avseende på kvalitet (se bilaga A). Om en mottagare väljer att se ett område av bilden helt förlustfritt krävs det att bilden från början är kodad utan förluster.

**A.** Bild i gråskala, 1 bpp, PSNR=20,57 dB. **B.** Bild i färg, 1.2 bpp, PSNR=21,07 dB.

Figur 2.6. Bilder kodade med cirkulär region av intresse.

I nätverk med liten bandbredd där trådlösa mobila enheter används, kan det komma att bli ett intressant tillvägagångssätt. I [8] exemplifieras dynamisk områdeskodning: När man i den bärbara enheten till en början får upp en bild med låg kvalitet, kan man som användare välja ut den del av bilden som ser mest intressant ut, för att på så sätt påskynda bildöverföringen. Tiden det tar att definiera ett subjektivt intressant område i en bild kan i vissa fall ta längre tid än att vänta på att hela bilden ska bli synlig med högsta bildkvalitet. I en progressiv bildvisuallisering kommer man dock att kunna välja att fokusera på ett intressant bildområde tidigt i bildöverföringsprocessen, då JPEG2000 ger en förståelig bild redan vid låga bitnivåer [8].

Förfarandet att definiera en given region i en bild med högre kvalitet än omkringliggande områden kallas statisk områdeskodning [20]. Den statiska kodningen äger i motsats till den dynamiska avkodningen rum vid kodningen av bilden och kan inte ändras efter det att bilden blivit kodad [8]. Det kan förekomma flera regioner i en och samma bild. En region kan även existera inuti en annan, och varje region kan vara av godtycklig form och storlek. Vid statisk områdeskodning kommer fler bitar att krävas för det angivna området, och för att hålla den totala filstorleken förhållandevis liten, kommer bakgrunden som inte ingår i det specificerade området, representeras med färre bitar [3].

### 2.4.3 Filformatet JP2 / JPX

Ett filformat har definierats för JPEG2000-komprimerade bilder som uppfyller del I i standarden [2]. En JPEG2000-kodad bild i ett datorsystem identifieras genom filändelsen .jp2

(eller .JP2). Alla applikationer behöver dock inte använda sig av denna ändelse. Filändelsen utgör en hjälp att identifiera en bildfil som är kodad genom den nya standarden. Filformatet rymmer den nödvändiga kodade bilddatan tillsammans med ytterligare kritisk information som behövs vid avkodning. Utöver det rymmer formatet även olika typer av metadata. Metadatan är information om bilden som inte krävs för själva avkodningen. Exempelvis kan datan innehålla information om vilken gråskala eller färgrymd bilden har, eller information som är specifik för en viss tillverkare. Ett annat användningsområde för metadatan är möjligheten att indexera bilder i stora bilddatabaser. Det underlättar sökningen av exempelvis medicinska röntgenbilder. All information (bilddata och metadata) delas upp i olika kategorier och underkategorier som sedan organiseras i filen. Exempelvis kan en kategori ge information om att filformatet är definierat av JPEG2000-standarden, en annan om storleken på bilden. Varje kategori är oberoende av andra, och en avkodare, som inte kan tolka en kategori i en fil, hoppar helt enkelt över den. Grundkravet för alla läsare (applikationer) av JPEG2000-filer är att de ska kunna avkoda alla filer som innehåller grundinformation om bilderna [20].

Kodare som grundar sig på standardens andra del (del II) kan ta med ytterligare kategorier [3]. Vilka kategorier som väljs beror på den applikation som ska tolka och använda formatet. Likt det grundläggande formatet för del I, har kodare som uppfyller standardens andra del, en egen filändelse, .jpx (eller .JPX). Här kan metadatan exempelvis tala om bildens titel, beskriva hur bilden skapades (till exempel genom fotografering), och hur bilden har ändrats sedan originalet. Datan kan även beskriva innehållet i bilden, som namn på personer och platser som bilden visar. Till skillnad från .jp2-formatet tillåter .jpx-formatet fragmentering av innehållet. Bildinformationens plats kan variera från att ligga i mitten eller slutet av filen, till det att informationen läggs i helt andra filer. Detta öppnar möjligheter för ett flertal applikationer. Vid redigering av en bild kan till exempel alla ändringar läggas i slutet av filen, för att senare möjliggöra jämförelse med originalbilden. I Internet-sammanhang kan en fil delas upp i flera mindre filer som sedan distribueras över Internet. Vissa användare kan sedan tillåtas få tillgång till en bild med en hög bildkvalitet, medan andra får en sämre version av samma bild. Ytterligare ett exempel-scenario på .jpx-formatet återfinns i bilaga C.

# 3 Grundläggande kodningstekniker

JPEG2000 innehåller fyra viktiga kodningstekniker för att uppnå en hög kompressionskvot. Dessa tekniker är wavelettransformering, kvantisering av koefficienter, runlength-kodning och aritmetisk kodning. Teknikerna presenteras i följande delkapitel.

## 3.1 Allmänt om wavelets

Utvecklingen av wavelets har framförallt motiverats av behovet av snabba algoritmer och för att på ett så kompakt sätt som möjligt representera datamängder. Wavelets har sin främsta användning i signalbehandling. Waveletteknikenhar även lett till ett antal effektiva algoritmer för användning i bland annat bildkompression, animering och multiupplösning av bilder.

Grunden för den diskreta wavelettransformen lades av Stephen Mallat och Ingrid Daubechies på 80- och 90-talet. Redan 1909 introducerades dock begreppet av Alfred Haar [27]. En wavelet är en funktion eller våg, som är koncentrerad med svängningar runt origo i ett xy-koordinatsystem. Denna funktion planar ut längs med x-axeln för stora värden på x (till skillnad från den diskreta cosinustransformen (DCT), som används för att koda JPEG-bilder, som svänger harmoniskt runt x-axeln). Waveleten är alltså en lokal funktion, och på grund av den egenskapen får den stor betydelse för signalhanteringen, då signaler kan representeras på ett mera kompakt sätt.

Syftet med en wavelettransform är att transformera en signal till en frekvensdomän. Intressant är att en hög signal kommer att mappas mot en låg frekvens i en wavelettransform, medan en låg signal mappas mot en hög frekvens. Vid waveletkodning av en bild kommer sådant beteende att innebära att höga värden transformeras till låga värden, alltså värden nära eller lika med noll. Höga signaler är något ögat inte uppfattar lika lätt som låga signaler, och därmed kan dessa värden tas bort, trunkeras. Vid återskapandet av originalsignalen kommer den trunkerade informationen inte att spela en avgörande roll för den visuella upplevelsen av en bild [31]. Om värden nära noll trunkerats kommer dessa värdena vid avkodningstillfället att ges ett approximativt värde, nämligen noll. Detta leder dock till en förlustkomprimerad version av exempelvis en bild.

### 3.1.1 Haar-wavelet - 1-dimension

Haar-waveleten för signaler i en dimension är troligen den enklaste att förstå. Givet är *a* och *b*, som är två tal som ligger intill varandra i en sekvens av tal. Genom en enkel linjär transformation kan *a* och *b* bytas ut mot medelvärdet *s*. Differensen *d* representerar avståndet mellan värdet *b* och medelvärdet *s*:

$$s = \frac{a+b}{2} \quad (1)$$

$$d = b - s \quad (2)$$

Formel (1) kallas lågpassfilter och formel (2) kallas högpassfilter. Dessa filter utgör Haar-wavelettransformen. Syftet är att reducera antalet bitar som krävs för att representera *a* och *b*. Beräkningen kan inverteras för att återfå *a* och *b*:

$$a = s - d \quad (3)$$

$$b = s + d \quad (4)$$

Detta är idén bakom Haar-waveleten. För en signal $sig_n$ baserad på $2^n$ värden, existerar det $2^{n-1}$ par av värden (*a*,*b*) och lika många medelvärden *s* och differenser *d*. Medelvärdena kan ses som en samling som representerar en lägre upplösning av originalversionen, och differenserna som värden för att återgå till den högre upplösningen. Det är fullt möjligt att applicera ovanstående transform ännu en gång på den approximerade signalen, genom att dela upp i ytterligare en approximerad version och fler differenser. Transformen kan appliceras flera gånger och sedan avsluta då endast ett signalvärde kvarstår tillsammans med ett stort antal differensvärden, nämligen $2^n$-1 värden. Den slutliga signalen representeras av medelvärdet för hela originalsignalen.

Följande resonemang kan förtydligas genom ett exempel från [31]. Givet är en 1-dimensionell bild med en upplösning på fyra pixlar. Bilden har följande pixelvärden:

**[9 7 3 5]**

Genom att parvis approximera ovanstående sekvens (enligt formel (1)) uppnås följande två medelvärden:

**[8 4]**

Differenserna är inte representerade här. Dessa värden kallas detaljkoefficienter och måste finnas tillhands, om originalsekvensen ska återskapas. Således måste även dessa lagras. De lagras i en så kallad filterbank. För paret (9 7) är medelvärdet enligt formel (1) ett mindre än 9 och ett mer än 7. Genom värdet 1 kan nu de två första pixlarna återskapas. Vidare, för det andra och sista paret (3 5), är medelvärdet 4, och därmed är medelvärdet ett mer än 3 och ett mindre än 5, vilket leder till –1 i detaljkoefficient (se tabell 3.1).

| **upplösning** | **approximation** | **filterbank** |
|---|---|---|
| **4** | **[9 7 3 5]** | **-** |
| **2** | **[8 4]** | **[1 -1]** |

Tabell 3.1. Haar koefficienter, upplösning 4 och 2.

Genom att applicera Haar-waveleten ytterligare en gång fås en slutlig upplösning på 1 enligt tabell 3.2.

| **upplösning** | **approximation** | **filterbank** |
|---|---|---|
| **1** | **[6]** | **[2 1 -1]** |

Tabell 3.2. Haar koefficienter, upplösning 1.

Den slutliga wavelettransformen av originalsignalen på fyra pixlar, kan nu representeras genom ett approximerat pixelvärde, följt av detaljkoefficienterna:

**[6 2 1 –1]**

Wavelettransformen appliceras enligt formel (1) och (2), och genom upprepning beräknas medelvärdet och differenskoefficienterna. Ingen information har lagts till eller förlorats. Ursprungsbilden på fyra pixlar är lika stor som den beräknade. Av den slutliga wavelettransformen kan olika upplösningar framställas, genom att lägga till och ta bort de detaljerade koefficienterna från medelvärdena från de lägre upplösningarna. Den första siffran (siffra 6) är alltså den sista approximationen, som är möjlig av originalbilden. Den kommer att vara den pixel som är den lägst representerande upplösning som går att få av originalbilden.

Progressiv överföring av bilder, med avseende på upplösning, kan dra fördel av denna teknik. Den information som sänds först, är den approximerade koefficienten. Mottagarsidan kan då visualisera en liten bild med låg upplösning, och när mer information mottas, i form av detaljkoefficienter, kan en bild med högre upplösning presenteras. Progressivitet med avseende på kvalitet kan också utföras. Genom att approximera detaljkoefficienterna (2, 1, -1 ovan) till 0, kan en bild i full skala presenteras men med en sämre kvalitet än originalbilden:

**[6 0 0 0] -> [6 6 0 0] -> [6 6 6 6]**

Den största fördelen, med att göra en wavelettransform enligt ovan, är att en stor del av de detaljerade koefficienterna antar värden nära noll. Genom att approximera dessa värden kan en högre kompression uppnås. Det sker dock på bekostnad av att originalsignalen inte går att återskapa exakt. Approximeringen introducerar oftast endast mindre felaktigheter i den kodade bilden [31]. Haar-waveleten är den enklaste formen av en transform. Effektivare wavelettransformer existerar. De kan producera fler värden nära noll eller lika med noll för detaljkoefficienterna än Haar-waveleten. Detta gör en större kompression möjlig. Beroende på vilken wavelet som används, och hur den genererar de approximerade och detaljerade koefficienterna för en bild, kommer kompressionskvoten för den kodade bilden att variera.

### 3.1.2 Wavelet - 2-dimensioner

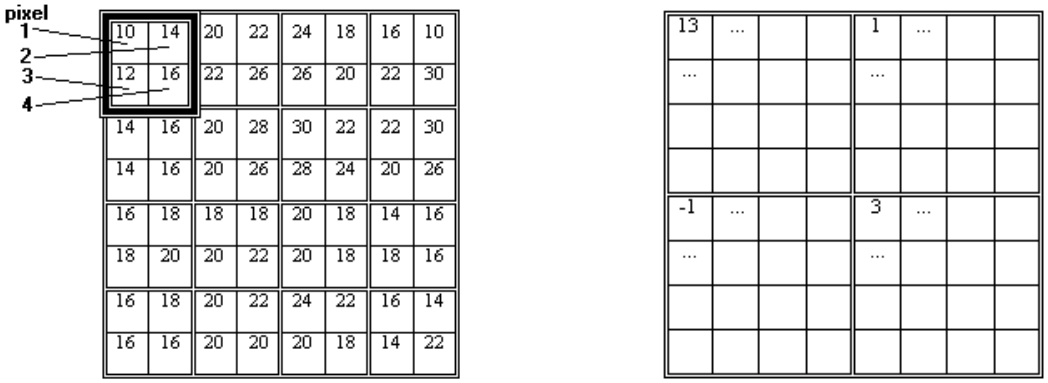
Exemplet ovan är baserat på en 1-dimensionell bild. Följande exempel visar en wavelet-transform av en 2-dimensionell bild. Principen är densamma, som för det tidigare givna 1-dimensionella Haar-exemplet. Givet är en symmetrisk bild med storleken $2^m*2^m$ pixlar. Om bilden inte är kvadratisk i sin storlek, är det möjligt att förstora och skala den genom att addera nollor för de områden av bilden, som måste utökas. Hädanefter antas bilden vara kvadratisk. Bilden delas in i plattor om fyra pixlar enligt figur 3.1 A. Då bilden är $2^m*2^m$ stor, kommer det att existera $2^{2m-2}$ plattor. Följande formler används för att beräkna ett medelvärde *s* från de fyra pixelvärdena och tre detaljkoefficienterna $d_n$:

$$s = \frac{pixel(1) + pixel(2) + pixel(3) + pixel(4)}{4}$$ , vilket implicerar

$$s = \frac{1}{4}\sum_{n=1}^{4} pixel(n) \qquad (5)$$

$$d_n = pixel(n) - s \text{ , för pixel } n=2, 3, 4 \qquad (6)$$

I varje platta finns fyra pixlar (se figur 3.1 A). Från dessa fyra värden beräknas medelvärdet 13 och detaljkoefficienterna 1, -1 och 3 fram från formel (5) respektive (6). De beräknade värdena fördelas sedan i mindre kvadrater, som kallas subband (att jämföra med filterbanker) enligt figur 3.1 B.




**A.** En platta om 4 pixlar.

**B.** Resultatet från formel (5) och (6) fördelade på fyra subband.

Figur 3.1. Wavelettransformering av en bild (bild i gråskala).

Ovanstående process upprepas sedan för varje platta. Medelvärdena och detaljkoefficienterna fördelas (enligt figur 3.1 B) tills det att subbanden är fyllda. Det vänstra subbandet högst upp kommer sedan att hanteras på samma sätt som originalbilden, nämligen att bli uppdelad i plattor om fyra pixlar för att sedan kunna generera de fyra subbanden. Subbanden kommer då att vara 1/16 av originalbildens storlek. De är baserade på de approximerade pixelvärdena. Tre subband kommer alltså att vara detaljkoefficienter, och den fjärde kommer att innehålla medelvärdena. Denna process kommer att fortgå tills en given upplösningsnivå är nådd, eller tills endast en pixel återstår. Den pixeln är då en approximation för alla pixlar i originalbilden. Denna transformering är reversibel.

Figur 3.2 illustrerar när wavelet-transformen applicerats på varje platta om fyra pixlar i två bilder. Pixlarna i varje subband beräknas på motsvarande sätt som ekvation (5) och (6). Någon dataförlust äger inte rum, och informationsmängden ökar inte. Upplösningsnivån i figur 3.2 är satt till nivå tre och därav en synlig miniatyrbild i övre vänstra hörnet.

Detaljkoefficienterna som antar värdet noll representeras genom de vita regioner som syns i subbanden.

Figur 3.2. Upplösningsnivå 3, koefficienter beräknade på block om fyra pixlar.

Resonemanget ovan (se figur 3.1 och 3.2) grundas på att bilden är en bild i grå färgskala. Wavelettransformer går dock lika bra att applicera på färgbilder som har flera komponenter för att representera en pixel [31]. Wavelettransformen appliceras då på varje enskild färgkomponent exempelvis $YC_bC_r$ [30]. Det innebär att kodaren får tre bilduppdelningar till skillnad från gråskaliga bilder, som ofta bara har en.

Med den 2-dimensionella waveleten kan en progressiv överföringsteknik appliceras. Användaren får till en början endast en lågt upplöst version av bilden, miniatyrbilden. Om användaren vill ha mer bildinformationen kan fler detaljkoefficienter från subbanden sändas över och bilden kan avkodas med en högre upplösning. Progression med avseende på kvalitet är likt den 1-dimensionella Haar-wavleten: istället för att överföra **[13 1 –1 3 …]** från figur 3.1 B kan **[13 0 0 0 …]** överföras till mottagaren. Denna information motsvaras då av en bild med lägre kvalitet än originalet. Se vidare i bilaga A.2 om progressiv bildpresentation.

För att få en visuell upplevelse, och eventuellt en ökad förståelse för ovanstående resonemang, rekommenderas följande två Internetlänkar [18, 34].

## 3.2 Kvantisering

Efter det att transformen har genererat koefficienterna, appliceras en kvantiserare [30]. Den har till uppgift att reducera koefficienterna i precision [33]. Vidare kan kvantiseringen approximera ett givet antal koefficienter med avseende på den bit-per-pixel (bpp) nivå som ska gälla [24]. Därmed kan en högre kompressionskvot (se bilaga B) uppnås. Processen utgörs av division av koefficienterna och sedan avrundning nedåt.

$$kvantiserad\text{-}koefficient(x,y) = \left\lfloor \frac{koefficient(x,y)}{kvantiseringssteg} \right\rfloor \quad (7)$$

Genom formel (7) introducerar kvantiseringen dataförluster. Kvantiseringen kan ske i olika steg beroende på vilken kvalitet och kompression bilden ska anta. Kvantiseraren appliceras en gång på varje subband, och olika subband kan kvantiseras i olika steg [24]. Stegen utgör storleken på talet som varje koefficient ska divideras med. Ett subbands koefficienter kvantiseras dock alltid lika mycket [2]. När kvantiseringssteget är 1 innebär det att ingen kvantisering äger rum, vilket i sin tur innebär, att inga förluster äger rum. Steget används vid icke förlustkomprimering. Icke förlustkomprimering förutsätter att koefficienterna är producerade med en wavelettransform som genererar heltalskoefficienter, och inte reella tal (närmevärden) [30]. Genom kvantiseringen kan antalet bitar, som ska representera varje enskild pixel i bilden, bestämmas [24]. Det innebär att en given bpp-nivå kan anges som argument till kodaren (exempel på denna typ av kodare är [16]). Kvantiseringsstegen kan alltså justeras med avseende på önskad kvalitet. Kvantisering sker i omvänd ordning när bildinformationen avkodas. Koefficienterna återskapas, och om de kvantiserades med förluster, kommer de att anta närmevärden.

## 3.3 Runlength-kodning

Wavelettransformen är en teknik som används i JPEG2000 för att öka komprimeringskvoten indirekt genom att den kan producera långa sekvenser med lika värden från originaldatan. För att uppnå markant större kompressionskvoter än vad wavelettransformeringen och kvantifieringen kan åstadkomma, krävs ytterligare två komprimeringstekniker varav den första kallas runlength-kodning.

När transformerad data innehåller klustrade sekvenser med lika värden, kan runlength-kodningen appliceras innan den aritmetiska kodningen (se delkapitel 3.4) för att få upp kompressionskvoten ytterligare. Tekniken låter en symbol representera hela den klustrade datan. Exempelvis kan en sekvens med tusen nollvärden bytas mot en symbol som representerar 1000, vilket tar mycket mindre plats:

**00000000 . . . 00000000 => 1000**

Anledning till att denna teknik appliceras före den efterföljande aritmetisk kodningen är att den sistnämnda i sig inte kan komprimera mer än 2:1 eller 3:1 [33] på naturliga bilder.

## 3.4 Komprimering med aritmetisk kodning

Aritmetisk kodning sker efter det att wavelettransformen, kvantifieringen och runlength-kodningen ägt rum. Den har till uppgift att komprimera informationen. Kompression uppnås genom att koda symboler, som förekommer med stor sannolikhet med ett mindre antal bitar än de symboler, som förekommer med liten sannolikhet. Den aritmetiska kodningsmetoden bygger på att meddelandet som ska komprimeras först representeras genom intervallet [0, 1] av reella tal. I följande exempel är det givet ett meddelande med ett alfabet {*a, b, c, !*} tillsammans med följande sannolikhetstabell, tabell 3.3:

| **Symbol** | **Sannolikhet** | **Spann** |
|---|---|---|
| *a* | 0.4 | [0, 0.4) |
| *b* | 0.3 | [0.4, 0.7) |
| *c* | 0.2 | [0.7, 0.9) |
| *!* | 0.1 | [0.9, 1] |

Tabell 3.3. Sannolikhetstabell.

Sannolikhetstabellen beräknas från den totala mängd av symboler som ska komprimeras. Det aktuella meddelandet, som ska komprimeras, behöver dock inte ha exakt denna sannolikhetsfördelning på symbolerna. Om det råder en annan fördelning uppnås dock inte samma effektiva kompression.

Två ekvationer, formel (8) och (9), används vid komprimeringen av meddelandet. Varje ny beräkning ger ett nytt mindre intervall [21]. Det slutliga intervallet representerar det komprimerade meddelandet.

*ny-lägsta-gräns = föregående-lägsta-gräns + symbolens-lägsta-gräns * föregående-storlek* (8)

*ny-storlek = föregående-storlek * symbolens-storlek* (9)

Från ekvation (8) kan intervallets nya lägsta gräns beräknas och från (9) spannet på intervallet. Ju mindre ett spann är för en symbol som ska kodas (se tabell 3.3), desto mindre kommer det slutliga intervallet bli. Det är därför viktigt att symboler som förekommer mer

frekvent än andra, representeras med ett så stort spann som möjligt i sannolikhetstabellen, då spannet ej kommer påverka det tidigare beräknade intervallet lika mycket.

För att illustrera med ett exempelmeddelande, tillsammans med ovanstående sannolikhetstabell, sänds meddelandet *abca!*. Först kodas *a* i kodaren, då den symbolen är den första i meddelandet. Det kodade intervallet minskas nu från [0, 1] till [0.0, 0.4] enligt (7, 8). Efter symbolen *b* blir intervallet mindre, [0.16, 0.28], och det med avseende på *b*'s sannolikhet på 0.3. När alla symboler har passerat kodaren har intervallgränser enligt tabell 3.4 växt fram.

| **Sekvens** | **Intervall** |
|---|---|
| Initiellt | [0, 1] |
| *a* | [0, 0.4) |
| *b* | [0.16, 0.28) |
| *c* | [0.244, 0.268) |
| *a* | [0.244, 0.2536) |
| *!* | [0.25264, 0.2536) |

Tabell 3.4. Intervallgränser.

