

TEKNISK RAPPORT

EDGY – Edge Detection Group of Y

Erik Mellström

Version 1.0

21 maj 2007

Status

Granskad		
Godkänd		

PROJEKTIDENTITET

Medicinsk bildanalys, VT 2007

Linköpings tekniska högskola, Institutionen för Systemteknik

Namn	Ansvar	Telefon	Student-ID
Per Thyr	Projektleddare	073-9846611	perth945
Erik Mellström	Dokumentansvarig	073-9095229	erime801
Alexander Tuttle	Designansvarig	070-4323836	aletu996
Henrik Wolkesson	Kvalitetsansvarig	073-0221415	henwo442
Erik Ringaby	Testansvarig	070-2752821	eriri062

E-postlista för hela gruppen: edgy@googlegroups.com**Kund:** R&D, Context Vision AB, Storgatan 24, 582 23 Linköping,
tel: 013-358550, fax: 013-101902, info@contextvision.se**Kontaktperson:** Gunnar Farneback, 013-358552, gunnar.farneback@contextvision.se**Kursansvarig:** Klas Nordberg, 013-281634, klas@isy.liu.se**Handledare:** Johan Wiklund, 013-281359, jowi@isy.liu.se

Innehåll

1	Inledning	6
1.1	Testdata	6
2	Översikt av systemet	7
2.1	Huvudprogram	7
2.2	Delsystem 1 (GUI)	7
2.3	Delsystem 2-4	7
2.4	Indata	8
3	Delsystem 1: GUI – Graphical User Interface	9
3.1	Översikt av delsystemet	9
3.2	Implementering	9
3.3	Att använda GUI	10
3.3.1	Ange inbild	10
3.3.2	Ange <i>Stick direction</i>	11
3.3.3	Välja om <i>Scan conversion</i> ska utföras	11
3.3.4	Välja algoritm	12
3.3.5	Begära hjälp om vald algoritm	12
3.3.6	Ställa in användarparametrar	12
3.3.7	Kalla på vald algoritm	13
3.3.8	Utvärdera resultatet	13
3.4	Felhantering	13
4	Delsystem 2: Anisotrop diffusion	16
4.1	Översikt av delsystemet	16
4.2	Kort om teorin	16
4.3	Blockschema och blockbeskrivning	16
4.3.1	Block 1 - Indata	18
4.3.2	Block 2 - Användarparametrar	18
4.3.3	Block 3 - Derivering	18
4.3.4	Block 4 - Strukturtensor	20
4.3.5	Block 5 - LP-filtrer	20
4.3.6	Block 6 - Egenvärdesanalys	21
4.3.7	Block 7 - Viktning	22
4.3.8	Block 8 - Diffusionstensor	23
4.3.9	Block 9 - Diffusionsekvationen	24
4.3.10	Block 10 - Iterering	26
4.3.11	Block 11 - Utdata	26
4.4	Resultat	26
4.4.1	Tidsåtgång	26
4.4.2	Bildförbättring	27
5	Delsystem 3: Wavelets	30
5.1	Översikt av delsystemet	30
5.2	Brusmodellen	30
5.3	Wavelettransform	30
5.3.1	Teori	30
5.3.2	Implementering	32

5.4	Brusreducering (DeN)	32
5.4.1	Teori	32
5.4.2	Implementering	34
5.5	Hård tröskling och kantförstärkning (Enh)	35
5.5.1	Teori	35
5.5.2	Implementering	36
5.6	Beskrivning av användarparametrar	36
5.7	Resultat	37
5.7.1	Exekveringstid	37
5.7.2	Bildförbättring	38
6	Delsystem 4: Adaptiv medianvärdesfiltrering	39
6.1	Översikt av delsystemet	39
6.1.1	Block 1 - Val av indata	39
6.1.2	Block 2 - Användarparametrar	40
6.1.3	Block 3 - Lokalt medelvärde	41
6.1.4	Block 4 - Lokal varians	42
6.1.5	Block 5 - Viktkoefficienter	43
6.1.6	Block 6 - Gråskalehistogram	44
6.1.7	Block 7 - Medianberäkning	46
6.1.8	Block 8 - Utdata	46
6.2	Kantproblem	47
6.3	Iteration	47
6.4	Resultat	47
	Referenser	50
A	Licens	51
B	Kod	52
B.1	Kod för delsystem 1: GUI	52
B.1.1	btn_run_Callback() i EDGY.m	52
B.1.2	DoResult() i EDGY.m	54
B.1.3	PolarToCartesian()	56
B.2	Kod för delsystem 2: Anisotrop diffusion	58
B.2.1	AnisotropicDiffusion.m	58
B.2.2	createDerivKernel.m	61
B.2.3	createGaussKernel.m	62
B.2.4	fastD.c	63
B.2.5	solveAOSM.m	66
B.2.6	solveAOSN.m	67
B.2.7	divergenceC.c	68
B.3	Kod för delsystem 3: Wavelets	71
B.3.1	multiScaleEnh.m	71
B.3.2	mxSimpleDiff.c	77
B.4	Kod för delsystem 4: Adaptiv medianvärdesfiltrering	80
C	Tidsanalys	83
C.1	Tidsanalys för delsystem 2	84
C.2	Tidsanalys för delsystem 3	88

C.2.1	Med sampling	88
C.2.2	Utan sampling	96
C.3	Tidsanalys för delsystem 4	104
D	Tidsanalys utan C-implementering	109
D.1	Tidsanalys för delsystem 2 utan C-implementering	109
D.2	Tidsanalys för delsystem 3 utan C-implementering	113
D.2.1	Med sampling	113
D.2.2	Utan sampling	121

DOKUMENTHISTORIK

Version	Datum	Utförda förändringar	Utförda av	Granskad
0.1	2007-03-22	Mall	EM	PT
0.2	2007-05-09	Första utkast	Alla	Alla
0.3	2007-05-10	Andra utkast	Alla	PT
0.4	2007-05-20	Ändrat testbilder	Alla	EM
1.0	2007-05-21	Lagt till info om testdata	EM	PT

1 Inledning

Det här dokumentet utgör en teknisk dokumentation av EDGY Ultrasound Image Enhancer, som i huvudsak utgörs av implementeringar av tre bildförbättringsalgoritmer för ultraljudsbilder. Produkten är utvecklad av fem studenter vid Linköpings universitet i samarbete med ContextVision AB. För mer information, se projektidentiteten på sidan 1.

Dokumentet inkluderar övergripande information om systemets struktur och detaljinformation om hur användargränssnittet och algoritmerna är implementerade. För att lättare kunna följa diskussionerna i rapporten bifogas koden i appendix.

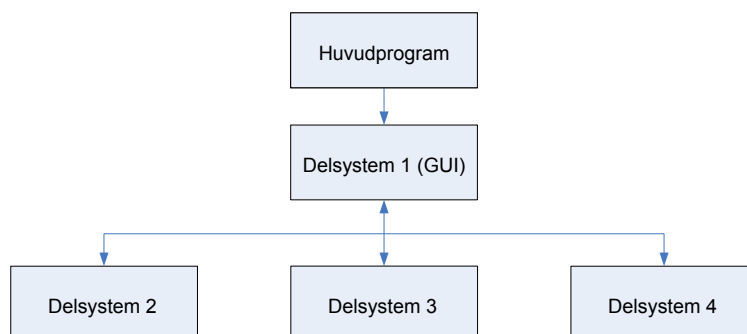
1.1 Testdata

Projektgruppen EDGY har inte upphovsrätt till de ultraljudsbilder som användes under utvecklingen och utvärderingen av produkten. På grund av upphovsrättsskäl används i denna rapport i stället en annan ultraljudsbild, vars licensiering medger fri spridning. Denna bild är i projektgruppens mening inte lika bra som testdata.

Överensstämmelsen mellan texten och figurerna i den här rapporten är därför inte alltid perfekt. Detta medför att vissa resultat och analyser kan vara svårtolkade. Beställaren [6] har tillgång till de ursprungliga bilderna och kan återskapa de i dokumentet nämnda exemplen.

2 Översikt av systemet

Systemet har huvudsakligen implementerats som ett program i Matlab och har en struktur enligt figur 1. Användaren kan interagera med systemet genom att antingen starta GUI genom huvudprogrammet eller genom att kalla direkt på en algoritm från Matlab.



Figur 1: Systemöversikt.

2.1 Huvudprogram

Huvudprogrammet anropas i Matlab genom att i rotkatalogen skriva `startEDGY`. Kommandot laddar applikationen och startar GUI (*Graphical User Interface*).

2.2 Delsystem 1 (GUI)

I användargränssnittet kan användaren bland annat göra följande:

- Välja inbild.
- Välja algoritm.
- Ställa in användarparametrar.
- Kalla på vald algoritm.
- Utvärdera resultatet.

2.3 Delsystem 2-4

Vart och ett av dessa delsystem utgör en självständig funktion i Matlab som implementerar var sin bildförbättringsmetod. Algoritmerna fungerar alltså även utan GUI. Implementeringarna är skrivna i Matlab och i C. De funktioner som skrivs i C kopplas till Matlab som *Matlab executables* (MEX-funktioner). Funktionerna tar emot en inbild och användarparametrar och returnerar en filtrerad bild samt tidsåtgång, enligt mönstret nedan.


```
[outImage, elapsedTime] = algorithmName(inImage, param1, param2, param3)
```

Delsystem 1 implementerar anisotrop diffusion, som bygger på att diffusion tillåts ske i bilden för att reducera brus, samtidigt som kanter detekteras och diffusion över dessa stoppas.

Delsystem 2 förutsätter att specklet i huvudsak är multiplikativt och utgår från olinjär bearbetning av waveletkoefficienter i en filterbank för att filtrera bilden.

Delsystem 3 använder sig av adaptiv medianvärdesfiltrering.

2.4 Indata

När man diskuterar bildflödet vid ultraljud talar man ofta om två typer av dataformat. Det ena är rådata (eng *raw lines* eller *sticks*) som är det format på data från ultraljudsutrustningen innan den har mappats om för att visas på skärm. Detta dataformat beskrivs med polära koordinater på grund av den tekniska utrustningens konstruktion. Det andra är scankonverterad data som är det ommappade resultatet av rådatan. Man kan beskriva mappningen som en transformation från polära till kartesiska koordinater. Ett annat namn för denna typ av data är formaterad data.

Enligt författarna till [1] finns det två stora fördelar med att använda rådata från en ultraljudsbild. Den ena är att man får ett noggrannare resultat då data beskrivs med samma typ av koordinatsystem som ultraljudsproben. Den andra är att man kan få upp till 3 gånger snabbare algoritmer, då man har färre pixlar som behöver behandlas. Även ContextVision [6] föredrar, av liknande skäl, att behandla rådata.

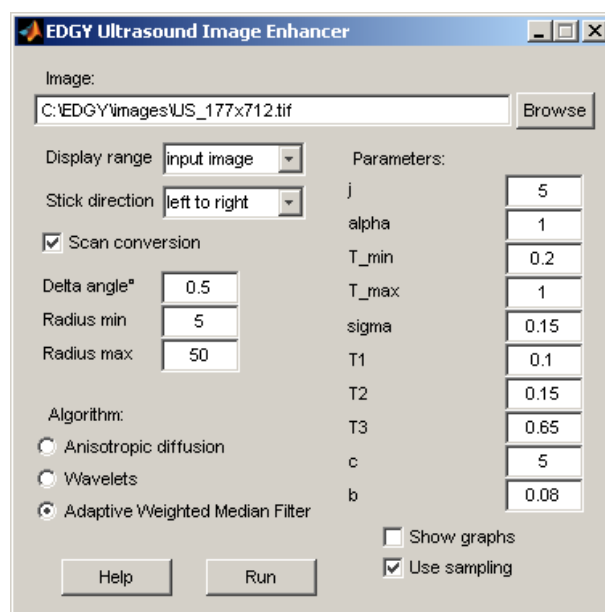
3 Delsystem 1: GUI – Graphical User Interface

Vid evaluering av algoritmerna är det viktigt att det går att få en överblick över tillgängliga användarparametrar för respektive algoritm. För att underlätta för användaren finns ett grafiskt verktyg där dessa kan justeras. Parameterinställningar och algoritmer kan jämföras med varandra med avseende på bildkvalitet och tidsåtgång.

3.1 Översikt av delsystemet

Produkten innehåller ett GUI, skrivet i Matlab. Det är kopplat till de olika algoritmerna, vilka anropas i form av Matlabfunktioner. Genom att justera parametrar i GUI kan användaren påverka funktionernas inargument. Den filtrerade bilden och tidsåtgången returneras till GUI och presenteras för användaren i ett resultatfönster. GUI ser ut som i figur 2.

GUI startas från huvudprogrammet, se avsnitt 2.1.



Figur 2: I GUI kan användaren på ett smidigt sätt evaluera algoritmerna.

3.2 Implementering

De olika algoritmerna är skrivna i Matlab och därför är även GUI skapat i Matlab. Vårt system ska inte kopplas ihop med något externt system utan körs självständigt direkt från Matlab. Matlab har ett grafiskt verktyg för att skapa GUI. Detta anropas med kommandot `guide()`. Där går det att lägga in nya objekt genom dra-och-släpp-metoden från ett verktygsfält. GUI sparas som en fig-fil (`gui/EDGY.fig`) som innehåller den mesta informationen om objekten, samt en m-fil (`gui/EDGY.m`) som innehåller all nödvändig kod. Det går även att göra inställningar för objekten i m-filen, såsom position och färg. Det viktigaste i m-filen är dock de funktioner

som körs när användaren interagerar med GUI. När man lägger in nya objekt i GUI skapas automatiskt funktioner för de flesta objekten och kan till exempel se ut enligt:

```
% — Executes on button press in pushbutton1.  
function pushbutton1_Callback(hObject, eventdata, handles)  
% hObject    handle to pushbutton1 (see GCBO)  
% eventdata  reserved — to be defined in a future version of MATLAB  
% handles    structure with handles and user data (see GUIDATA)
```

Objekten är uppbyggda genom *handles* och man erhåller till exempel data från en textinmatningsruta genom:

```
get(handles.textBoxID, 'String')
```

För att ställa in vad den ska innehålla användes `set()` istället.

3.3 Att använda GUI

Användargränssnittets uppbyggnad förstås enklast om beskrivningen utgår från användarens synvinkel. GUI i princip indelat i sex delar:

- Ange inbild.
- Ange *Stick direction*.
- Välja om *Scan conversion* ska utföras.
- Välja algoritm.
- Begära hjälp om vald algoritm.
- Ställa in användarparametrar.
- Kalla på vald algoritm.
- Utvärdera resultatet.

Var och en av dessa beskrivs noggrannare nedan.

3.3.1 Ange inbild

Ett bra matlabkommando för att erhålla sökvägen till en fil från en dialogruta är:

```
[filename, fpath] = uigetfile('*..*', 'Open');
```

När användaren klickar på *Browse*-knappen i GUI körs denna rad. Den valda filens sökväg ligger kvar i textrutan och kan läsas när det är dags att köra algoritmen. Sökvägen skickas då som inargument till Matlabs `imread()`, som kan läsa de vanligaste bildfilformaten. Systemet klarar av att hantera alla sådana bildfiler, där färgdatan i filen är minst 2 bitar och högst 8 bitar per pixel. Anledningen är att `imread()` då returnerar data i formatet `UINT8`, som övriga delsystem stödjer.

3.3.2 Ange *Stick direction*

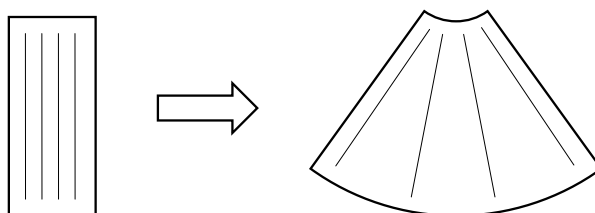
Stick direction måste anges om *Scan conversion* (se nedan) ska användas, annars är det valfritt. Anledningen är att *Scan conversion* förutsätter att probens placering är högst upp i bilden för att transformationen ska bli korrekt. Bilden roteras alltid så att *Stick direction* blir uppifrån och ned. Bildens rotation kan vara av intresse för användaren när resultatet ska beskådas, även om inte *Scan conversion* har använts.

Möjligen skulle en specificerad ultraljudsalgoritm kunna utnyttja det faktum att den vet i vilken riktning ljudpulserna har fortplantat sig vid bildkonstruktionen, men ingen av våra algoritmer är anpassade för detta.

3.3.3 Välja om *Scan conversion* ska utföras

Scan conversion innebär att bilden efter filtrering transformeras från polär representation till en kartesisk motsvarighet som ger en geometriskt korrekt beskrivning, se figur 3. De parameter-värden som beskriver hur data är insamlat och som krävs vid rekonstruktionen är:

- *Delta angle* – antalet grader mellan två intilliggande *sticks*.
- *Radius min* – antalet längdenheter till mätvärdet närmast proben.
- *Radius max* – antalet längdenheter till mätvärdet längst bort från proben.



Figur 3: *Scan conversion* innebär att polärt representerad data transformeras till en kartesisk, geometrisk korrekt, bild. De tunna linjerna illustrerar hur *sticks* transformeras.

Den funktion som utför transformationen är separerad från GUI och ligger i `gui/PolarToCartesian.m`. Funktionsanropet ser ut som nedan. De fyra första inargumenten har beskrivits ovan. Det femte, `method`, anger vilken typ av interpolation som ska ske. Vi använder linjär interpolation, men detta går alltså att ändra i `gui/EDGY.m` där funktionsanropet sker. Koden, inklusive noggrannare specifikation av inargumenten, hittas i avsnitt B.1.3.

```
cartesian = PolarToCartesian(polar, r0, r1, v0deg, method)
```

Observera att `v0deg` i funktionsanropet ska ange vinkeln mellan de två yttersta kolumnerna i datat (i GUI anges vinkeln mellan två intilliggande *sticks*).

3.3.4 Välja algoritm

Så kallade *radio buttons* skapas enklast genom att lägga dem i en *button group* eftersom Matlab då ser till att övriga alternativ automatiskt avmarkeras när ett nytt alternativ väljs. Följande kod sätter variabeln `algorithm` till den valda algoritmen och gör tillhörande användarparametrar synliga.

```
function uipanel_algorithm_SelectionChangeFcn(hObject, eventdata, handles)
global algorithm;
% hObject    handle to uipanel_algorithm (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
switch hObject
case handles.rdb_diffusion
    algorithm = 'diffusion';
    set(handles.uipanel_par_wavelet, 'Visible', 'off');
    set(handles.uipanel_par_median, 'Visible', 'off');
    set(handles.uipanel_par_diffusion, 'Visible', 'on');
case handles.rdb_wavelet
    algorithm = 'wavelet';
    set(handles.uipanel_par_median, 'Visible', 'off');
    set(handles.uipanel_par_diffusion, 'Visible', 'off');
    set(handles.uipanel_par_wavelet, 'Visible', 'on');
otherwise
    algorithm = 'median';
    set(handles.uipanel_par_diffusion, 'Visible', 'off');
    set(handles.uipanel_par_wavelet, 'Visible', 'off');
    set(handles.uipanel_par_median, 'Visible', 'on');
end
```

3.3.5 Begära hjälp om vald algoritm

Help-knappen hämtar den hjälptext som genereras då vald algoritm kallas på med `help algorithmName` och visar denna i en dialogruta. På så sätt kan användare få viss hjälp med parameterinställningar och övrig information om algoritmen, utan att behöva titta i tekniska dokumentationen.

3.3.6 Ställa in användarparametrar

Varje algoritm tar emot ett antal användarparametrar som påverkar hur filtreringen av bilden ser ut. Dessa kan justeras i GUI. Som skönsvärden används de parametrar som EDGY uppfattar som bäst. För att ändra skönsvärden måste GUI redigeras via Matlabs `guide()`-kommando.

Parametrarnas värden läses in då användaren kallar på algoritmen. Värdena i textrutorna tolkas av Matlab som strängar och konverteras till numeriska tal med `str2num()`.

Samtliga algoritmer har parametern *Show graphs* som innebär att informativa grafer och delresultat genereras under körning av algoritmen. En annan speciellt parameter är Waveletsalgoritmens *sigma*. Om den sätts till 0 får användaren med hjälp av ett klickbart fönster välja ett område i bilden för brusskattning. För mer information om användarparametrarna för de olika algoritmerna, se dokumentationen för respektive delsystem.

3.3.7 Kalla på vald algoritm

När användaren klickar på *Run*-knappen läses alla parametervärden in och den kodsträng som används för att kalla på algoritmen byggs upp. För att köra det kommando som finns i strängen används Matlabs `eval(codeString)`. Som har nämnts tidigare ser `codeString` typiskt ut som exemplet nedan.

```
[outImage, elapsedTime] = algorithmName(inImage, param1, param2, param3)
```

Algoritmerna returnerar två utargument; utbild och tidsåtgång. Anledningen till att tidsåtgången beräknas inne i algoritmen i stället för i GUI är att det ger en mer rättvisande bild om vissa kommandon som körs i algoritmen inte räknas in i tiden. Exempel på sådan är engångsgenerering av filterkärnor och dylikt, som varierar med parametervärdena men som för givna inargument egentligen inte skulle behöva omgenereras.

När användaren klickar på *Run*-knappen ändras texten till "Running...", vilket visar att GUI väntar på att vald algoritm ska returnera eller kasta ett fel. Under tiden går det inte att klicka på knappen för att starta en nya exekvering. Så fort algoritmen returnerar ändras knapptexten till "Run" igen och väntar på ett nytt kommando.

3.3.8 Utvärdera resultatet

När algoritmen har returnerat erhåller GUI den filtrerade bilden samt uppmätt tidsåtgång. Om användaren så önskar utförs *scan conversion* (se avsnitt 3.3.2) av både in- och utbild.

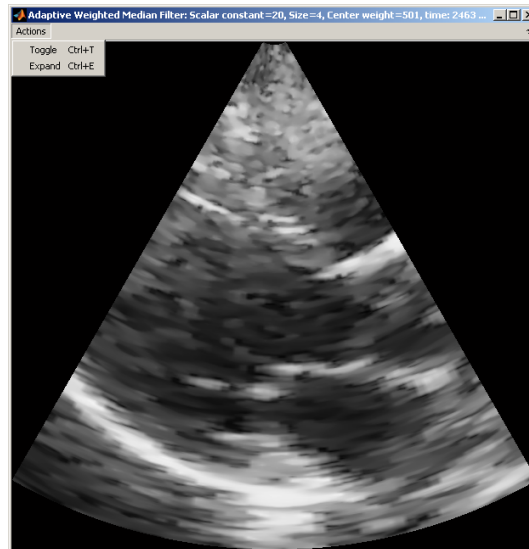
För att presentera dessa tillsammans med använda parametervärden laddas ett Resultatfönster, som utgörs av ett nytt Matlab-GUI vars inställningar och kod finns i `gui/ResultWindow.fig` respektive `gui/ResultWindow.m`. Resultatfönstret består i princip av en bild och två menyalternativ, se figur 4.

Initialt visas resultatbilden. För att växla till originalbilden används kommandot *Toggle* på *Actions*-menyn, alternativt tangentkombinationen `Ctrl+T` på tangentbordet. På samma sätt kan användaren växla tillbaka till den filtrerade bilden. Möjligheten att via tangentbordet växla mellan in- och utbild gör det enkelt att uppfatta skillnader dem emellan.

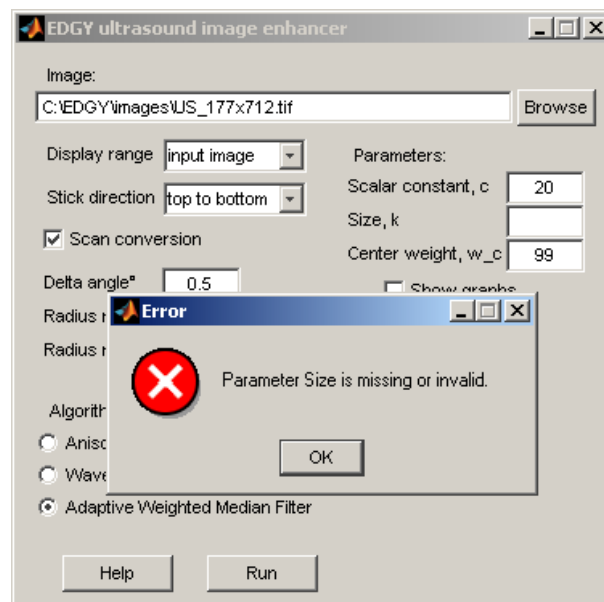
Användare vill ibland skriva ut eller spara undan bilder och parameterinställningar för framtida bruk. För att enkelt kunna erbjuda ett sådant alternativ finns alternativet *Expand* på *Actions*-menyn (`Ctrl+E`). Kommandot genererar två *figures* i Matlab, en för respektive bild, med de vanliga verktygsfälten och menyerna för figurer. Dessa inkluderar bland annat *Save as* och *Print* på *File*-menyn samt zoomningsverktyg.

3.4 Felhantering

I GUI inkapslas majoriteten av koden i ett `try-catch`-block. Fördelen är att fel som uppstår mellan Matlabs `try` och `catch` inte direkt levereras till användaren, utan kan hanteras på valfritt sätt i koden. Exempel: I `try`-blocket kallas en funktion. Om ett inargument är felaktigt kastas ett exception inne i funktionen genom att köra `error(errorString)`. I huvudfilen hoppar då exekveringsflödet direkt till `catch`-blocket, där en dialogruta genereras och exekveringen avbryts, se figur 5.



Figur 4: Exempel på ett resultatfönster. Ctrl+T växlar mellan in- och utbilden.



Figur 5: Genom att använda `try-catch`-block kan eventuella fel hanteras internt i programmet.

Om det uppstår ett fel utanför ett `try`-block tar Matlab hand om felet och presenterar felmeddelandet i kommandofönstret. Detta oönskade beteende undviks alltså genom användningen av `try-catch`-block.

Meddelandet i dialogrutan är sista raden i det genererade felmeddelandet. Anledningen till att endast sista raden presenteras för användaren är att felmeddelanden i Matlab normalt består av några rader med debuginformation (liknande ett *stack trace*) som inte passar i GUI, följt av en beskrivning av felet. Om användaren av någon anledning vill se hela felmeddelandet kan följande rad kommenteras i `gui/EDGY.m`:

```
message = message(tIndex(end)+1:end);
```


4 Delsystem 2: Anisotrop diffusion

4.1 Översikt av delsystemet

Artikeln *Real-Time Speckle Reduction and Coherence Enhancement in Ultrasound Imaging via Nonlinear Anisotropic Diffusion* [1] beskriver en brusreducerande men kantbevarande metod och förslag på implementering. Metoden kallas *nonlinear coherent diffusion (NCD)* modell och anpassar filtrets utseende beroende på den lokala omgivningen. Genom strukturtensorn kommer modellen att kombinera de tre filtren *isotropic linear diffusion*, *anisotropic diffusion* och *mean curvature motion* (MCM). Resultatet av de olika filtren finns i artikeln [1] i figur 1.

Artikeln är relativt noggrann i sin beskrivning av modellen men lämnar mycket åt implementatören att lösa. De antaganden vi har gjort beskrivs nedan i samband med att operationerna beskrivs. Enligt artikelförfattarna kan man få denna modell/implementering att komma upp i hastigheter runt 16 ms/iteration. Med ett snitt på 3 iterationer/bild kom man upp i en hastighet av 50 ms/bild på en PC med PII 366-MHz.

Algoritmen kallas på med följande kommando:

```
[I, timeSpent] = AnisotropicDiffusion(inImage, sigma1, delta1, sigma2, delta2, ...  
                                     s2, alpha, beta, tau, numIter, useAOS, graphs)
```

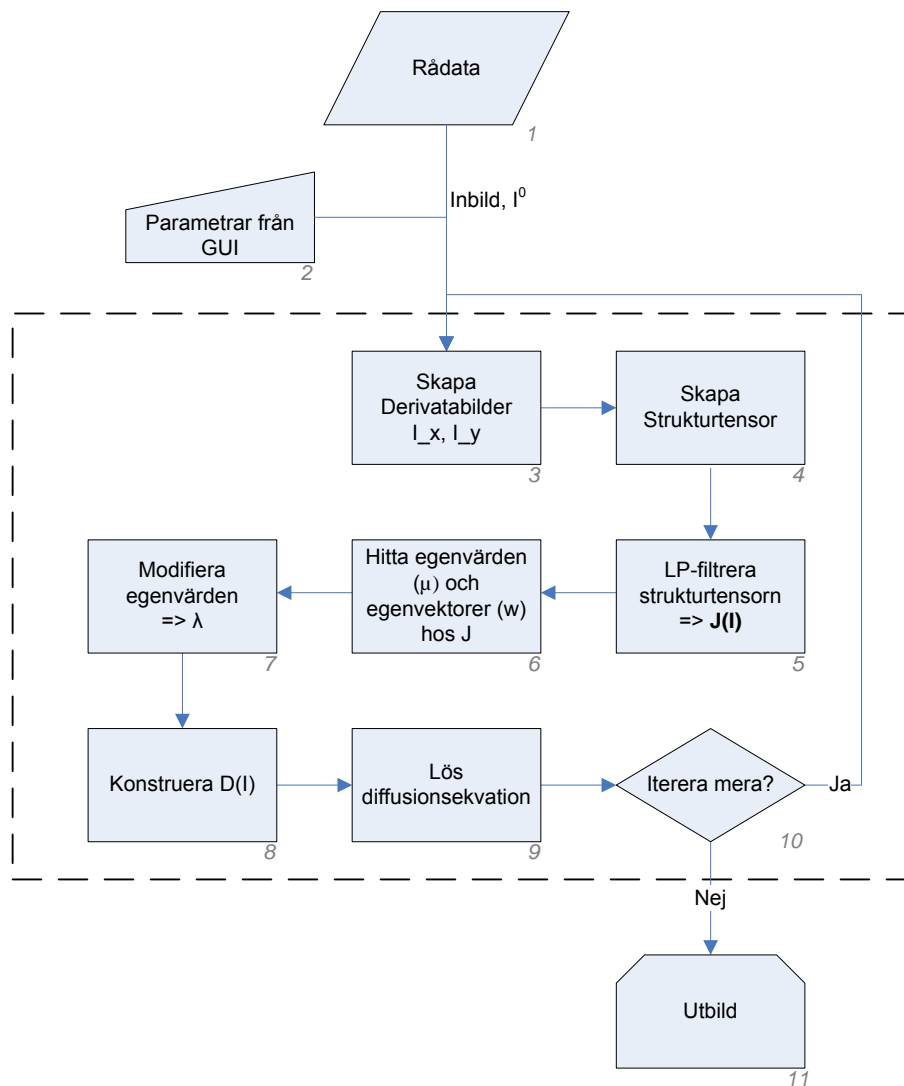
För information om inargumenten, se avsnitt 4.3.2. Koden återfinns i appendix, avsnitt B.2.

4.2 Kort om teorin

Algoritmen bygger på att man modellerar ett fysikaliskt termiskt system. Man tänker sig att bilden genereras och sedan diffunderar ut under viss tid och att detta ger effekter som utsuddning av kanter och brus. Man vill finna derivatan i tidsled hos bilden för att kunna gå framåt i tiden och på så vis jämna till bilden. Eftersom man bara har en bild och inte en tidsserie beräknar man tidsderivatan ur strukturtensorn och tar divergensen av denna. För att beräkna denna behöver man bildens derivator. Man antar också att diffusionen inte går över kanter och på så vis blir metoden kantbevarande och brusreducerande.

4.3 Blockschema och blockbeskrivning

I korta drag kan man säga att algoritmen som beskrivs i artikeln tar fram en strukturtensor och undersöker, med hjälp av egenvärdesanalys, om omgivningen innehåller kanter eller är mer isotrop. Egenvärdena viktas om för att filtrera styrkan hos diffusionen i vardera riktning. Diffusionen utförs i båda riktningarna förutom fallet då det är stor skillnad på egenvärdena. Egenvärdet till principalriktningen sätts då till noll så att diffusionen inte går över kanten utan bara filtrerar längs med den. Ett blockschema över implementationen finns i figur 6.



Figur 6: Blockschemat över metoden

4.3.1 Block 1 - Indata

Den bild som läses in återfinns i parametern *inImage* eller i formler I . I^t där ($t > 0$) är den brusreducerade bilden efter t antal iterationer. $I^0 = I$ motsvarar inbilden om $t = 0$. Formatet på inbilden är rådata (eng. *raw lines* eller *sticks*) från ultraljudsutrustningen innan det har om-mappats för att visas på skärm. På grund av den tekniska utrustningens konstruktion beskrivs bilden i något som kan liknas vid polära koordinater. Den radiella riktningen definieras som y-led och den angliska/vinkeln som x-led. Alla pixlar detekterade i samma *stick* hamnar då i en kolumn i matrisen och alla pixlar på en rad är detekterade på samma avstånd från detektorn.

Ett alternativt format på inbilden är kartesiska koordinater, det vill säga redan ommappad för visning på skärm. Val av indataformat tar dock GUI hänsyn till, så själva algoritmen påverkas inte. Se mer avsnitt 2.4. Datalogiskt är formatet en matris av UINT8 i Matlab som skickas från GUI.

4.3.2 Block 2 - Användarparametrar

De parametrar som styrs från GUI kallas *användarparametrar* och listas i tabell 1 och därefter följer en närmare beskrivning av dess funktion.

Funktion	Param.	Beskrivning
Data	I	Inbild i form av en matris i Matlab.
Deriveringskärna	σ_1 δ_1	Standardavvikelse för gaussfunktionen i deriveringsfunktionen i ekv. 2. Parameter som styr när gausskurvan (för derivering) ska trunckas.
Strukturtensor	σ_2 δ_2	Standardavvikelse till gaussfunktionen i $J(I)$. Parameter som styr när gausskurvan som tillhör σ_2 ska trunckas.
Diffusion	s^2 α β τ $numIter$ $useAOS$	Stoppnivå, styr när diffusionen i gradientriktningen upphör. Styr graden av isotrop diffusion, dvs generella brusundertryckningen. Parameter som styr maximal förändring för varje pixel. Tidssteg vid användning av AOS-schema (se avsnitt 4.3.9) Antalet iterationer. Anger om Additive Operation Splitting scheme ska användas. eller om vanlig divergens ska användas.

Tabell 1: Användarparametrar

4.3.3 Block 3 - Derivering

Artikeln beskriver inte hur derivatorna I_x och I_y ska tas fram. Vi väljer att beräkna derivator genom faltning med en deriverad gaussfunktion med standardavvikelse σ_1 . Det är också den filterkärna som beställaren rekommenderar. Gaussfunktionen i två dimensioner återfinns i ekvation 1 om man förutsätter att medelvärde hos signalen är noll. Detta är inget problem i vårt fall eftersom kärnan ska deriveras.

$$K_\sigma(x, y) = \frac{1}{2\pi\sigma_1^2} \cdot \exp\left(\frac{-(x^2 + y^2)}{2\sigma_1^2}\right) \quad (1)$$

Det har undersökts ifall en enkel differenskärna ($[-1 \ 1]$) utför samma operation men derivatan blir då ytterst bruskänslig. Detta kompenseras till viss del i LP-filtreringen av strukturtensor som görs i block 5. Men då derivatan där har kvadrerats blir resultatet inte så bra.

Vi vill ha operatören separabel i x- och y-led och som dessutom är normerad. Detta görs genom att filtret K , efter diskretisering och trunkering, normeras så att $\sum K(k) = 1$. Vi utgår ifrån den endimensionerella gaussfunktionen (ekvation 2) och dess derivata (ekvation 3).

$$K_{\text{onorm}}(k) = \exp\left(\frac{-k^2}{2\sigma_1^2}\right) \quad (2)$$

$$K'_{\text{onorm}}(k) = \frac{\partial K_{\text{onorm}}(k)}{\partial k} = -\frac{k}{\sigma_1^2} \cdot \exp\left(\frac{-k^2}{2\sigma_1^2}\right) \quad (3)$$

Dessa funktioner samplas, trunkeras vid δ_1 och normeras sedan för att skapa filterkärnorna. Vi har valt att ha användarparametern δ_1 för att trunkera gaussfunktionen vid en lämplig nivå, för att få en lämplig storlek på kärnan. Alternativt skulle man kunna styra storleken på kärnan direkt, men detta blir onödigt svårt att ställa in. Dock blir vårt sätt lite svårare att implementera eftersom man inte vet vilken storlek kärnan får direkt.

$$K(k) = \frac{K_{\text{onorm}}(k)}{\sum K_{\text{onorm}}(k)} \quad (4)$$

$$K'(k) = \frac{K'_{\text{onorm}}(k)}{\sum k \cdot K'_{\text{onorm}}(-k)} \quad (5)$$

För att få kärnan att ha minimal storlek görs detta iterativt i hjälpfunktionen `createDerivKernel()`. Matlabkod för att skapa gausskärna och deriverade kärna iterativt finns i appendix B.2.2. Tiden detta tar är försvinnande liten gentemot tiden det tar att falta bilden med kärnan. Dessutom behöver respektive kärna bara göras om ifall användarparametrarna ändras. Därför läggs denna utanför den redovisade tidsåtgången. Givetvis behövs derivatan tas fram även i y-led, dvs det behövs även en transponerad variant.

Faltningen utförs med `imfilter()`, då denna ingår i *Image Processing Toolbox*, vilken ContextVision har licens för. Dessutom behåller den dataformatet på bilderna och hanterar kantproblem genom att kopiera kantvärdena rakt ut från bilden om man använder parametern `'replicate'`. Denna lösning av kantproblematiken föreslogs av beställaren då den ofta används på ContextVision. Artikelförfattarna tar inte alls upp denna problematik.

Man bör uppmärksamma att `imfilter()` utför korrelation, det vill säga man måste spegla kärnan först eller använda parametern `'conv'` för att få faltning. För att få samma beteckningar som i artikeln i ekvationerna definieras variabler i ekvation 6 upp.

$$\begin{aligned} I_x &= \frac{\partial I}{\partial x} \\ I_y &= \frac{\partial I}{\partial y} \end{aligned} \quad (6)$$

Anrop för derivering med separabla kärnor i x- och y-led blir nu som nedan.

```
I_x=imfilter(I,derivKernel, 'conv', 'replicate');
I_y=imfilter(I,derivKernel', 'conv', 'replicate');
```

4.3.4 Block 4 - Strukturtensor

I artikeln [1] beskrivs två uppställningar av tensorerna för beskrivning av lokal koherens, hessian och strukturtensor. Med motiveringen att hessianen är mer känslig för brus har artikelförfattarna valt att använda strukturtensorerna. Vi undersökte laborativt detta och kunde bekräfta att så var fallet.

$$(\nabla I)(\nabla I)^T = \underbrace{\begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix}}_{\text{Strukturtensor}} \quad (7)$$

För att spara minne och processorbelastning utnyttjas symmetrin genom att tensorerna lagras som en matris med tre lager. Matlabkoden blir trivialt som följer nedan.

```
nablaI(:, :, 1) = I_x.*I_x;
nablaI(:, :, 2) = I_x.*I_y;
nablaI(:, :, 3) = I_y.*I_y;
```

4.3.5 Block 5 - LP-filtrer

Strukturtensorerna är fortfarande känsliga för brus trots att de enskilda derivatorna skapats med gaussfilter. Detta problem minskas genom att lågpasfiltrera strukturtensorerna med ett normerat gaussfilter beroende på användarparametrarna σ_2 och δ_2 . Strukturtensorerna döps, i enlighet med artikeln, efter filtreringen till $J(I)$ och kan uttryckas enligt ekvation 8.

$$J(I) = \begin{pmatrix} K_\sigma * I_x^2 & K_\sigma * (I_x I_y) \\ K_\sigma * (I_x I_y) & K_\sigma * (I_y^2) \end{pmatrix} \quad (8)$$

Kärnan kommer även här tas fram iterativt av funktionen `createGaussKernel(sigma, delta)`. Standardavvikelsen på kärnan styrs av σ_2 och storleken av trunkeringströskeln δ_2 . Framtagning av filtren behöver bara göras om användarparametrarna ändras. Därför räknas, precis som `createDerivKernel(sigma, delta)`, inte heller denna operation med i den redovisade tidsåtgången för algoritmen. Kod för funktionen återfinns i appendix B.2.3.

Filtreringen görs givetvis i x- och y-led och för att få upp hastigheten används separabla kärnor. I Matlab kan filtreringen uttryckas enligt nedan.

```
% The following two lines can be precalculated
Kx = createGaussKernel(sigma_1, delta_1);
Ky = Kx';
J = imfilter(imfilter(nablaI, Kx, 'conv', 'replicate'), Ky, 'conv', 'replicate');
```

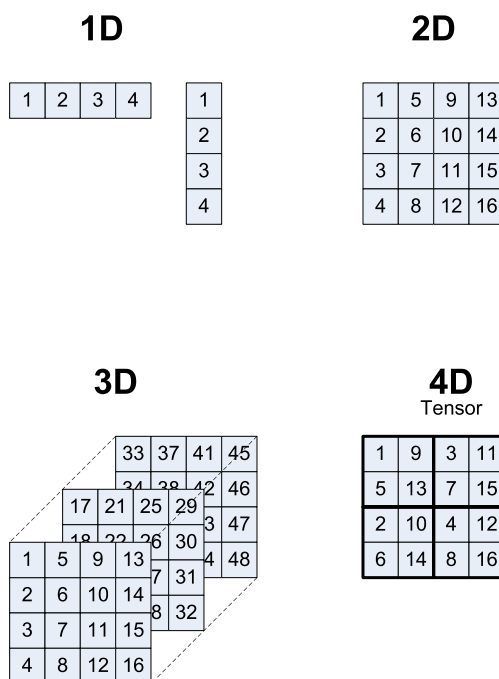
Detta steg tar mycket tid trots att `imfilter()` enligt beställaren är kraftigt optimerad. Detta även på små kärnor av storlek $[1 \times 9]$.

4.3.6 Block 6 - Egenvärdesanalys

Av optimeringsskäl implementerades block 6 till 8 i C och källkod återfinns i appendix B.2.4. För att inte spilla tid på argumentvalidering och göra koden mer lättläst görs bara ett anrop från Matlab.

Eftersom funktionerna som är implementerade i C ska vara så optimerade som möjligt och att de inte är tänkta att användas fristående görs ingen kontroll av inargument. Denna kontroll görs bara i Matlab-koden i början av funktionen även om god programmeringssed är att göra det i varje funktion.

Mycket av beräkningarna i C görs med pekararitmetik. Fördelen med det är hastigheten, nackdelen är att koden blir något mer svårläst. För att förstå vad som händer behöver man veta hur Matlab lagrar matriser. Matriser lagras alltid kolumnvis, det vill säga om man har en bild så motsvara nästa adress pixeln rakt under. Om bilden har m rader och n kolumner och man vill nå pixeln direkt till höger måste man gå m steg framåt. Om man har en 3D-matris och vill nå nästa elementet i z-led måste man gå $m * n$ steg framåt. För 4D-tensorer av storlek $[m * n * x * 2 * x * 2]$ ligger de fyra elementen på avstånd $m * n$. Figur 7 försöker illustrera hur de ser ut i olika dimensioner. 4D-fallet har 4 pixlar med en $[2 \times 2]$ matris i varje pixel.



Figur 7: Matlabs lagring av matriser i minnet

En sak som gör att koden blir ännu mera svårläst är att kvadrering inte har gjorts med `pow()` utan genom att multiplicera talet med sig själv. Detta går förvånande snabbare, vilket antagligen beror på färre funktionsanrop och att `pow()` är en mycket mer generell funktion.

Egenvärdesanalysen av strukturtensorn J utförs i hjälpfunktionen `fastEig()`. Den utför en egenvärdesanalys på varje elementgrupp i tensorn och tar fram egenvektorer ω_i och egenvärden μ_i . Målet är att få matrisen diagonaliserad för att kunna skriva den på formen i ekvation 9.

$$J(I) = \begin{pmatrix} \omega_1 & \omega_2 \end{pmatrix} \begin{pmatrix} \mu_1 & 0 \\ 0 & \mu_2 \end{pmatrix} \begin{pmatrix} \omega_1^T \\ \omega_2^T \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{pmatrix} \quad (9)$$

Matlabs inbyggda funktion `eig` ansågs för långsam av beställaren och dessutom inte utnyttjar att matrisen J är en symmetrisk matris med storlek 2x2. Våra tester visar att `fastEig()` är minst 100 gånger snabbare.