Antag nu att avkodaren mottager det slutliga intervallet [0.25264, 0.2536] som kodaren beräknade, och sannolikhetstabellen ovan. Avkodaren kan direkt från intervallet fastslå att den första symbolen är *a*, eftersom intervallet ligger helt och hållet inom spannet för symbolen *a* enligt tabell 3.3. Avkodaren utgår nu från *a´*s intervallet [0, 0.4). Avkodaren räknar sedan ut den nya fördelningen för symbolerna i detta intervall: [0, 0.16) för *a*, [0.16, 0.28) för *b*, [0.28, 0.36) för *c* och [0.36, 0.4) för *!*. Från detta ges, att det mottagna intervallet ligger innanför spannet för symbol *b*, vilket är den andra symbolen i meddelandet. Avkodaren måste alltså räkna fram intervall i samma ordning som kodaren. Denna process fortskrider till dess att hela meddelandet *abca!* är avkodat.

Storleken på det slutliga intervallet, som kodaren producerar, bestämmer antalet bitar, som krävs för att representera det komprimerade meddelandet. Ju större spann intervallet har, som kodaren beräknar, desto mindre plats kommer det också att ta, då det kräver mindre antal bitar. Avkodaren kan utan problem avkoda det komprimerade meddelandet, även om inte hela intervallet [0.25264, 0.2536] skickas till den. För avkodarens del räcker det med, att ett tal inom detta intervall sänds över, exempelvis 0.2530. Samma sannolikhetstabell kan användas för flera meddelanden. Tabellen bör dock ominitieras, då nya meddelanden ska skickas, då de inte alls behöver ha samma sannolikhetsdistribution som tabellen.

# 4 Kodning med JPEG2000

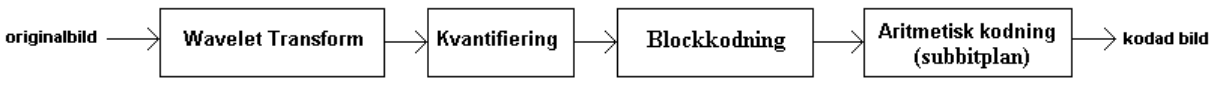
Följande kapitel abstraherar JPEG2000-standardens beskrivning av hur en bild ska kodas. Kodningen innebär komprimering av bildens pixelvärden, men även hur datan ska representeras och lagras så att avkodaren kan tolka bitströmmen på ett korrekt sätt. Kapitlet avslutas med en sammanfattning av hur denna kodning utförs.

## 4.1 Introduktion

I JPEG2000 grundutförande (del I) används två wavelettransformer [2]. De är mer utvecklade och anpassade för bildkodning än den tidigare presenterade Haar-waveleten. De är även optimerade med avseende på ögats förmåga att se detaljer i en bild. Transformerna kan även transformera fler detaljkoefficienter till värdet noll. Det leder i sin tur till att en högre kompression kan uppnås, då dessa värden trunkeras. De detaljkoefficienter som är nära noll kan approximeras genom kvantiseringen. Vilken av de två wavelettransformerna som används beror på om förlustkomprimering ska utföras eller inte. Vid förlustkomprimering används en flyttalstransform, och vid icke förlustkomprimering används en heltalstransform. Någon närmare redogörelse för hur dessa fungerar kommer ej att ges i detta arbete (se [2] för mer information). Principen är dock den som redogjorts för i kapitel 3. Resultatet blir en uppdelning av bilden i fyra subband. Det så kallade lågpassbandet kommer att innehålla den mest relevanta informationen i bilden. De resterande tre subbanden på samma upplösningsnivå, högpassbanden, innehåller information om detaljerna i bilden, vilket visuellt inte uppfattas lika lätt.

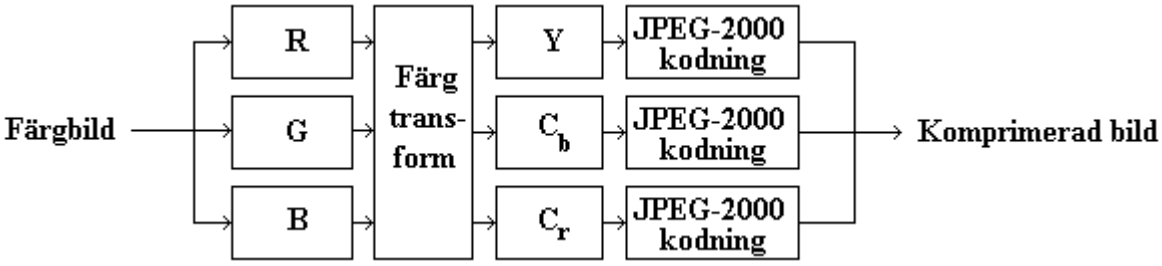
## 4.2 Kodomgång och komponenttransform

Figur 4.1 visar huvudstegen i JPEG2000-kodaren. Stegen i en JPEG2000-kodomgång är likt andra waveletbaserade bildkodare. Skillnaden är steget ”blockkodning” som förklaras senare.




Figur 4.1. Illustration av en kodomgång.

Det första som händer med en färgbild (till exempel i RGB-format) är att den kodas om till formatet $YC_bC_r$ (se figur 4.2, se bilaga A.1 för mer information angående representation av en bild genom olika format.) Detta sker innan wavelettransformen appliceras [2].




Figur 4.2. Färguppdelning av en flerkomponentbild.

Ögat är mer känsligt för olika ljusstyrkor (Y) än färgskiftningar ($C_bC_r$) [30, 31]. Således kan högre kompression uppnås, då de för ögat mindre känsliga komponenterna, $C_bC_r$, kan approximeras mer. Transformeringen mellan de olika färgformaten (RGB -> $YC_bC_r$) ger närmevärden, varför omkodning endast används vid förlustkomprimering. För att koda en bild i RGB förlustfritt krävs en annan färgtransformation, Reversible Component Transform (RCT), som transformerar färgkomponenterna mellan varandra med ändliga heltal som resultat.

## 4.3 Intern kodning

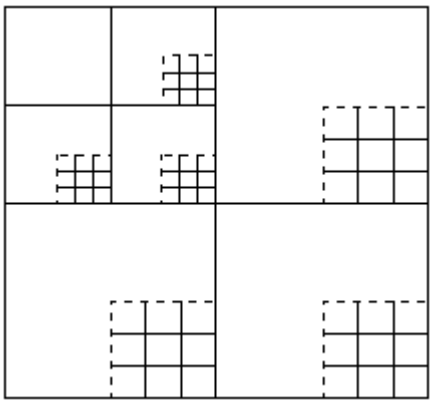
Efter det att wavelettransformen och kvantiseringen applicerats på bilden respektive koefficienterna, ordnas de genererade waveletkoefficienterna konceptuellt enligt figur 3.2. Figuren visar på 3 nivåer av upplösning medan JPEG2000 normalt har 5 [2]. Antalet nivåer går att variera från 0 till 32 nivåer. Då noll anges som antalet nivåer kommer ingen wavelettransformering att utföras.

### 4.3.1 Paketområden

Efter kvantiseringen delas varje subband in i så kallade paketområden (se figur 4.3). Anledningen till paketområdena är bland annat att få en minneseffektiv kodare. Paketområdena används senare till att ordna datan i filen.

### 4.3.2 Kodblock

Från paketområdena skapas sedan lika stora rektangulära block innehållande koefficienter (se figur 4.3). Blocken är till en början normalt 64 * 64 pixlar stora, men kan ändras vid behov.

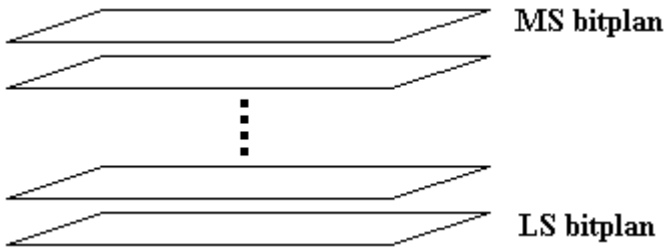


Figur 4.3. Paketområdena (streckade linjer) innehåller 9 block (heldragna linjer).

Varje sådant block fungerar sedan som indata till den aritmetiska kodaren. Denna kodning bestämmer sedan i vilken utsträckning varje blocks bitström ska trunkeras, för att på så sätt uppnå en given bpp-nivå. Denna process kallas EBCOT (Embedded Block Coding with Optimizied Truncation) [32] och förklaras i mer detalj i nästkommande delkapitel. Kvantiseringen kan som tidigare nämnts även den utföra trunkering, men den processen utförs inte med en exakt bpp-nivå som resultat.

### 4.3.3 Bitplan

Varje kodblock, som kan representeras av en array med kvantiserade koefficienter, delas först upp i bitplan där den mest signifikanta biten (MSB) från varje koefficient, placeras i det mest signifikanta bitplanet. Den näst MSB placeras i det nästkommande bitplanet (se figur 4.4). Processen fortskrider tills alla bitar är fördelade i bitplanen.



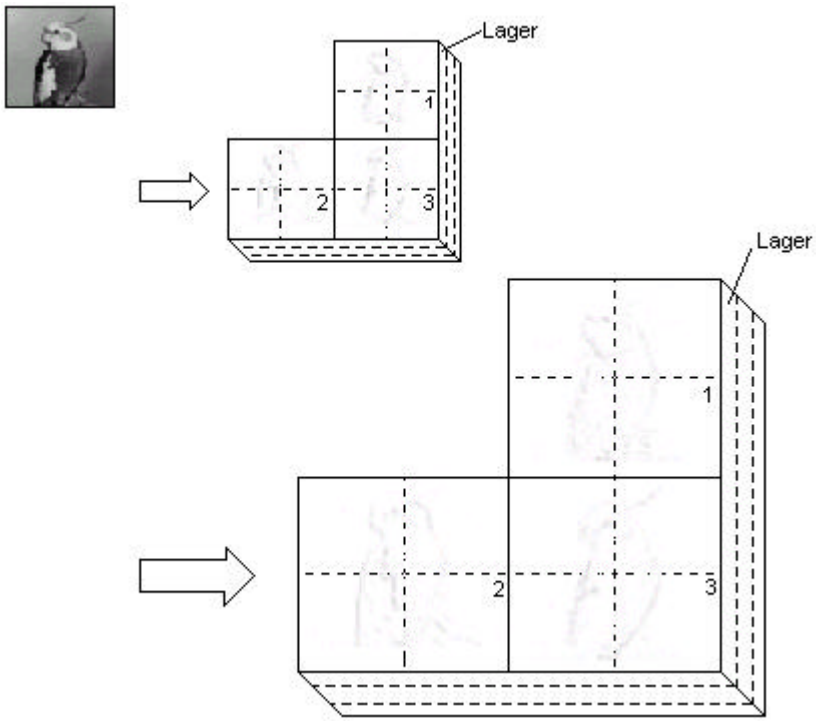

Figur 4.4. Kvantiserade koefficienter uppdelade i bitplan.

För varje bitplan i ett kodblock appliceras en kodningsalgoritm tre gånger. En bit kodas bara en gång i någon av dessa tre omgångar. Kodningsalgoritmens uppgift är att tolka bitarna i bitplanet utifrån deras närliggande grannar (någon närmare redogörelse av denna kodning görs ej). Från de tre kodomgångarna på bitplanen lagras sedan denna informationen i tre

ordnade subbitplan. Dessa tre subbitplan är nu föremål för den aritmetiska kodaren, som används i JPEG2000. Om möjligt appliceras även run-length-kodning före den aritmetiska kodaren för att på så sätt krympa informationsmängden ytterligare. Ett komprimeringspass innebär att subbitplanen komprimeras ett i taget. Den aritmetiska kodaren arbetar alltså med små datamängder, och efter varje komprimeringspass kan sannolikhetstabellen nollställas och kodaren kan termineras med en termineringsmarkör (se kapitel 5). För en mer ingående beskrivning av bitplanskodningen se [2].

### 4.3.4 Lager

Den kodade datan i varje kodblock distribueras i ett eller flera lager i kodströmmen. Varje lager innehåller ett antal på varandra följande bitplan från alla kodblock, från alla subbanden på samma upplösningsnivå (se figur 4.5) [2]. Varje lager tillför bilden mer information och därmed bättre kvalitet för varje nytt lager, som når avkodaren (se bilaga A.2).



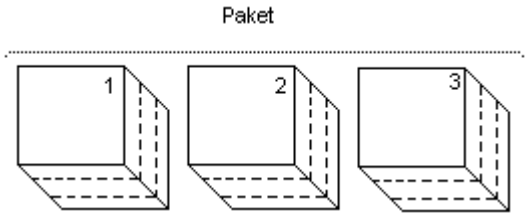

Figur 4.5. Subbanden delas i lager.

### 4.3.5 Paket

Tre paketområden från olika subband, men på samma upplösningsnivå och från samma spatiala område, bildar ett paket (se figur 4.5, paketområdena angivna 1, 2 och 3 bildar ett paket). Paketen innehåller nu noll eller flera kodblock (se figur 4.3), eftersom en del kodblock

inte innehåller någon data. Det beror på att kodblocken inte innehåller några subbitplan överhuvudtaget, vilket i sin tur beror på att waveletkodaren genererat koefficienter som alla antagit värdet noll. Dessa koefficienter har sedan trunkerats.

Varje paket innehåller data från tre paketområden (se figur 4.6). Paketet föregås av en header. Den talar bland annat om hur mycket data som finns och längden på kodblocken i paketet. Information om vilket område i bilden datan tillhör finns också.

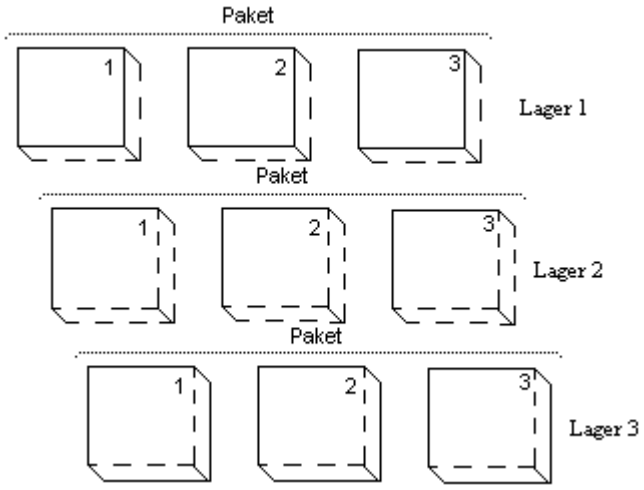



Figur 4.6. Tre paketområden bildar paket.

Ett pakets information kan också fördelas olika i den kodade bitströmmen. Var någonstans ett paket eller delar av ett paket hamnar, beror på vilket progressivt läge, som ska stödjas (se bilaga A.2). Ett paketområde i motsatts till figur 4.3, kan även täcka ett helt subband. Då uppnås den största kompressionskvoten, eftersom antalet paketheadrar minimeras. Minneskravet för en hård- eller mjukvaruimplementation minskar dock då fler paketområden används. Vidare tillåts applikationer att godtyckligt få tillgång till bitströmmen på fler ställen, då denna typ av uppdelning används [5].

Progressivitet med avseende på endast upplösning är då alla paketen innehåller alla lager och därmed alla bitplan från givna paketområden (se figur 4.6). Bitströmmen måste även ordnas så att subband som ligger på lägre upplösningsnivåer kommer före subband på högre nivåer.

Ska bilden däremot visas med kvalitetprogressivitet, kan inte längre ett paket innehålla all information om ett specifikt paketområde. Paketen delas nu in i flera mindre paket (se figur 4.7).




Figur 4.7. Paket delas in i mindre paket.

Dessa mindre paket innehåller då bara ett lager (ett fåtal bitplan) med information. Skillnaden för det kvalitetsprogressiva läget är att här kan inte varje kodblock innehålla alla bitplanen. De mindre paketen innehåller samma kodblock som tidigare, fast kodblocken innehåller nu endast en delmängd av bitplanen. För att få en bättre bildkvalitet måste avkodaren ha tillgång till ytterligare ett mindre paket. Dessa paket innehåller då information om resterande lager. Denna typ av omorganisation av paket till mindre paket kräver då fler paketheadrar enligt [24] – från en paketheader till exempelvis fyra paketheadrar.

En omkodare kan tämligen enkelt tolka bitströmmen med avseende endast på paketheadrarna. Ingen aritmetisk avkodning behöver utföras utan endast omorganisering av paketen i bitströmmen. Omkodaren kan då förse mottagare med olika slag av bildvisualiseringar.

## 4.4 Sammanfattning

Följande steg används för att koda en originalbild till en JPEG2000-bild:

- Bilden delas upp i färgkomponenter.
- Varje komponent wavelettransformeras och härmed skapas upplösningsnivåer. Upplösningsnivåerna representerar olika upplösningar för bilden.
- Varje nivå består av subband av koefficienter.
- Koefficienterna i subbanden kvantiseras och delas upp i rektangulära kodblock.
- Bitplanen från koefficienterna i ett kodblock kodas med en aritmetisk kodare.

Nu är bilden kodad. Återstående steg syftar till att ordna bitströmmen:

- De komprimerade bitplanen från kodblocken samlas i lager.
- Paketområden representerar ett antal kodblock från samma spatiala område och upplösningsnivå.
- Paket bildas från olika paketområden. Varje paket innehåller ett antal lager.
- Paketen kan ordnas beroende på hur bilden ska presenteras (exempelvis upplösningsprogression).
- Hela bitströmmen startar med en header som beskriver bilden och hur den är kodad för att möjliggöra vidare avkodning av bilden.

# 5 Robust bildkodning i JPEG2000

I detta kapitel beskrivs de mekanismer som används i JPEG2000 för att uppnå en hög robusthet i bitströmmen. I nästa kapitel kommer dessa robusthetsmekanismer att testas och utvärderas.

## 5.1 Introduktion

Ericsson, Nokia, Motorola, och Samsung har deltagit i utvecklingen av den nya bildkodningsstandarden. Dessa företag utvecklar produkter, som används i trådlösa nätverk. Tyvärr introduceras ofta fel i den data, som ska transporteras över dessa nätverk. De feldistributioner som förekommer har avspeglat vilka robusthetsmekanismer JPEG2000-kodaren använder sig av för att kunna avkoda en bild. Robustheten avser här avkodarens förmåga att upptäcka och hantera fel i en bitström.

## 5.2 Feldistributioner

Databussen mellan moderkortet och en hårddisk i en dator, eller en radiokanal mellan en sändare och en mottagare är exempel på överföringskanaler.

Oberoende av vilken kanal som används, kan tyvärr olika typer av fel uppstå i bitströmmen. Felen kan exempelvis vara bitfel, där en bit ändras från att vara '1' till att anta '0' (eller tvärtom).

En allvarligare feltyp är skurfel, som introducerar långa sekvenser av bitfel i bitströmmen. Skurfel påträffas ofta i trådlösa överföringar och satellitförbindelser. Felet uppkommer från fenomenet fädning (se kapitel 6).

Totala antalet bitfel, i form av skurar eller inte, som kan inträffa över en kanal uttrycks genom ett Bit Error Rate-värde (BER, se bilaga B.2).

Det kanske allvarligaste fel som kan inträffa är att data försvinner helt och hållet. Detta fel inträffar då paket av information förloras (där paket avser den datauppdelning som underliggande kommunikationsprotokoll gör). Detta typ av fel kallas paketförluster. Internet är ett medium som kan förlora hela datapaket. Dock finns mekanismer för att komma till rätta med dessa fel, exempelvis omsändning av det förlorade paketet.

## 5.3 Robusthetsmekanismerna

Vid komprimering ökar beroendet mellan dataelementen. Om ett eller flera element drabbas av fel, kan det leda till att en bild inte längre kan avkodas korrekt. En robust överföring av komprimerad data, över opålitliga kommunikationskanaler, är därför ett viktigt applikationskrav. Robusthetsmekanismerna som presenteras nedan är till för att underlätta för avkodaren, som är en del av applikationen, att avkoda en felaktig bitström.

Det går att dela upp begreppet robusthet i tre nivåer. Den första nivån är att upptäcka fel, den andra är att dölja fel och den tredje att rätta fel. JPEG2000 arbetar framförallt på den första nivån, alltså att upptäcka fel, men har även en mekanism för att dölja bitfel [2]. JPEG2000 har som mål att klara en robust avkodning i exempelvis trådlösa dataöverföringar då bit- och skurfel kan inträffa. Senare tester (utförda i kapitel 6) kommer även att visa att paketförluster kan tillåtas inträffa, men på en stor bekostnad av den visuella kvaliteten.

Robusthetsmekanismerna realiseras på paket-, block- och bitplansnivå i den befintliga datauppdelning [2]. Robusthetsmekanismerna kan alltså befinna sig på olika nivåer i JPEG2000-standardens naturliga bitström. Följande underkapitel är en redovisning av dessa mekanismer.

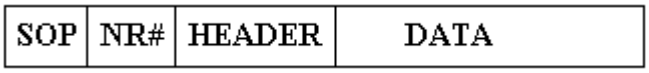
### 5.3.1 Paketnivå

Datan i en JPEG2000-kodad bild är hierarkiskt ordnad på grund av den wavelettransform och de kodningstekniker som används (se kapitel 4). Bildens pixlar kodas till waveletkoefficienter, som sedan lagras i subband. Varje subband är i sin tur uppdelade i paketområden. Ett paketområde innehåller ett antal kodblock, och kodblocken består av ett antal lager med bitplan. Bitplanen kodas med den aritmetiska kodaren. Tre paketområden från olika subband, men från samma spatiala område, bildar slutligen ett paket.

Den enklaste formen av paketuppdelning sker då paketområdet innefattar hela subbandet. Det implicerar att ett paket innehåller all information om den aktuella upplösningsnivån (sånär som på kvaliteten). Varje paket föregås av en header, följt av den kodade bildinformationen. Paketheadern innehåller information om bland annat kodblockens antal, och antalet bitplan i varje block.

#### 5.3.1.1 Återsynkronisering

Då bitströmmen ska klara av att upptäcka ett antal fel, räcker inte paketinformationen för en robust bildavkodning. Istället kan en återsynkroniseringsmarkör, Start Of Packet header (SOP), och ett sekvensnummer tillföras framför varje paketheader (se figur 5.1) [2].



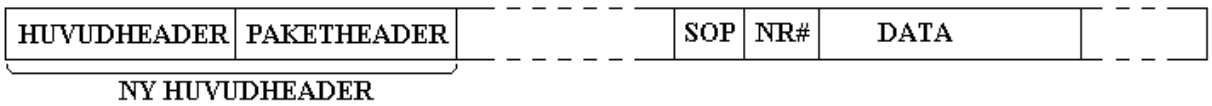

Figur 5.1. SOP-markör och sekvensnummer.

SOP-markören är en reserverad bitsekvens i standarden. Avkodaren måste unikt kunna avläsa varje återsynkroniseringsmarkör från bitströmmen, varför kodaren utför bitstuffing i dataströmmen [2]. Avkodaren använder SOP-markörerna för att kontinuerligt kunna synkronisera med bitströmmen, och för att kunna avgöra vilka koefficienter som är drabbade av fel. Om sekvensnumret inte är synkroniserat med det avkodaren förväntar sig att få, genereras ett fel. Denna mekanism klarar alltså bara av att upptäcka fel i bitströmmen. Sekvensnumren användas för att ange var i bitströmmen paketen hör hemma. Numren startar från noll, och ökas med ett för varje nytt paket.