I `fastEig()` tas egenvärdena, μ_1 och μ_2 , fram på vanligt manér genom att lösa den karateristiska ekvationen. Men man utnyttjar att storleken är känd och att det råder symmetrin, det vill säga $a_{12} = a_{21}$. Värt att notera är att `fastEig()` till skillnad från `Eig()` returnerar största egenvärdet (principalkomponenten) först.

$$\begin{aligned} \left| \begin{pmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{pmatrix} - \mu \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \right| &= (a_{11} - \mu)(a_{22} - \mu) - a_{12}^2 = \\ &= \mu^2 - \mu(a_{11} + a_{22}) + a_{11}a_{22} - a_{12}^2 = 0 \Rightarrow \\ \mu_{\pm} &= \frac{a_{11}+a_{22}}{2} \pm \sqrt{\frac{(a_{11}+a_{22})^2}{4} - a_{11}a_{22} + a_{12}^2} = \frac{1}{2} \left[a_{11} + a_{22} \pm \sqrt{(a_{11} - a_{22})^2 + 4a_{12}^2} \right] \end{aligned} \quad (10)$$

Dessa egenvärden används nu när vi söker nollrummet till systemet för att ta fram egenvektorna. Egenvektorer bör normeras så detta görs också.

$$\begin{aligned} \begin{pmatrix} a_{11} - \mu & a_{12} \\ a_{12} & a_{22} - \mu \end{pmatrix} \begin{pmatrix} \omega_1 \\ \omega_2 \end{pmatrix} &= 0 \Rightarrow \\ (a_{11} - \mu)\omega_1 + a_{12}\omega_2 &= a_{12}\omega_1 + (a_{22} - \mu)\omega_2 \Rightarrow \frac{a_{11}-a_{12}-\mu}{a_{22}-a_{12}-\mu}\omega_1 = \omega_2 \Rightarrow \\ |\omega_1| &= |\omega_2| = 1 \Rightarrow \\ \omega_1 &= \begin{pmatrix} \omega_{11} \\ \omega_{12} \end{pmatrix} = \begin{pmatrix} \frac{-a_{12}}{\sqrt{(\mu_1-a_{11})^2+a_{12}^2}} \\ \frac{\mu_1-a_{11}}{a_{12}*\omega_{11}} \end{pmatrix} \omega_2 = \begin{pmatrix} \omega_{21} \\ \omega_{22} \end{pmatrix} = \begin{pmatrix} \frac{a_{12}}{\sqrt{(\mu_2-a_{11})^2+a_{12}^2}} \\ \frac{\mu_2-a_{11}}{a_{12}*\omega_{21}} \end{pmatrix} \end{aligned} \quad (11)$$

Källkod återfinns som sagt i appendix B.2.4 i hjälpfunktionen `fastEig()`.

4.3.7 Block 7 - Viktning

Egenvärdena μ_1 och μ_2 ger ett mått på förändringen i principalriktningen respektive dess ortogonala riktning. Egenvärdena används för att dela upp filtreringen i olika fall. Saknar omgivningen kanter blir egenvärdena ungefär lika stora, det vill säga $(\mu_1 - \mu_2)$ är liten. Då är olinjär isotrop diffusion lämplig. Vid lite större differenser har man antagligen en toning i bilden och olinjär anisotrop diffusion är lämplig. Denna övergång blir glidande eftersom $(\mu_1 - \mu_2)$ ingår viktningen av λ_1 . Vid stort värde på differensen har man en kant i omgivningen och då används *Mean Curvature Motion*. Detta styrs genom att sätta $\lambda_1 = 0$

Exempel på filtrens resultat för olika fall finns i figur 1 i artikeln [1]. Ekvation 12 hämtas från artikeln och beskriver hur egenvärdena behandlas.

$$\begin{aligned} \lambda_1 &= \begin{cases} \alpha \cdot \left(1 - \frac{(\mu_1 - \mu_2)^2}{s^2}\right), & \text{om } (\mu_1 - \mu_2)^2 \leq s^2 \\ 0, & \text{annars} \end{cases} \\ \lambda_2 &= \alpha \end{aligned} \quad (12)$$

Vi introducerar även s^2 som användarparameter för att ange gränsen för vad som räknas som en kant det vill säga när man ska använda *Mean Curvature Motion*.

$|I(0) - I(t)|$ i ekvation 13 mäter hur stor skillnad diffusionen hittills givit upphov till. β anger därav när diffusionen blir orimligt stor och ska minskas. Om man inte i förväg vet hur många iterationer som ska göras kommer modellen att fortsätta mot den triviala lösningen (grå bild). Användarparametern β införs för att få en algoritm som är lättare att ställa in.

En detalj som vi misstänker har förbisetts i artikeln är att δ kan bli negativt och därmed resultera i att algoritmen i onödan diffunderar fram och tillbaka kring "korrekt" värde. Vi modifierar därför uttrycket med ett extra bivillkor på δ .

$$\delta = 1 - \beta \cdot (|I(0) - I(t)|), \quad 0 \leq \beta \leq 1, \delta \geq 0 \quad (13)$$

Införandet av δ ger att ekvation 12 måste modifieras. Ny ekvation för viktning av $J(I)$:s egenvärden blir

$$\begin{aligned} \lambda_1 &= \begin{cases} \alpha \cdot \delta \cdot \left(1 - \frac{(\mu_1 - \mu_2)^2}{s^2}\right), & \text{om } (\mu_1 - \mu_2)^2 \leq s^2 \\ 0, & \text{annars} \end{cases} \\ \lambda_2 &= \alpha \cdot \delta \end{aligned} \quad (14)$$

Denna omskrivning av ursprungsmodellen i artikeln kallas *Pseudobiased Diffusion* och är rekommenderad av artikelförfattarna. Vi väljer att enbart implementera denna eftersom den ursprungliga modellen används om man sätter $\beta = 0$ ty δ blir då konstant 1. Jämför med ekvation 12.

Vi har valt att inte presentera källkod här eftersom den bör ses i sitt sammanhang. Kod återfinns i appendix B.2.4 i huvudfunktionen under block 7.

4.3.8 Block 8 - Diffusionstensorn

Nästa block i schemat bildar diffusionstensorn D .

$$\begin{aligned} D(I) &= \begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} \omega_{11} & \omega_{12} \\ \omega_{21} & \omega_{22} \end{pmatrix} \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \begin{pmatrix} \omega_{11} & \omega_{21} \\ \omega_{12} & \omega_{22} \end{pmatrix} = \\ &\begin{pmatrix} \omega_{11}^2 \lambda_1 + \omega_{12}^2 \lambda_2 & \omega_{11} \omega_{21} \lambda_1 + \omega_{12} \omega_{22} \lambda_2 \\ \omega_{11} \omega_{21} \lambda_1 + \omega_{12} \omega_{22} \lambda_2 & \omega_{21}^2 \lambda_1 + \omega_{22}^2 \lambda_2 \end{pmatrix} \end{aligned} \quad (15)$$

Källkod återfinns i appendix B.2.4 i huvudfunktionen under block 8.

4.3.9 Block 9 - Diffusionsekvationen

Modellen i ekvation 7 kan nu enligt artikeln uttryckas enligt

$$\frac{\partial I}{\partial t} = \text{div} \left[\underbrace{\begin{pmatrix} a & b \\ c & d \end{pmatrix}}_{\text{Diffusionsmatris (D)}} \cdot \underbrace{\begin{pmatrix} I_x \\ I_y \end{pmatrix}}_{\text{Gradientvektor}} \right] \quad (16)$$

Den lösningsgång som artikelförfattarna rekommenderar för implementation är mycket svårtolkad och vi är osäkra om vi har implementerat den på ett korrekt sätt. Vi har även konsulterat artikeln "Efficient and Reliable Scheme for Nonlinear Diffusion Filtering" [5] som även artikelförfattarna refererar till. Den lösningsgång vi presenterar här är vår tolkning av artikeln [1].

Den lösningsgång artikelförfattarna rekommenderar är en modifikation av *additive operator splitting (AOS) scheme* som i sin tur är en modifikation av en semi-implicit lösningsgång. Nästa utbild genereras enligt ekvation 17.

$$I^{k+1} = \frac{1}{2} \sum_{l=1}^2 (1 - 2\tau A_l(I^k))^{-1} (J^{k+1/2}) \quad (17)$$

Vår tolkning av lösningsgången illustreras i den schematiska skissen i figur 8.

Den tridiagonala matrisen $A_l(I^k)$ har element som byggs upp enligt ekvation 18

$$a_{i,j} = \begin{cases} \frac{\bar{a}_{i,j} + \bar{a}_{i,j\pm 1}^k}{2}, & j = i \pm 1, l = 1 \\ \frac{\bar{d}_{i,j} + \bar{d}_{i,j\pm 1}^k}{2}, & j = i \pm 1, l = 2 \\ a_{i,i-1} + a_{i,i+1}, & j = i \\ 0, & \text{annars} \end{cases} \quad (18)$$

$\bar{a}_{i,j}$, $\bar{b}_{i,j}$, $\bar{c}_{i,j}$ och $\bar{d}_{i,j}$ är elementen i strukturtensorn som beskrivs i ekvation 16.

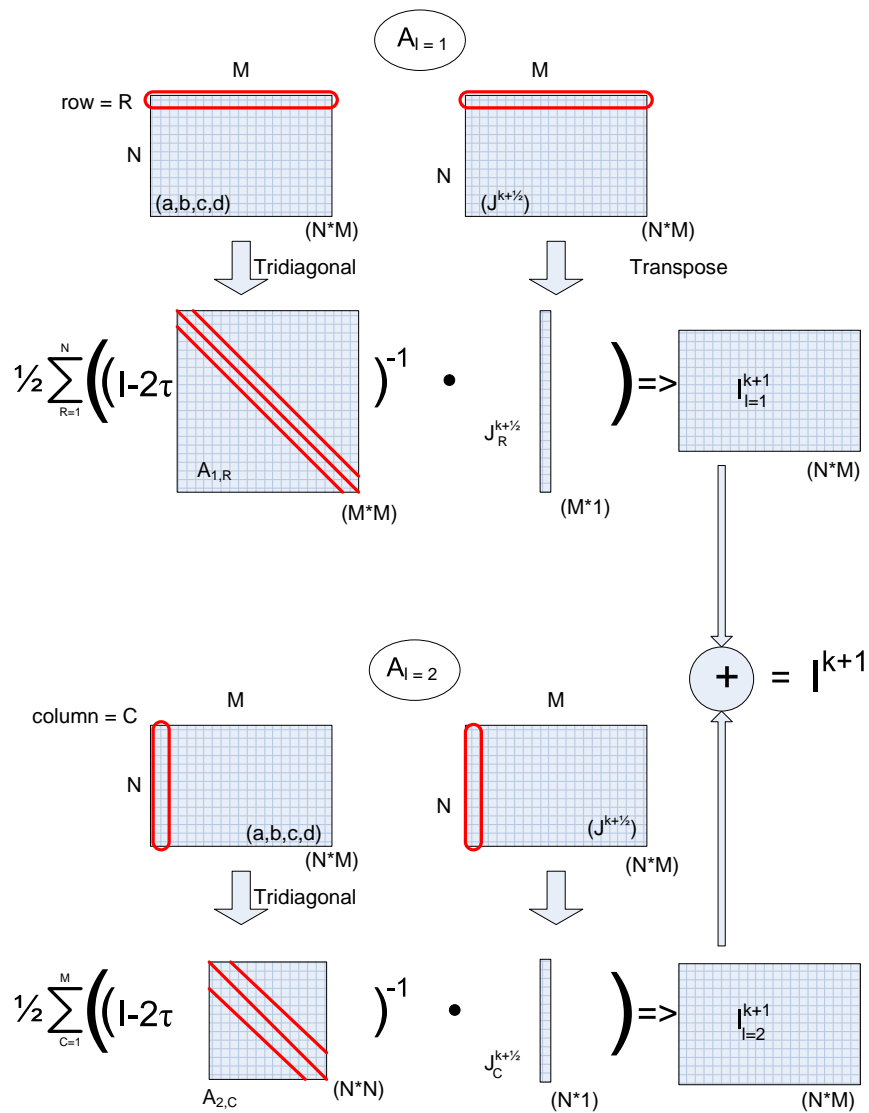
$j = i \pm 1$ pekar på grannvärdena till den aktuella pixeln i och dessa värden som blir grannarna till diagonalen i horisontell riktning för $l = 1$. För $l = 2$ blir det istället diagonalens grannar i vertikal riktning. Då varje A_l består av en rad respektive kolumn är även $J^{k+1/2}$ av samma dimension, det vill säga $J = [1 \times M]$ för $l = 1$ respektive $J = [N \times 1]$ för $l = 2$.

$J^{k+1/2}$ räknas fram i ekvation 19.

$$J_{j,l}^{k+1/2} = I_{j,l}^k + \frac{\Delta t}{4} \{ \bar{c}_{j+1,l}^k (I_{j+1,l+1}^k - I_{j+1,l-1}^k) - \bar{c}_{j-1,l}^k (I_{j-1,l+1}^k - I_{j-1,l-1}^k) \} + \frac{\Delta t}{4} \{ \bar{b}_{j,l+1}^k (I_{j+1,l+1}^k - I_{j-1,l+1}^k) - \bar{b}_{j,l-1}^k (I_{j+1,l-1}^k - I_{j-1,l-1}^k) \} \quad (19)$$

Vi har inte lyckats förstå poängen med schemat då det tar mycket längre tid än att beräkna divergensen direkt som ekvation 16. Detta tror vi beror på de många inversoperationerna i 17. Även en C-implementering av schemat skulle antagligen ta mycket längre tid än Matlabs inbyggda divergensoperator `divergence()`. På grund av detta har vi valt att varken optimera schemat i Matlabkod eller implementera det i C. Vi har inte heller brytt oss om att hantera kantproblemen.

Vi tycker även att resultatet av beräkning med schemat blir sämre än med divergensoperatorn. Vi har lagt till en användarparameter `useAOS` för att kunna utvärdera skillnaden i resultat. Resultat av våra tester finns i avsnitt 4.4.2.



Figur 8: Schematisk skiss över lösningsgången

Ifall man sätter $useAOS = 0$ används istället vanlig divergensberäkning. Enligt beställarens önskemål implementerades Matlabs inbyggda funktion `divergence()` i C. I huvudsak beräknas divergensen enligt ekvation 20 genom att beräkna derivatorna med centraldifferenser i vardera riktning enligt ekvation 21. I randpunkterna används dock framåt- respektive bakåt-differens enligt ekvation 22.

$$\nabla F = \frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y} \quad (20)$$

$$\nabla F = \frac{F_x(k-1) + F_x(k+1) + F_y(k-1) + F_y(k+1)}{2} \quad (21)$$

$$\frac{\partial F}{\partial i} = \frac{F_i(k \pm 1) + F_i(k)}{2} \quad (22)$$

Den nya implementationen utför exakt samma sak som originalet även om den är mindre generell och bara hanterar tvådimensionella fallet och samplingsavstånd 1. Vår implementation är dock fyra gånger snabbare. Källkod finns i appendix B.2.7.

4.3.10 Block 10 - Iterering

Artikelförfattarna nämner inte hur eller över vilka operationer de har itererat. Beställaren miss-tänkte att för att man ska kunna nå upp i den hastighet som artikeln påstår att modellen kan komma upp i borde bara block 4-9 iterera. Det vill säga från efter att man beräknat derivatorna. Detta kommer dock inte ge bra resultat eftersom om man inte uppdaterar derivatorna kommer diffusionstensorn blir exakt samma i varje iteration. Då kunde man lika gärna ha använt ett större α från början. Det finns en användarparameter, *numIter*, som anger antalet iterationer.

4.3.11 Block 11 - Utdata

Sista steget är att producera en färdig bild, vilket bara är resultatet av sista iterationen. Om man använder rådata, det vill säga man har polära koordinater, krävs där en om-mappning till kartesiska koordinater för att få en användbar bild. Denna funktionalitet är implementerad i GUI så algoritmen returnerar bilden dit i 8-bitars unsigned integer (UINT8).

4.4 Resultat

4.4.1 Tidsåtgång

Tidsåtgången för algoritmen börjar mätas efter att kärnor skapats och felkontroll av användarparametrar gjorts. Om algoritmen skulle användas i en färdig produkt skulle man inte behöva felkontrollen eftersom man då vet vad man matar in. Likaså skulle kärnorna bara behöva genereras på nytt ifall användaren ändrat någon parameter.

För att få upp hastigheten på beräkningarna och få en känsla av ifall algoritmen är tillämpbar på realtidstillämpningar ville beställaren att vi implementerade de mest tidskrävande avsnitten

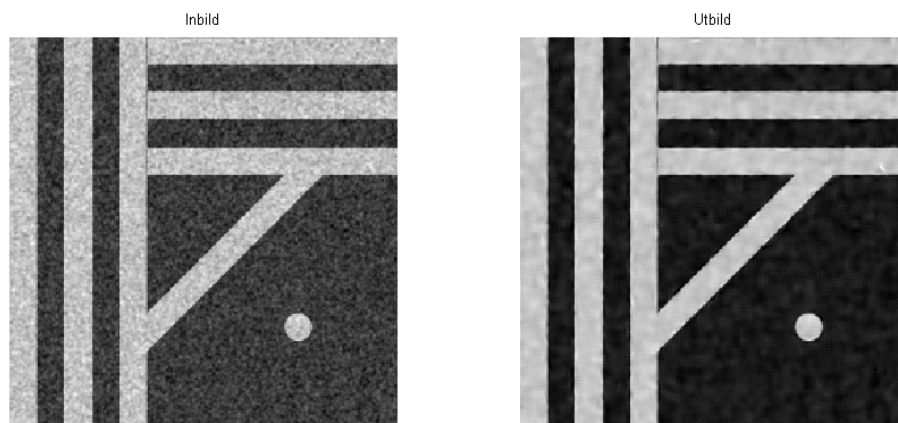
i C. De block som beställaren önskade ha implementerat var övergången från J till D (block 6-8) samt beräkningen av divergens motsvarande Matlabs `divergence()`. Dessa var också de som implementerades.

Att göra egenvärdesberäkningen (block 6) i C gav stora förbättringar i hastighet. Implementeringen av viktningen av λ samt skapandet av elementen i D ger dock inte lika mycket hastighetsvinst, bara cirka dubbelt så snabbt. Divergensberäkningarna blir väldigt många gånger snabbare och tar nu inte mätbar tid. AOS tar dock väldigt lång tid. Denna var inget krav att implementera i C och eftersom det inte verkar som att det kan ge bättre resultat än divergensberäkning har detta inte heller gjorts.

Den totala tidsåtgången är fortfarande en bra bit ifrån att kunna användas i realtid. Realtid anser vi vara cirka 30 bilder/s, det vill säga cirka 0.03 sekunder/bild. Det som framförallt skulle kunna optimeras bättre är `imfilter()` som används för framtagning av derivatabilder samt lågpasfiltrering av strukturtensorn. Denna ska dock enligt beställaren redan vara kraftigt optimerad, men om man gjorde den mindre generell kanske hastigheten skulle kunna komma upp något.

4.4.2 Bildförbättring

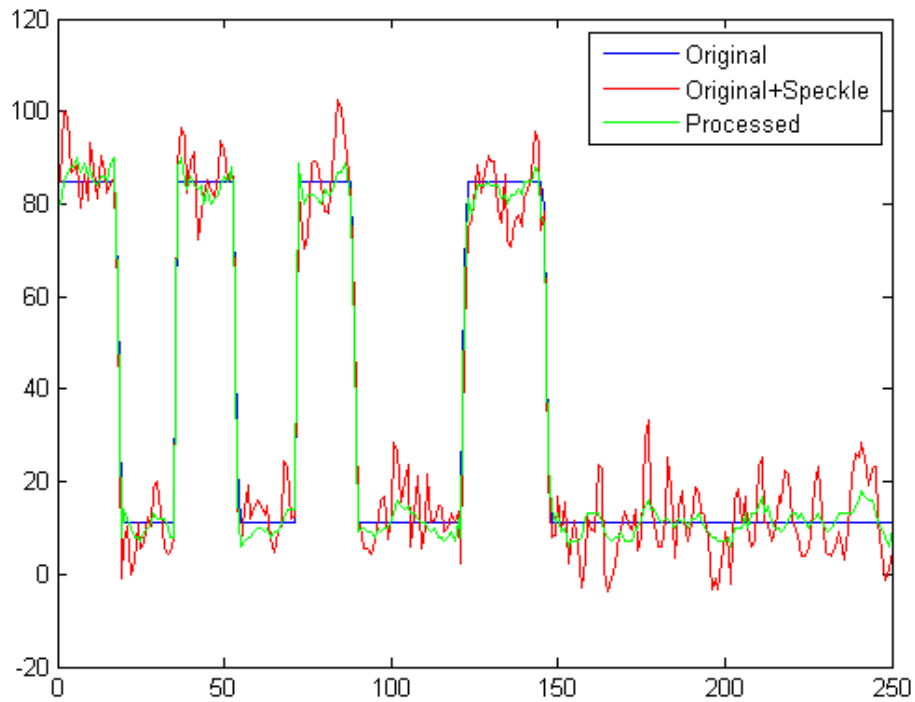
Vi tycker att algoritmen ger ett bra resultat på förhållandevis kort tid. En testbild vi skapat ger kraftig brusundertryckning och kanterna bevaras helt intakta. Före- och efterbild finns i figur 9. Figur 10 visar en graf över förbättringen i en dimension på rad 145 i testbilden. Vi anser att man tydligt ser att den processerade bilden följer originalet i kanter, men är lågpasfiltrerad för övrigt. På detta område anser vi att algoritmen fungerar riktigt bra.



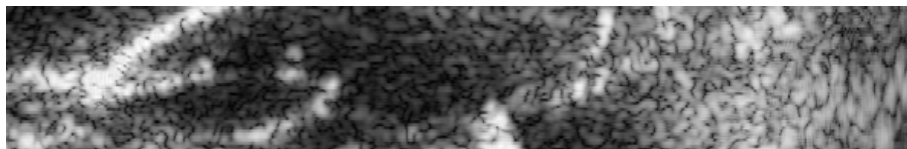
Figur 9: Före- och efterbild. Parametrar: $\sigma_1 = 0.5$, $\delta_1 = 0.6$, $\sigma_2 = 1$, $\sigma_2 = 0.06$, $s^2 = 12000$, $\alpha = 0.9$, $\beta = 0.05$, $numIter = 2$, $useAOS = 0$

Testbilden beskriver dock inte vad som händer på en verklig ultraljudsbild. Den har till exempel additivt brus och inte speckel. Figur 11 är en mer naturlig testbild. Figur 12 visar resultatet på en verklig testbild på samma sätt som testbilden ovan.

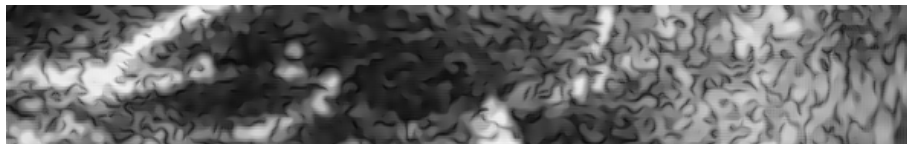
I figur 12 har man använt 5 iterationer vilket kanske är lite mycket. Man kan öka α istället även om risken ökar att man får artefakter kring kraftigt brus. σ_2 är den största skillnaden från testbilden ovan, vilket är naturligt eftersom gaussfiltret för lågpasfiltrering ska matcha brusets



Figur 10: Graf över rad 145 i testbilden.



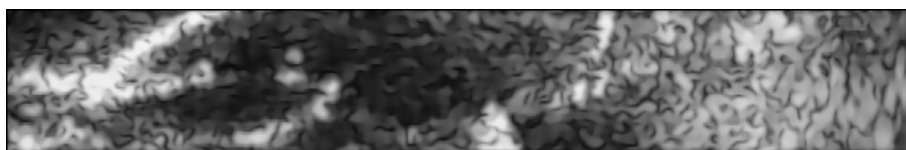
Figur 11: Naturlig testbild.