Om synkroniseringen förloras efter fel genom att nästkommande sekvensnummer inte är det förväntade, kan avkodaren anta strategin att leta efter nästa markör med tillhörande sekvensnummer i bitströmmen [22]. Feldöljning kan därefter appliceras genom att initiera detaljkoefficienterna till noll, för att på så sätt dölja den felaktiga data som är placerad mellan de påträffade och korrekta markörerna. En annan strategi är att avsluta avkodningen om nästkommande markör ej kan avkodas korrekt (strategin används i [16]).

#### 5.3.1.2 Korta paket

Paketen i bitströmmen kan göras kortare [2]. Korta paket tillhandahålls genom att flytta paketheadern till huvudheadern (se figur 5.2).




Figur 5.2. Paketheader flyttas till huvudheadern.

Huvudheadern är den header som lagrar information om exempelvis bildens storlek, vilken typ av progressivt läge som har använts, vilken wavelettransform som transformerat pixlarna, och vilket kvantifieringssteg som reducerat koefficienterna. Drabbas huvudheadern av fel, och den inte är skyddad, kommer bilden i alla fall inte att kunna avkodas. Skydd av headrar är inte standardiserat i JPEG2000-standarden, men skulle kunna utföras genom redundant data. Ett alternativ är att sända huvudheadern tillsammans med paketheadrarna separat över en annan

kanal som erbjuder en bättre service [25]. Flyttas paketheadrarna till huvudheadern kan fortfarande rätt paket associeras till rätt data i kodströmmen. Kopplingen mellan paketheadern och paketets data, alltså de komprimerade koefficienterna, sker med hjälp av sekvensnumren. Sekvensnumren införs samtidigt som ovan nämnda återsynkroniseringsmarkör [2].

### 5.3.2 Robusthet genom den aritmetisk kodningen

Den aritmetiska kodaren genererar komprimerad data av varierande längd. Om ett fel inträffar i en variabel längdkodare förloras synkroniseringen med bitströmmen [22]. För att undvika detta kan ytterligare robusthetsmekanismer införas under komprimeringssteget.

#### 5.3.2.1 Blocknivå

Komprimeringen av de kvantifierade koefficienterna utförs inom varje kodblock. Eftersom kodning respektive avkodning av ett kodblock är oberoende av andra block kommer bitfel i ett kodblock isoleras, och därmed kommer endast det aktuella kodblocket att drabbas och inte närliggande kodblock. Denna robusthetsmekanism, som är på blocknivå, används för att isolera fel i ett kodblock. Mekanismen följer naturligt från den tidigare nämnda datauppdelningen.

#### 5.3.2.2 Terminering av kodaren

Den aritmetiska kodaren kan termineras antingen i slutet av varje kodomgång i ett bitplan, eller i slutet för varje kodblock. Detta tillåter avkodaren att fortsätta att avkoda även efter det att ett fel har inträffat. Om kodaren termineras efter varje ny kodomgång, kommer ett bättre felskydd att existera. Termineras kodaren endast efter varje kodblock, kommer bitfel i ett bitplan att drabba avkodningen även för nästkommande bitplan.

#### 5.3.2.3 Selektiv komprimering

För att minska beroendet mellan de olika dataelementen i bitströmmen, kan kodaren välja att inte applicera den aritmetiska kodaren överhuvudtaget på utvalda delar av bitplanen [2]. Fel som inträffar på den råa datamängden, kommer alltså inte att beröra den annars sårbara aritmetiska avkodaren. Däremot kommer koefficienterna, som är drabbade av bitfel, bli wavelettransformerade till felaktiga pixelvärden.

#### 5.3.2.4 Segmentsymbol på bitplansnivå

I slutet av ett bitplan kan en segmentsymbol adderas. Symbolen kodas tillsammans med koefficienterna med den aritmetiska kodaren. En korrekt avkodning av denna symbol signalerar till avkodaren att bitplanet avkodades korrekt. Avkodaren använder alltså symbolen

för att kunna upptäcka fel. Om segmentsymbolen inte avkodas korrekt, innebär det att ett fel har inträffat på det aktuella bitplanet. Avkodaren blir då tvungen att hoppa över avkodningen av det aktuella bitplanet, och eventuellt gå vidare till nästa bitplan eller nästa kodblock.

När bitströmmen drabbas av många bitfel, kommer många kodblocks bitplan att påverkas. Om ett kodblock, som innehåller de mest signifikanta bitplanen, träffas av bitfel, kommer resterande bitplan vara oanvändbara, eftersom de endast innehåller förfiningar av de högre planen [29].

### 5.3.3 Dölja bitplansfel

Med hjälp av segmentsymbolen, som upptäcker fel, kan en feldöljningsalgoritm appliceras. Avkodaren kontrollerar först att segmentsymbolen är närvarande i slutet för ett bitplan. Om symbolen inte är närvarande, innebär det att bitströmmen innehåller någon felaktig bit, som har föranlett en inkorrekt avkodning av datan och symbolen. Datan avser här hela det aktuella bitplanet. Funktionaliteten som handhar feldöljningen appliceras sedan.

Hur avkodaren döljer felen är inte specificerat i standarden. Det är upp till dem som implementerar en JPEG2000-kodare att ta ställning till [2]. Efter det att avkodaren dolt felen i datan, bör dock inte fler bitplan avkodas för det aktuella kodblocket. Den enklaste formen av feldöljning, är att sätta de felaktiga koefficienterna till noll [22]. Förfarandet kan dock ge negativa effekter på den visuella bildupplevelsen.

Om den högre nivån av robusthet skulle gälla i standarden, som innebär rättning av fel, måste dock hela den felaktiga datan återskapas korrekt. Det kan realiseras med så kallad redundanskodning, vilket innebär att extra information tillförs originalbitströmmen. Rättade fel påverkar inte bildkvaliteten. Denna robusthetsnivå förklaras inte närmare i detta arbete.

# 6 Test och analys av robusthetsmekanismerna

Syftet med detta kapitel är att visa hur väl JPEG2000-standardens robusthetsmekanismer skyddar mot fel, och vilka typer av feldistributioner som en kodare kan förväntas klara. Tre olika testkonfigurationer har använts, en utan och två med olika uppsättningar mekanismer. Testning har skett med bitfel, skurfel och paketförluster. Testresultaten av de olika testfallen presenteras utifrån lyckade avkodningar och beräknade PSNR-värden. Paketförluster presenteras även utifrån progressionsval.

## 6.1 Generell testmiljö

Testerna har genomförts på en maskin med Windows NT 4.0® installerat. Genom ett script har en procedur genomförts för varje bild, som testas med antingen bitfel, skurfel eller paketförluster. Bilderna kodas enligt denna procedur med olika robusthetskonfigurationer (testfall, se nästkommande delkapitel) på JPEG2000-kodaren beroende på vilka mekanismer som ska testas. För varje testfall har ett antal bilder kodats: 100 stycken bilder som drabbas av bitfel och 80 bilder som drabbas av skurfel (för testning av paketförluster se kapitel 6.6). Proceduren är enligt följande (proceduren för paketförluster sträcker sig endast till och med steg 4):

1. Bilden kodas till JPEG2000-formatet med en given konfiguration på robusthets-mekanismerna.
2. I den kodade bilden introduceras fel i någon förutbestämd form (bitfel, skurfel eller paketförluster).
3. Bilden som nu innehåller fel avkodas med JPEG2000-avkodaren.
4. PSNR-värde beräknas på bilden utifrån originalbilden.
5. Proceduren upprepas från steg 1 (100 upprepningar för bitfel och 80 upprepningar för skurfel).

## 6.2 Konfiguration av kodaren

En JAVA™-implementation av JPEG2000-kodaren, JJ2000, har genomgående använts för nedanstående testningar (implementationen återfinns vid [16]). Två anmärkningar gäller för denna implementation. Den första är att den avslutar avkodningen då SOP-markören inte kan avkodas korrekt. Den andra anmärkningen är att avkodaren avbryter avkodningen då avkodaren förlorar synkroniseringen med sekvensnumret (se delkapitel 5.3.1.1).

Följande fyra konfigurationer har använts för att koda bilderna:

- Kodning utan mekanismer. (Benämns hädanefter som testfall 1.)
    - *Ingen modifiering av dataströmmen har ägt rum.*
- Kodning med mekanismer inklusive paketflyttning och återsynkroniseringsmarkörer. (Benämns hädanefter som testfall 2.)
    - *Paketflyttning till huvudheadern.*
    - *SOP-markörer inlagt i bitströmmen.*
    - *Segmentsymbol har införts efter aritmetiska kodaren.*
    - *Sannolikhetstabellen har ominitieras efter varje kodpass.*
    - *Aritmetiska kodaren termineras och skapar nytt kodord efter varje kodpass.*
    - *Aritmetiska kodaren tillåts hoppa över kodning av mindre signifikanta bitplan.*
- Kodning med mekanismer men utan paketflyttning och återsynkroniseringsmarkörer. (Följande konfiguration benämns hädanefter som testfall 3.)
    - *Segmentsymbol har införts efter aritmetiska kodaren.*
    - *Sannolikhetstabellen har ominitieras efter varje kodpass.*
    - *Aritmetiska kodaren termineras och skapar nytt kodord efter varje kodpass.*
    - *Aritmetiska kodaren tillåts hoppa över kodning av mindre signifikanta bitplan.*
- Kodning med endast paketflyttning och återsynkroniseringsmarkörer. (Benämns hädanefter som testfall 4.)
    - *Paketflyttning till huvudheadern.*
    - *SOP-markörer inlagt i bitströmmen.*

Testfall 1-3 används vid testningen av bitfel och skurfel. Testfall 4 används endast vid test med dataförluster. Vid simuleringen kan all data träffas av fel och mekanismen innebärande flyttning av paketheadrar har därför inte någon effekt. Detta är anledningen till varför kombinationen med att flytta paketheadrarna tillsammans med samtliga resterande mekanismer inte finns med som ytterligare ett testfall.

Bilden som använts vid testningen har genomgående kodats med 2 bpp (originalbilden på 8 bpp återfinns i bilaga D, bild 1). Det finns ingen visuell skillnad mellan de fyra olika kodade bilderna då bitströmmen är felfri. En viss skillnad kan dock subjektivt upplevas mellan dessa och originalet som är kodat med 8 bpp. I tabell 6.1 presenteras kvaliteten för respektive kodad bild under felfria förhållanden. Presenterade PSNR-värde motsvarar kvaliteten mellan givet testfall jämfört med originalbilden på 8 bpp. Värdena är endast beräknade utifrån bilder med maximal upplösning.

| | **Testfall 1** | **Testfall 2** | **Testfall 3** | **Testfall 4** |
|---|---|---|---|---|
| **PSNR** | *44,3026* | *43,6856* | *43,6996* | *43,6996* |

Tabell 6.1. Testfallens kvalitet jämfört med originalbilden.

Filstorleken och kvaliteten, varierar beroende på vilka mekanismer som används enligt tabell 6.2.

| | **Testfall 1** | **Testfall 2** | **Testfall 3** | **Testfall 4** |
|---|---|---|---|---|
| **Storlek i bytes** | *16 387* | *16 403* | *16 390* | *16160* |
| **Bpp-nivå** | *2,0004* | *2,0023* | *2,0007* | *1,9727* |

Tabell 6.2. Testfallens filstorlek och bpp-nivå.

Önskvärt hade varit om testningen kunnat ske med bilder med flera färgkomponenter, exempelvis RGB. Den programvara som använts för att beräkna bildkvaliteten har dock inte klarat detta format. För testningen av själva mekanismerna har det dock ingen betydelse, då de beter sig på samma sätt oberoende av komponentantalet. Däremot skulle en reflektion vara önskvärd över hur en bild visuellt påverkas, då endast en komponent av flera träffas av fel.

Det förlustfria kompressionsalternativet är heller inte testat. Exempelvis medicinska bilder som inte kodas med förluster bör sändas om, om de transporteras över felintroducerande nätverk. Konsekvensen av att inte sända om, utan istället vid mottagarsidan upptäcka fel och dölja dem, kan få katastrofala konsekvenser.

## 6.3 Införda begrepp

Vid presentationen av hur avkodaren förmår att avkoda bilder, används följande tre begrepp (gäller endast vid presentation av bitfel och skurfel):

- **Felfri** - den avkodade bilden är återskapad utan några fel.
- **Ej felfri** - den avkodade bilden har sämre kvalitet jämfört med motsvarande felfria version. Kodaren har även rapporterat att bilden innehåller fel.
- **Krasch** - avkodaren rapporterade ett felmeddelande och lyckades inte avkoda bilden överhuvudtaget.

## 6.4 Tolkning av diagram

I presentationen av testresultaten för bitfel och skurfel används två olika diagram. Det första digrammet, där figur 6.1 kan ges som ett exempel, sammanfattar resultaten från ett testfall X och med en given feldistribution. Från detta digram kan det utläsas hur många av bilderna som blev felfritt återskapade, ej felfritt återskapade, och där avkodaren inte kunnat avkoda bilden överhuvudtaget. För varje feldistribution presenteras de tre testfallen med ett sådant diagram.

Kvalitetsvärdena som genereras från dessa tre testfall, sammanställs sedan i det andra diagrammet (för exempel se figur 6.11). Värdena som ligger till grund för detta digram är de värden som generas från steg 4 (se kapitel 6.1). I detta diagram kan skillnaden mellan de olika testfallen studeras när det gäller de avkodade bildernas kvalitet. För varje testfall ses en stapel som visar kvaliteten med hjälp av så kallade kvartilavstånd. Den undre och övre delen av en stapel består av 25 procent av de minsta värdena, respektive 25 procent av de största värdena. Den mittersta delen, som är formad som en smal rektangel, består av 50 procent av de mittersta värdena. Endast avkodade bilder från kategorin "ej felfria" finns med i detta diagram. De kraschade bilderna, som har PSNR-värdet 0, och de bilder som blev felfritt återskapade, som antar ett ändligt PSNR-värde (se tabell 6.1), finns inte representerade här.

Beskrivning av diagrammen från testningen av paketförluster behandlas separat (se kapitel 6.7.1).

## 6.5 Bitfel

Bitfel förekommer i nästan alla typer av nätverk. Oftast klarar underliggande protokoll av att reda ut denna typ av fel genom exempelvis omsändningar av datan. Men omsändningar är inte alltid önskvärda, då det kan ta för lång tid att slutföra denna åtgärd. Det ställer i sin tur krav på de kodare som den överförda datan är kodad med. Testningen av bitfel syftar till att se hur mekanismerna i JPEG2000 hanterar denna typ av fel.

### 6.5.1 Simulering av bitfel

Fyra olika BER-värden har simulerats: $10^{-3}$, $10^{-4}$, $10^{-5}$ och $10^{-6}$. De rapporterade resultaten är baserade på 100 körningar för varje nytt BER-värde. Olika slumpfrön har använts för varje ny bild som ska träffas av fel. Då de fel som simuleras inte tar hänsyn till var någonstans i bitströmmen de inträffar har testning av att inte använda paketflyttning, men med SOP-markörer förkastats. Detta fall anses vara likvärdigt med testfall 2, där båda mekanismerna används. Detta gäller även vid testning av mekanismerna vid skurfel.

Genomsnittliga antalet totala bitfel i en bildfil framgår från tabell 6.3. Värdena är teoretiskt beräknade, och vid simuleringarna varierar antalet bitfel.

| **BER** | **$10^{-3}$** | **$10^{-4}$** | **$10^{-5}$** | **$10^{-6}$** |
|---|---|---|---|---|
| **Antal bitfel** | 131,1 | 13,1 | 1,3 | 0,1 |

Tabell 6.3. Förväntat antal bitfel.

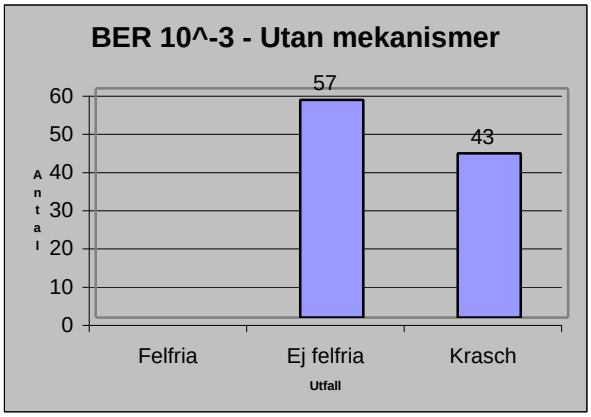
Simuleringarna testar de tre olika fallen av kodad bitström: kodning då inga robusthetsmekanismer används, kodning då alla robusthetsmekanismer används och slutligen kodning med alla robusthetsmekanismer utom mekanismerna paketflyttning och SOP-markörer. Syftet med dessa tre olika fall är att se hur avkodaren reagerar, i form av lyckade avkodningar. Anledningen till att dela upp mekanismerna med ett fall utan paketflyttning och SOP-markörer och ett fall med dem, är att bitfelen kan inträffa godtyckligt i bitströmmen. Vid denna feldistribution lämpar sig således inte paketflyttning. Anledningen är att paketflyttningsmekanismen endast är användbar då huvudheadern kan sändas över en kanal som inte är drabbad av fel eller att den sänds tillsammans med redundant data. Vidare syfte är att se hur väl mekanismerna fungerar för olika BER-värden, jämfört utan mekanismer. Testresultaten jämförs utifrån de kvalitetsvärden som uppmätts från varje avkodad bild.

### 6.5.2 Testresultat

Nedan presenteras resultaten efter simuleringarna med bitfel.

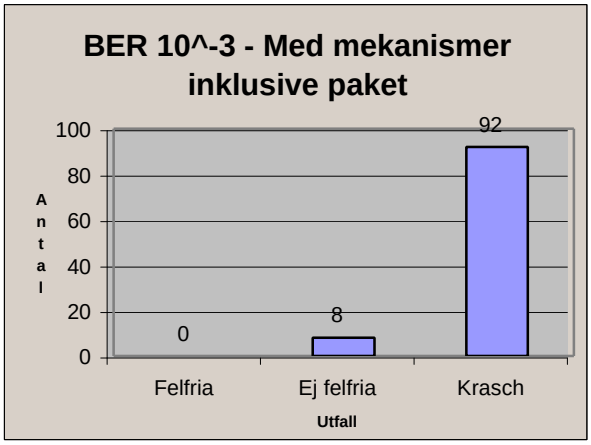
#### 6.5.2.1 BER $10^{-3}$

Följande tre figurer (figur 6.1-6.3) visar fördelningen av avkodarens framgång att avkoda bilderna. Genomgående för de tre testfallen är att ingen felfri bild kunnat återskapas från det data som fanns tillgänglig för avkodaren.



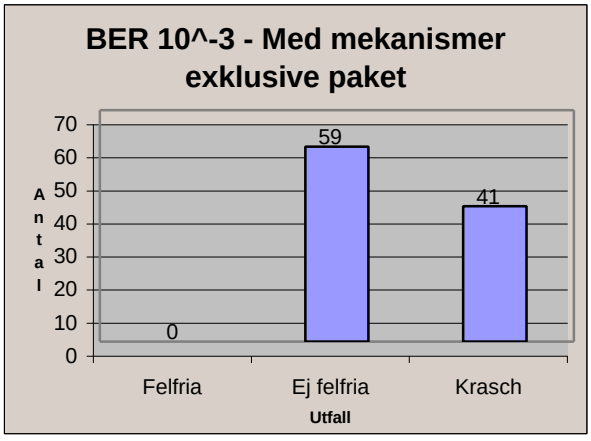

Figur 6.1. BER $10^{-3}$ - testfall 1.

För testfall 1 och 3 har mer än hälften av bilderna kunnat avkodas. I testfall 2 (se figur 6.2) kunde en stor mängd bilder inte avkodas överhuvudtaget. Den främsta anledningen är att då SOP-markören drabbades av fel (vilket innebär att kodaren inte kan avkoda markören med tillhörande sekvensnummer korrekt ur bitströmmen) avslutar kodaren [16] avkodningen och terminerar programexekveringen. Det är i enlighet med kodarens dokumentation. Paketflyttningen har ingen betydelse för hur väl kodaren lyckas i avkodningen. Det beror på tidigare nämnda aspekt, vilket är att denna mekanism endast är användbar då paketinformationen sänds över en kanal med felfria sändningsförhållanden.




Figur 6.2. BER $10^{-3}$ - testfall 2.

För testfallen 1 och 3 (se figur 6.1 och 6.3) beror krascherna främst på att avkodaren inte kan tolka bitströmmen korrekt, exempelvis på grund av att bilden har en negativ storlek, maxantalet (färg)komponenter är överskridna eller att paketheadern är korrupt. Ytterligare fel som ledde till krascherna är att antalet lager blev felaktiga. Dessa fel är exempel på bitfel i huvudheadern.




Figur 6.3. BER $10^{-3}$ - testfall 3.

Även paketheadern kan ange för korta datamängder vilket leder till att avkodaren använder för lite data. Denna händelse kan bara ske för testfall 1 och 3. I testfall 2 är det SOP-markören som förhindrar detta från att inträffa, eftersom avkodaren inte kan avkoda bilden om markören är inkorrekt. När paketheadern anger för korta datamängder leder det till att bilden blir negativt påverkad. Ju mer data som försvinner (eller rättare sagt inte används) desto större risk finns det att bilden består av skadade områden likt svarta och vita fält (se figur 6.4). En

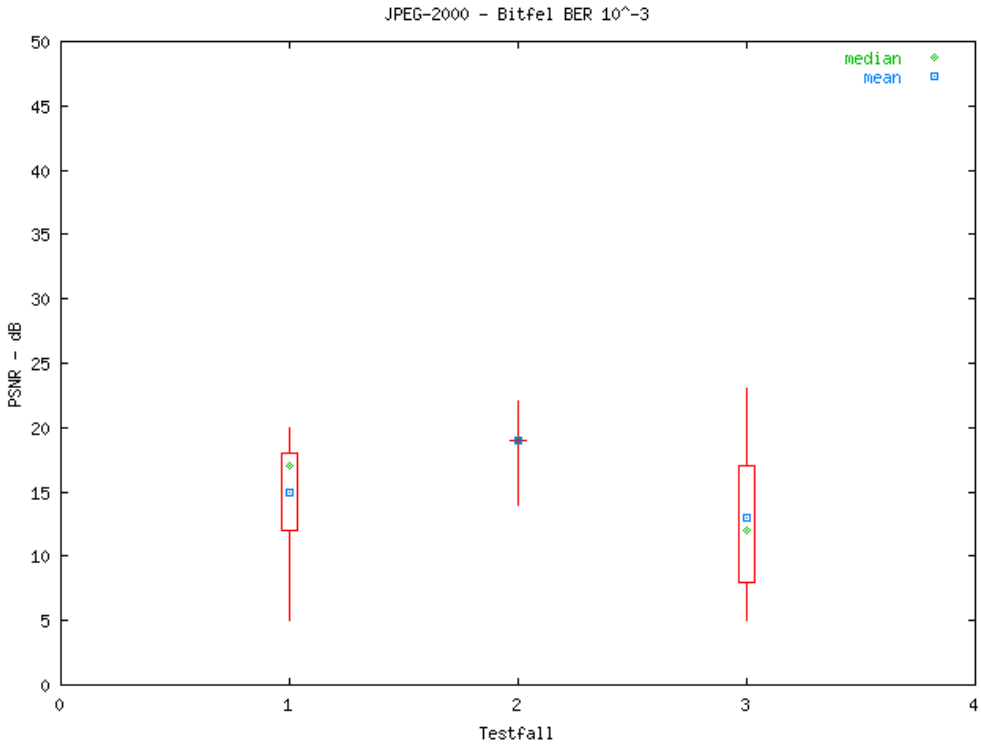
simulering som inte har utförts, men som skulle vara önskvärd, är att inte låta paketinformationen drabbas av fel. Detta skulle kunna avhjälpa ovanstående nämnda problem, där avkodaren läser för lite data från bitströmmen vilket leder till kraftigt svårtolkade bilder. En annan anledning, som också framkommit vid testningen, till varför bilden får svarta och vita fält, är att detaljkoefficienter i subband på lägre upplösningsnivåer drabbas av fel.