Figur 12: Efterbild med vanlig divergens. Parametrar: $\sigma_1 = 0.5$, $\delta_1 = 0.4$, $\sigma_2 = 1$, $\sigma_2 = 0.04$, $s^2 = 10000$, $\alpha = 0.9$, $\beta = 0.05$, $numIter = 5$, $useAOS = 0$

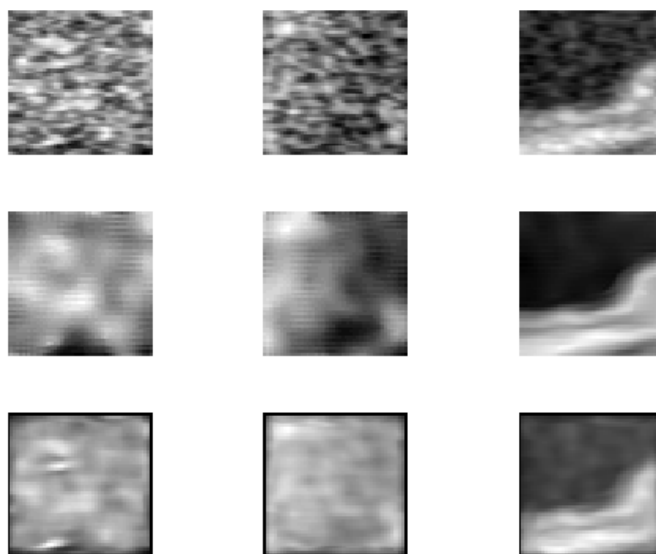
storlek. Den ska vara stor nog för att inte reagera på brus men liten nog för att inte smeta ut kanterna tillräckligt. s^2 är också ändrad eftersom kanternas styrka är mindre i denna bild. Om vill minimera antalet användarparametrar för operatören skulle man spara σ_2 och s^2 .

För att jämföra resultatet mellan att använda vanlig divergens och additive operator splitting (AOS) scheme visar figur 13 resultatet av AOS med i för övrigt samma parametrar. Värt att notera är dock att AOS tog 25 gånger så lång tid. Läs mer om varför i avsnitt 4.3.9.



Figur 13: Efterbild med AOS. Parametrar: $\sigma_1 = 0.5$, $\delta_1 = 0.4$, $\sigma_2 = 1$, $\sigma_2 = 0.04$, $s^2 = 10000$, $\alpha = 0.9$, $\beta = 0.05$, $\tau = 1$, $numIter = 5$, $useAOS = 1$

För att verifiera att algoritmen är adaptiv plockades tre områden ut och tittades närmare på; en isotrop, en med svag toning i och en med tydlig kant. Samma parametrar som ovan användes.



Figur 14: Filtreringen på olika områden. Första raden är inbilder, andra är filtrerade med vanlig divergens och den tredje är filtrerat med AOS

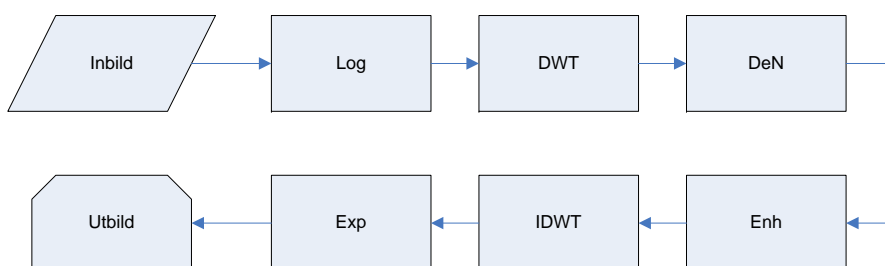
Med de parametrar som använts i till exempel figur 12 anser vi att man får subjektivt bästa resultat. Vi är dock inte medicinskt kvalificerade att avgöra hur användbara bilderna är.

5 Delsystem 3: Wavelets

I det här avsnittet beskrivs hur bildbehandlingsmetoden i artikel [2] har implementerats.

5.1 Översikt av delsystemet

Vid implementeringen har vi gjort samma blockindelning av algoritmen som artikelförfattarna. Indelningen illustreras i figur 15.



Figur 15: Kort översikt över algoritmen. DWT: Diskret wavelettransformation. DeN: *De-noising*. Enh: *Edge enhancement*.

Nedan följer en detaljerad beskrivning av hur varje block har implementerats samt en del teoretiska beskrivningar och motiveringar till den valda implementeringen.

5.2 Brusmodellen

Metoden använder sig av en speckelbrusmodell enligt:

$$f(x, y) = g(x, y)\eta_m(x, y) + \eta_a(x, y) \quad (23)$$

där $g(x, y)$ är originalbilden man vill återskapa, $f(x, y)$ är den brusiga observationen av $g(x, y)$, $\eta_m(x, y)$ multiplikativt brus och $\eta_a(x, y)$ additivt brus. Det gäller att det additiva bruset är mycket mindre än det multiplikativa ($\|\eta_a(x, y)\|^2 \ll \|g(x, y)\eta_m(x, y)\|^2$). (23) kan då approximeras med

$$f(x, y) = g(x, y)\eta_m(x, y) \quad (24)$$

och för att separera bruset från originalbilden använder man en logaritmisk transform $\log(f(x, y)) = \log(g(x, y)) + \log(\eta_m(x, y))$ som även kan skrivas som $f^l(x, y) = g^l(x, y) + \eta_a^l(x, y)$. Det är datat efter den logaritmiska transformen man använder under resten av algoritmen. Efter bearbetning av den logaritmiska bilden exponeras den och man erhåller en förbättrad version av inbilden.

5.3 Wavelettransform

5.3.1 Teori

Den diskreta (dyadiska) wavelettransformen utvecklad av Mallat och Zhong har tidigare används inom områden som kantdetektion, texturanalys, brusreducering och bildförbättring. De har visat

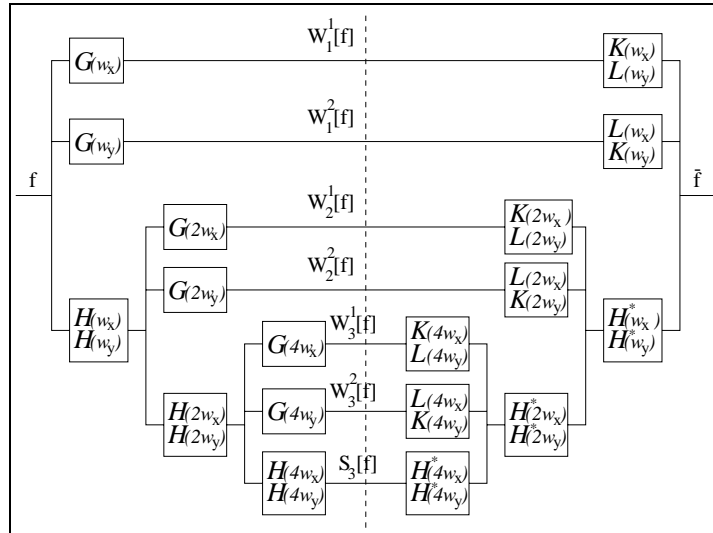
i [4] att man med första ordningens derivator av en utjämningsfunktion (eng *spline smoothing function*) kan skapa en perfekt rekonstruerande filterbank men hjälp av 4 stycken filter ($H(\omega)$, $G(\omega)$, $K(\omega)$ och $L(\omega)$). Impulssvaren för dessa filter fanns givna i artikeln och återfinns i tabell 2. För att erhålla perfekt rekonstruktion måste filtrena uppfylla ekvationerna 25 och 26.

Wavelettransformen går till så att man delar upp frekvensinnehållet i signalen genom att filtrera den med de två filtrena $H(\omega)$ och $G(\omega)$. $G(\omega)$ är ett högpasfilter (enkelt derivafilter) och $H(\omega)$ ett lågpasfilter. Man får två högpaskanaler då man filterar bilden en gång i x-led och en gång i y-led och sparar dem separat. Ur lågpasdelen får man endast ut en kanal då man använder utresultatet från filtreringen i ena riktningen som indata till den andra. Efter de två lågpasfiltreringarna kan man sedan iterera denna kanal och utföra samma filtreringar igen. Högpaskanalerna benäms W_j^d , där $d = 1, 2$ står för att signalen filterats i x- respektive y-led och j talar om i vilken iteration/nivå i filterbanken man är i. I artikeln står det att man för varje nivå ska nollinbaka filtrena med $2^j - 1$ nollor mellan varje koefficient.

Det är sedan på datat i dessa kanaler som man gör själva brusreduceringen och bildförbättringen. Om man lämnar kanalerna obehandlade kan man rekonstruera signalen genom att filtrera högpaskanalerna W_j^1 i x-led med $K(\omega)$ och i y-led med $L(\omega)$, filtrera W_j^2 i x-led med $L(\omega)$ och i y-led med K samt filtrera lågpaskanalen i båda lederna med komplexa konjugatet till $H(\omega)$ (se bild 16).

$$|H(\omega)|^2 + G(\omega)K(\omega) = 1 \quad (25)$$

$$L(\omega) = \frac{1 + |H(\omega)|^2}{2} \quad (26)$$



Figur 16: En 3-nivås DWT-uppdelning och rekonstruktion.

FIR-filter

n	$h(n)$	$g(n)$	$k(n)$	$l(n)$
-4				0.001953125
-3			-0.00390625	0.015625
-2	0.0625		-0.03515625	0.0546875
-1	0.25	1.0	-0.14453125	0.109375
0	0.375	-1.0	-0.36328125	0.63671875
1	0.25		0.36328125	0.109375
2	0.0625		0.14453125	0.0546875
3			0.03515625	0.015625
4			0.00390625	0.001953125

Tabell 2: Impulssvar för DWT-filer

5.3.2 Implementering

Eftersom ContextVision inte har wavelettoolboxen till Matlab och vår filterbank skiljer från vanliga dito på så sätt att den delar upp högpasiskanalen i x -led och y -led, valde vi att skriva vår implementering själva med hjälp av filtreringar med funktionen `imfilter()`. För att kunna erhålla perfekt rekonstruktion är vi tvungna att använda cirkulär faltning genom inargumentet `'circular'` i `imfilter()`. Det är dock inte önskvärt att få in kanteffekter som beror av motstående sida av bilden så vi angriper kantproblemen med hjälp av inargumentet `'replicate'` till `imfilter()` istället. Då kopieras bildens yttersta värde i kanten ut, istället för att tolkas som 0. Två av filtren som var givna har ett jämnt antal koefficienter. Man måste då lägga till en nolla för att få ett udda antal så att `imfilter()` tolkar origo på rätt ställe. Eftersom vårt $H(\omega)$ är väldigt enkelt valde vi att implementera denna filtrering som en matrisoperation i C som vid enskilda tester med kommandona `tic` och `toc` visade sig vara snabbare än Matlabs `imfilter()` på en matris med dimensionerna 512×512 . Tyvärr kan detta inte verifieras när man studerar tidsrapporten som genererats med matlabkommandot `profile()` på referensdatorn, vi är inte säkra på vad detta beror på. Koden för denna hjälpfunktion återfinns i avsnitt B.3.2. För att göra funktionen så snabb som möjligt har vi kommenterat ut alla kontroller av inargument.

I filterbanker vill man ofta sampla ner signalen en faktor två efter varje nivå. För oss introducerar detta ett fel så att vi inte får perfekt rekonstruktion, men man får däremot mycket mindre data att arbeta med. I GUI kan man välja om man vill använda nollinbakade filter för respektive nivå eller om man vill sampla ner bilden varje nivå. När man ska rekonstruera bilden igen använder vi kommandot `interp2()` med interpoleringsmetoden `'spline'`. Vi ansåg att linjär interpolation inte gav tillräckligt bra resultat och `'cubic'` var långsammare.

5.4 Brusreducering (DeN)

5.4.1 Teori

Brusreduceringen i den här algoritmen består av mjuk tröskling av koefficienterna i de två översta nivåerna i wavelettransformen (se figur 16). Den mjuka trösklingen utförs endast på de två första nivåerna eftersom man kan förmoda att dessa innehåller större delen av brusets energi.

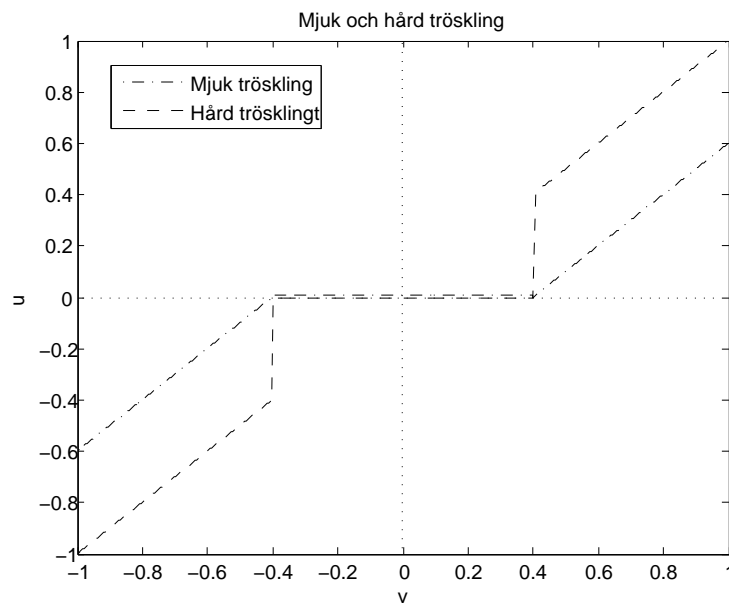
Mjuk tröskling av en funktion v kan beskrivas av följande uttryck

$$u = \mathcal{T}(v, t) = \text{sgn}(v)(|v| - t)_+ \quad (27)$$

där

$$(|v| - t)_+ = \begin{cases} |v| - t & \text{då } |v| > t \\ 0 & \text{annars} \end{cases} \quad (28)$$

Ett exempel på en tröskelfunktion visas i figur 17.



Figur 17: Exempel på mjuk och hård trösklingsfunktion.

De trösklade waveletkoefficienterna ges av

$$W_j^{d,*}[f(m, n)] = \mathcal{T}(W_j^d[f(m, n)], t_j^d) \quad (29)$$

där $d = 1, 2$ anger orienteringen, $j = 1, \dots, k$, ($k \leq J$) anger nivån i filterbanken och t_j^d är ett tröskelvärde som beror av d , j och brusenergin i den aktuella kanalen. Eftersom nivå j innehåller signalkomponenter med högre frekvens än de komponenter som finns på nivå $j + 1$ kommer andelen energi som härstammar från brus att vara mindre på nivå $j + 1$ än på nivå j . Värdet på tröskeln t_j^d beräknas därför som en funktion som avtar linjärt med j enligt:

$$t_j^d = \begin{cases} (T_{\max} - \alpha(j - 1))\sigma_j^d & \text{då } T_{\max} - \alpha(j - 1) > T_{\min} \\ T_{\min}\sigma_j^d & \text{annars} \end{cases} \quad (30)$$

där σ_j^d är en uppskattning av standaravvikelsen hos bruset i signalen, α en avtagandefaktor mellan två efterföljande nivåer i filterbanken, T_{\max} och T_{\min} är maximala respektive minimala skalningsfaktorer för σ_j^d , ($1 \leq j \leq J$), och $d \in \{1, 2\}$. Ett exempel på en funktion som beräknar tröskeln ovan finns i figur 18 på sidan 35.

5.4.2 Implementering

Som vi nyss nämnt bygger beräkningen av tröskelvärde ovan på att man kan göra en uppskattning av standardavvikelsen σ_j^d hos bruset i varje kanal i filterbanken. Detta visade sig vara en av de svårare delarna av implementeringsfasen, varken vi eller våra handledare kände till någon självklar metod. Vi bestämde oss för att jämföra tre olika metoder som skilde sig åt på en punkt: beräkningen av σ_j^d .

Den första metoden använde ett och samma värde på σ_j^d för alla j och d . Den andra metoden byggde på att σ_j^d beräknades som standardavvikelsen av *hela* signalen i den aktuella kanalen. Till slut bestämde vi oss för att använda den tredje metoden som bygger på att låta värdet på brusets standardavvikelse i originalbilden, σ , vara en användarparameter.

För att få fram ett lämpligt utgångsvärde för användarparametern valde vi ut ett område i en av våra testbilder som såg ut att innehålla enbart brus och beräknade standardavvikelsen i området. För att slippa en användarparameter för varje kanal räknade vi ut σ_j^d för varje j och d och tog fram koefficienter som beskriver relationerna mellan varje σ_j^d och σ för testbilden. För att beräkna tröskelvärde för varje j och d används alltså användarparametern skalad med respektive koefficient. Eftersom relationerna till stor del bör bero av vilka filter som används i filterbanken kan man förmoda att de koefficienter vi räknat fram för testbilden är hyfsat representativa för generella ultraljudsbilder. Anledningen till att vi valde denna metod var dels att den var mest konsekvent med beskrivningen i artikeln och den dessutom ger bra resultat. För att användaren inte ska behöva gissa värden på σ för den aktuella inbilden har vi en funktion som genererar ett figurfönster i vilket inbilden visas så att användaren kan välja ut ett lämpligt område att beräkna σ i. Denna funktion anropas om villkoret $\sigma = 0$ är uppfyllt.

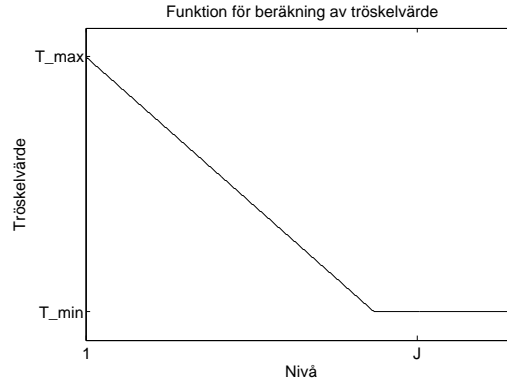
I syfte att göra den mjuka trösklingsfunktionen så snabb som möjligt har vi använt oss av Matlabs inbyggda matrisoperatorer $<$, $>$, $.*$ och $./$. Funktionen är snabb och inte en av flaskhalsarna varför vi valde att inte implementera den i C. Eftersom koden för den mjuka trösklingen är så kort redovisas den för bekvämlighets skull nedan.

```
function U = softThresh( V, tMax, tMin, sigma, alpha, lev )

c = tMax - alpha*(lev-1);

if c > tMin
    t = c*sigma;
else
    t = tMin*sigma;
end

% Apply soft thresholding
U = sign(V).*(abs(V)-t).*(abs(V) > t);
```



Figur 18: Exempel på funktion för beräkning av tröskelvärde.

5.5 Hård tröskling och kantförstärkning (Enh)

5.5.1 Teori

Hård tröskling och kantförstärkning av bilden sker med hjälp av en adaptiv förstärkningsoperator (eng *Generalized Adaptive Gain* (GAG) operator) som definieras av

$$E_{GAG}(v) = \begin{cases} 0 & \text{då } |v| < T_1 \\ \text{sgn}(v)T_2 + \bar{u} & \text{då } T_2 \leq |v| \leq T_3 \\ v & \text{annars} \end{cases} \quad (31)$$

$$\bar{u} = a(T_3 - T_2)(\text{sigm}(c(u - b)) - \text{sigm}(-c(u + b))) \quad (32)$$

$$\text{sigm}(x) = \frac{1}{1 + \exp(-x)} \quad (33)$$

$$u = \text{sgn}(v)(|v| - T_2)/(T_3 - T_2) \quad (34)$$

$$v \in [-1, 1] \quad (35)$$

$$b \in [0, 1] \quad (36)$$

För tröskelvärdena gäller att $0 \leq T_1 \leq T_2 < T_3 \leq 1$. Koefficienten c är en förstärkningskoefficient och a beräknas enligt

$$a = \frac{1}{\text{sigm}(c(1 - b)) - \text{sigm}(-c(1 + b))} \quad (37)$$

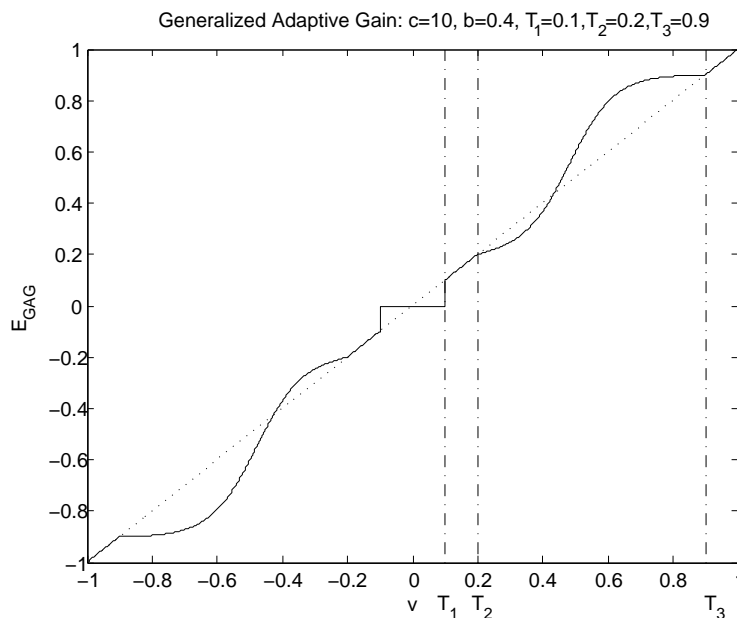
T_1 är värdet på den hårda tröskeln och intervallet $[T_2, T_3]$ kan justeras för att välja ut vilka karakteristika i bilden som ska förstärkas. Operators verkan på DWT-koefficienterna kan uttryckas som

$$W_j^{d,*}[f(m, n)] = M_j^d E_{GAG}(W_j^d[f(m, n)]/M_j^d) \quad (38)$$

$$M_j^d = \max_{m, n} (|W_j^d[f(m, n)]|) \quad (39)$$

$$d = 1, 2 \text{ (riktning)} \quad (40)$$

$$j \in \{k, \dots, J\} \text{ där } 1 \leq k \leq J \quad (41)$$



Figur 19: Exempel på icke-linjär förstärkningsfunktion med hård tröskel.

5.5.2 Implementering

Eftersom vår matlabimplementering av trösklings- och förstärkningsfunktionen `egag()` tog upp en stor andel av exekveringstiden i vår algoritim föreslog beställaren att vi skulle implementera denna som en uppslagning av funktionsvärden i en tabell med funktionsvärden som beräknas en gång för varje uppsättning av parameterinställningar. Förslaget var att vi skulle implementera uppslagningen med linjärinterpolation, men eftersom vi inte ansåg att det var en begränsning att ha en tabell med relativt hög upplösning implementerade vi uppslagningen med hjälp av en högupplöst tabell och närmsta-granne-interpolation. På så sätt blev implementeringen såpass snabb att det inte gjorde någon skillnad om vi implementerade den i C och eftersom matlabkoden är lättare att läsa valde vi att använda matlabfunktionen.