Figur 6.4. Svart/vita fält.

Vidare kan avkodning ske med data från endast ett fåtal upplösningsnivåer tillgängliga, vilket även det leder till felaktiga bilder.

Testerna visar att avkodaren, med hjälp av mekanismerna (testfall 2 och 3), fungerar i den bemärkelsen att den upptäcker och rapporterar fel. Man kan dock inte utläsa från sammanställningen i figur 6.5 att mekanismerna (gällande testfall 3) i medel hjälper till att återskapa bilderna med ett bättre resultat som följd, jämfört med testfall 1 då inga mekanismer används. Mekanismen som döljer fel fungerar på tidigare rekommenderat vis, nämligen att inte fortsätta avkoda nästkommande bitplan då bitfel upptäckts. Om bitfelet hamnar i bitplan som har större betydelse för kvaliteten än mindre signifikanta bitplan kommer detta påverka bildkvaliteten till det sämre. Detta innebär en större dataförlust och därmed en kvalitetsförlust. Det kan förklara varför bilderna vid testfall 3 fick en större spridning på bildkvaliteten än testfall 1 enligt figur 6.5. De avkodade bilderna som ändock blev bättre i både testfall 2 och 3, kan fallet vara så att bitfelet hamnade i ett mindre signifikant bitplan, vilket i sin tur påverkar bildkvaliteten mindre eftersom detta innebär en mindre dataförlust vid döljningen av bitfelet. Då testfall 2 endast har ett fåtal värden representerade (se figur 6.2) i figur 6.5, kan inga klara slutsatser och paralleller dras från dessa resultat jämfört med de andra testfallen. Klart är dock att SOP-mekanismen i testfall 2 inte lämpar sig för att öka kvaliteten på mottagna bilderna

jämfört med de andra testfallen. Anledning till att de bilder som trots allt blev avkodade i testfall 2 har högre kvalitet är att de inte träffades av lika många bitfel.




Figur 6.5. Testresultat $10^{-3}$.

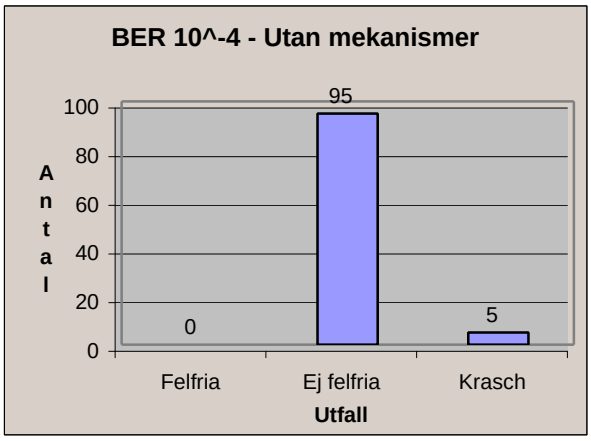
Den visuella granskning som utförts på alla bilderna tyder inte heller på att testfallet utan mekanismer (testfall 1) skulle ge ett sämre resultat. De bilder som hade det bästa PSNR-värdet för respektive testfall presenteras i figur 6.6.

**A.** Testfall 1 PSNR=20.6085dB. **B.** Testfall 2 PSNR=22.0198dB. **C.** Testfall 3 PSNR=23.3677dB.

Figur 6.6. Urval av bilder.

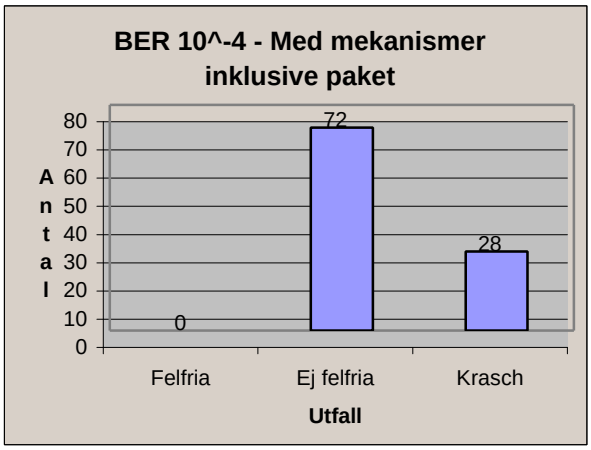
### 6.5.2.2 BER $10^{-4}$

Testfallet med BER $10^{-4}$ har utförts på motsvarande sätt som tidigare BER-värde. Vid detta testfall har det totala antalet bitfel minskat med en faktor 10. Som väntat är också kraschfrekvensen lägre, vilket beror på att ovan nämnda fel inte inträffar lika ofta. Det är dock dessa fel som även ligger bakom krascherna för BER $10^{-4}$ (se figur 6.7-6.9).

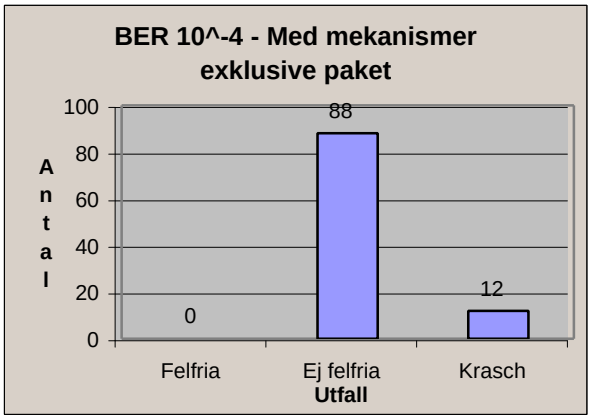



Figur 6.7. BER $10^{-4}$ - testfall 1.

Anledningen till att krascherna är fler i testfall 2 beror fortfarande på att SOP-markören är drabbad av fel (se figur 6.8). Någon perfekt bild har inte kunnat återskapats för något testfall.

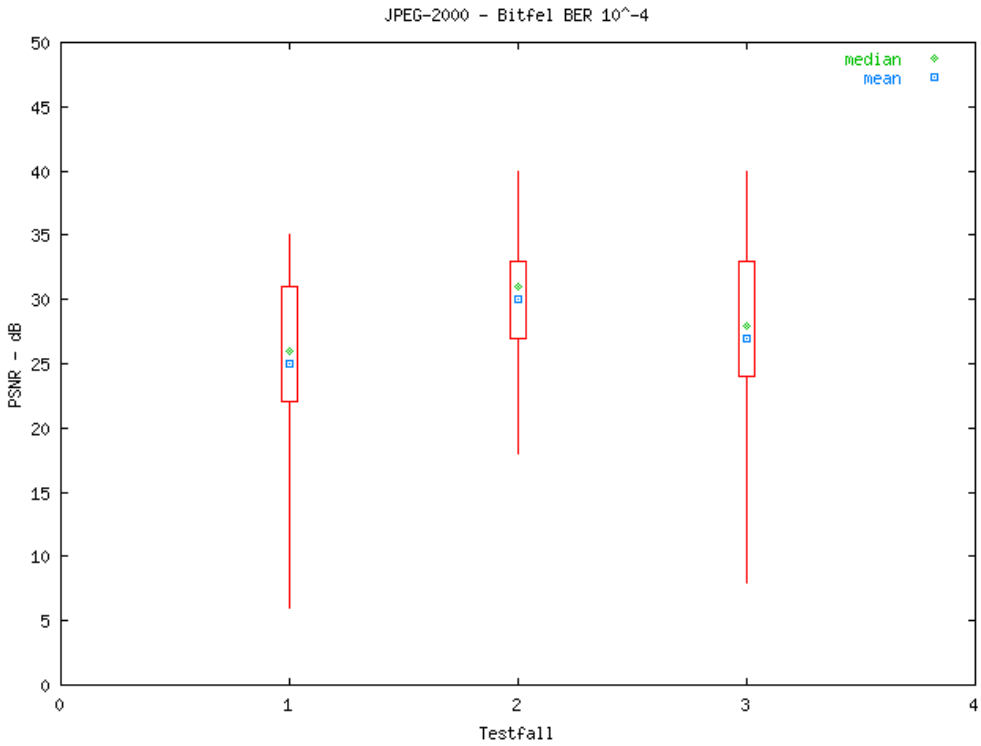



Figur 6.8. BER $10^{-4}$ - testfall 2.




Figur 6.9. BER $10^{-4}$ - testfall 3.

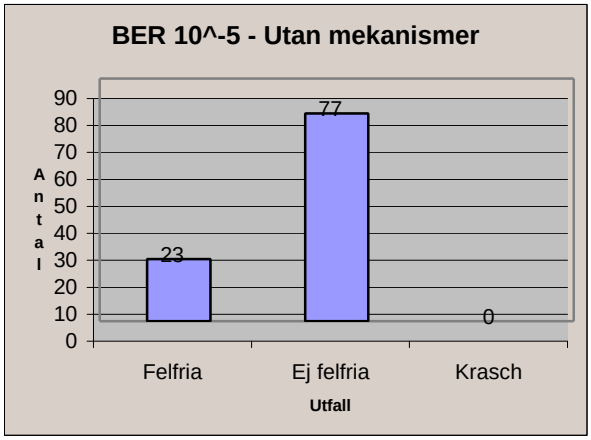
Då antalet bitfel är färre jämfört med tidigare testat BER-värde, $10^{-3}$, är följden den att den genomsnittliga kvalitetsfaktorn PSNR är högre för de ej felfria bilderna (se figur 6.5 och 6.10). Detta är en genomgående trend för alla testfallen. Avkodaren ger information om hur avkodningen av bilden gick. Informationen talar bland annat om hur många fel den upptäckt. Denna information visar att mekanismerna fungerar genom att de upptäcker alla fel i dataströmmen. Resultaten visar att en något högre kvalitet kan uppnås med mekanismerna. Hänsyn måste dock tas till det extra bortfall (se figur 6.8) som sker med krascherna med SOP-markörerna i testfall 2. Detta bortfall är fortfarande anledningen till varför testfall 2 visar bättre kvalitetsresultat jämfört med testfallen 1 och 3.



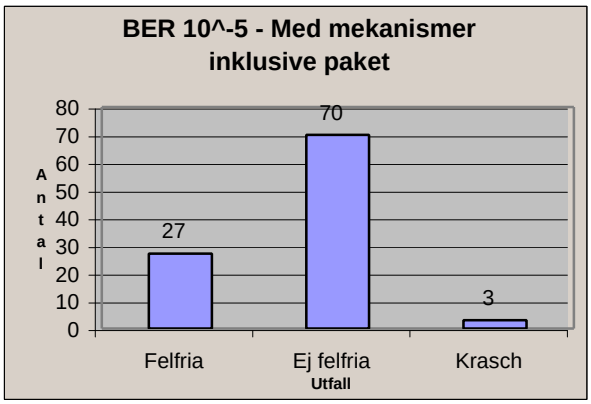

Figur 6.10. Testresultat $10^{-4}$.

### 6.5.2.3 BER $10^{-5}$

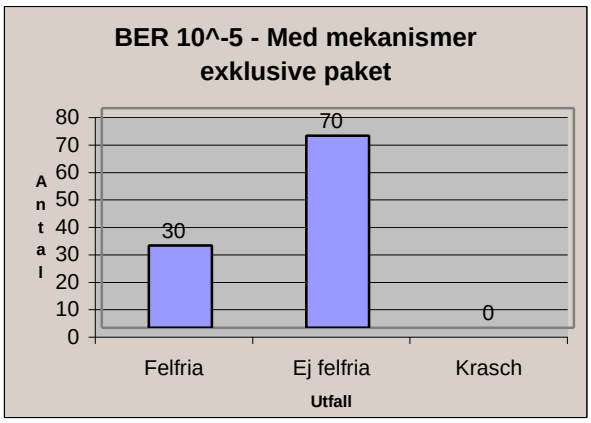
Från tabell 6.3 ges att för BER-värdet $10^{-5}$ inträffar det i genomsnitt drygt 1 bitfel i bitströmmen. Vid simuleringen kan det även inträffa fler bitfel, ända upp till 4 stycken. Vidare gäller att simuleringen även kan generera bilder utan fel. Det beror på att en sannolikhet beräknas för att bitfel ska inträffa för varje originalbit i bilden. Med detta som bakgrund kommer således en stor mängd avkodade bilder få ett förhållandevis lågt PSNR-värde, medan andra bilder återskapas helt felfria (se figur 6.11-6.13).




Figur 6.11. BER $10^{-5}$ - testfall 1.



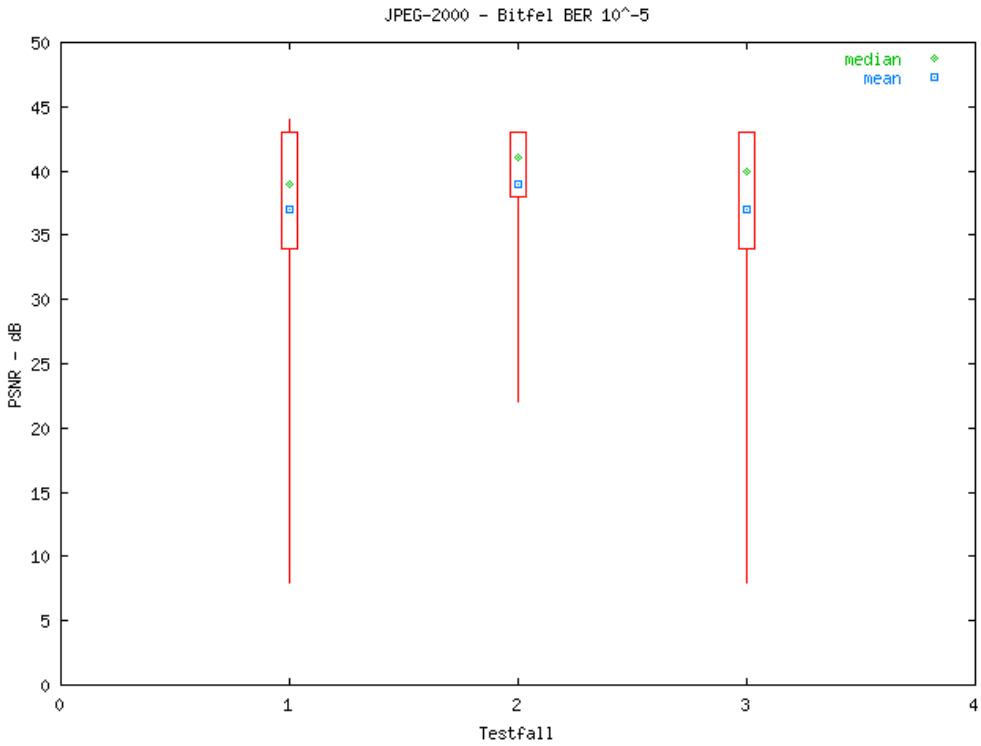

Figur 6.12. BER $10^{-5}$ - testfall 2.




Figur 6.13. BER $10^{-5}$ - testfall 3.

Ett bitfel skulle dock kunna innebära att bilden fortfarande återskapas perfekt. Tester har visat att markören EOC (End Of Codestream), som markerar att filen är slut, kan drabbas av bitfel utan att bilden blir skadad. Beroende på vilken kodare som används kan dock utfallet bli annorlunda.

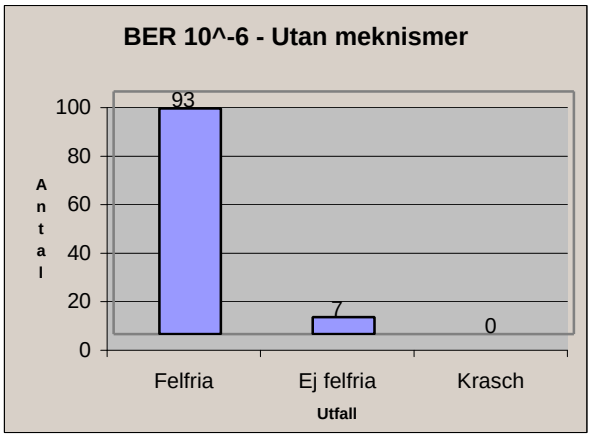
Då de felfritt återskapade bilderna innebär ett statiskt värde på PSNR, och krascherna innebär 0 i PSNR-värde, tas dessa bilders kvalitetsfaktor inte i beaktning i figur 6.14. Fallet "ej felfria" är däremot representerat. Testfall 1 och 3 har tämligen lika utfall på bildkvaliteten. Testfallen 2 drabbades av tre krascher (se figur 6.12) och detta är en faktor som spelar roll vid jämförelsen med de resterande två testfallen. Bitfel som har lett till krasch i testfall 2 leder nödvändigtvis inte till krasch i testfall 1 och 3 och det kan vara anledningen till varför somliga bilder för de två sistnämnda testfallen fick ett lägre PSNR-värde. Med detta i åtanke kommer även medelvärdet på bildkvaliteten för dessa testfall sjunka. Från figur 6.14 kan således inga klara slutsatser dras att mekanismerna har en positiv påverkan på bildkvaliteten.




Figur 6.14. Testresultat $10^{-5}$.

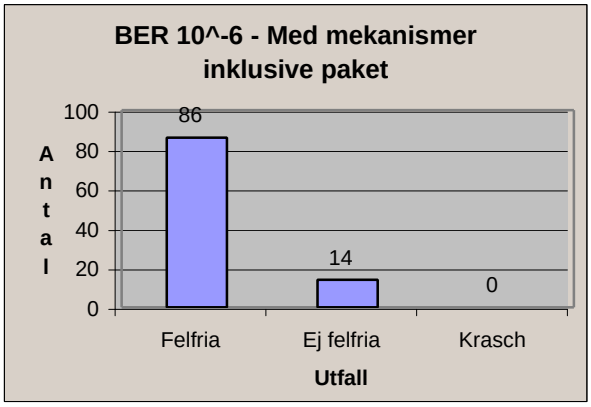
#### 6.5.2.4 BER $10^{-6}$

För BER-värde $10^{-6}$ inträffar nästan inga fel överhuvudtaget (se tabell 6.3). Detta faktum kan även utläsas från figur 6.15 till 6.17.

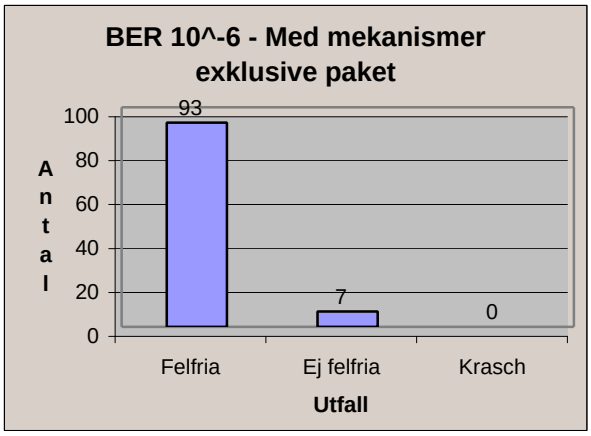



Figur 6.15. BER $10^{-6}$ - testfall 1.

Från figur 6.16 ges att färre bilder fick perfekt kvalitet jämfört med testfall 1 och 3. Det skulle kunna förklaras med att fler bitfel inträffade vid denna simulering, och grundar sig därmed inte i valet av mekanism.

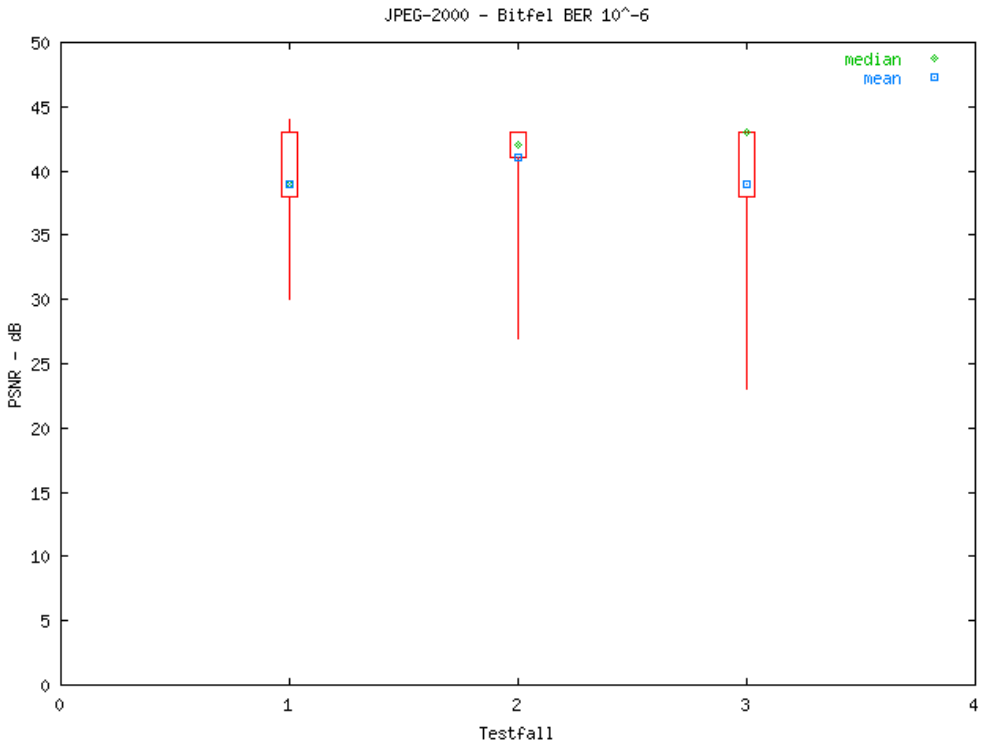



Figur 6.16. BER $10^{-6}$ - testfall 2.




Figur 6.17. BER $10^{-6}$ - testfall 3.

Då underlaget är endast ett fåtal simuleringsfall kan inga klara slutsatser dras från sammanställningen i figur 6.18. Att döljmekanismen, som är aktiv i testfall 2 och 3, skulle inverka positivt på kvaliteten och därmed ge ett högre PSNR-värde för bilder drabbade av ett lågt antal bitfel, kan inte observeras. Tidigare notering om att inte avkoda fler bitplan efter bitfel är dock en rekommendation att föredra, då ett närmevärde på en koefficient är bättre än ett felaktigt värde.




Figur 6.18. Testresultat $10^{-6}$.

### 6.5.3 Sammanställning

Slutsatsen är att vid höga BER-värden ger mekanismerna ingen större skillnad i den generella bildkvaliteten (se tabell 6.4) i jämförelse med att inga mekanismer alls finns kodade i bitströmmen. BER $10^{-3}$, som innebär ett stort antal bitfel, är ett sändningsförhållande som JPEG2000-bilder inte bör skickas i, utan att det finns externa möjligheter tillhands att hantera felen med. En åtgärd som skulle leda till en bättre bildkvalitet är då paketinformationen eller den s.k. styrinformationen kan sändas över till mottagaren på ett felfritt sätt. Detta gäller både då bilderna kodas med eller utan mekanismer. Vid testfall 1 och 3 lyckades avkodaren avkoda nästan lika många bilder för respektive fall utan krasch, dock visar det tidigare testfallet på en högre kvalitet på de resulterande bilderna. Förklaringen ligger i att döljalgoritmen i testfall 3 anropas frekvent vilket skulle kunna leda till en informationsförlust. Testfall 2 bidrar endast med ett lågt antal bilder för att ge ett kvalificerat uttalande om dess positiva påverkan på den slutliga bildkvaliteten för denna bitfelstakt.