I den nuvarande implementeringen utförs kantförstärkningen endast på de lägsta nivåerna i filterbanken, men det finns egentligen ingen anledning att inte göra kantförstärkningen på fler eller samtliga nivåer i filterbanken och då med olika värde på förstärkningsparametern c i de olika nivåerna. Tyvärr hade vi svårt att tolka artikeln på den här punkten och vi insåg detta lite sent i implementeringsfasen, men det är inga större ändringar som krävs för att åstadkomma funktionaliteten.

5.6 Beskrivning av användarparametrar

I tabell 3 redovisas samtliga inargument till funktionen `multiScaleEnh()`.

Parameter	Beskrivning	Typ
INIMG	Inbilden.	Alla format som stöds av Matlabfunktionen <code>im2single()</code> .
J	Antalet nivåer som skall beräknas i filterbanken	Heltal mellan 1 och 5.
ALPHA	Avtagandefaktor för skalfaktorn som används vid beräkningen av den mjuka tröskeln.	Skalär.
TMIN	Minsta värde på skalfaktorn för den mjuka tröskeln	Skalär.
TMAX	Största värde på skalfaktorn för den mjuka tröskeln	Skalär.
SIGMA	Uppskattning av brusets standardavvikelse i inbilden.	Skalär.
T1	Hård tröskel för trösklings- och kantförstärkningsfunktionen <code>egag()</code> .	Skalär.
T2	Undre gräns för förstärkningsområde i <code>egag()</code> . (skalär).	Skalär.
T3	Övre gräns för förstärkningsområde i <code>egag()</code> .	Skalär.
C	Förstärkningsfaktor i <code>egag()</code> .	Skalär.
B	Parameter som kontrollerar utseende på den förstärkande delen av <code>egag()</code> .	Skalär.
GRAPHS	Logisk parameter som styr om grafer för trösklings- och förstärkningsfunktioner ska visas eller inte.	Heltal 0 eller 1.
SAMPL	Logisk parameter som styr om <code>dwt()</code> ska använda sig av ned-sampling av bilden (<code>SAMPL=1</code>), eller nollinbakning av filtren (<code>SAMPL=0</code>) mellan nivåerna i filterbanken.	Heltal 0 eller 1.

Tabell 3: Beskrivning av användarparametrar

5.7 Resultat

5.7.1 Exekveringstid

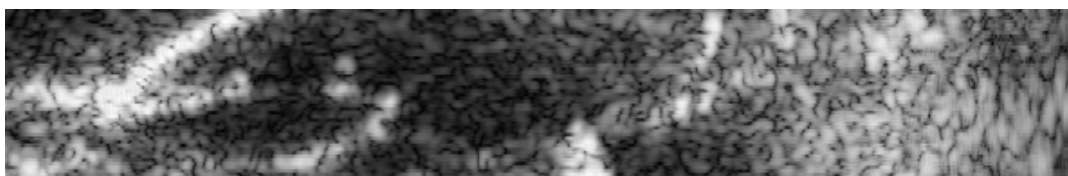
Den tid som alltid redovisas överst i det fönster som presenterar resultatbilden är den tid det tar att köra själva bildförbättringsalgoritmen, utan kontroller av inargument etc. Tidsåtgången för en körning av hela programmet redovisas i avsnitt C.2. Enligt beställare är det önskvärt att kunna köra implementeringen i realtid vilket motsvarar att man kan behandla runt 30 bilder

per sekund vilket motsvarar en exekveringstid på runt 0.03 sekunder. Tyvärr har vi inte lyckats få en iteration av algoritmen att gå snabbare än runt 0.3 sekunder på referensdatorn som finns beskriven i appendix C. De anrop som tar upp huvuddelen av exekveringstiden är anropen till `imfilter()` om man väljer att inte sampla ned bilderna mellan nivåerna i filterbanken, om man däremot gör det kommer även funktionen `interp2()` att ta upp en stor del av tiden. Dessa funktioner förmodar vi vara såpass bra programmerade att vi inte skulle vinna tid på att försöka implementera egna versioner av dem i C-kod.

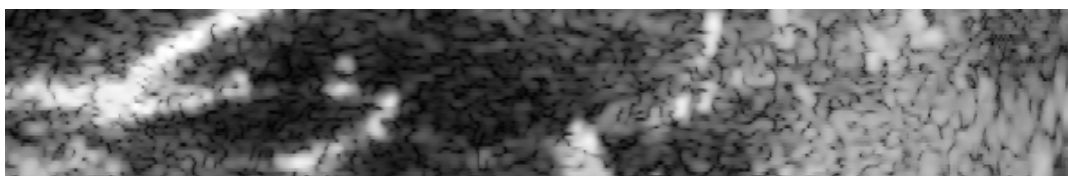
5.7.2 Bildförbättring

Som framgår av tabell 3 finns det en hel del parametrar att justera och vi såg ingen uppenbar strategi för att komma fram till optimala värden, men genom att prova olika inställningar och göra en subjektiv bedömning av resultatet har vi kommit fram till den uppsättning parametervärden som vi tycker ger bäst resultat på bilden `US1.tiff`, en testbild som vi erhållit av beställaren. Dessa parametervärden är som följer: `J=5`, `ALPHA=1`, `TMIN=0.2`, `TMAX=1`, `SIGMA=0.15`, `T1=0.1`, `T2=0.15`, `T3=0.65`, `C=5`, `B=0.08`. Det är också dessa värden som är förinställda när man startar huvudprogrammet. Av resultatet som fås på bilden `US1.tiff` med parametervärden enligt ovan drar vi slutsatsen att vår implementering av metoden ger en tydlig undertryckning av brus och att den har en kantförstärkande effekt. Dock tenderar kantförstärkningen för vissa parameterinställningar att överstyra signalen i områden med hög intensitet vilket leder till att strukturer i dessa områden kan bli osynliga om man visar utbilden i samma område som inbilden. Detta kan dock undvikas genom att man väljer alternativet “no cutoff” i GUI.

Av upphovsrättsliga skäl kan vi inte visa resultatet för bilden `US1.tiff` i detta dokument utan vi får nöja oss att visa resultatet på en bild med mycket sämre upplösning vilket tyvärr ger ett väldigt intetsägende resultat. Testbilden syns i figur 20 och resultatet av bildförbättringen i figur 21.



Figur 20: Testbild.



Figur 21: Resultat av bildförbättringsalgoritm på testbilden i figur 20.

6 Delsystem 4: Adaptiv medianvärdesfiltrering

Artikeln *An Adaptive Weighted Median Filter for Speckle Suppression in Medical Ultrasonic Images* [3] beskriver hur man kan reducera speckel både genom viktade medianfilter och adaptivt viktade medianfilter. De viktade medianfiltren kan visserligen vara effektiva vid brusreducering men samtidigt förloras viktig information i form av mindre objekt samt kantskärpa. Därav introduceras *Adaptive Weighted Median Filter (AWMF)* vilket adaptivt viktar om filtret beroende på kvoten mellan variansen och medelvärdet i den lokala omgivningen. Denna artikel publicerades 1989.

6.1 Översikt av delsystemet

Utöver de moment som är gemensamma med övriga artiklar (val av användarparametrar, visning av in- och utbild) så innehåller AWMF-modellen grovt sett två stycken statistiska beräkningsmoment, konstruktion av histogram samt medianfiltrering av detta histogram. I figur 22 visas ett blockschema över AWMF och i nedanstående avsnitt förklaras blocken mer noggrant. Tillvägagångssättet för att filtrera med AWMF finns implementerat i matlabfunktionen `AWMF()` och anropas genom

```
[outPic time] = AWMF(inPic, c, k, wCenter, graphs)
```

Mer ingående beskrivning av parametrarna finns i avsnitten 6.1.1, 6.1.2 och 6.1.8.

Fullständig Matlabkod till `AWMF()` finns i avsnitt B.4.

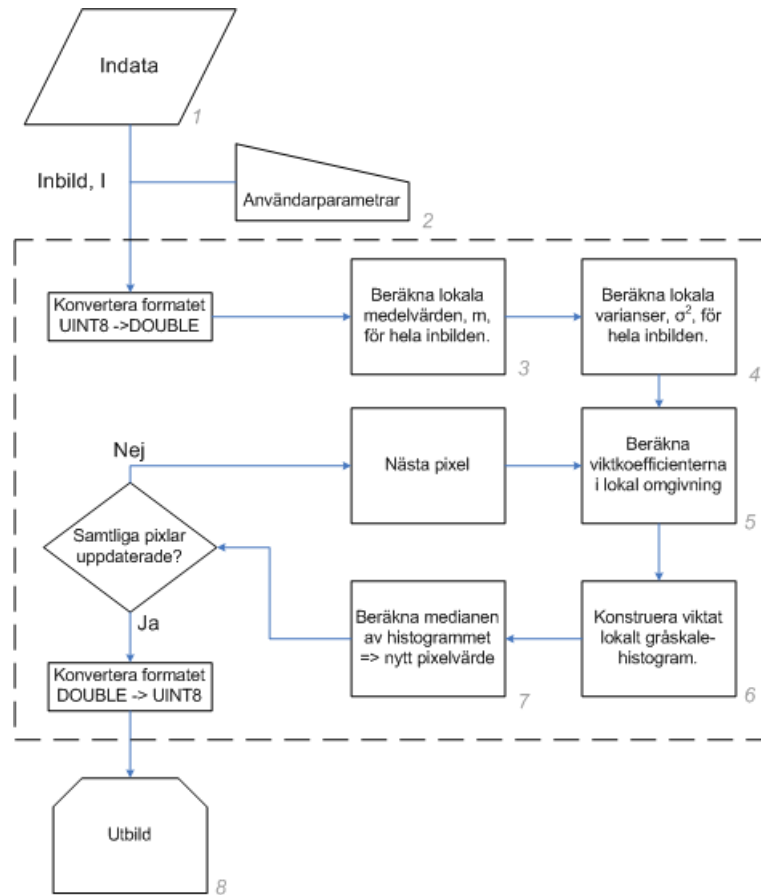
6.1.1 Block 1 - Val av indata

Artikel går ej igenom vilken typ av indata som skall användas i modellen men eftersom modellen använder en konstant storlek på den lokala omgivningen kan vi använda antingen rådata eller formaterad indata. Vi kommer att använda rådata då denna typ av indata kan ge en snabbare process samt ett noggrannare resultat. Rådata är enligt beställaren [6] att rekommendera som indata. Den data som används som indata kallas *inbild*. Den testdata som använts är i formatet *Tagged Image File Format (TIFF)* och vid inläsning i Matlab används funktionen `imread()` som lagrar inbilden i formatet *Unsigned Integer 8-bit (UINT8)*. Därför är detta formatet som AWMF-funktionen ska ha som indata.

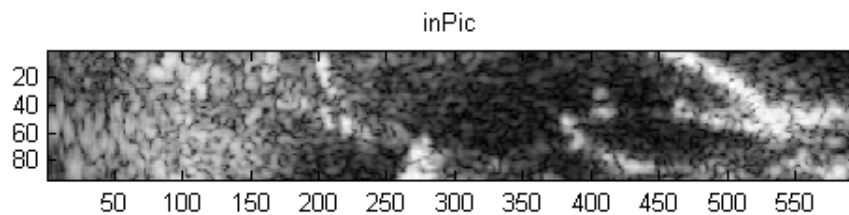
Innan filtrering av inbilden konverteras inbilden från `UINT8` till `DOUBLE` med Matlabfunktionen `double()`. För att undvika division med noll i framtida beräkningar adderar jag även med ett.

```
inPic = double(inPic) + 1;
```

Den inbild som kommer att användas som exempel i denna rapport visas i figur 23 och i figur 24 visas ett inzoomat område av testbilden.



Figur 22: Blockschema AWMF

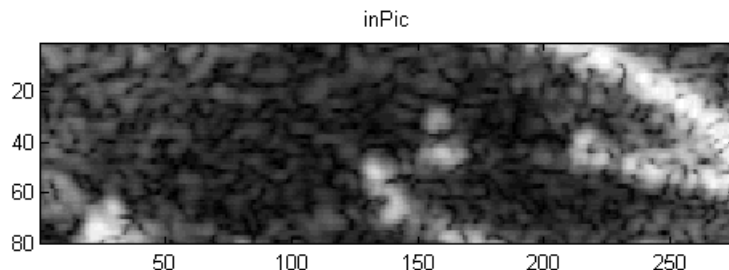


Figur 23: Ultraljudsbild som används som testbild.

6.1.2 Block 2 - Användarparametrar

De parametrar som används i AWMF som kommer att kunna styras från GUI kallas *användarparametrar* och finns i tabell 4.

Då $N = (2K + 1)$ blir uttrycket för storleken på den lokala omgivningen $N \times N$ där centerpixel är den aktuella pixelpositionen, (i, j) . När K ökar i storlek blir utbilden mer "utsmetad" och



Figur 24: Inzoomat område av testbilden.

K	Storleken på lokala omgivningen blir $(2K + 1) \times (2K + 1)$
c	Skalningsparameter vid viktcoefficientberäkning
$w_{center} = w(k + 1, k + 1)$	Initialvärde för centrumvikten i den lokala omgivningen
$graphs$	Visar <i>Development graphs</i> om 1 (se avsnitt 6.1.5)

Tabell 4: Användarparametrar till AWMF

detaljer filtreras bort. Även beräkningsdatan ökar och filtreringen tar längre tid. Sätts $K = 0$ blir utbilden lika med inbilden.

När man ändrar på K behöver man ofta justera övriga parametrar och för att underlätta detta val kan man använda *graphs*-parametern och då se plottar (*Development graphs*) på kvoterna som styr den adaptiva medianfiltreringen. I figur 24 visas ett urklipp ur en ultraljudsbild och i figur 25 visas resultatet efter att filtrerat denna inbild med olika inställningar på parametern K . Här syns hur resultatet blir utsmetat desto större K .

I detta exempel är parametrarna c och w_{center} konstanta. Figur 26 visar kvoten mellan variansen och medelvärdet och här ser man att kanterna inte hittas vid för lågt K och de blir allt för utsuddade om K är för stort. Lägg även märke till att pixelvärdena ökar för kanter vi olika K vilket medför att parametrarna måste justeras då K ändras. Parametrarna c och w_{center} funktion beskrivs mer ingående i avsnittet 6.1.5.

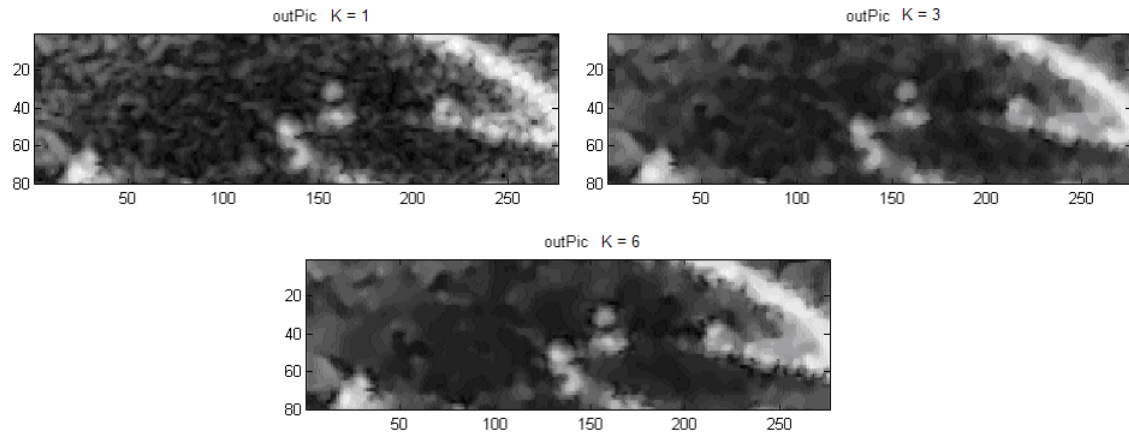
6.1.3 Block 3 - Lokalt medelvärde

Beräkna medelvärdet m för en lokal omgivning (2D). Detta görs för alla pixlar i inbilden.

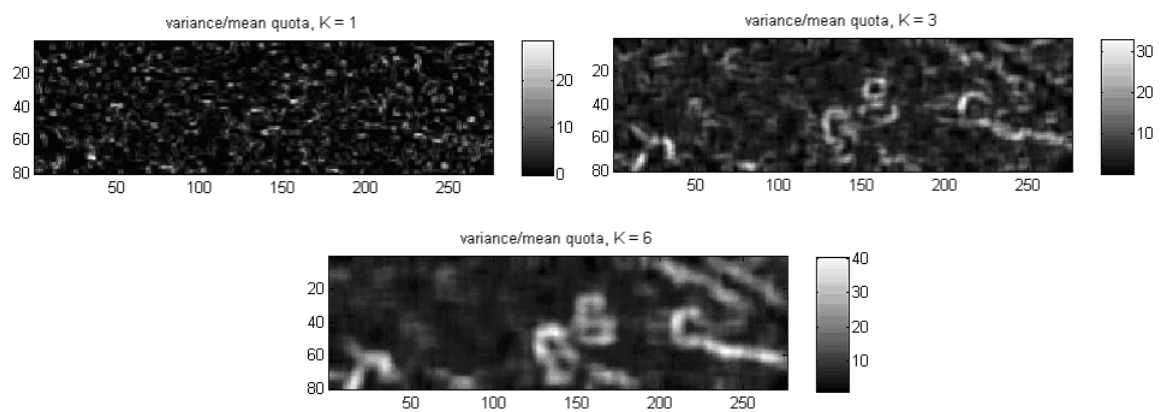
$$m = \bar{x} \equiv \frac{1}{N^2} \sum_{i=1}^{N^2} x_i = E[X] \quad (42)$$

Ett matlabkommando som utför detta är `mean()` men en bättre lösning är att beräkna medelvärdet genom faltning. Faltningen utförs med `imfilter()`, då denna ingår i *Image Processing Toolbox*, vilken ContextVision har licens för. Dessutom behåller den dataformatet på bilderna och hanterar kantproblem genom att kopiera kantvärdena rakt ut från bilden om man använder inargumentet `replicate`. Denna lösning av kantproblematiken föreslogs av beställaren [6] då den ofta används på ContextVision. Artikelförfattarna tar inte alls upp denna problematik.

Medelvärdet m heter i Matlabfunktionen `inPicMean` och tas fram genom



Figur 25: Utbild (inzoomat område) med olika värden på parametern K .



Figur 26: Kvotbild (inzoomat område) med olika värden på parametern K .

```
N = 2*k+1; % Ger lokala omgivningens sidlängd
kernelMean = ones(N,N)/(N*N);
inPicMean = imfilter(inPic, kernelMean, 'conv','replicate');
```

6.1.4 Block 4 - Lokal varians

Beräkna variansen σ^2 för den lokala omgivningen (2D). Detta görs för alla pixlar i inbilden.

$$\sigma^2 = \frac{1}{N^2} \sum_{i=1}^{N^2} (x_i - m)^2 \quad (43)$$

Ett matlabkommando som utför detta är `var()` men en bättre lösning är att återigen använda `imfilter()` och beräkna medelvärdet enligt

$$\sigma^2 = E[X^2] - E[X]^2 \quad (44)$$

där $E[X]$ är det medelvärde som beräknades fram i avsnitt 6.1.3 och X^2 är inbilden punktvis i kvadrat.

Då det är kvoten $\frac{\sigma^2}{m}$ som skall beräknas vid viktningen, avsnitt 6.1.5, utförs en förenkling av kvoten och tidsåtgången minskas.

$$\frac{\sigma^2}{m} = \frac{E[X^2] - E[X]^2}{E[X]} = \frac{E[X^2]}{E[X]} - E[X] \quad (45)$$

vilket resulterar i följande Matlabkod

```
inPicMean = imfilter(inPic, kernelMean, 'conv', 'replicate');
inPicMeanSq = imfilter(inPic.^2, kernelMean, 'conv', 'replicate');
quotaWeight = c*(inPicMeanSq./inPicMean - inPicMean);
```

Här har kvoten multiplicerats med skalningsparametern c .

6.1.5 Block 5 - Viktkoefficienter

Viktkoefficienterna för den aktuella lokala omgivningen beräknas enligt

$$w(i, j) = [w_{center} - c \times d \times \frac{\sigma^2}{m}] \quad (46)$$

Variablerna i ekvation 46 förklaras i tabell 5. I ekvation (46) ser man hur skalningsparametern c påverkar kvoten $\frac{\sigma^2}{m}$. Om centervikten w_{center} är för liten i förhållande till $c \times d \times \frac{\sigma^2}{m}$ kommer alla vikter att bli noll (förutom centerpixeln då $d = 0$ ger $w = w_{center}$) och utbilden kommer inte att filtreras. Om man sätter $c = 0$ blir alla pixlar i den lokala omgivningen lika viktade och den adaptiva viktade medianfiltreringen övergår till en ren medianfiltrering. För att lättare se vilka värden man bör sätta på parametrarna w_{center} och c kan man titta på *Development graphs*, se avsnitt 6.1.2. I *Development graphs* ser man kvoterna $\frac{\sigma^2}{m}$ och avståndskartan d . Ett exempel på hur denna *Development graphs* kan se ut visas i figur 27.

Även valet av parametervärdena på w_{center} och c resulterar i balansgång mellan brusreducering och detaljbevaring.

w_{center}	Initial viktparameter
c	Skalningsparameter vid viktcoefficientberäkning
m	Lokala medelvärde
σ^2	Lokala variansen
d	Avståndet från punkt (i, j) till centrum av fönstret, $(K + 1, K + 1)$
$[\cdot]$	Enligt ekvation 47

Tabell 5: Parametrar till viktningfunktionen

$$[x] = \begin{cases} \text{närmsta heltal till } x & , \text{ då } x \geq 0 \\ 0 & , \text{ annars} \end{cases} \quad (47)$$

Artikelförfattarna [3] diskuterar inte vilket typ av avståndsmått d motsvarar så det euklidiska längdmåttet används.