När BER antar värden runt $10^{-4}$, visar testfallen med mekanismerna i medel ett bättre resultat än testfall 1. Endast under dessa förhållanden ger testfall 3 högre kvalitetsvärden än testfall 1 (se tabell 6.4). Här har även varje testfall kunnat bidra med ett underlag på mer än 70 lyckade avkodade bilder.

Vid BER $10^{-5}$ och $10^{-6}$ kan inte några klara skillnader observeras mellan de olika testfallen. Generellt ger testfall 2 alltid högre värden gällande bildkvaliteten. Testfall 2 och 3 skiljer sig inte mer än på de punkter som framgår från kodarens konfiguration (se kapitel 6.2). Med detta som diskussionsunderlag plus det faktum att bitfelen placeras godtyckligt i bildfilen kan resultatet angående det höga medelvärdet för bildkvaliteten för testfall 2 inte sammankopplas med att en avkodare bör utnyttja just de mekanismer som gäller för testfall 2 för att uppnå bäst bildkvalitet. För BER $10^{-6}$ är det ett mycket litet antal värden representerade för alla testfallen eftersom många av bilderna avkodades felfritt. På grund av detta kan heller ingen slutsats dras.

| **BER** | **$10^{-3}$** | **$10^{-4}$** | **$10^{-5}$** | **$10^{-6}$** |
|---|---|---|---|---|
| **Testfall 1** | 15,3270 | 25,5927 | 37,5263 | 39,7112 |
| **Testfall 2** | 19,1926 | 30,7676 | 39,7283 | 41,2833 |
| **Testfall 3** | 13,4670 | 27,3515 | 37,4974 | 39,1627 |

Tabell 6.4. Medelvärden för kvaliteten på ej felfria bilder.

Den viktigaste observationen som gjorts är att simuleringarna av de fyra olika BER-värdena visar att mekanismerna upptäcker fel, oberoende av BER-värde. När felmekanismerna används vid höga BER-värden ($10^{-3}$) är avkodaren känslig för fel på grund av att den tolkar datan felaktigt. Avkodaren kan då generera ett programfel och följden blir att ingen bild blir avkodad. Detta gäller även för testfall 1.

För att dölja fel på bitplansnivå anropas en döljalgoritm i kodaren [16]. Denna åtgärd innebär att nästkommande mindre signifikanta bitplan inte används. Om många fel inträffar kommer således många bitplan inte användas. Det innebär då ytterligare en kvalitetsförsämring. Någon detaljerad studie av hur denna mekanism påverkar bildkvaliteten har inte utförts.

Hur mycket en bild påverkas av fel beror även på var någonstans felet inträffar. Drabbas de approximerade värdena av bilden, eller detaljkoefficienter i subband på lägre upplösningsnivåer, får det större negativt genomslag på hela bilden, än om detaljkoefficienter i subband på högre upplösningsnivåer drabbas (se kapitel 3). En korrekt dataström ger följande korrekta avkodning som resultat:

**[6 2 1 -1] ® [8 4 1 -1] ® [9 7 3 5]**

Om det approximerade värdet blir felaktigt, kommer varje värde i slutresultatet skilja sig från originalsignalen ovan:

**[7 2 1 -1]® [9 5 1 -1] ® [10 8 4 6]**

Om dataströmmen drabbas av fel i detaljkoefficienterna i högre subband, kommer inte alla värden i signalen bli felaktiga:

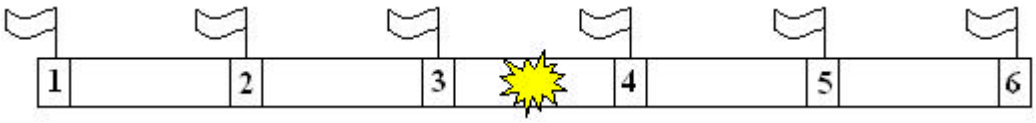
**[6 2 1 -2]® [8 4 1 -2] ® [9 7 2 6]**

Markörer, som talar om för avkodaren vilken data den arbetar med, är känsliga för fel. Det är mekanismernas uppgift att skydda bilddatan, som utgör större delen av hela datamängden. Ett överraskande resultat är att avkodaren utan mekanismer i dataströmmen, klarar av att avkoda en bild överhuvudtaget.

Det kan inträffa olika felsituationer i bitströmmen. Exempelvis kan huvudheadern innehålla felaktig information till avkodaren. Oftast leder dessa fel till att avkodaren avbryter avkodningen helt och hållet. Andra fel har också observerats. Figur 6.19 visar när ett fel träffar återsynkroniseringsmarkören SOP och tillhörande identifikationsnummer på paketnivå.


Figur 6.19. Markör drabbad av fel.

I figur 6.19 förlorar avkodaren synkroniseringen med bitströmmen, vilket leder till att den inte kan avkoda rätt identifikationsnummer. Den andra situationen är när fel träffar datan mellan två SOP-markörer (se figur 6.20). Här är det de andra mekanismernas uppgift att upptäcka att fel har inträffat i bitströmmen.




Figur 6.20. Data mellan markörer drabbad.

## 6.6 Skurfel

Skurfel inträffar ofta i mobila trådlösa nätverk. Som bakomliggande faktor till skurfel ligger fenomenet fädning. Det finns två huvudtyper av fädning [26]. Den första typen kallas skuggfädning. Den uppstår då en mobil rör sig i en omgivning som innefattar hinder av olika slag. Hindren är exempelvis berg, hus och tunnlar. Dessa hinder kommer att hindra signalen mellan basstationen och mobilen. För att förebygga skuggfädning, kan fler basstationer och basstationer högt placerade sättas upp. Den andra typen kallas Rayleigh-fädning. Den typen kan också uppstå i omgivningar som har många hinder, men här reflekteras signalen mot hindren. Konsekvensen blir att mottagaren tar emot flera signaler som reflekterats mot omgivningens hinder, och på grund av att signalerna har färdats olika långt har de även olika faser, som kan leda till att signalerna kan motverka varandra.

### 6.6.1 Simulering av skurfel

Simuleringen av skurfel har genomförts med sex stycken testfiler för ett Wideband Code Division Multiple Access-nät (WCDMA-nät) från [14]. Testfilerna är genererade från simulerade WCDMA-nät med inlagda felmönster av typen skurfel. Testfilerna används för att kunna utvärdera robustheten för olika applikationer.

WCDMA är ett radioaccess-protokoll för att stödja bredbandsaccessen till tredje generationens multimediatjänster. Protokollet är optimerat för höghastighets-multimediatjänster där accesshastigheten kan uppgå till 2 Mbit/s [10]. Tjänsterna är exempelvis röst- och Internettjänster och videokonferenser.

Testfilerna är simulerade med en överföringshastighet på 64 kbit/s. Skurfel kan uttryckas med genomsnittslängden på skurarna och ett BER-värde. BER-värdet ger hur många bitar som är felaktiga i hela strömmen. Alla bitar i en skur löper samma risk att drabbas av bitfel enligt det angivna BER-värdet. Vid högre BER-värden ökar längden på skurarna och / eller totala antalet skurar. Då mobilenheten framförs i olika hastigheter kommer karaktären på skurfelen förändras (se tabell 6.5). Med det som bakgrund finns också hastigheten med som en parameter i simuleringen. Följande tre hastigheter är aktuella; 3, 40 och 120 km/h. För varje hastighet har även två olika BER-värden använts, $10^{-3}$ och $10^{-4}$. Dessa värden har endast fungerat som riktvärden vid framtagningen av testfilerna. De exakta BER-värdena framgår av tabell 6.5. BER-värdet beskriver den genomsnittliga sannolikheten för bitfel i en bild.

Testningen har alltså genomförts med sex stycken testfiler, vilka är en kombination av de tre hastigheterna och de två BER-värdena. För varje testfil kan testbilden appliceras 80

gånger. En testomgång innebär således att testbilden överförs 80 gånger sekventiellt. Då den givna överföringshastigheten är 64 kbit/s, sker överföringen av testbilden 80 gånger på cirka 3 minuter.

Tabell 6.5 redogör för hur många skurar i genomsnitt (teoretisk beräkning) som har inträffat i bildfilen, och hur lång varje skur i genomsnitt var. I en skur av fel är nödvändigtvis inte alla bitar fel. Mellan två på varann följande skurar antas det finnas minst 8 felfria bitar [14].

| BER | Testfilskonfiguration | Skurlängd i bitar | Antal skurar / bild |
|---|---|---|---|
| $10^{-3}$ | 3 km/h (BER $1{,}35*10^{-3}$) | 16 | 11 |
| $10^{-4}$ | 3 km/h (BER $8{,}17*10^{-5}$) | 11 | 1 |
| $10^{-3}$ | 40 km/h (BER $1{,}26*10^{-3}$) | 17 | 10 |
| $10^{-4}$ | 40 km/h (BER $1{,}21*10^{-4}$) | 13 | 1 |
| $10^{-3}$ | 120 km/h (BER $9{,}73*10^{-4}$) | 15 | 8 |
| $10^{-4}$ | 120 km/h (BER $9{,}37*10^{-5}$) | 11 | 1 |

Tabell 6.5. Konfiguration av testfilerna.

Även om det förekommer ett antal skurar totalt i en bildfil, innebär det inte att avkodaren kommer rapporterar alla dessa. När fel upptäcks (upptäckt inträffar då felmekanismerna är påslagna), och därmed döljs, kommer som bekant döljfunktionen inte att använda resterande bitplan. Eftersom ytterligare skurfel och bitfel kan inträffa i just dessa bitplan, kommer således heller inte avkodaren att rapportera dem. Samma resonemang gäller även för bitfel.

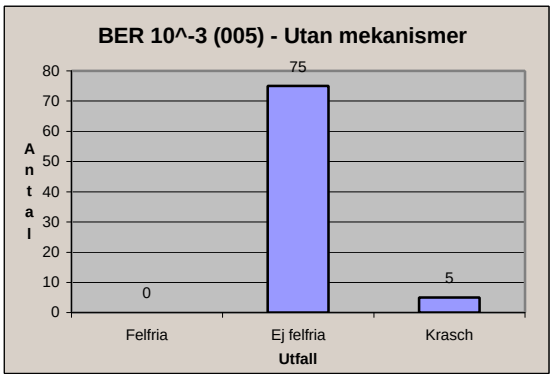
### 6.6.2 Testresultat

Nedan följer testresultaten från de sex testomgångarna med skurfel.

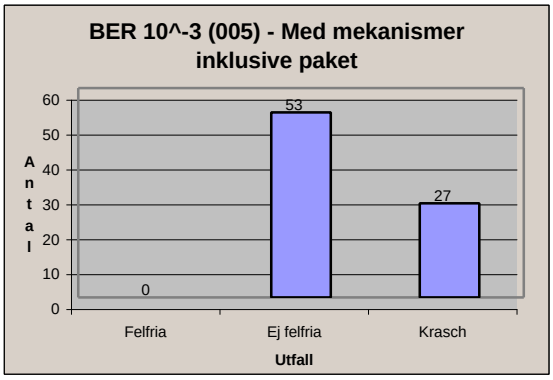
#### 6.6.2.1 Skurfel vid 3 km/h med BER $10^{-3}$

Från testresultaten i figur 6.21-6.23 kan man utläsa att testfall 2 klarade sig sämre mot denna typ av fel jämfört med testfall 1 och 3. Testfall 1 och 3 är likvärdiga gällande antalet avkodade bilder och krascher. Testfall 3 har dock en större spridning på kvaliteten jämfört med testfall 1 (se figur 6.24). En förklaring skulle kunna vara att eftersom mekanismerna uppmärksammar fel och utför åtgärder därefter skulle detta kunna bidra till en sämre kvalitet (exempelvis feldöljningen) framförallt då skurfelen inträffar i bitplan med högre signifikans. Att somliga bilder från testfallen 3 får sämre kvalitet än testfall 1, ska inte kopplas samman

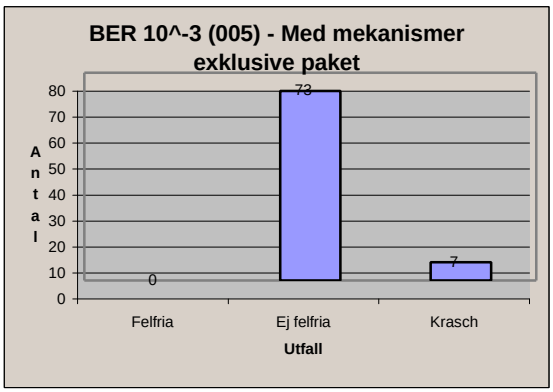
med att kodarens robusthet är dålig. Snarare visar dessa resultat på att robustheten i JPEG2000 fungerar väl eftersom de endast skall upptäcka och dölja bitfel. Är däremot robusthetsmekanismerna av den karaktär att de också ska rätta fel, så spelar kvaliteten en mer avgörande roll.




Figur 6.21. Testfall 1, hastighet 3 km/h BER $10^{-3}$.



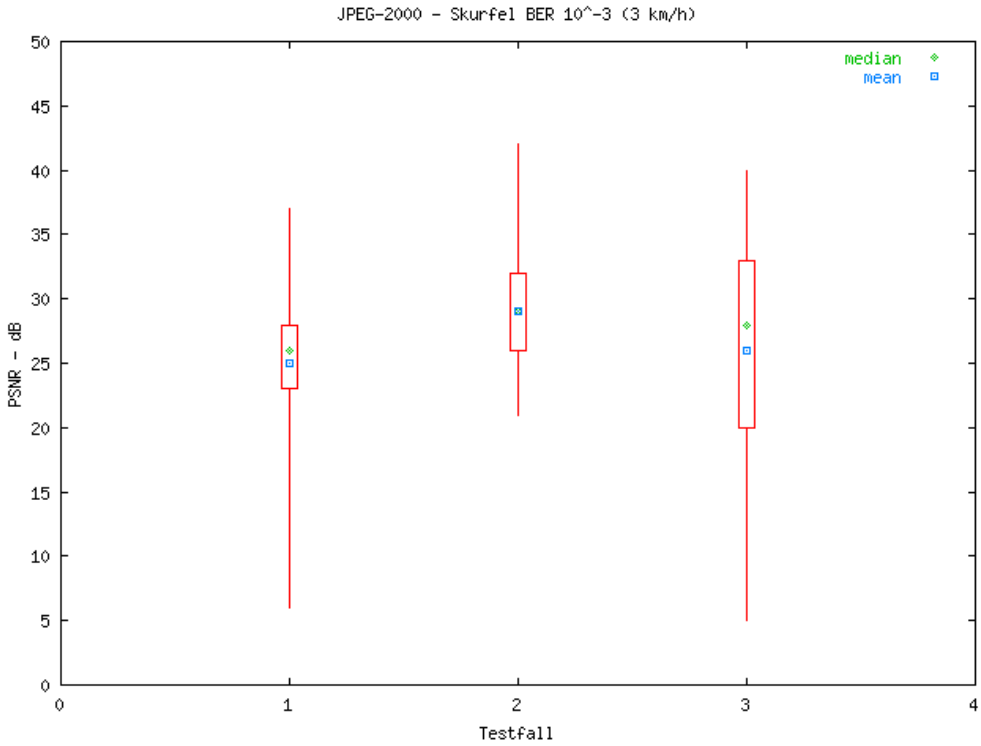

Figur 6.22. Testfall 2, hastighet 3 km/h BER $10^{-3}$.




Figur 6.23. Testfall 3, hastighet 3 km/h BER $10^{-3}$.

Från figur 6.5 och nedanstående figur 6.24 kan man utläsa att skurfel med samma BER-värde inte påverkar avkodaren lika negativt som slumpvisa bitfel (hänsyn måste dock tas till

det faktum att skurfel endast är testade med 80 bilder jämfört med 100 bilder för bitfel och att underlaget för figur 6.24 är större än för figur 6.5).

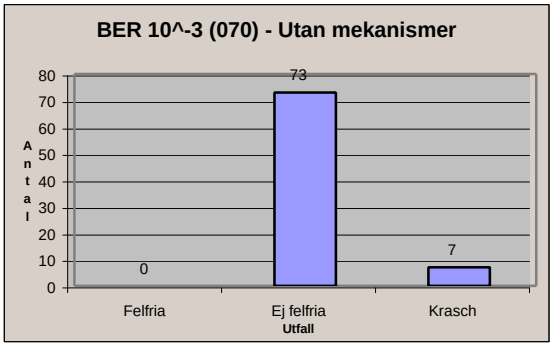



Figur 6.24. Testresultat 3 km/h BER $10^{-3}$.

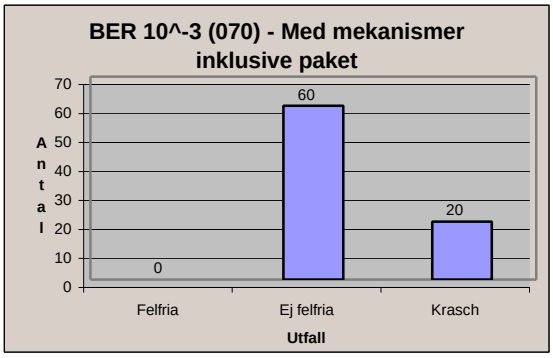
### 6.6.2.2 Skurfel vid 40 km/h med BER $10^{-3}$

I konfigurationen för hastigheten 40 km/h minskar antalet skurar i genomsitt med ett (se tabell 6.5), vilket skulle kunna förklara de något positivare resultaten för testfall 1 och 3 som figur 6.28 visar. Att antalet bitar för varje enskild skur ökar med en bit har mindre betydelse på bildkvaliteten än vad antalet skurar har. I testfall 2 har den genomsnittliga bildkvaliteten försämrats något jämfört med den tidigare konfigurationen vid 3 km/h, däremot har ett fåtal bilder kunnat återskapats med högre kvalitet för detta testfall.

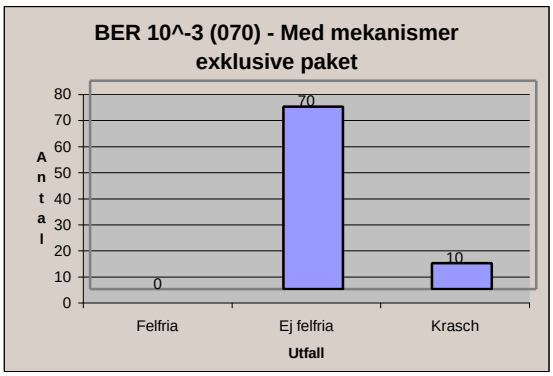
Fortfarande sker krascher vid avkodningen (se figur 6.25-27.). För testfall 2 beror de flesta krascherna på att SOP-markören och dess tillhörande sekvensnummer inte kan avkodas korrekt. För testfall 1 och 3 (se figur 6.25 och 6.27) beror krascherna främst på att huvudheadern blir felaktig.




Figur 6.25. Testfall 1, hastighet 40 km/h BER $10^{-3}$.

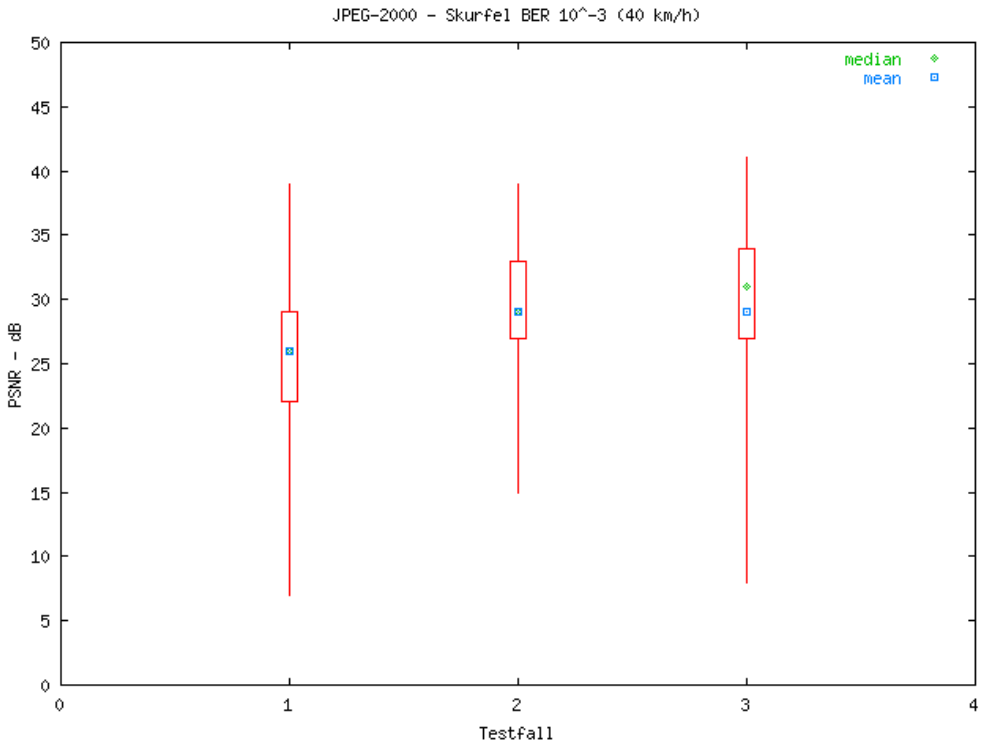



Figur 6.26. Testfall 2, hastighet 40 km/h BER $10^{-3}$.




Figur 6.27. Testfall 3, hastighet 40 km/h BER $10^{-3}$.

En jämförelse mellan de olika testfallen visar att avkodning med mekanismerna är att föredra då en bättre kvalitet eftersträvas (se figur 6.28). För testfall 1, då inga mekanismer används, förekommer däremot färre krascher. Den genomgående fördelen med att använda mekanismerna är dock att de alltid hjälper kodaren att i någon grad rapportera att fel inträffat i bitströmmen. Denna information skulle senare kunna användas för att göra omsändningar av den bildinformation som är skadad.




Figur 6.28. Testresultat 40 km/h BER $10^{-3}$.

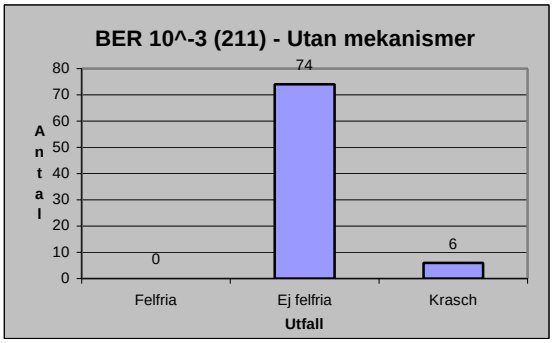
I figur 6.29 visas exempel på bilder från testfall 3. I figur 6.29 A har bildens bitplan belägna i den lägsta och de två högsta upplösningarna blivit skadade av skurfel. Skillnaden mellan figur 6.29 A och 6.29 B är att i den senare har avkodaren inte använt all data och dessutom felaktig data. Datan är felaktig på så sätt att den tillhör lägre upplösningars subband. Detta fel kan även inträffa för bilder i testfall 1. Med hjälp av synkroniseringsmarkörer hade avkodaren kunnat komma till rätta med detta problem. Förvånande är att avkodaren lyckas avkoda en bild överhuvudtaget. I den avkodade bilden i figur 6.29 A har däremot alla data använts korrekt, därav den bättre kvaliteten. I båda fallen sker fortfarande avkodning på ett robust sätt, eftersom avkodaren meddelar att bilderna innehåller fel.