Nedan visas ett exempel på hur ekvation (46) och (47) kan implementeras i Matlab. För att se hur detta är implementerat i AWMF se avsnitt B.4. Observera att här tas ingen hänsyn till kantproblematiken utan detta behandlas i avsnittet 6.2.

```
% Calculate distancematrix
[dMeshCol, dMeshRow] = meshgrid(-k:k,-k:k);
distMap = sqrt((dMeshRow.^2) + (dMeshCol.^2));

WCenter = wCenter*ones(N,N);

for iRow = 1 : numRows
    for iCol = 1 : numCol
        % Calculate local weightmatrix, wLocal
        wLocal = (WCenter - distMap.*quotaWeight(iRow,iCol));
        wLocal = round(max(wLocal,0)); % negative 2 zero
        ...
    end
end
```

6.1.6 Block 6 - Gråskalehistogram

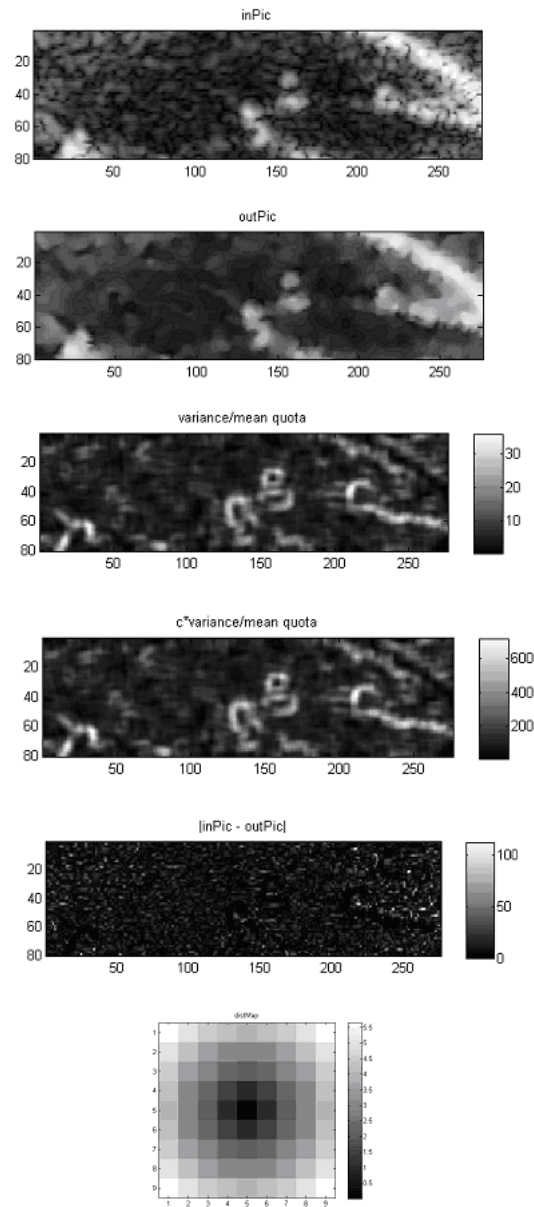
Konstruera ett gråskalehistogram i den lokala omgivningen, $H(l)$. Pixelvärdena, x_i , räknas med w_i gånger i histogrammet. Endast pixelvärden i den lokala omgivningen med nollskiljda vikter räknas med. $H(l)$ är sorterat med lägsta pixelvärdet först.

Exempel: fyra pixlar, $\bar{x} = [x_1 \ x_2 \ x_3 \ x_2]$, med gråskalevärden som förhåller sig som $x_3 < x_2 < x_1$, sorteras först till $\tilde{x} = [x_3 \ x_2 \ x_2 \ x_1]$ (lägsta värdet först). Parallellt sorteras respektive pixels vikt $\bar{w} = [4 \ 2 \ 0 \ 1]$ och $\tilde{w} = [0 \ 2 \ 1 \ 4]$ erhålls. Sedan viktas \tilde{x} med \tilde{w} vilket resulterar i $\hat{x} = [x_2 \ x_2 \ x_2 \ x_1 \ x_1 \ x_1 \ x_1]$ med motsvarande histogram $H(l) = [3 \ 4]$ och tillhörande index-vektor $h_{index} = [x_2 \ x_1]$.

För att minska tidsåtgången vid implementering beräknas inte \hat{x} utan endast \tilde{x} och index-vektorn som erhålls vid sorteringen av denna utnyttjas för att räkna ut medianen, se avsnitt 6.1.7. Om vi åter ser på ovanstående exempel skulle de vektorer som används vid medianberäkningen i implementationen se ut: $\tilde{x} = [x_3 \ x_2 \ x_2 \ x_1]$, $\tilde{x}_{index} = [3 \ 2 \ 4 \ 1]$ och $\bar{w} = [4 \ 2 \ 0 \ 1]$.

Nedan visas ett exempel på hur detta kan implementeras i Matlab. För att se hur detta är implementerat i AWMF () se avsnitt B.4. Observera att här tas ingen hänsyn till kantproblematiken utan detta behandlas i avsnittet 6.2. Ovan nämnda \bar{x} , \tilde{x} och \tilde{x}_{index} motsvaras av `inPicLocal`, `sortInPicLocal` och `index`. \bar{w} återfinns som `wLocal` i koden i avsnitt 6.1.5.

```
...
% Calculate histogram, H(l) for Local Neighbourhood
inPicLocal = inPic(iRow: iRow, iCol - k : iCol + k);
% Sort
[sortInPicLocal index] = sort(inPicLocal(:), 1);
...
```



Figur 27: Exempel på hur *Development graphs* ser ut.

6.1.7 Block 7 - Medianberäkning

I artikeln [3] söks medianen av histogrammet $H(l)$ det vill säga medianen av de viktade gråskälvärdena $M\{H(l)\} = y_{WM}$ där y_{WM} är det lägsta värde som uppfyller ekvation 48

$$\sum_{l=1}^{y_{WM}} H(l) \geq (\sum w(m,n) + 1)/2 \quad (48)$$

där $\sum w(m,n)$ är summan av alla viktkoefficienter. Det nya pixelvärdet x_{ny} fås sedan genom $h_{index}(y_{WM}) = x_{ny}$.

Genom att utnyttja index-vektorn, \tilde{x}_{index} , behöver vi vid implementation inte gå igenom ett viktat histogram utan bara summera ihop vikterna i den ordning som finns i index-vektorn och sedan avbryta när summan av vikterna överstigit eller sammanfallit med $(\sum w(m,n) + 1)/2$. Den antal, y_{WM} , summeringar man behöver göra ger det nya pixelvärdet, x_{ny} , genom $\tilde{x}(y_{WM}) = x_{ny}$, vilket är den sökta medianen. Nedan visas hur denna del är implementerad i Matlab.

```

...
newPxPos = (sumW+1)/2;
sumW = 0;
for iPos = 1:sqrN
    sumW = sumW + wLocal(index(iPos));
    if sumW >= newPxPos
        break
    end
end
yWM = sortInPicLocal(iPos);
outPic(iRow, iCol) = yWM;
end
end

```

6.1.8 Block 8 - Utdata

Före utbilden returneras konverteras utbilden från DOUBLE till UINT8 med Matlabfunktionen `uint8()`. Kompenserar även för den etta som adderades i avsnittet 6.1.1.

```
inPic = double(inPic-1)
```

Utbilden som returneras har samma storlek som inbilden och genom den sista konverteringen så returneras utbilden även i samma dataformat som inbilden.

Förutom den utbild som konstruerats skickas även en tidsvariabel med som innehåller den tid körningen av `AWMF()` tog. Matlabkommandona `tic/toc` användes för att beräkna tiden.

```

tic
...
time = toc;

```

6.2 Kantproblem

I och med att modellen arbetar med pixlar och deras närliggande område så kommer det uppstå problem vid kanterna. Lösningen på detta problem diskuteras inte i artikeln så tanken är att vi kopierar kantvärdena rakt ut från bilden. Denna lösning av kantproblematiken föreslogs av beställaren då den ofta används på ContextVision [6]. Detta medför att de K yttersta pixlarna på inbilden inte kommer att filtreras på samma sätt som övriga delar av bilden. Genom att använda Matlab funktionen `imfilter()` och parametern `replicate` löses detta vid faltningar. Övriga tillfällen när detta problem uppstår har ursprungs matriserna utökas med K extra rader och kolumner åt alla håll. Exempel hur detta genomfördes visas i nedanstående kod

```
% Makes extended InPic
tmpInPicExt = [ repmat(inPic(:,1),1,K) inPic ...
               repmat(inPic(:,end),1,K) ];
inPicExt = [ repmat(tmpInPicExt(1,:),K,1); tmpInPicExt; ...
            repmat(tmpInPicExt(end,:),K,1) ];
clear tmpInPicExt;
```

Se även avsnitt 6.1.3 för hur `imfilter()` implementerades.

6.3 Iteration

Författarna [3] diskuterar inte, varken i modellbeskrivningen eller någon annan del av artikeln, om man bör iterera algoritmen eller vinsten av en sådan iteration. Därav kommer denna implementation endast att genomföra en iterering.

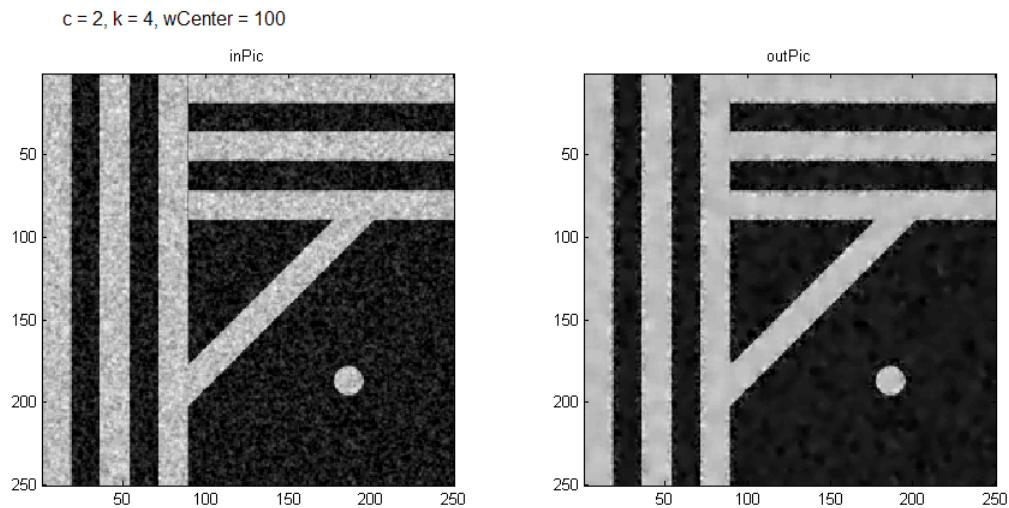
6.4 Resultat

Artikelförfattarna [3] påstår att *Adaptive Weighted Median Filter* är bra på reducera brus samt att bevara kanter och andra detaljer i ultraljudsbilder. Bilderna i artikeln är av så pass dålig kvalitet att detta inte kan verifieras direkt och de resultat vi kommit fram till visar att AWMF inte är att rekommendera på ultraljudsbilder. Största anledningen till detta är att allt som inte tolkas som skarpa kanter i stor sett kommer att medianfiltreras. Detta fungerar bra på de ytor som inte innehåller några detaljer men övriga delar blir suddiga och det är svårt att urskilja variationer av intensitet. Ytterligare anledning till att överväga andra tekniker istället för AWMF är tidsåtgången. Med denna implementering tar det i storleksordningen 8 sekunder på referensdatorn att filtrera en (177×712) stor inbild med parameterinställningarna $c = 20$, $K = 4$ och $w_{center} = 99$. En utskrift av Matlabs tidsanalys `profile()` finns i avsnitt C.3.

Tittar man på vår enklare testbild ser man att AWMF klarar av att filtrera bort brus och behålla kanterna, se figur 28.

Använder man AWM-filtrering på ultraljudsbilder med speckel ser man att filtreringen inte räcker till för att, enligt oss, ge ett acceptabelt resultat. Figur 30 visar resultatet med de parameterinställningar som artikeln använder. Visserligen varierar parametervärdena mellan olika bilder men i storleksordning är de $c = 20$, $w_{center} = 99$ och $K = 4$. I figur 29 visas återigen inbilden och i figur 30 den utbild som erhålls.

Förmodligen är specklet större i den testbild vi har försökt förbättra än den bild artikelförfattarna [3] har utgått ifrån. Om man ser på vår inbild så ligger specklet i storleksordning 10 – 15



Figur 28: Före och efter AWM-filtrering av testbild.



Figur 29: Inbild.

Adaptive Weighted Median Filter: Scalar constant=20, Size=4, Center weight=99

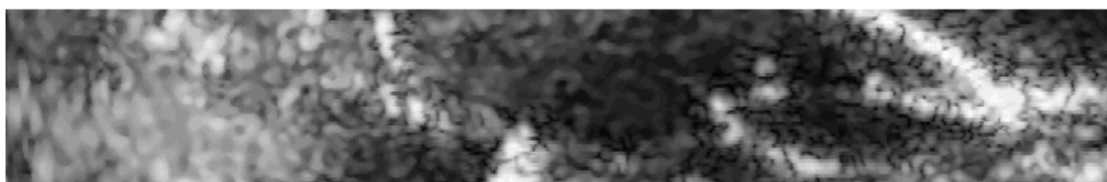


Figur 30: Utbild ($c = 20, w_{\text{center}} = 99$ och $K = 4$).

pixlar vilket, enligt oss, borde resultera i ett filter som minst har storleken (15×15) för att spekket inte ska ge kantutslag. I och med denna ändring av inparametern behövde vi även justera parametrarna c och w_{center} . I detta fall räckte det med att w_{center} justerades tills att så

mycket som möjligt av inbilden filtrerades utan att detaljer försvann. Det som händer när man justerar c och w_{center} kan grovt beskrivas som att man väljer hur "nära" kanter som man ska utföra ren medianfiltrering och sedan börja vikta centerpixeln högre (viktrade medianfiltrering) och på så sätt få mindre filtrering. Det nya utresultatet, se figur 31, är enligt oss bättre filtrerat på de mörka områdena och har bevarat kanterna bättre än det resultat som erhöles vid figur 30 men det har tillförts nya falska konturer samt att tidsåtgången ökade till 35 sekunder.

Adaptive Weighted Median Filter: Scalar constant=20, Size=14, Center weight=600



Figur 31: Utbild ($c = 20$, $w_{center} = 600$ och $K = 14$).

Utbilden är enligt oss inte ett bra resultat utan för mycket utsmetad i intressanta områden och nästan ingen brusreducering vid kanter. Visserligen kan man variera parametrarna och få något bättre detaljbevaring men då minskar brusreduceringen. Det vi saknar är en mjukare övergång av "filtreringsgraden" vid detaljer i bilden och att intensitetsvariationerna vid ljusa områden inte smetas ut i den grad den nu gör.

Vi valde att inte implementera något i C då utbildresultatet var av så dålig kvalitet och tidvinsten förmodligen inte skulle bli så pass bra att det närmar sig en realtidsimplementation. Vi har även provat att skriva om funktionen så att den i större utsträckning jobbar med matrisoperationer istället för loopar utan att lyckas förbättra tiden nämnvärt. Då artikeln skrevs, 1989, säger även artikelförfattarna att AWMF inte är en realtidsimplementation.

Ser man på ekvation 46 tycker vi att det finns ett samband mellan parametrarna c och w_{center} . Om man delar uttrycket med c fås

$$\frac{w(i,j)}{c} = \left[\frac{w_{center}}{c} - d \times \frac{\sigma^2}{m} \right] \quad (49)$$

där man kan se att det endast är kvoten $\frac{w_{center}}{c}$ som styr hur vikterna ska räknas fram. Visserligen så skalas $w(i,j)$ men detta görs för alla vikter och bör inte spela någon roll för utresultatet. Vi har konfirmerat detta genom att filtrera med $c = 20$ och $w_{center} = 600$ ($\frac{w_{center}}{c} = 30$) samt $c = 10$ och $w_{center} = 300$ ($\frac{w_{center}}{c} = 30$) utan att kunna se någon skillnad på utbilderna. Utöver den visuella bekräftningen såg vi även på skillnaderna mellan utbilderna genom

```
sum(sum(abs(outPic1-outPic2)))
```

vilket returnerade 0.

Referenser

- [1] Abd-Elmoniem, Youssef, Kadah “Real-Time Speckle Reduction and Coherence Enhancement in Ultrasound Imaging via Nonlinear Anisotropic Diffusion”, *IEEE Transactions on biomedical engineering*, vol 49 (#9, sep 2002): sid 997–1013.
- [2] Zong, Laine, Geiser “Speckle Reduction and Contrast Enhancement of Echocardiograms via Multiscale Nonlinear Processing”, *IEEE Transactions on medical imaging*, vol 17 (#4, aug 1998): sid 532–540.
- [3] Loupas, McDicken, Allan “An Adaptive Weighted Median Filter for Speckle Suppression in Medical Ultrasonic Images”, *IEEE Transactions on circuits and systems*, vol 36 (#1, jan 1989): sid 129–135.
- [4] Mallat, Zhong “Characterization of signals from multiscale edges”, *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 14 (jul 1992): sid 710–732.
- [5] Joachim Weickert, Bart M. ter Haar Romeny, Max A. Viergever “Efficient and Reliable Schemes for Nonlinear Diffusion Filtering”, *IEEE Trans. on Image Processing*, vol. 7 (#3 march 1998): sid 398–410.
- [6] Gunnar Farnebäck, ContextVision AB, Linköping, 013-358552, `gunnar.farneback@contextvision.se`

APPENDIX

A Licens

Hela systemet, inklusive all kod, är licensierad under en BSD-licens, som liknar texten nedan. Denna text är endast avsedd som en vägledning för läsaren att bilda sig en uppfattning om hur licensen ser ut. Om texten nedan skiljer sig från texten i `readme.txt` ska den sistnämnda gälla.

Licensen kan kort sammanfattas som att det är fritt att kopiera och sälja produkten eller delar av produkten i modifierad eller omodifierad form så länge licensen reproduceras.

```
* Copyright (c) 2007,  
* Erik Mellström, Erik Ringaby, Per Thyr, Alexander Tuttle and Henrik Wolkesson  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
* modification, are permitted provided that the following conditions are met:  
*   * Redistributions of source code must retain the above copyright  
*     notice, this list of conditions and the following disclaimer.  
*   * Redistributions in binary form must reproduce the above copyright  
*     notice, this list of conditions and the following disclaimer in the  
*     documentation and/or other materials provided with the distribution.  
*   * The names of the contributors may not be used to endorse or promote  
*     products derived from this software without specific prior written  
*     permission.  
*  
* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS ‘‘AS IS’’ AND ANY  
* EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED  
* WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE  
* DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDERS BE LIABLE FOR ANY  
* DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES  
* (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;  
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND  
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT  
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS  
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

B Kod

B.1 Kod för delsystem 1: GUI

B.1.1 btn_run_Callback() i EDGY.m

Denna funktion finns i gui/EDGY.m och anropas när användaren startar algoritmen från GUI.

```

1 function btn_run_Callback(hObject, eventdata, handles)
2 global algorithm;
3 % hObject    handle to btn_open_run (see GCBO)
4 % eventdata  reserved - to be defined in a future version of MATLAB
5 % handles    structure with handles and user data (see GUIDATA)
6
7 userData = get(handles.figure1, 'UserData');
8 if (userData.running == 1)
9     return;
10 end
11 setBusy(handles, 1);
12
13 imPath = get(handles.editOpen, 'String');
14 try
15     if (isempty(imPath))
16         error('Please enter a file name.');
17     end
18     try
19         inImage = imread(imPath);
20     catch
21         errorTitle = 'Error when reading file';
22         rethrow(lasterror);
23     end
24
25     if isequal(algorithm, 'diffusion')
26         parameters = struct('editSigma1', 'sigma1', ...
27                             'editDelta1', 'delta1', ...
28                             'editSigma2', 'sigma2', ...
29                             'editDelta2', 'delta2', ...
30                             'editS2', 's2', ...
31                             'editAlpha', 'alpha', ...
32                             'editBeta', 'beta', ...
33                             'editTau', 'tau', ...
34                             'editNumIter', 'numIter', ...
35                             'checkboxAOS', 'aos', ...
36                             'anisCheckGraphs', 'graphs');
37         algInfo = struct('desc', 'Anisotropic diffusion', ...
38                         'func', 'anisotropicDiffusion');
39     elseif isequal(algorithm, 'wavelet')
40         parameters = struct('editJ', 'j', ...
41                             'editWalpha', 'alpha', ...
42                             'editTmin', 'Tmin', ...
43                             'editTmax', 'Tmax', ...
44                             'editSigma', 'sigma', ...
45                             'editT1', 'T1', ...
46                             'editT2', 'T2', ...
47                             'editT3', 'T3', ...
48                             'editCwavelet', 'c', ...
49                             'editB', 'b', ...
50                             'waveCheckGraphs', 'graphs', ...
51                             'checkSampling', 'sampling');
52         algInfo = struct('desc', 'Wavelets', ...

```

```
53                                     'func', 'multiScaleEnh');
54 elseif isequal(algorithm, 'median')
55     parameters = struct('editCmedian', 'Scalar constant', ...
56                         'editK', 'Size', ...
57                         'editWcenter', 'Center weight', ...
58                         'awmfCheckGraphs', 'graphs');
59     algInfo = struct('desc', 'Adaptive Weighted Median Filter', ...
60                     'func', 'AWMF');
61 else
62     errorTitle = 'Error in GUI';
63     error('Please select an algorithm.');
```

```
64 end
65 DoResult(algInfo, inImage, handles, parameters);
66 catch
67     err = lasterror;
68     message = err.message;
69     tIndex = find(double(message)==10);
70     if (numel(tIndex) >= 1)
71         message = message(tIndex(end)+1:end);
72     end
73     if (~exist('errorTitle'))
74         errorTitle = 'Error';
75     end
76
77     errordlg(message, errorTitle);
78     setBusy(handles, 0);
79 end
```

B.1.2 DoResult() i EDGY.m

Denna funktion finns i gui/EDGY.m och anropas från btn_run_Callback().