**A.** **B.**

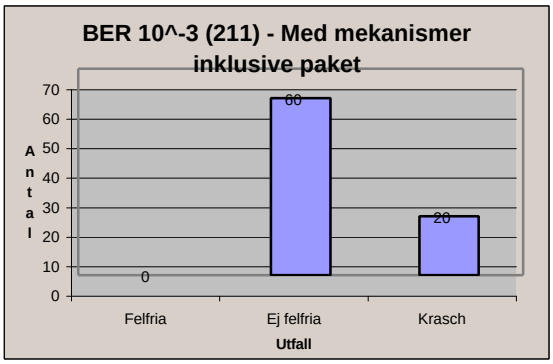
Figur 6.29. Felaktigt avkodade bilder.

### 6.6.2.3 Skurfel vid 120 km/h med BER $10^{-3}$

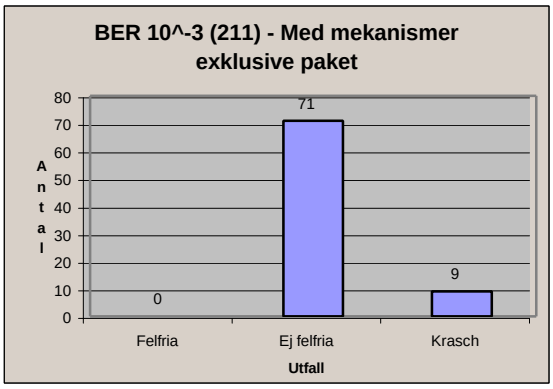
Vid konfigurationen med 120 km/h och med BER-värdet $10^{-3}$ är antalet skurar färre och längden på varje skur något kortare än tidigare fall (se tabell 6.5). Det inträffar alltså något färre krascher än det senaste redovisade fallet (se figurer 6.25-6.27 och 6.30-6.32), vilket är i enlighet med tabell 6.5. För denna konfiguration kan inget samband kopplas till att något kortare skurar skulle leda till bättre bildkvalitet.




Figur 6.30. Testfall 1, hastighet 120 km/h BER $10^{-3}$.

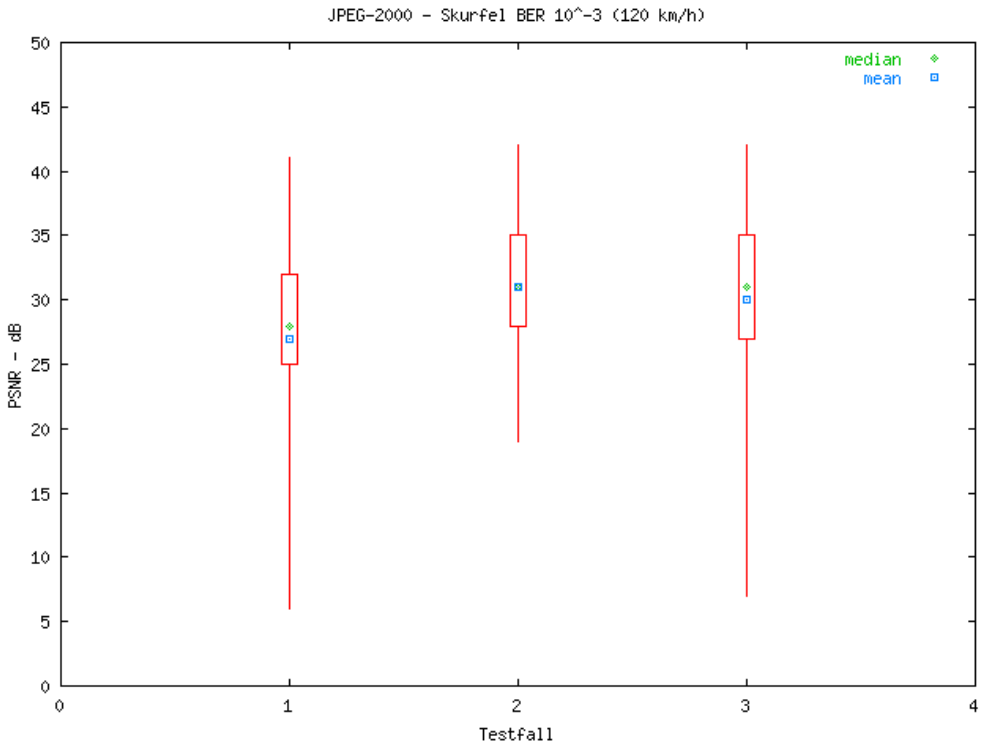



Figur 6.31. Testfall 2, hastighet 120 km/h BER $10^{-3}$.



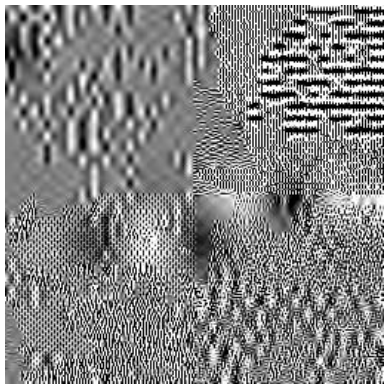

Figur 6.32. Testfall 3, hastighet 120 km/h BER $10^{-3}$.

Även om det rent teoretiskt, med stöd av tabell 6.5, existerar färre och kortare skurfel vid denna konfiguration, kan ingen klar slutsats dras om att en bättre kvalitet uppnås. Tendensen är ändå att vissa avkodade bilder, från alla testfallen, har bättre kvalitet är tidigare redovisade konfigurationer (se figur 6.33). Den troligaste förklaring är dock att skurfelen är färre. Ytterligare en förklaring skulle kunna vara det att skurfelen placeras på känsligare ställen (med avseende på bildkvaliteten) i bildfilen i den föregående konfigurationen.




Figur 6.33. Testresultat 120 km/h BER $10^{-3}$.

Figur 6.34 visar på hur ett avkodningsresultat kan se ut från testfall 3 när koefficienterna i minsta upplösningen blir felaktiga. Här har även paketinformationen blivit felaktig, vilket har lett till att felaktig data lästs in för resterande upplösningsnivåer och bitplan. Värt att påpeka, även i denna avkodning, är att bilden överhuvudtaget har gått att avkoda. Datamängden till de resterande subbanden på högre upplösningar, har varit liten, endast ett fåtal bytes. Om SOP-markören hade använts som ytterligare mekanism, hade bilden inte gått att avkoda. Med hjälp av SOP-markören hade avkodaren kontinuerligt haft möjlighet att synkronisera med bitströmmen, vilket inte är fallet i avkodningen för bilden i figur 6.34.



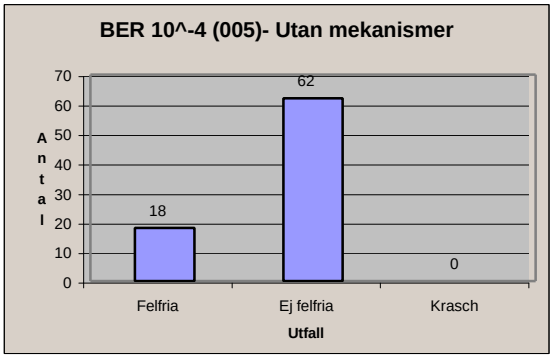
Figur 6.34. Felaktigt avkodad bild.

I skurfelstesterna som redovisats hitintills, har det klart framkommit att då ingen synkronisering med bitströmmen sker kan fel leda till felaktig tolkning av bitströmmen. Om synkroniseringen förloras, vilket kan vara fallet utan SOP-markör, kan alltså avkodaren tolka datan helt godtyckligt. Testerna visar vidare att när avkodaren tappat synkroniseringen vid testfall 1 och 3, kommer hela den fortsatta bitströmmen tolkas felaktigt. Med SOP-markören kommer paketheadrar alltid att tolkas korrekt, nämligen som paketheadrar. Om paketheadern innehåller en felaktig längdinformation för kommande data, kommer avkodaren inte att kunna avkoda nästföljande SOP-markör korrekt.

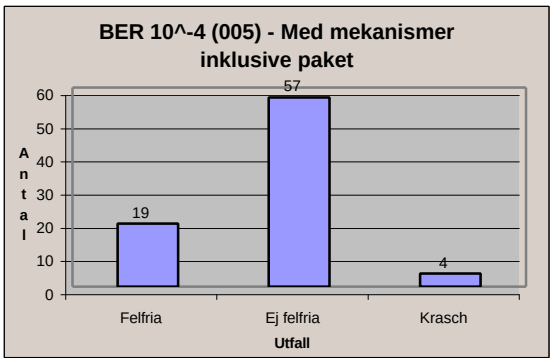
Avkodaren som tolkar en korrupt bitström, kodad med testfall 3, kan skapa paket från felaktig data. Avkodaren kommer därefter, i de allra flesta fall, inte att kunna avkoda en segmentsymbol i slutet på ett bitplan. Konsekvensen av detta blir att döljfunktionen anropas.

#### 6.6.2.4 Skurfel vid 3 km/h med BER $10^{-4}$

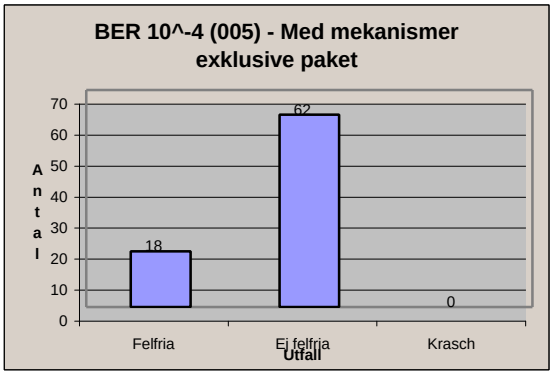
I följande tre skurfelstester har antalet skurfel per bild sjunkit drastiskt. Detta återspeglas också i positivare avkodningsresultat (se figur 6.35-6.37). I de bilder som blir felfritt återskapade har inga skurfel inträffat.




Figur 6.35. Testfall 1, hastighet 3 km/h BER $10^{-4}$.

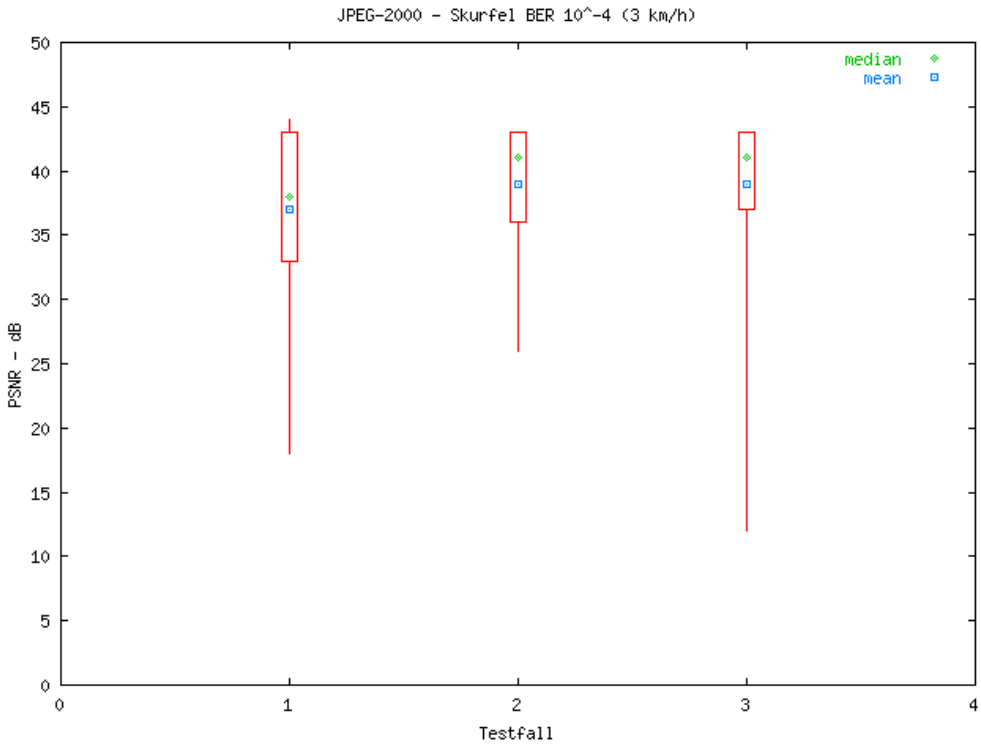



Figur 6.36. Testfall 2, hastighet 3 km/h BER $10^{-4}$.




Figur 6.37. Testfall 3, hastighet 3 km/h BER $10^{-4}$.

Alla bilderna har testats mot samma testfil men eftersom filstorlekarna är olika för de tre testfallen infaller skurfelen på olika ställen. För testfall 1 och 3 har dock lika antal bilder avkodats. Ett fåtal krascher inträffar för testfall 2. Det är SOP-markören som inte kunnat avkodas korrekt och huvudheadern som angivit orimliga storlekar för bilden.

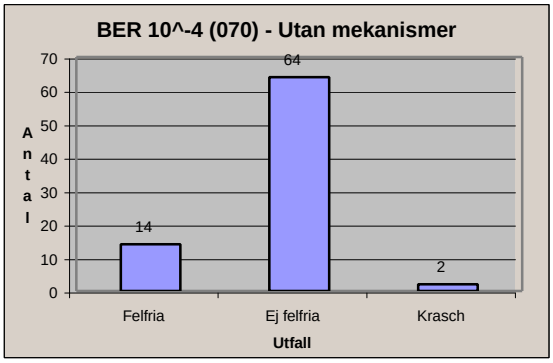



Figur 6.38. Testresultat 3 km/h BER $10^{-4}$.

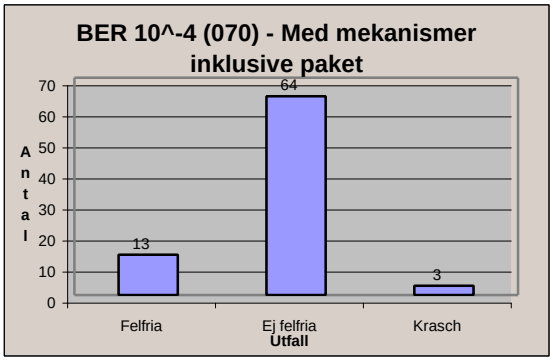
De sammanfattade resultaten i figur 6.38 visar att med konfigurationen enligt testfall 2 och 3 är medelkvaliteten för de avkodade bilderna något högre jämfört med testfall 1.

#### 6.6.2.5 Skurfel vid 40 km/h med BER $10^{-4}$

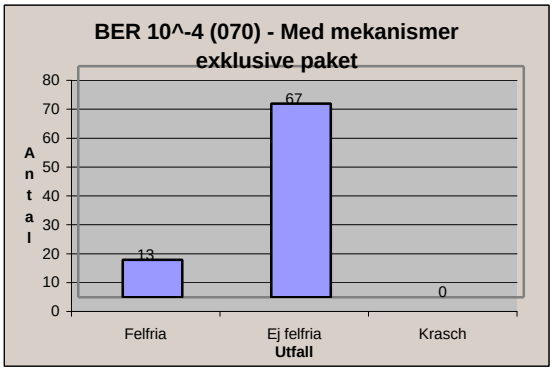
Resultaten för skurfel vid 40 km/h med BER $10^{-4}$ skiljer sig inte nämnvärt från föregående test. Det inträffar ett fåtal krascher för testfall 1 och 2 (se figur 6.39-6.41). Krascherna härstammar från fel i huvudheadern och inkorrekt avkodning i SOP-markören. Antalet skurfel är lika många som föregående test men däremot är längden på varje skur något längre. Detta, tillsammans med placeringen av skurfelen, kan förklara varför färre bilder har återskapats felfritt.




Figur 6.39. Testfall 1, hastighet 40 km/h BER $10^{-4}$.

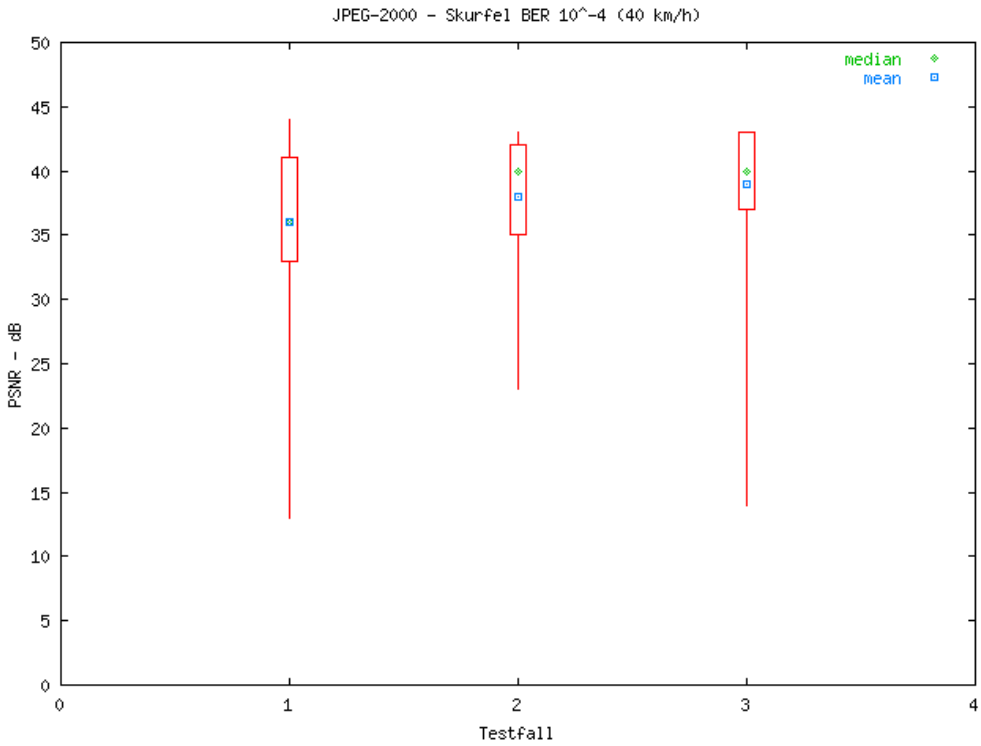



Figur 6.40. Testfall 2, hastighet 40 km/h BER $10^{-4}$.




Figur 6.41. Testfall 3, hastighet 40 km/h BER $10^{-4}$.

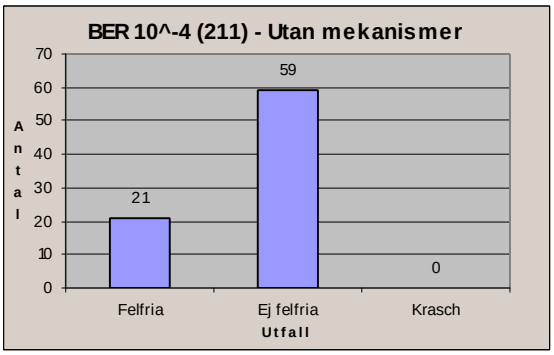
Då kodaren är konfigurerad med mekanismer ges en något högre medelkvalitet för de bilder som inte blev perfekt återskapade (se figur 6.42).



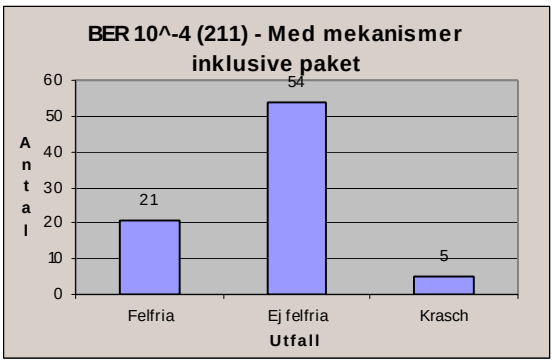

Figur 6.42. Testresultat 40 km/h BER $10^{-4}$.

#### 6.6.2.6 Skurfel vid 120 km/h med BER $10^{-4}$

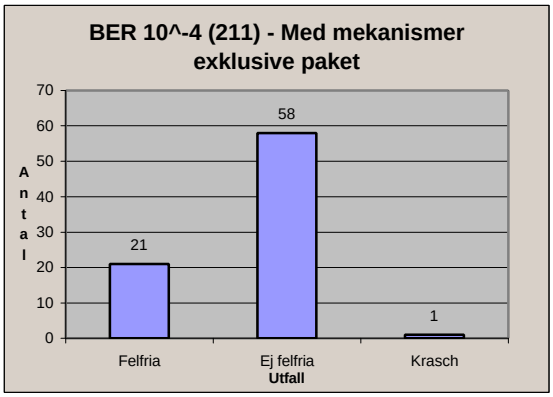
Skurfel vid en hastighet av 120 km/h och felsannolikhet på $10^{-4}$ ger inte lika stor påverkan på bitströmmen i jämförelse med föregående fall. Det framgår från figur 6.43-6.45 där det kan utläsas att antalet perfekta bilder har ökat. Dessa resultat visar inte att mekanismerna fungerar bättre utan endast att det är mindre fel i bitströmmen. För testfall 2 (se figur 6.44) är bitströmmen mera känslig mot fel än testfall 3 (se figur 6.45) på grund av att det senare fallet inte använder SOP-markören.




Figur 6.43. Testfall 1, hastighet 120 km/h BER $10^{-4}$.

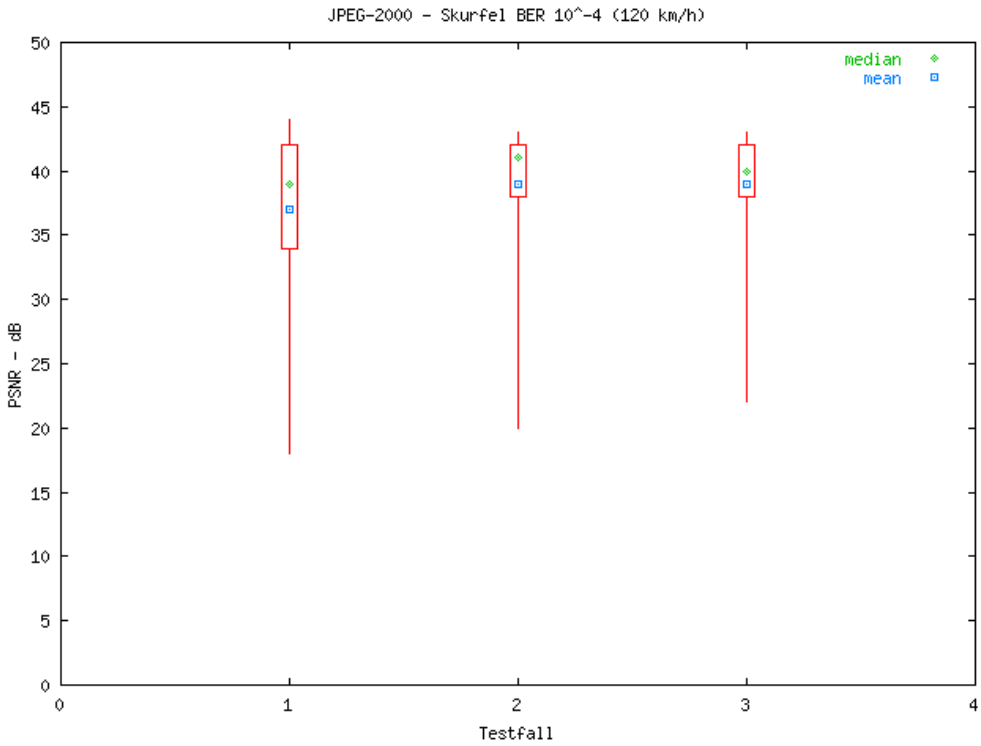



Figur 6.44. Testfall 2, hastighet 120 km/h BER $10^{-4}$.