```

1 function DoResult(algorithm, inImage, handles, parameters)
2
3 titleStr = '';
4 toRun = '';
5 parameterNames = fieldnames(parameters);
6 for i=1:length(parameterNames)
7     tID = char(parameterNames(i));
8     tName = eval(['parameters.', tID]);
9     tStyle = char(eval(['get(handles.', tID, ', ''Style'')']));
10    if (strcmp(tStyle, 'edit'))
11        tValue = eval(['get(handles.', tID, ', ''String'')']);
12        if (isempty(str2num(tValue)))
13            error(['Parameter ', tName, ' is missing or invalid.']);
14        end
15        if (strcmp(titleStr, ''))
16            titleStr = [algorithm.desc, ': ', tName, '=', tValue];
17            toRun = [algorithm.func, '(inImage, [' , tValue, '])'];
18        else
19            titleStr = [titleStr, ', ', tName, '=', tValue];
20            toRun = [toRun, ', [' , tValue, '])'];
21        end
22    elseif (strcmp(tStyle, 'checkbox'))
23        tValue = eval(['get(handles.', tID, ', ''Value'')']);
24        tMax = eval(['get(handles.', tID, ', ''Max'')']);
25        tChecked = (tValue == tMax);
26        if (strcmp(titleStr, ''))
27            if (isempty(strfind(tID, 'CheckGraphs')))
28                % Lägg inte till graphs i rubriken
29                titleStr = [algorithm.desc, ': ', tName, '=', ...
30                    num2str(tChecked)];
31            end
32            toRun = [algorithm.func, '(inImage, ', num2str(tChecked)];
33        else
34            if (isempty(strfind(tID, 'CheckGraphs')))
35                % Lägg inte till graphs i rubriken
36                titleStr = [titleStr, ', ', tName, '=', num2str(tChecked)];
37            end
38            toRun = [toRun, ', ', num2str(tChecked)];
39        end
40    end
41 end
42 toRun = [toRun, ');'];
43
44 dirIndex = get(handles.popupDirection, 'Value');
45 if (dirIndex == 1)
46     %top to bottom
47     %ignore
48 elseif (dirIndex == 2)
49     %left to right
50     inImage = rot90(inImage, -1);
51 elseif (dirIndex == 3)
52     %right to left
53     inImage = rot90(inImage, 1);
54 elseif (dirIndex == 4)
55     %bottom to top
56     inImage = rot90(inImage, 2);
57 end

```

```

58
59 eval(['outImage, time] = ', toRun]);
60 time = round(time*1000);
61
62 titleStr = [titleStr, ', time: ', num2str(time), ' ms'];
63
64 if (get(handles.checkScanConv, 'Value') == get(handles.checkScanConv, 'Max'))
65
66     r0 = str2num(get(handles.editRadiusMin, 'String'));
67     r1 = str2num(get(handles.editRadiusMax, 'String'));
68     deltaAngle = str2num(get(handles.editDeltaAngle, 'String'));
69
70     if (isempty(r0) || isempty(r1) || isempty(deltaAngle))
71         error('All scan conversion parameters must be entered correctly.');
```

```

72     end
73     angle = deltaAngle*(size(inImage,2)-1)
74     if (angle >= 180)
75         error(['Delta angle (', deltaAngle, ...
76             ') times number of sticks (', size(inImage,2), ...
77             ') exceeds 180 degrees.']);
78     end
79
80     showIn = PolarToCartesian(double(inImage), r0, r1, angle);
81     showOut = PolarToCartesian(double(outImage), r0, r1, angle);
82 else
83     showIn = inImage;
84     showOut = outImage;
85 end
86
87 rangeIndex = get(handles.popupRange, 'Value');
88 if (rangeIndex == 1)
89     %input image
90     limits = [min(double(showIn(:))) max(double(showIn(:)))];
91 elseif (rangeIndex == 2)
92     %no cutoff
93     limits = [min(min(double(showIn(:))), min(double(showOut(:))), ...
94         max(max(double(showIn(:))), max(double(showOut(:))))]);
95 end
96
97 userData = struct('titleStr', titleStr, ...
98     'inPic', showIn, ...
99     'outPic', showOut, ...
100     'showingOutput', 0, ...
101     'limits', limits);
102 ResultWindow('UserData', userData);
103 setBusy(handles, 0);

```


B.1.3 PolarToCartesian()

Denna funktion finns i `gui/PolarToCartesian.m` och kallas från `DoResult()`. Den utför eventuell *Scan conversion* (om användaren så önskar).

```

1 function cartesian = PolarToCartesian(polar, r0, r1, v0deg, method)
2 %POLARTOCARTESIAN Converts polar data to cartesian representation
3 %   CARTESIAN = POLARTOCARTESIAN(POLAR, R0, R1, V0, METHOD) Takes one
4 %   image, represented in polar coordinates, and converts it to a
5 %   representation of cartesian coordinates.
6 %
7 %   POLAR is the input image. It must be at least 2x2. The stick direction
8 %   must be "top to bottom" in the matrix.
9 %
10 %   Class support for input POLAR:
11 %       single, double
12 %
13 %   R0 is the distance from the probe to the first row in POLAR, measured
14 %   in units. One unit is the distance between two adjacent rows in POLAR.
15 %
16 %   Class support for input R0:
17 %       single, double, *int*
18 %
19 %   R1 is the distance from the probe to the last row in POLAR, measured
20 %   in the same unit as R0.
21 %
22 %   Class support for input R1:
23 %       single, double, *int*
24 %
25 %   V0 is the field of vision, i.e. the angle between the two outermost
26 %   columns in POLAR. The angle is measured in degrees. It must be positive
27 %   and less than 180 degrees.
28 %
29 %   Class support for input V0:
30 %       single, double, *int*
31 %
32 %   The interpolation is made using METHOD interpolation. METHOD must be
33 %   either 'linear', 'cubic' or 'spline'. If the METHOD argument is
34 %   omitted, linear interpolation will be assumed.
35 %
36 %   Class support for input METHOD:
37 %       char
38 %
39 %   Licensed under BSD as a part of EDGY project source code,
40 %   see readme.txt file. For more information see project documentation.
41 %
42 %   Revision: 1.0
43 %   Date: 2007/05/09
44 %   Author: Erik Mellström
45
46 if (nargin == 4)
47     method = 'linear';
48 end
49
50 if (nargin ~= 4 && nargin ~= 5)
51     error('Bad number of input arguments. Should be 4-5.');
```

```

57     error('Input image must be a single or double matrix.');
```

```

58 elseif (~isa(polar,'single') && ~isa(polar,'double'))
59     error('Input image must be a single or double matrix.');
```

```

60 elseif (~IsScalar(r0))
61     error('Minimal radius must be a numeric scalar.');
```

```

62 elseif (r0 < 0)
63     error('Minimal radius cannot be negative.');
```

```

64 elseif (~IsScalar(r1))
65     error('Maximal radius must be a numeric scalar.');
```

```

66 elseif (r1 <= r0)
67     error('Maximal radius must be greater than minimal radius.');
```

```

68 elseif (~IsScalar(v0deg))
69     error('The angle must be a numeric scalar.');
```

```

70 elseif (v0deg <= 0 || v0deg >= 180)
71     error('The angle must be positive and less than 180 degrees.');
```

```

72 elseif (~strcmp(method, 'linear') && ~strcmp(method, 'cubic') && ...
73         ~strcmp(method, 'spline'))
74     error('Bad value of interpolation method argument.');
```

```

75 end
76
77 if (nargout ~= 1)
78     error('Bad number of output arguments. Should be one.');
```

```

79 end
80
81 [inRows, inCols] = size(polar);
82
83 v0 = v0deg*pi/180;
84 tAngles = linspace(-v0/2, v0/2, inCols);
85 tRadius = linspace(r0, r1, inRows);
86 [vOrig, rOrig] = meshgrid(tAngles, tRadius);
87
88 rUnit = (r1-r0)/inRows;
89
90 % Calculate maximal image width using sine law
91 imgWidth = 1:ceil(sin(v0)*ceil(r1/rUnit)/sin((pi-v0)/2));
92 imgWidth = (imgWidth-mean(imgWidth))*rUnit;
93
94 % Calculate maximal image height using cosine for cutting upper part
95 imgHeight = r0*cos(v0/2):rUnit:r1;
96
97 % Generate mesh grid for interpolation
98 [x, y] = meshgrid(imgWidth, imgHeight);
99 r = sqrt(x.^2+y.^2);
100 v = atan(x./y);
101
102 cartesian = interp2(vOrig, rOrig, polar, v, r, method);
103 cartesian(cartesian==NaN) = min(min(cartesian));
104 end

```

B.2 Kod för delsystem 2: Anisotrop diffusion

B.2.1 AnisotropicDiffusion.m

Denna är huvudfunktionen för delsystem 2.

```
1 function [I, timeSpent] = AnisotropicDiffusion( inImage, sigma1, delta1,...
2     sigma2, delta2, s2, alpha, beta, tau, numIter, useAOS, graphs )
3 %AnisotropicDiffusion Implements Nonlinear Anisotropic Diffusion
4 % [I, timeSpent] = AnisotropicDiffusion( inImage, sigma1, delta1,
5 %     sigma2, delta2, s2, alpha, beta, tau, useAOS, numIter, useAOS, graphs )
6 %
7 % Returns:
8 % I          Processed Image after selected iterations
9 % timeSpent  Time spent processing exclusive creation of kernels etc.
10 %
11 % Parameters:
12 % inImage    Image to process. Must be gray scale and type double.
13 % sigma1     Sigma for derivating kernel
14 % delta1     Cut of tail of derivating kernel at this value
15 % sigma2     Sigma for low pass kernel
16 % delta2     Cut of tail of low pass kernel at this value
17 % s2         Stop value for diffusion in gradient direction
18 % alpha      Isotropic diffusion number
19 % beta       Maximum amount of diffusion. Small beta allows big changes
20 % tau        Time step
21 % numIter    Maximum number
22 % useAOS     Choose to use an Additive Operatorsplitting Scheme instead
23 %            of a divergence operator
24 % graphs     1 - Show graphs, 0 - Show no graphs
25 %
26 % Requirements IMFILTER and IMSHOW in Image Processing Toolbox
27 % See also IMFILTER, IMSHOW.
28 %
29 % Licensed under BSD as a part of EDGY project source code,
30 % see readme.txt file. For more information see project documentation.
31 %
32 % Revision: 1.0
33 % Date: 2007/05/09
34 % Author: Henrik Wolkeesson
35
36 if ~IsScalar(sigma1) || sigma1 < 0 || ...
37     ~IsScalar(delta1) || delta1 < 0 || ...
38     ~IsScalar(sigma2) || sigma2 < 0 || ...
39     ~IsScalar(delta2) || delta2 < 0
40     error('Sigma1, Sigma2, Delta1 and Delta2 must be a positive scalar');
41 end
42 if ~IsScalar(s2) || s2 < 0 || ~IsScalar(alpha) || alpha < 0 || ...
43     ~IsScalar(beta) || beta < 0 || ~IsScalar(tau) || tau < 0
44     error('s2, alpha, beta and tau must be a positive scalar');
45 end
46 if ~IsPositiveInteger(numIter)
47     error('numIter must be positive integer');
48 end
49 if useAOS < 0 || useAOS > 1 || round(useAOS) ~= useAOS
50     error('useAOS must be either 0 or 1');
51 end
52 if graphs < 0 || graphs > 1 || round(graphs) ~= graphs
53     error('Graphs must be either 0 or 1');
54 end
55
```

```

56 % Section 1: Initize
57 inImage = double(inImage);
58 I=inImage;
59 N = size(I,1);
60 M = size(I,2);
61 nablaI=zeros(N,M,2);
62 if (useAOS == 1)
63     unitMatrixM = sparse(eye(M));
64     unitMatrixN = sparse(eye(N));
65 end
66
67 % Section 2: Create kernels
68 derivKernel = createDerivKernel(sigma1, delta1);
69 Kx = createGaussKernel(sigma2,delta2);
70
71 if (graphs == 1)
72     figure('Name','Kernels');
73     subplot(1,2,1);plot(Kx);
74     title(['Gauss Kernel (Size: ' num2str(size(Kx,2)) ')']);
75     subplot(1,2,2);plot(derivKernel);
76     title(['Derivating Kernel (Size: ' num2str(size(derivKernel,2)) ')']);
77 end;
78
79 if (size(Kx,2)>size(I,2))
80     warning('Deriviting kernel is bigger than image');
81 end
82
83 tic;
84 for k = 1:numIter
85     % Section 3: Calculate derivatives
86     I_y=imfilter(I,derivKernel, 'conv', 'replicate');
87     I_x=imfilter(I,derivKernel, 'conv', 'replicate');
88
89     % Section 4: Create structure matrix
90     nablaI(:, :, 1)=I_x.*I_x;
91     nablaI(:, :, 2)=I_x.*I_y;
92     nablaI(:, :, 3)=I_y.*I_y;
93
94     % Section 5: LP filter structure matrix
95     Ky = Kx';
96     J = imfilter(imfilter(nablaI,Kx, 'conv', 'replicate'), ...
97         Ky, 'conv', 'replicate');
98
99     % Section 6-8: Eigenanalays, modulation and construction of D
100    [a, b, c, d] = fastD(J, alpha, beta, s2, inImage,I);
101
102    % Section 9: Solve diffusion ekvation
103    if (useAOS == 0)
104        I = I + divergenceC(a.*I_x + b.*I_y, c.*I_x + d.*I_y);
105    else
106        InextN = solveAOSN(I, N, M, a, b, c, d, tau, unitMatrixN); % Solves row by row
107        InextM = solveAOSM(I, N, M, a, b, c, d, tau, unitMatrixM); % Solves col by col
108        I = InextM + InextN;
109    end
110
111    % Optional plotting
112    if (graphs == 1)
113        figure;
114        subplot(3,1,1);imshow(I, []);title(['Iteration ' num2str(k)]);
115        title(['I of iterataion ' num2str(k)]);
116        subplot(3,1,2);imshow(I_x, []);
117        title(['Derivate dI/dx of iterataion ' num2str(k)]);

```

```

118         subplot(3,1,3);imshow(I_y, []);
119         title(['Derivate dI/dy of iterataion ' num2str(k)]);
120         figure('Name', ['LP-filtered Tensor J of iterataion ' num2str(k)]);
121         subplot(3,1,1);imshow(J(:, :, 1), []);
122         title('dI/dx*dI/dx');
123         subplot(3,1,2);imshow(J(:, :, 2), []);
124         title('dI/dx*dI/dy');
125         subplot(3,1,3);imshow(J(:, :, 3), []);
126         title('dI/dy*dI/dy');
127     end
128 end
129
130 % Optional Plotting
131 if (graphs == 1)
132     figure;plot(inImage(floor(N/2), :), 'r');hold on; plot(I(floor(N/2), :), 'b');
133     legend('Original with Speckle', 'Processed');
134 end
135
136 % Section 10: Return time and processed image
137 timeSpent = toc;
138 I = uint8(I);

```

B.2.2 createDerivKernel.m

Denna funktion skapar en deriverande gausskärna.

```
1 function kernel = createDerivKernel(sigma,delta)
2 %CREATEDERIVKERNEL Returns an iteratively created derivating Gaussian
3 % kernel with given sigma trunkaded at a level selected by delta.
4 %
5 % Note: Parameters are not checked if valid of reasons of optimization
6
7 if (sigma == 0)
8     kernel = 0;
9     return;
10 end
11
12 if (delta > 1)
13     error('Delta is truncating limit and should be small');
14 end
15
16 N=1;
17 while 1
18     k=(-N:N);
19     kernel=-k/sigma^2.*exp(-k.^2/(2*sigma^2));
20     kernel=kernel./sum(k.*kernel(end:-1:1));
21     if (abs(kernel(2*N+1)) <= delta)
22         break;
23     else
24         N = N + 1;
25     end
26 end
```

B.2.3 createGaussKernel.m

Denna funktion skapar en gausskärna.

```
1 function kernel = createGaussKernel(sigma,delta)
2 %CREATEGAUSSKERNEL Returns an iteratively created gaussian kernel with
3 %   given sigma trunkaded at a level selected by delta.
4 %
5 %   Note: Parameters are not checked if valid of reasons of optimization
6
7 if (sigma == 0)
8     kernel = 0;
9     return;
10 end
11
12 if (delta > 1)
13     error('Delta is truncating limit and should be small');
14 end
15
16 N=1;
17 k=0;
18 while 1
19     k=[k N];
20     kernel=exp(-k.^2/(2*sigma.^2));
21
22     if (kernel(end) < delta)
23         break;
24     else
25         N = N + 1;
26     end
27 end
28
29 kernel = [kernel(end:-1:2) kernel];
30 kernel = kernel ./ sum(kernel);
```

B.2.4 fastD.c

Denna funktion beräknar D optimerat i c-kod. Kompileras för användning i Matlab med `mex fastD.c`.

```

1  /* =====
2  * EDGY - Computes D out of J
3  * fastD.c
4  *
5  * Description: This is a MEX-file for MATLAB. Computes eigenvalues and
6  * eigenvectors of a symmetric 2 by 2 matrix [ 11 12; 21 22 ] represented
7  * as [ 11 12 22 ]. These are weighted according to the paper and
8  * eventually a diffusion matrix is calculated.
9  *
10 * Note: The name of the function will be the name of this file.
11 * To compile run: mex fastD.c
12 * In Matlab use:
13 * [a, b, c, d] = fastEigC(J, alpha, beta, s2, I_0, I);
14 *
15 * Programmer: Henrik Wolkesson (henwo442@student.liu.se)
16 *
17 * Licensed under BSD as a part of EDGY project source code,
18 * see readme.txt file. For more information see project documentation.
19 *
20 * Revision: 1.0 (2007-05-09)
21 * =====
22 */
23
24 #include <mex.h>
25 #include <math.h>
26
27 void fastEig(double* J, double* Mu, double* W, int nfields)
28 {
29     int nPixels, nPixels2, nPixels3;
30     double rp,lp;
31     double *startJ, *startMu, *startW;
32
33     /* Store initial pointers for later resetting */
34     startJ = J;
35     startMu = Mu;
36     startW = W;
37
38     nPixels = nfields/3;
39     nPixels2 = 2 * nPixels;
40     nPixels3 = 3 * nPixels;
41
42     while(J<startJ+nPixels)
43     {
44         /* Already diagonalized elements gets special treatment */
45         if (*(J+nPixels) == 0)
46         {
47             *(Mu+nPixels3) = *(J+nPixels2);
48             *Mu = *J;
49             *W = 1;
50             *(W+nPixels) = 0;
51             *(W+nPixels2) = 0;
52             *(W+nPixels3) = 1;
53         }
54         else
55         {
56             /* Calculate eigenvalues */
57             lp = *J + *(J+nPixels2);

```



```

58         rp = sqrt ((*J - *(J+nPixels2)) * (*J - *(J+nPixels2)) + 4 * *(J+nPixels) * *(J+nPixels));
59         *Mu = 0.5 * (lp + rp);
60         *(Mu+nPixels3) = 0.5 * (lp - rp);
61
62     /* Calculate eigenvectors */
63     *W = -*(J+nPixels) / sqrt ((*Mu-*J) * (*Mu-*J) + *(J+nPixels) * *(J+nPixels));
64     *(W+nPixels) = (*Mu-*J) / *(J+nPixels) * *W;
65
66     *(W+nPixels2) = *(J+nPixels) / sqrt ((*Mu+nPixels3)-*J) * ((*Mu+nPixels3)-*J) + *(J+nPixels) *
67     *(W+nPixels3) = ((*Mu+nPixels3)-*J) / *(J+nPixels) * *(W+nPixels2);
68     }
69     J++;
70     Mu++;
71     W++;
72 }
73
74 /* Reset pointers */
75 J = startJ;
76 Mu = startMu;
77 W = startW;
78 }
79
80
81 void mexFunction(int nlhs, mxArray *plhs[], int nrhs,
82 const mxArray *prhs[])
83 {
84     double *Mu, *J, *W;
85     int mrows, ncols, nfields, nPixels;
86     int dims[4];
87     double alpha, beta, delta, lambda1, lambda2, s2;
88     double *originalImage, *nowImage;
89     double *a,*b,*C,*d;
90     double *W11,*W12,*W21,*W22;
91     mxArray *pMu,*pW;
92
93     /* Check for proper number of arguments. */
94     if (nrhs != 6) {
95         mexErrMsgTxt("Six inputs required.");
96     } else if (nlhs != 4) {
97         mexErrMsgTxt("Exactly four outputs required");
98     }
99
100     /* No check of arguments since optimization is preferred */
101
102     /* Create matrix for the return argument. */
103     mrows = mxGetM(prhs[0]);
104     ncols = mxGetN(prhs[0]);
105     dims[0] = mrows;
106     dims[1] = ncols/3;
107     dims[2] = 2;
108     dims[3] = 2;
109     pMu = mxCreateNumericArray(4,dims,mxDOUBLE_CLASS,mxREAL);
110     pW = mxCreateNumericArray(4,dims,mxDOUBLE_CLASS,mxREAL);
111     plhs[0] = mxCreateDoubleMatrix(mrows, ncols/3 , mxREAL);
112     plhs[1] = mxCreateDoubleMatrix(mrows, ncols/3 , mxREAL);
113     plhs[2] = mxCreateDoubleMatrix(mrows, ncols/3 , mxREAL);
114     plhs[3] = mxCreateDoubleMatrix(mrows, ncols/3 , mxREAL);
115
116     nfields = mxGetNumberOfElements(prhs[0]);
117
118     /* Assign pointers to matrixes */
119     W = mxGetPr(pW);

```

```

120     Mu = mxGetPr(pMu);
121
122     /* Assign pointers to each input and output. */
123     J = mxGetPr(prhs[0]);
124     alpha = *mxGetPr(prhs[1]);
125     beta = *mxGetPr(prhs[2]);
126     s2 = *mxGetPr(prhs[3]);
127     originalImage = mxGetPr(prhs[4]);
128     nowImage = mxGetPr(prhs[5]);
129     a = mxGetPr(plhs[0]);
130     b = mxGetPr(plhs[1]);
131     c = mxGetPr(plhs[2]);
132     d = mxGetPr(plhs[3]);
133
134     // Section 6 //////////
135     fastEig(J,Mu,W,nfields);
136     //////////
137
138     /* Initialize pointers */
139     nPixels = nfields/3;
140     W11 = W;
141     W12 = W+nPixels;
142     W21 = W+2*nPixels;
143     W22 = W+3*nPixels;
144
145     /* Loop the image */
146     while(W11 < W+nPixels)
147     {
148         // Section 7 //////////
149         delta = 1 - beta*(fabs(*(originalImage++) - *(nowImage++)));
150         if (delta<0) {delta = 0;}
151
152         if ((*Mu-(Mu+3*nPixels))*(*Mu-(Mu+3*nPixels)) <= s2)
153         {
154             lambda1 = alpha*delta*(1-((*Mu-(Mu+3*nPixels))*(*Mu-(Mu+3*nPixels)))/s2);
155         }
156         else{lambda1 = 0;}
157         lambda2 = alpha * delta;
158         //////////
159
160         // Section 8 //////////
161         *(a++) = lambda1 * *W11 * *W11 + lambda2 * *W21 * *W21;
162         *b = lambda1 * *W11 * *W12 + lambda2 * *W21 * *W22;
163         *(c++) = *(b++);
164         *(d++) = lambda1 * *W12 * *W12 + lambda2 * *W22 * *W22;
165
166         W11++;
167         W12++;
168         W21++;
169         W22++;
170         Mu++;
171         //////////
172     }
173
174 }

```

B.2.5 solveAOSM.m

Denna funktion beräknar vår additive operator splitting scheme x-led.