Figur 6.45. Testfall 3, hastighet 120 km/h BER $10^{-4}$.

Den slutliga kvaliteten på alla ej felfria bilder är sammanställd i figur 6.46. I likhet med tidigare fall med skurfel ges en något högre medelkvalitet med mekanismer.




Figur 6.46. Testresultat 120 km/h BER $10^{-4}$.

### 6.6.3 Sammanställning

Testerna har visat på betydelsen av var någonstans ett fel inträffar för utgången av avkodningen. Om ett fel inträffar på koefficienter på lägre subband får det större genomslag, än om liknande fel inträffar på högre subbands koefficienter. Anledningen är det starka beroende som de högre subbandens koefficienter har av de lägre subbandens.

Vidare slutsatser utifrån testerna är att vid BER $10^{-3}$ och $10^{-4}$ spelar hastigheten in genom att skurfelen då blir fler och längre. En konstellation av skurfel som är lokala och som inträffar i tät följd, jämfört med enstaka skurfel som inträffar med längre avstånd mellan varandra, kan det tidigare fallet vara mer gynnsamt i avkodningshänseende. När fel inträffar i ett bitplan döljs resterande plan och därmed ytterligare skurfel. Medan i det senare fallet kommer felen vara utspridda över hela bilden. Positionen av skurfel i bilden har dock fortfarande stor betydelse för den slutliga bildkvaliteten.

Mekanismerna verkar i de flesta fall alltid kunna återskapa några bilder med högre kvalitet än om inga mekanismer används. Kvalitetsresultaten är mer samlade vid testfall 2 och 3 vid kortare och färre skurar. Testerna har visat att i vissa fall är spridning på kvalitetsresultatet för testfall 3 dock större jämfört med testfall 1 (se figur 6.24). Det gäller vid BER-värdet $10^{-3}$. Enligt tabell 6.6, vilket visar medelkvaliteten för respektive fall, verkar mekanismerna ge en förbättring i bildkvaliteten. Detta talar för att använda mekanismerna i JPEG2000 vid denna

typ av feldistribution. Det som talar emot att konfigurera kodaren efter testfall 2, är att avkodaren kraschar vid felaktig avkodning av SOP-markören och tillhörande sekvensnummer. Uppmärksamma samtidigt att paketflyttning implicerar att denna information ska sändas över länkar som kan garantera en viss kvalitetsnivå. Detta har inte testats.

För BER-värde $10^{-4}$ är mönstret för skurfelen tämligen lika för alla hastigheter (se tabell 6.5). Den stora skillnaden gällande längden på skurfelen är för BER-värde $10^{-3}$. Längre skurfel behöver i sig inte betyda att det vållar avkodaren större problem jämfört med kortare skurar. Från testerna framgår det klart att positionen på skurfelet har en mer avgörande roll för slutresultatet.

| BER | Testfall | 3 km/h | 40 km/h | 120 km/h |
|---|---|---|---|---|
| **$10^{-3}$** | **1** | 25,4646 | 26,2519 | 27,9491 |
| | **2** | 29,9545 | 29,7318 | 31,5409 |
| | **3** | 26,1660 | 29,9743 | 30,2787 |
| **$10^{-4}$** | **1** | 37,0873 | 36,0747 | 37,4157 |
| | **2** | 39,1563 | 38,7074 | 39,4303 |
| | **3** | 39,8377 | 39,1334 | 39,4871 |

Tabell 6.6. Medelvärden för kvaliteten på ej felfria bilder.

Sammanfattningsvis kan sägas att mekanismerna bör användas för att bli varse om felaktigheter i bitströmmen. Att använda mekanismerna i syfte att dessutom höja bildkvaliteten är inte att rekommendera. Det kan dock noteras att tendens till högre bildkvalitet uppnås vid förhållanden där skurfel gör sig gällande.

## 6.7 Paketförluster

Bildkodare har normalt inte robusthetsmekanismer som är anpassade för att klara av paketförluster (avser paket på överföringsnivå). Modifieringar av den befintliga bildkodningsstandarden JPEG existerar dock för att komma till rätta med detta problem [11]. JPEG2000 har inga mekanismer som är direkt anpassade för paketförluster. Men på grund av den wavelettransformering som sker, och beroende på hur datan ordnas i dataströmmen, kan ändå godtagbara resultat uppnås vid kraftiga dataförluster. Detta redovisas i följande delkapitel.

### 6.7.1 Simulering av paketförluster

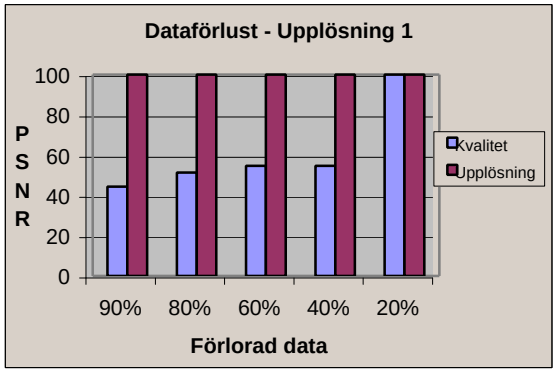
Vid simuleringen av paketförluster är bilden kodad enligt testfall 4. Anledningen till att endast använda mekanismerna paketflyttning och SOP-markörer, är att resterande mekanismer inte är relevanta, då de endast klarar av bitfel. En mer passande benämning av paketförluster i denna testning är dataförluster. En dataförlust innebär att all data, från och med förlustpunkten och till slutet av filen, förloras. Tester har visat att kodaren inte använder datan efter förlusten på rätt sätt, och inte heller ges det några indikationer på hur stora datamängder som är borta. Simuleringen har utförts genom att endast ett antal procent av dataströmmen, som motsvarar ett antal bytes, har använts av avkodaren.

För att återge kvaliteten i ett PSNR-värde används en referensbild på 2 bpp. Referensbilden är den bild som kodaren producerar utan dataförluster men med angiven upplösning. Vid presentation av resultatet har därför ett antagande gjorts: värdet 100 dB används för att representera perfekt återgivelse av bilden. Anledningen är endast för att kunna visualisera resultaten i figurer på ett tillfredsställande vis. Antagandet gäller endast i det följande delkapitlet.

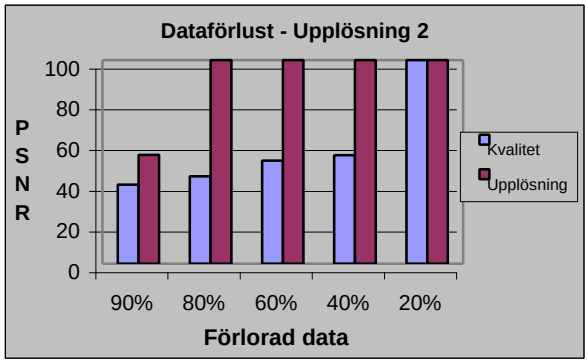
Två progressionslägen har använts: upplösning och kvalitet. Syftet med testningen av dessa val är att se vilket progressionsval som är mer fördelaktigt än det andra. Eftersom relativt stora dataförluster sker redovisas resultaten därför även med avseende på olika upplösningar. Upplösningen är en intressant aspekt på grund av att en mindre bild kan visas med endast en liten datamängd tillgänglig för avkodaren.

### 6.7.2 Testresultat

Resultaten visar att vid låga upplösningsnivåer, nivå 1 och 2, är upplösningsprogression klart bättre. Vid kodning med upplösningsprogression kan avkodaren återskapa perfekta bilder, sånär som vid 90 % förlust på nivå 2. Redan här visar också kodning med kvalietetsprogressionen visuellt bra resultat (se figur 6.47 och 6.48).

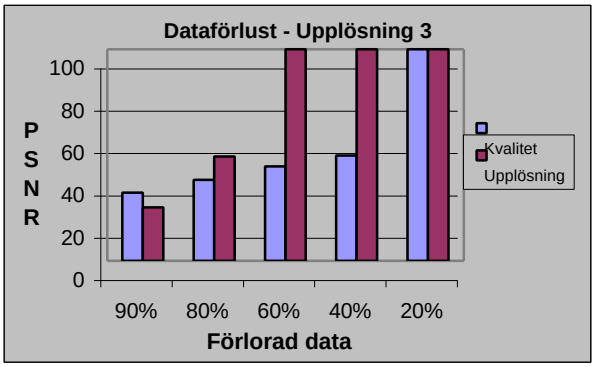



Figur 6.47. Dataförlust vid upplösning 1.

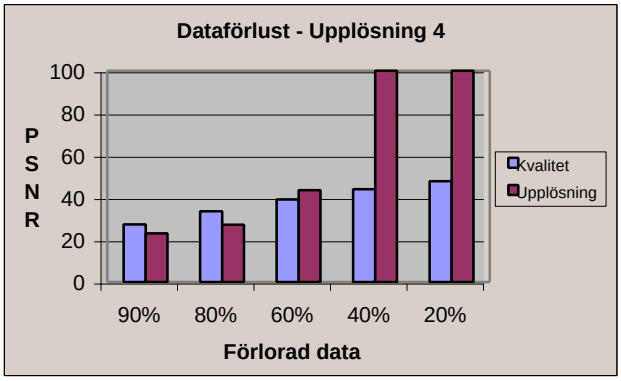



Figur 6.48. Dataförlust vid upplösning 2.

När upplösningsnivån blir högre, och bilden närmar sig originalstorleken, är kvalitetsprogression bättre vid stora dataförluster enligt figur 6.49 och 6.50. Vid lägre förluster är upplösningsprogressionen däremot fortfarande ett bättre val, vilket också är ett förväntat resultat.

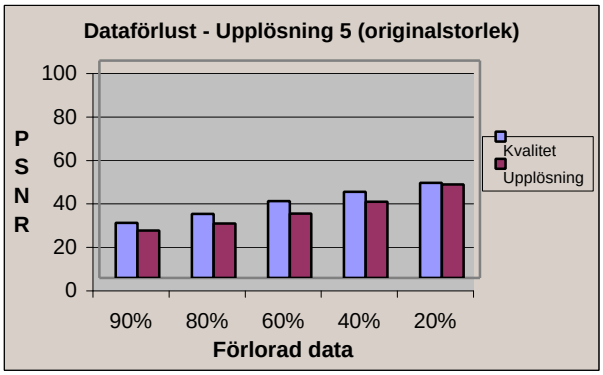



Figur 6.49. Dataförlust vid upplösning 3.




Figur 6.50. Dataförlust vid upplösning 4.

När den avkodade bilden har samma storlek som originalet (se figur 6.51), är kvalitetsprogression bättre vid alla förlustfallen. Då stora dataförluster inträffar har kodaren i upplösningsprogressionsfallet endast tillgång till lägre nivåer, medan kodaren i kvalitetsprogressionsfallet har tillgång till alla nivåer.




Figur 6.51. Dataförlust vid upplösning 5.

Figur 6.52 visar den visuella skillnaden mellan bilder kodade med upplösningsprogression vid upplösning 5 testade med olika dataförluster.

**A.** 90 % förlust. **B.** 80 % förlust. **C.** 60 % förlust.

**D.** 40 % förlust. **E.** 20 % förlust. **F.** inga förluster.

Figur 6.52. Visuell skillnad med upplösningsprogression vid upplösning 5.

Figur 6.53 visar den visuella skillnaden mellan bilder kodade med kvalitetsprogression vid upplösning 5 testade med olika dataförluster.

**A.** 90 % förlust. **B.** 80 % förlust. **C.** 60 % förlust.

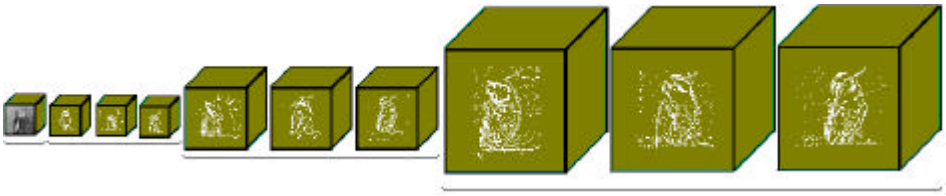
**D.** 40 % förlust. **E.** 20 % förlust. **F.** inga förluster.

Figur 6.53. Visuell skillnad med kvalitetsprogression vid upplösning 5.

### 6.7.3 Sammanställning

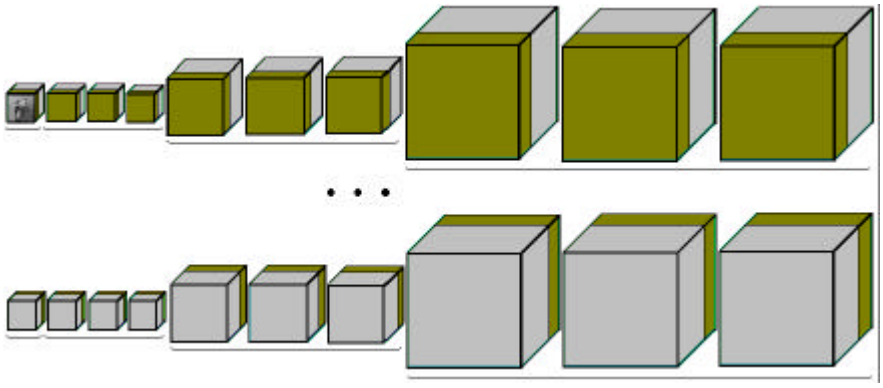
När paketförluster inträffar, spelar antalet förlorade överföringspaket inte någon roll. Vad som däremot spelar roll är var det första paket förloras. Därmed är det också första paketet som avgör hur mycket data det finns kvar för att rekonstruera bilden.

Figur 6.54 och 6.55 tydliggör hur dataströmmen är ordnad vid de två olika progressionsfallen. I figur 6.54 används upplösningsprogression. Vid denna typ av kodning kommer all data från de lägsta upplösningarna komma fram först.



Figur 6.54. Upplösningsprogression.

Vid kvalitetsprogression, som illustreras i figur 6.55, kommer den mest signifikanta datan från varje detaljkoefficient komma fram först. Redan vid små datamängder finns det tillräckligt med data, för att rekonstruera en bild med originalstorlek, vilket också bekräftas av genomförda simuleringar. Bilden har dock fått ge avkall på kvaliteten.



Figur 6.55. Kvalitetsprogression.

Om avkodaren påträffar en paketförlust eller ett bitfel som den inte klarar, kan den försöka att avkoda bilden fram till felet (något som kan utföras manuellt med [16]). Kombinerat med upplösningsprogression kommer det innebära en mindre bild men att dess kvalitet är bra.

Den så kallade SOP-markören används för att hålla synkroniseringen med bitströmmen. Avkodas den inkorrekt kommer avkodaren att avsluta och meddela att bitströmmen innehåller allvarliga fel. Om SOP-markören blir felaktig, finns det ingen algoritm i [16] för att återfå

synkroniseringen med bitströmmen genom att leta efter efterföljande markör. Notera dock att SOP-markörens uppgift är att hjälpa avkodaren så att den kan upptäcka fel. För att minska risken att förlora för mycket data vid en dataförlust på grund av att markörena är för få, finns det möjlighet att lägga in fler SOP-markörer. När bitströmmen består av flera lager och därmed flera paket kommer fler SOP-markörer att finnas. Att lägga in fler utöver de som kommer med paketgränserna har dock ingen positiv påverkan på bildkvaliteten. Då inte en hel upplösningsnivå X finns tillhands för att återskapa bilden med angiven upplösning X, ska heller inte delar av upplösningsnivån, vilket utgörs av subbanden, användas (se kapitel 3). En möjlig väg för att komma runt problemet, skulle kunna vara att ersätta hela subbandet eller delar av det som fattas med nollor. Några tester har dock inte utförts för att stödja det påståendet. Ett tilläg till kodaren [16] för att realisera en resulterande bild skulle kunna vara att då avkodaren kommer ur synkronisering med dataströmmen genereras ett fel, men avkodaren slutar inte att avkoda utan använder den data som finns för att återskapa en bild.

Resultaten från testningen av dataförluster kan sammanfattas i följande två tumregler:

- Upplösningprogression ger skarp liten bild vid små datamängder.
- Kvalitetsprogression ger suddig stor bild vid små datamängder.

Simuleringen visar vidare på att om en fullskalig bild efterfrågas så är kvalitetsprogression att föredra vid paketförluster. Denna typ av progression kan med hjälp av liten datamängd fortfarande rekonstruera en bild med samma upplösning som originalet. Kvaliteten är dock den faktor som blir lidande när endast en liten mängd information finns tillgänglig på mottagarsidan.

Fortsatta intressanta tester är huruvida personer vill göra avkall på bildupplevelsen, beroende på vilken progression bilden kodas med. Föredrar personer bildens originalstorlek med lite sämre kvalitet, eller vill de hellre ha mindre bilder med bättre kvalitet? Faktorer som spelar roll är inte bara bilden i sig utan även det sammanhang bilden används i exempelvis en webbsidas uppbyggnad. En webbsidas upplägg kan förändras i och med att bilderna ändrar storlek. Det ställer då krav på att sidan är testad för de olika fallen. Om en bild drabbas av stora paketförluster, och beroende på vilken progression den är kodad med, skulle avkodaren kunna göra avkall på upplösningsnivån eller kvaliteten på bilden, utifrån användarens krav. Alternativet är att ingen bild blir synlig alls.

# 7 Sammanfattning

Den redovisade bildkodningsstandarden, JPEG2000, gör att användningsområden och applikationer, som exempelvis Internet, digitala kameror, mobila applikationer, medicinska röntgen- och undersökningsbilder och bildarkiv kan kodas genom en och samma standard. På grund av waveletkodning av bilder kan nu en högre kompressionskvot erhållas, jämfört med tidigare bildkodare. I applikationer, som inte har behov av den höga funktionalitet som JPEG2000-standarden erbjuder, kommer den nya standarden troligen inte att konkurrera med den befintliga JPEG. Applikationer, exempelvis vid bildöverföring över nätverk, som bland annat kräver hög bildkvalitet vid låga bitnivåer, kommer dock att kunna dra nytta av JPEG2000. Den nya standarden har som mål att kunna fungera tillfredsställande även över dåliga kanalförhållanden och har därmed utrustats med en viss robusthet. Tester visar att en bildkodare som implementerat standarden, har mekanismer som klarar av att upptäcka bitfel och skurfel i bitströmmen. Mekanismerna bör användas främst för att upptäcka fel i dataströmmen. Mekanismer för att dölja fel existerar också, men ger inte alltid ett positivt genomslag på bildkvaliteten. Vidare tester har även visat att med hjälp av waveletkodningen kan även dataförluster accepteras men att detta kräver avkall på kvaliteten eller upplösningen.

Detta arbete har till stor del inneburit litteraturstudier och inläsning av JPEG2000-standarden. Genomförda tester, som bland annat inneburit att identifiera vilka testfall som ska redovisas och analysen av testresultaten, har också varit en stor del av detta arbete. Testningen av framförallt robusthetsmekanismerna men även kodning av andra egenskaper, som JPEG2000 tillhandahåller, har gett en stor förståelse för standarden. Önskvärt gällande förbättringar av arbetet är en mer ingående redovisning av hur wavelettransformen fungerar.

Förslag på vidare arbete är att vidareutveckla den JPEG2000-kodare som använts i detta arbete, genom att göra den robust även mot dataförluster. I originalutförandet slutar denna kodare att avkoda när den tappar synkroniseringen med SOP-markören och tillhörande sekvensnummer. Någon bild skapas då överhuvudtaget inte. En åtgärd skulle kunna vara att den avslutar avkodningen som tidigare, men att den sedan återskapar en bild av det data den har. Vidare kan utvärdering ske av kodaren för att se hur datoranvändare ställer sig till bilder med dataförluster. I anslutning till dataförluster vore det också intressant att utveckla döljning av förlorade subband och upplösningsnivåer, för att sedan undersöka vad döljningen får för effekter på bildkvaliteten.

# Referenser


[1] Osama K. Al-Shaykh, Iole Moccagatta, Homer Chen. JPEG-2000: *A New Still Image Compression Standard*. Asilomar Conference on Signals, Systems and Computers, Pacific Grove, California, redigerad av Michael B. Matthews, VOL. 1, sidor 99-103, november 1-4, 1998.

[2] Martin Boliek, Charilaos Christopoulos, and Eric Majani. *ISO/IEC JTC1/SC29 WG1, Coding of still pictures, JPEG 2000 Editor Martin Boliek, Co-editors Charilaos Christopoulos, and Eric Majani. JPEG 2000 Part I Final Committee Draft Version 1.0*. 16 mars 2000.

[3] Martin Boliek, Eric Majani, J. Scott Houchin, James Kasner, and Mathias Larsson. *ISO/IEC JTC 1/SC 29/WG 1, Coding of still pictures, JPEG 2000 Editor Martin Boliek, Co-editors Eric Majani, J. Scott Houchin, James Kasner, and Mathias Larsson. JPEG 2000 Part II Committee Draft (revised and corrected)*. Augusti 2000.

[4] Maryline Charrier, Diego Santa Cruz, Mathias Larsson. *JPEG-2000, the next millennium compression standard for still images*. In Proceeding of the IEEE ICMCS´99, 1(2):pp. 131-132, June 7-11, Florence Italy.

[5] Charilaos Christopoulos, Athanasios Skodras, Touradj Ebrahimi. Tutorial - *JPEG2000 The next generation still image coding system*. IEEE Int. Conference on Image Processing (ICIP 99), in Kobe, Japan, 24-28 Oct 99.

[6] Mjukvara Cygwin (Unix-bash-skal under Windows), kommando `cjpeg.exe`, Cygwin, hemsida http://www.cygwin.com, juli 2001.

[7] Touradj Ebrahimi, Bernard Brower, Daniel Lee. *Coding of Still Pictures*. ISO/IEC JTC1/SC29/WG1 (ITU-T SG8), 20th WG1 Arles Meeting Press Release. Arles, juli 2000.

[8] Touradj Ebrahimi, Mathias Larsson, Joel Askelöf, Charilos Cristopoulos. *Region of Intrest Coding in JPEG-2000 for interactive client/server applicaitons*. Proceedings of the IEEE Third Workshop on Multimedia Signal Processing, sidor 389-394, september 13-15, 1999.

[9] Nick Efford. *Digital image processing, a practical introduction using Java™*. Addison Wesley, Pearson Education Limited 2000.

[10] Ericsson, Nya Zeeland, hemsida http://www.ericsson.co.nz/ericsson/home/default.asp, april 2001.

[11] Johan Garcia and Anna Brunström. *A Robust JPEG Coder for a Partially Reliable Transport Service*. Proceedings 7th Int. Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS 2000), October, 2000.

[12] Micheal J. Gormish, Nekka Matsuura och Takahiro Yagashita. *An adaptive transform for compression of mixed con-tone and graphics*. Picture Coding Symposium, Portland, Oregon, May 1999.

[13] Internet FAQ Archives, Internet FAQ Consorttium, hemsida http://www.faqs.org/faqs/jpeg-faq/, november 2000.

[14] ITU - Telecommunications Standadization Sector, Study group 16. Dokument Q11-I-11 och Q11-F-05, 1998.

[15] Mjukvara Jasper (C-implementation), The Jasper project, hemsida http://www.ece.ubc.ca/~mdadams/jasper/, november 2000.

[16] Mjukvara JJ2000 version 4.1 (Java™-implementation), JJ2000, hemsida http://jj2000.epfl.ch/, mars 2001.

[17] JPEG-2000, JPEG kommitté, hemsida http://www.jpeg.org/JPEG2000.htm november 2000.

[18] JPEG-2000 wavelet demo, JPEG kommitté, hemsida http://www.jpeg.org/JPEG2000.htm, januari 2001.

[19] JPEG kommitté, hemsida http://www.jpeg.org/, november 2000.

[20] Requirements ad hoc group. *JPEG2000 requirements and profiles version 6.3*. ISO/IEC JTC 1/SC 29/WG 1 N1803, Convener Dr. Daniel T. Lee, Juli 2000.

[21] Debra A. Lelewer, Daniel S. Hirschberg. *Data compression*. ACM computing surveys, volym 19, nummer 3, september 1987.

[22] Jie Liang, Raj Talluri. *Tools for robust image and video coding in JPEG2000 and MPEG4 standards*. Part of the IS&T/SPIE conference on communications and image processing '99, San Jose, California, januari 1999.

[23] Luratech, hemsida http://www.luratech.com, december 2000.

[24] Michael W. Marcellin, Michael J. Gormish, Ali Bilgin, Martin P. Boliek. *An Overview of JPEG-2000*. Proceedings of IEEE Data Compression Conference, sidor 523-541, 2000.

[25] Iole Moccagatta, Salma Soudagar, Jie Liang and Homer Chen. *Error-resilient coding in JPEG-2000 and MPEG-4*. IEEE journal on selected areas in communications, vol. 18. No. 6, juni 2000.

[26] Michel Mouly, Marie-Bernadette Paulet. The GSM System for Mobile Communications. Cell & Sys. Correspondence 1992. ISBN 0-945592-15-9.

[27] Stefan Nilsson, Patrik Jacobson. *Data compression using the wavelet transform*. Högskolan i Luleå, master thesis, april 1998.

[28] Ricoh Silicon Valley, hemsida http://www.rsv.ricoh.com/aboutus/crc/crc_crew.html, november 2000.

[29] D. Santa-Cruz, T. Ebrahimi, J. Askelöf, M. Larsson och C. A. Christopoulos. *JPEG 2000 still image coding versus other standards*. Signal processing laboratory - Swiss federal institute of technology, Switzerland, Ericsson research, Sweden, framtida publicering i proceedings of SPIE vol. 4115, 2000.

[30] A. N. Skodras, C. A. Chrisopoulos, T. Ebrahimi. JPEG2000: *The upcoming still image compression standard*. Proceedings of the 11th Portuguese Conference on Pattern Recognition, Porto, Portugal, Maj 11-12, sidor 359-366, 2000.

[31] Eric J. Stollnitz, Tony D. Derose, David H. Salesin. *Wavelets for computer graphics, theory and applications*. Morgan Kaufmann Publishers, 1996.

[32] David Taubman. *High performance scalable image compression with EBCOT*. IEEE Trans. Image Proc., mars 1999.

[33] Pankaj N. Topiwala. *Wavelet image and video compression*. Kluwer Academic Publishers, 1998.

[34] Simple wavelet transform generator wavelet demo, Worcester Polytechnic Institute Computer Science Department, hemsida http://www.cs.wpi.edu/~matt/courses/cs563/talks/Wavelet_Presentation/java/wavelet-demo.html januari 2001.

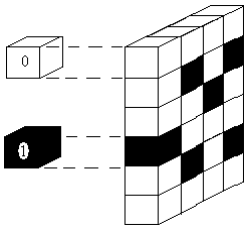
# Förkortningar

| | |
|---|---|
| **BER** | Bit Error Rate |
| **bpp** | Bits Per Pixel |
| **dB** | Decibel |
| **DCT** | Discrete Cosine Transform |
| **EOC** | End Of Codestream |
| **gif** | Graphic Interchange Format |
| **IS** | International Standard |
| **JPEG** | Joint Photographic Experts Group |
| **JPEG2000** | Joint Photographic Experts Group 2000 |
| **JPEG-LS** | Joint Photographic Experts Group Lossless |
| **LWF** | LuraWave Format |
| **MS** | Mest signifikanta |
| **LS** | Minst signifikanta |
| **PDA** | Portable Digital Assistent |
| **pgm** | Portable Grey Map |
| **ppm** | Portable Pix Map |
| **PSNR** | Peak Signal-to-Noise Ratio |
| **RCT** | Reversible Component Transform |
| **RGB** | Färgmodell, Red Green Blue |
| **SOP** | Start Of Packet |
| **VM** | Verifikationsmodell - mjukvara |
| **WCDMA** | Wideband Code Division Multiple Access |
| **$YC_bC_r$** | Färgmodell, Y luminans, $C_bC_r$ färgkomponenter |

# A Allmänt om bildkodning

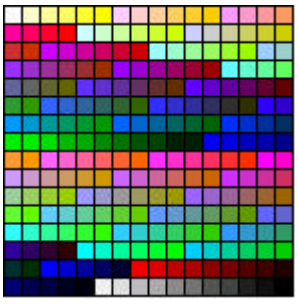
## A.1 Representation av en bild

Det finns olika sätt att presentera en bild på datorskärmen. På skärmen är bilden uppbyggd av pixlar (färgpunkter). Varje pixel kan vara svart eller vit eller någon färg. Beroende på vilket färgdjup en pixel har, kräver den också olika mycket minne. En svartvit bild kan representeras av en bit, 0 för vit och 1 för svart (se figur A.1).




Figur A.1. Bitmap.

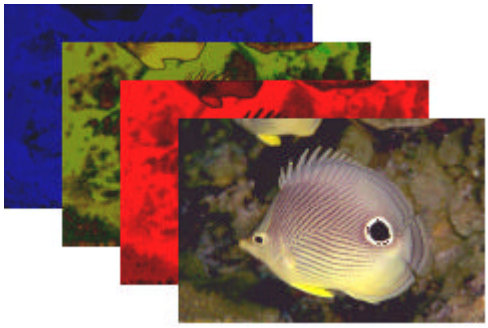
När en bild representeras av som gråskala krävs det mer minne. Den vanligaste lösningen är att varje pixel lagras i en byte. Vit motsvaras där av värdet 0, svart av värdet 255 och gråskalorna av värdena 1 till 254. Även enkla färgbilder kan representeras genom att varje pixel lagras som en byte. Värdet på pixeln måste då mappas mot en färgkarta (se figur A.2) med maximalt 256 färger (principen används i .gif-formatet).



Figur A.2. Färgkarta.

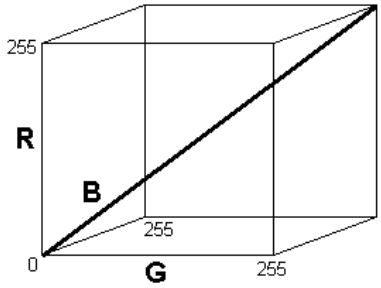
Kan mer minne allokeras för en pixel kan också fler färger visas. En stor del av det för ögat synliga färgspektrumet kan återges genom att blanda tre olika färgkomponenter, röd, grön och blå. Denna modell kallas RGB och varje pixel i RGB-formatet lagras med 24 bitar, en byte för

röd, en för grön och en för blå färg. Genom att dela upp bildens färger i dessa tre färglager kan mycket naturtrogna bilder presenteras på en datorskärm (se figur A.3).



Figur A.3. RGB-format.

Gråskaliga bilder kan fortfarande presenteras med denna modell, då dessa uppnås när den röda, gröna och blåa färgen antar samma värde. Den tjocka linjen i figur A.4 nedan, illustrerar de gråa färgerna, där vit är i origo och svart i (255, 255, 255).




Figur A.4. Representation av gråskaliga bilder genom RGB-formatet.

Ytterligare ett vanligt färgbildsformat är $YC_bC_r$ [9]. Det används för att dela upp en bild i en ljuskomponent Y (luminans) och två färgkomponenter $C_b$ och $C_r$ (se figur A.5). Till skillnad från RGB, lagrar detta format färginformationen och intensiteten separat. Därav kan en av dessa informationskällor ändras utan att påverka den andra. Denna typ av uppdelning har visat sig mer användbar i bildkodning.

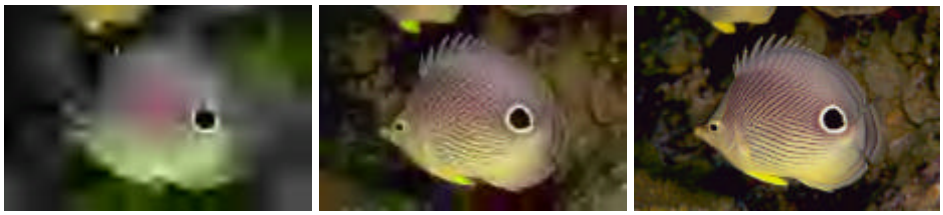
**Y** **$C_b$** **$C_r$**

Figur A.5. $YC_bC_r$ –format.

## A.2 Progressivitet

Den vanligaste formen av bildvisualisering i Internet-sammanhang är sekventiell framställning av bilden. Så fort all bildinformation har sänts över till mottagaren kan bilden presenteras i webbläsaren. I många fall visas bildens övre del först, för att sedan byggas på nedåt då mer data kommer fram. Det beror också på vilket format bilden är kodad i.

Det finns två sorters progressiva framställningar i JPEG2000 [4]. I dessa progressiva lägen, visas alltid hela bilden i någon form. Vanligast förekommande i Internetsammanhang är progressivitet med avseende på bildkvalitet. Användaren får då en bild med låg kvalitet presenterad framför sig. Sedan ökar kvaliteten allt eftersom mer bildinformation kommer fram (se figur A.6).



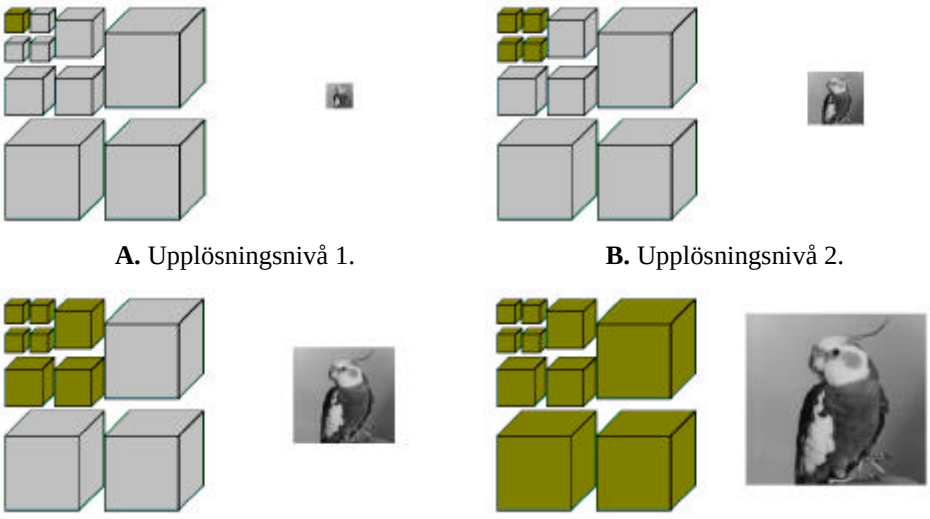
Figur A.6. Progressivt läge med avseende på kvalitet.

Den andra formen av progressivitet är den som avser upplösning av en bild. En användare kommer först att se en liten bild (liten upplösning) som sedan ökar i storlek då mer information sänds över till mottagaren (se figur A.7). Bilder med liten upplösning skalas dock oftast så att dimensionerna på bilden blir samma som originalet.

Figur A.7. Progressivt läge med avseende på upplösning.

Båda progressionsvarianterna är möjliga med JPEG2000. Skillnaden mellan lägena är beroende på i vilken ordning den kodade bilddatan skrivs till kodströmmen (datafilen). Då bilder kodas med en JPEG2000-kodare kommer kodarens wavelettransform att generera subband. Ett subband innehåller information om den kodade bilden (se kapitel 3). Subbanden i sin tur möjliggör progressiv bildvisning då kodströmmen är organiserad så att de mindre subbanden (subband som befinner sig på en lägre upplösningsnivå) förekommer tidigare än

vad de större subbanden gör (subband som befinner sig på en högre upplösningsnivå). Upplösningsprogression följer naturligt av kodaren. Resonemanget illustreras med figur A.8 [5].

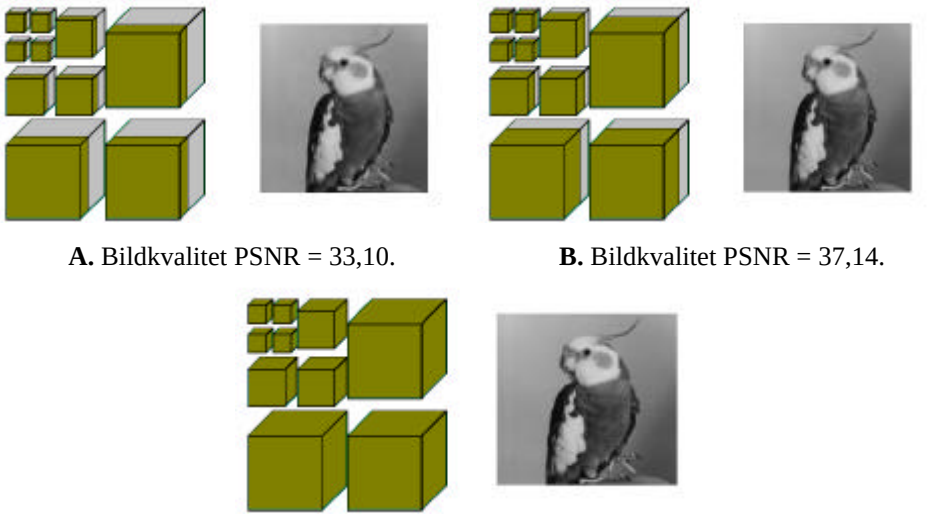


**A.** Upplösningsnivå 1. **B.** Upplösningsnivå 2.

**C.** Upplösningsnivå 3. **D.** Upplösningsnivå 4.

Figur A.8. Progressivt bildläge med avseende på upplösning.

På motsvarande sätt kan också den andra typen av progressivitet realiseras. I detta fall kommer alla subbanden att förekomma i början av kodströmmen. Men subbanden kommer inte vara representerade med all sin information i början av kodströmmen, utan endast med den mest signifikanta informationen. Resterande bilddata förekommer senare i kodströmmen. Kodningen blir mer beräkningsintensiv då denna typ av kvalitetsskalning används. Resonemanget illustreras med figur A.9 [5].



**A.** Bildkvalitet PSNR = 33,10. **B.** Bildkvalitet PSNR = 37,14.

**C.** Bildkvalitet PSNR = - (original bild).

Figur A.9. Progressivt bildläge med avseende på bildkvalitet (PSNR).

Genom att en bild är en kontinuerlig dataström, kommer en Internet-användare att genast kunna se en lågt upplöst version av bilden, eller en bild med lite sämre kvalitet beroende på vilken progression bilden kodats med. Rent praktiskt kan nedladdningen avbrytas, om en viss nivå av upplösning eller kvalitet har uppnåtts. Som användare skulle det senare finns möjlighet att öka kvaliteten eller upplösningen, genom att ladda hem ytterligare information.

# B Formler och begrepp

## B.1 Kompressionseffektivitet

En bildkodares kompressionseffektivitet mäts både för förlustfri och förlustkompression. Det är framförallt tre mätningar som är intressanta att utföra vid bildkomprimeringen. Först utreds hur mycket bilden har komprimerats i jämförelse med originalbilden. Den s.k. kompressionskvoten ges genom division av ursprungsbildens och den komprimerade bildens filstorlekar:

$$Kompressionskvot = \frac{gammal - filstorlek}{ny - filstorlek}$$

En bild med kompressionskvot 10:1 är alltså 10 gånger mindre än originalbilden.

Den andra mätningen för att utvärdera kompressionseffektiviteten talar om hur många bitar varje pixel representeras av i genomsnitt i den komprimerade bilden. Mätvärdet uttrycks i bit per pixel (bpp).

Den slutliga mätningen visar hur mycket information som har förlorats för att erhålla den uppnådda kompressionen (gäller ej den förlustfria komprimeringen). Följande mätning är även intressant i miljöer, som innebär att bit- eller bytefel införs i datan. Om den återskapade bilden inte är lika med originalbilden är det intressant att mäta bildkvaliteten på bilden. Root Mean Square Error (RMSE) bygger på jämförelsen mellan originalsignalen och den komprimerade signalen och används för att få fram Peak Signal Noise Ratio (PSNR) [29]:

$$RMSE = \sqrt{\frac{\sum_{n=1}^{w*h} (a[n] - a'[n])^2}{w*h}}, \; n = 1, 2, \ldots, w*h$$

$$PSNR = 20 \log_{10} \frac{255}{RMSE}$$

där *w* och *h* är vidden och höjden på bilden och *a* och *a'* är de två bilderna som jämförs. Ett riktmärke för en bra förlustkomprimerad bild är när PSNR antar värden runt 40 dB eller högre. Det kan då vara svårt att med blotta ögat se en skillnad mellan originalbilden och den förlustkomprimerade bilden. Men även om PSNR-värdet är högt för en bild svarar det inte i alla fall till en bild med god visuell kvalitet. Därför kan en subjektiv bedömning vara nödvändig (se figur B.1). När PSNR-värdet blir lågt, implicerar det en försämrad bildkvalitet. Formeln är inte användbar vid förlustfri kompression då värdet på RMSE antar 0 och PSNR är då inte definierat. PSNR-värden uppmätta i denna uppsats är utförda med verktyget LuraWave SmartCompress [23].

**A.** PSNR=32,11 bpp=0.5. **B.** PSNR=25,23 bpp=0.125.

Figur B.1. Bilder med olika PSNR och bpp-nivåer.

Ytterligare ett mätvärde för komprimeringseffektivitet är entropi. Entropin uttrycker det minsta antalet bitar, som krävs för att koda en symbol med en given komprimeringsmodell. Entropin används framförallt för att mäta en komprimeringsalgoritms effektivitet och är inte direkt knuten till bildkomprimering.

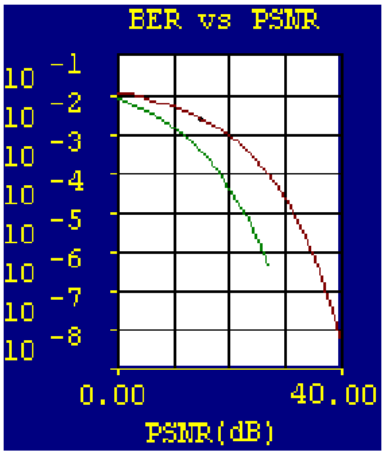
## B.2 Bit Error Rate

I telekommunikationsöverföringar är Bit Error Rate (BER ~ bitfelstakt) definierat som kvoten mellan antalet felaktiga bitar, och det totala antalet överförda bitar. En dataöverföring kan exempelvis ge ett BER-värde av $10^{-6}$. Det betyder att av 1000000 bitar är en felaktig.

BER är en indikation på hur ofta ett paket eller andra dataenheter, måste omsändas på grund av ett fel. Ett för högt BER-värde kan indikera att en lägre dataöverföringshastighet är att föredra, för att förbättra den sammanlagda överföringshastigheten för en given datamängd. BER-värdet kan då reduceras, vilket innebär att färre paket behöver sändas om.

I telekommunikationsöverföringar ger BER även ett uttryck för effektiviteten av en robusthetsmekanism. Ju fler fel som kan tillåtas i den mottagna datan för en robustetsmekanism, utan att datan blir skadad eller avkodningen misslyckas, desto bättre är mekanismens förmåga att hantera felet.

Eftersom bitfelen ändrar bitarna i bitströmmen, är BER kopplat till PSNR (för exempel se figur B.2). När fler fel inträffar i datan, och därmed större BER, kommer PSNR-värdet (bildkvaliteten i bildsammanhang) att påverkas negativt.




Figur B.2. - Koppling mellan BER och PSNR.

# C Exempel på tillämpning av .jpx-formatet

Exempel-scenariot hämtat från [20].

*"Jag tog en bild av John i Frankrike. Bilden har ett slott i bakgrunden. John vill nu e-posta en förbättrad version av JPEG2000-bilden till sin bror Tom. John har bestämt att han ska införa extra information i två regioner av bilden. Regionerna begränsas med två rektanglar och den första rektangel ritar John runt sitt eget ansikte, och den andra runt slottet. Genom att John använder sig av en passande applikation, kan han integrera ett ljudklipp i bilden och knyta en URL till sin privata hemsida till rektangeln som omringar honom själv på bilden. Ljudklippet är ett personligt meddelande till hans bror Tom, och Johns hemsida innehåller mer bilder från Frankrike. John lägger även till en text, som innehåller historien om slottet, i rektangeln runt slottet på bilden."*

När Tom tar emot ovanstående bild följer en sekvens av händelser i den applikation som visar bilden:

| | |
|---|---|
| Händelse 1: | Applikationen avkodar och visar bilden. |
| Händelse 2: | Tom upptäcker de två regionerna i bilden som innehåller information från John. |
| Händelse 3: | Genom att utforska bildens regioner (de synliga rektanglarna som John skapade) upptäcker Tom vilken typ av information bilden innehåller (i detta fall ljud och en URL). |
| Händelse 4: | Tom väljer sedan från sitt eget tycke och smak vilken information han vill se. |

# D Originalbilder

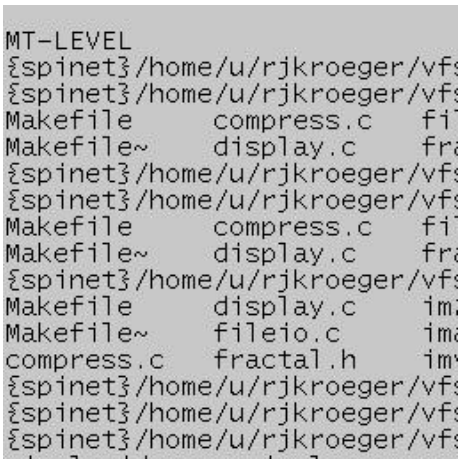
**1.**

**2.**

**3.**

**4.**

**5.**




**6.**

**7.**

**1.** Originalbild lena2.ppm - storlek 256 * 256, 196623 bytes, 24 bitar / pixel.

**2.** Originalbild lena2.pgm - storlek 256 * 256, 65551 bytes, 8 bitar / pixel.

**3.** Originalbild bike-d.ppm - storlek 512 * 512, 786506 bytes, 24 bitar / pixel.

**4.** Originalbild goldhill.pgm - storlek 256 * 256, 65598 bytes, 8 bitar / pixel.

**5.** Originalbild bird.gif - storlek 256 * 256, 47516 bytes, 8 bitar / pixel.

**6.** Originalbild text.gif - storlek 256 * 256, 5121 bytes, 8 bitar / pixel.

**7.** Originalbild fisch.lwf - storlek 384 * 256, 294912 bytes, 24 bitar / pixel.

Figur D.1. Originalbilder.