```

1 function InextM = solveAOSM(I, N, M, a, b, c, d, tau, unitMatrixM)
2 %solveAOSM Calculates the additive operator splitting scheme row by row
3 %
4 % Parameters:
5 % N, number of rows
6 % M, number of columns
7 % a,b,c,d from Diffusion Matrix
8 %
9 % Note: Parameters are not checked if valid for reasons of optimization
10
11 % Create matrixes
12 AM = zeros(M, M, 2);
13 InextM = zeros(size(I));
14 JnextM = zeros(1, M);
15
16 % Starts at 2 row and column and ends one to early to avoid corner effects
17 for iRow = 2 : N - 1
18     for iCol = 2: M - 1
19         JnextM(iCol) = I(iRow, iCol) ...
20             + tau/4 * (c(iRow,iCol+1) * (I(iRow+1,iCol+1) ...
21                 - I(iRow-1,iCol+1)) - c(iRow,iCol-1) * (I(iRow+1,iCol-1) - ...
22                 I(iRow-1,iCol-1))) + tau/4 * (b(iRow+1,iCol) * ...
23                 (I(iRow+1,iCol+1) - I(iRow+1,iCol-1)) - b(iRow-1,iCol) * ...
24                 (I(iRow-1,iCol+1)-I(iRow-1,iCol-1)));
25
26         % Left in tridiagonal matrix
27         AM(iCol, iCol-1, 1) = (a(iRow, iCol) + a(iRow, iCol- 1)) / 2;
28         % Right in tridiagonal matrix
29         AM(iCol, iCol+1, 1) = (a(iRow, iCol) + a(iRow, iCol+ 1)) / 2;
30         % Center in tridiagonal matrix
31         AM(iCol, iCol, 1) = -((a(iRow, iCol) + a(iRow, iCol - 1)) + ...
32             (a(iRow, iCol) + a(iRow, iCol + 1))) / 2;
33
34     end
35     InextM(iRow,:) = 1/2*((unitMatrixM - 2*tau*sparse(AM(:,:,1))\JnextM');
36 end

```

B.2.6 solveAOSN.m

Denna funktion beräknar vår additive operator splitting scheme y-led.

```

1 function InextN = solveAOSN(I, N, M, a, b, c, d, tau, unitMatrixN)
2 %solveAOSM Calculates the additive operator splitting scheme column by
3 % column
4 %
5 % Parameters:
6 % N, number of rows
7 % M, number of columns
8 % a,b,c,d from Diffusion Matrix
9 %
10 % Note: Parameters are not checked if valid for reasons of optimization
11
12 % Create matrixes
13 AN = zeros(N, N, 2);
14 InextN = zeros(size(I));
15 JnextN = zeros(N, 1);
16
17 % Starts at 2 row and column and ends one to early to avoid corner effects
18 for iCol = 2 : M - 1
19     for iRow = 2 : N - 1
20         JnextN(iRow) = I(iRow, iCol) ...
21             + tau/4 * (c(iRow,iCol+1) * (I(iRow+1,iCol+1) ...
22                 - I(iRow-1,iCol+1)) - c(iRow,iCol-1) * (I(iRow+1,iCol-1) ...
23                 - I(iRow-1,iCol-1))) + tau/4 * (b(iRow+1,iCol) * ...
24                 (I(iRow+1,iCol+1) - I(iRow+1,iCol-1)) - b(iRow-1,iCol) * ...
25                 (I(iRow-1,iCol+1) - I(iRow-1,iCol-1)));
26
27         % Left in tridiagonal matrix
28         AN(iRow, iRow-1, 2) = (d(iRow, iCol) + d(iRow-1, iCol)) / 2;
29         % Right in tridiagonal matrix
30         AN(iRow, iRow+1, 2) = (d(iRow, iCol) + d(iRow+1, iCol)) / 2;
31         % Center in tridiagonal matrix
32         AN(iRow, iRow, 2) = -((d(iRow, iCol) + d(iRow-1, iCol)) + ...
33             (d(iRow, iCol) + d(iRow+1, iCol))) / 2;
34     end
35     InextN(:,iCol) = 1/2*((unitMatrixN - 2*tau*sparse(AN(:, :, 2)))\JnextN);
36 end

```

B.2.7 divergenceC.c

Denna funktion beräknar divergensen av ett tvådimensionellt vektorfält med samplingsavstånd

1. Kompileras för användning i Matlab med `mex divergenceC.c`.

```

1  /* =====
2  * EDGY - Computes divergence of a two dimensional vector field
3  * divergenceC.c
4  *
5  * Description: This is a MEX-file for MATLAB. It is a exact replica of the
6  * built in function i Matlab. The algorithm is using central difference for
7  * calculating the derivates, however at the borders forward and backward
8  * difference is used.
9  *
10 * Note: The name of the function will be the name of this file.
11 * To compile run: mex divergenceC.c
12 * In Matlab use:
13 * nablaF = divergenceC(F_x, F_y);
14 *
15 * Programmer: Henrik Wolkesson (henwo442@student.liu.se)
16 *
17 * Licensed under BSD as a part of EDGY project source code,
18 * see readme.txt file. For more information see project documentation.
19 *
20 * Revision: 1.0 (2007-05-09)
21 * =====
22 */
23
24 #include "mex.h"
25
26 void mexFunction(int nlhs,mxArray *plhs[],int nrhs,const mxArray *prhs[])
27 {
28     double *in_x, *in_y, *out_x, *out;
29     double *startIn_x, *startIn_y;
30     double *startOut_x, *startOut;
31     double *xb, *xf, *yb, *yf;
32     int N, M, NM;
33     mxArray *pout_x;
34
35     /* Get matrix size */
36     N = mxGetN(prhs[0]);
37     M = mxGetM(prhs[0]);
38     NM = N * M;
39
40     /* Create matrixes and get pointers */
41     pout_x = mxCreateDoubleMatrix(M,N,mxREAL);
42     plhs[0]=mxCreateDoubleMatrix(M,N,mxREAL);
43
44     in_x = mxGetPr(prhs[0]);
45     in_y = mxGetPr(prhs[1]);
46     out_x = mxGetPr(pout_x);
47     out = mxGetPr(plhs[0]);
48
49     /* Store initial pointers for later resetting */
50     startIn_x = in_x;
51     startIn_y = in_y;
52     startOut_x = out_x;
53     startOut = out;
54
55     // Calculate dF_x/dx //////////////////////////////////////
56     /* Calculate first column using forward difference*/

```

```

57     xb = startIn_x;
58     xf = startIn_x + M;
59     while(out_x < startOut_x + M)
60     {
61         *(out_x++) = *(xf++) - *(xb++);
62     }
63
64     /* Calculate main matrix using central difference*/
65     xb = startIn_x;
66     while (out_x < startOut_x + NM - M)
67     {
68         *(out_x++) = (*(xf++) - *(xb++)) / 2;
69     }
70
71     /* Calculate last column using backward difference*/
72     xf = startIn_x + NM - M;
73     xb = startIn_x + NM - 2*M;
74     while(out_x < startOut_x + NM)
75     {
76         *(out_x++) = *(xf++) - *(xb++);
77     }
78
79
80     // Calculate dF_y/dy //////////////////////////////////////
81     /* Calculate main matrix using central difference*/
82     yb = startIn_y;
83     yf = startIn_y + 2;
84     out = startOut + 1;
85     while (out < startOut + NM - 1)
86     {
87         *(out++) = (*(yf++) - *(yb++)) / 2;
88     }
89
90     /* Calculate first row using forward difference*/
91     out = startOut;
92     yb = startIn_y;
93     yf = startIn_y + 1;
94
95     while(out < startOut + NM - M + 1)
96     {
97         *out = *yf - *yb;
98         yb = yb + M;
99         yf = yf + M;
100        out = out + M;
101    }
102
103    /* Calculate last row using backward difference*/
104    out = startOut + M - 1;
105    yb = startIn_y + M - 2;
106    yf = startIn_y + M - 1;
107
108    while(out < startOut + NM)
109    {
110        *out = *yf - *yb;
111        yb = yb + M;
112        yf = yf + M;
113        out = out + M;
114    }
115
116    // Calculate output dF_x/dx + dF_y/dy //////////////////////////////////////
117    out = startOut;
118    out_x = startOut_x;

```

```
119     while(out < startOut + NM)
120     {
121         *out = *(out++) + *(out_x++);
122     }
123 }
```

B.3 Kod för delsystem 3: Wavelets

B.3.1 multiScaleEnh.m

Denna funktion har implementerats i en enda m-fil som innehåller alla hjälpfunktioner som är skrivna i matlabkod.

```

1 function [ outImg execTime ] = multiScaleEnh(inImg,j,alpha,tMin,tMax,...
2                                     sigma,t1,t2,t3,c,b,graphs,...
3                                     sampl)
4
5 %MULTISCALEENH Multi-scale enhancement of medical ultrasound images.
6 % [ENHANCEDIMAGE EXECUTIONTIME] =
7 % MULTISCALEENH(INIMG,J,ALPHA,TMIN,TMAX,SIGMA,T1,T2,T3,C,B,GRAPHS)
8 %
9 % Returns:
10 % ENHANCEDIMAGE    Processed Image.
11 % EXECUTIONTIME    Execution time for image enhancement operations.
12 %
13 % Parameters:
14 % INIMG    -    Input image.
15 % J        -    Number of channels to be computed by DWT.
16 % ALPHA    -    Decreasing factor for soft thresholds between subsequent
17 %               channels in DWT.
18 % TMIN     -    Minimum scaling factor for soft threshold.
19 % TMAX     -    Maximum scaling factor for soft threshold.
20 % SIGMA    -    Estimate of the standard deviation of the image noise.
21 % T1       -    Hard threshold.
22 % T2       -    Lower bound for edge enhancement through Generalized
23 %               Adaptive Gain of DWT coefficients.
24 % T3       -    Upper bound for edge enhancement through Generalized
25 %               Adaptive Gain of DWT coefficients.
26 % C        -    Gain in GAG-function.
27 % B        -    Controls shape of GAG-function.
28 % GRAPHS   -    Controls whether graphs of thresholding and enhancement
29 %               functions should be plotted (GRAPHS=1) or not (GRAPHS=0).
30 % SAMPL    -    SAMPL=0 gives zero padding of filters between levels in
31 %               DWT while SAMPL=1 replaces zero padding with downsampling
32 %               of signal between levels (greater distortion).
33 %
34 % Licensed under BSD as a part of EDGY project source code,
35 % see readme.txt file. For more information see project documentation.
36 %
37 % Revision: 1.0.1
38 % Date: 2007/05/08
39 % Author: Alexander Tuttle, Erik Ringaby
40
41 if ~(IsScalar(j) && IsScalar(alpha) && IsScalar(tMax) && IsScalar(tMin) ...
42     && IsScalar(t1) && IsScalar(t2) && IsScalar(t3) && IsScalar(c) ...
43     && IsScalar(b) && IsScalar(sigma) && IsScalar(graphs) ...
44     && IsScalar(sampl))
45     error('Parameters must be scalar.');
```



```

54 if ~(t1 >= 0 && t2 >= t1 && t3 > t2 && t3 <= 1)
55     error('t1, t2 and t3 must satisfy 0 <= t1 <= t2 < t3 <= 1.');
```

56 end

57

58

59 % Convert the image to gray scale.

60 inImg = im2single(mean(inImg,3));

61

62 if sampl && min(size(inImg)) <= 2^j

63 error('The input image is too small to be downsampled the number of times required.');

64 end

65

66 % If sigma is specified as 0 the user is prompted to select an area in the

67 % image that is adequate for estimating the standard deviation of the

68 % noise.

69 logImg = log(inImg+eps);

70 if sigma == 0

71 [upperLeft lowerRight] = getUserCoordinates(inImg);

72 rows = min(upperLeft(1,2),lowerRight(1,2)):max(lowerRight(1,2),upperLeft(1,2));

73 cols = min(upperLeft(1,1),lowerRight(1,1)):max(lowerRight(1,1),upperLeft(1,1));

74 imgSelection = logImg(rows,cols);

75 sigma = std(imgSelection(:));

76 end

77

78 % Compute the function table for the enhancement function E_GAG.

79 res = 0.001; %table resolution

80 s = -1; %lower bound for table

81 e = 1; %upper bound for table

82 egagTable = egag(s:res:e, b, c, t1, t2, t3);

83

84 % Plot graphs of thresholding and enhancement functions.

85 if graphs

86 showGraphs(j,alpha,tMin,tMax,sigma,egagTable,res);

87 end

88

89 % Start timer for measuring execution time of image enhancement algorithm.

90 tic;

91

92 % Compute the natural logarithm of the image to separate multiplicative

93 % noise.

94 inImg = log(inImg+eps);

95

96 % ----- DWT ----- %

97 [hpChannels lpChannels] = dwt(inImg,j,sampl);

98 % ----- %

99

100 % ----- Soft thresholding ----- %

101 % The coefficients below describe the relations between the standard

102 % deviation of the noise in the image and the respective channels.

103 sigmaxCoeff = [0.7122 0.2828 0.1907 0.1472 0.1217];

104 sigmayCoeff = [0.3416 0.1946 0.1468 0.1194 0.1012];

105

106 for level = 1:floor(j/2)

107 hpChannels{level}(:, :, 1) = softThresh(hpChannels{level}(:, :, 1), ...

108 tMax, tMin, sigmaxCoeff(level)*sigma, alpha, level);

109 hpChannels{level}(:, :, 2) = softThresh(hpChannels{level}(:, :, 2), ...

110 tMax, tMin, sigmayCoeff(level)*sigma, alpha, level);

111 end

112 % ----- %

113

114 % ----- Enhancement through Generalized Adaptive Gain ----- %

115 for iChannel = ceil(j/2):j

```

116     M1 = max(abs(reshape(hpChannels{iChannel}(:, :, 1), 1, [])));
117     M2 = max(abs(reshape(hpChannels{iChannel}(:, :, 2), 1, [])));
118     hpChannels{iChannel}(:, :, 1) = M1*lookUp(egagTable, res, ...
119         hpChannels{iChannel}(:, :, 1)/M1);
120     hpChannels{iChannel}(:, :, 2) = M2*lookUp(egagTable, res, ...
121         hpChannels{iChannel}(:, :, 2)/M2);
122 end
123 % ----- %
124 % ----- IDWT ----- %
125 % ----- %
126 outImg = exp(idwt(hpChannels, lpChannels, sampl));
127 % ----- %
128 execTime = toc;
129
130 % ##### Help functions ##### %
131
132 % ----- DWT ----- %
133 function [hpChannels lpChannel] = dwt(img, numChannels, sampl)
134 %DWT Frequency decomposition of 2D signal.
135 % [HP LP] = DWT(S,N,D) decomposes the signal into frequency channels
136 % according to the follow illustration:
137 %
138 %      |---HPx{1}
139 %      |
140 %      |---HPy{1}      |---HPx{N}
141 %      |               |
142 %      |               |---HPy{N}
143 %      |               |
144 %      |---LP--- ... ---|
145 %      |               |
146 %      |               |---LP
147 %
148 % S - Signal to be decomposed.
149 % N - Number of channels to compute.
150 % D - Downsample LP-component between channels.
151
152 % Filter coefficients for analyzing filters.
153 hx = [0.0625 0.25 0.375 0.25 0.0625];
154 gx = [0 1 -1];
155
156 lpChannel = img;
157
158 if(sampl)
159     for iChannel = 1:numChannels
160         hpChannels{iChannel}(:, :, 1) = mxSimpleDiff(lpChannel, 0);
161         hpChannels{iChannel}(:, :, 2) = mxSimpleDiff(lpChannel', 0)';
162
163         lpChannel = imfilter(imfilter(lpChannel, hx, 'replicate', 'conv'), ...
164             hx, 'replicate', 'conv');
165         lpChannel = lpChannel(1:2:end, 1:2:end);
166     end
167 else
168     for iChannel = 1:numChannels
169         hpChannels{iChannel}(:, :, 1) = mxSimpleDiff(lpChannel, ...
170             2^(iChannel-1)-1);
171         hpChannels{iChannel}(:, :, 2) = mxSimpleDiff(lpChannel', ...
172             2^(iChannel-1)-1)';
173
174         lpChannel = imfilter(imfilter(lpChannel, zeroPad(hx, iChannel), ...
175             'replicate', 'conv'), zeroPad(hx, iChannel), 'replicate', 'conv');
176     end
177 end

```

```

178 % ----- %
179
180 % ----- IDWT ----- %
181 function img = idwt(hpChannels, lpChannel, sampl)
182 %IDWT Reconstructs a signal that has been decomposed with DWT.
183 % S = IDWT(HPCHANNELS, LPCHANNEL, SAMPL)
184 % HPCHANNELS, LPCHANNEL - Output from DWT, write 'help dw'.
185 % SAMPL - Boolean that indicates whether the signal was
186 % downsampling during decomposition in DWT.
187
188 % Filter coefficients for reconstructing filters.
189 hx = [0.0625 0.25 0.375 0.25 0.0625];
190 kx = [-0.00390625 -0.03515625 -0.14453125 -0.36328125 0.36328125...
191 0.14453125 0.03515625 0.00390625 0];
192 lx = [0.001953125 0.015625 0.0546875 0.109375 0.63671875 0.109375...
193 0.0546875 0.015625 0.001953125];
194
195 % Split signal into channels with or without downsampling in between the
196 % channels.
197 iChannel = length(hpChannels);
198 if(sampl)
199     while iChannel > 0
200         hpChannelX = imfilter(imfilter(hpChannels{iChannel}(:, :, 1), ...
201             lx, 'replicate', 'conv'), kx, 'replicate', 'conv');
202
203         hpChannelY = imfilter(imfilter(hpChannels{iChannel}(:, :, 2), ...
204             kx, 'replicate', 'conv'), lx, 'replicate', 'conv');
205
206         lpChannel = interp2(lpChannel, 'spline');
207         if mod(size(hpChannelX, 1), 2) == 0
208             lpChannel = [lpChannel; lpChannel(end, :)];
209         end
210         if mod(size(hpChannelX, 2), 2) == 0
211             lpChannel = [lpChannel lpChannel(:, end)];
212         end
213
214         lpChannel = imfilter(imfilter(lpChannel, hx, 'replicate'), hx, ...
215             'replicate');
216         lpChannel = hpChannelX + hpChannelY + lpChannel;
217         iChannel = iChannel - 1;
218     end
219 else
220     while iChannel > 0
221         hpChannelX = imfilter(imfilter(hpChannels{iChannel}(:, :, 1), ...
222             zeroPad(lx, iChannel), 'replicate', 'conv'), ...
223             zeroPad(kx, iChannel), 'replicate', 'conv');
224
225         hpChannelY = imfilter(imfilter(hpChannels{iChannel}(:, :, 2), ...
226             zeroPad(kx, iChannel), 'replicate', 'conv'), ...
227             zeroPad(lx, iChannel), 'replicate', 'conv');
228
229         lpChannel = imfilter(imfilter(lpChannel, zeroPad(hx, iChannel), ...
230             'replicate'), zeroPad(hx, iChannel), 'replicate');
231
232         lpChannel = hpChannelX + hpChannelY + lpChannel;
233         iChannel = iChannel - 1;
234     end
235 end
236 img = lpChannel;
237 % ----- %
238
239 % ----- SOFTTHRESH ----- %

```

```

240 function U = softThresh( V, tMax, tMin, sigma, alpha, lev )
241 %SOFTTHRESH performs soft thresholding of the values in a vector or matrix.
242 % Y = softThresh( X, TMAX, TMIN, S, A, L ) performs
243 % thresholding of X with the threshold t computed as
244 %
245 % ( TMAX-A*(L-1))*S if TMAX-A*(L-1)>TMIN
246 % t = <
247 % ( TMIN*S otherwise
248
249 % Calculate a threshold
250 c = tMax - alpha*(lev-1);
251
252 if c > tMin
253     t = c*sigma;
254 else
255     t = tMin*sigma;
256 end
257 % Apply soft thresholding
258 U = sign(V).*(abs(V)-t).*(abs(V) > t);
259 % ----- %
260
261 % ----- EGAG ----- %
262 function Y = egag( X, b, c, t1, t2, t3 )
263 %EGAG Signal enhancement of X through Generalized Adaptive Gain.
264 % Y = EGAG( X, B, C, T1, T2, T3 ).
265 % X is the signal that you want to enhance, it can be either a vector or
266 % a matrix. Parameters B and C control the amount of gain. Parameters T1,
267 % T2 and T3 are thresholds. Values in [0 T1] are mapped to 0, values in
268 % [T2 T3] are amplified and values in ]T1 T2[ and above T3 are not
269 % altered.
270
271 a = 1./(sigm(c*(1-b))-sigm(-c*(1+b)));
272 signx = sign(X);
273 U = signx.*(abs(X)-t2)./(t3-t2);
274 U_bar = a*(t3-t2)*(sigm(c*(U-b))-sigm(-c*(U+b)));
275
276 Y = (signx.*t2+U_bar).*((abs(X) <= t3).*((abs(X) >= t2))) +...
277     X.*((abs(X) > t3) + (abs(X) < t2)).*(abs(X)>t1);
278
279 function Y = sigm( X )
280 %SIGM A sigmoid function
281 % Y = SIGM(X) = 1./(1+exp(-X));
282
283 Y = 1./(1+exp(-X));
284 % ----- %
285
286 % ----- ZEROPAD ----- %
287 function out = zeroPad(in, n)
288 %ZEROPAD Insert zeros between elements of a vector or matrix.
289 % Y = ZEROPAD(X,N) inserts 2^(N-1)-1 zeros between the elements in the
290 % vector or matrix X. If X is a matrix X will be zero padded in both
291 % dimensions.
292
293 if (n <= 0)
294     error('n must be equal to or larger than one');
295 end
296 if (n == 1)
297     out = in;
298 else
299     n = 2^(n-1);
300     out = zeros(size(in,1),n*size(in,2)-n+1);
301     out(:,1:n:end)=in;

```

```

302 end
303 % ----- %
304
305 % ----- LOOKUP ----- %
306 function Y = lookUp(funTable,res,X)
307 %LOOKUP Look up values in a function table with NN-interpolation.
308 % Y = LOOKUP(FUNTABLE, R, X)
309 % FUNTABLE - table of function values.
310 % R - table resolution.
311 % X - values whos correspodng function values
312 % are to be looked up.
313
314 % Dummy variables intended to optimize speed
315 tVar1 = (1/res); % Multiplication is faster than division
316 tVar2 = length(funTable)/2;
317
318 Y = funTable(ceil(X*tVar1+tVar2));
319 % ----- %
320
321 % ----- SHOWGRAPHS ----- %
322 function showGraphs(j,alpha,tMin,tMax,sigma,egagTable,res)
323 %SHOWGRAPHS Displays graphs of thresholding and enhancement functions.
324 x = linspace(-1,1,1/res);
325 numC = floor(j/2);
326 % Plot softThresh for fine scale levels of DWT
327 fig1 = figure;
328 set(fig1,'Name','Thresholding and enhancement functions');
329 for channel = 1:numC
330     subplot(1,numC+1,channel);
331     plot(x,softThresh(x, tMax, tMin, sigma, alpha, channel));
332     axis square;title(sprintf('softThreshold in channel %d',channel));
333     xlabel x;ylabel('softThresh(x)');
334 end
335 %Plot egag function
336 subplot(1,numC+1,numC+1);plot(x,lookUp(egagTable,res,x));axis square;
337 title('Generalized Adaptive Gain function');xlabel x;ylabel E_{GAG}(x);
338 % ----- %
339
340 function [ x1 x2 ] = getUserCoordinates(img)
341 %GETUSERCOORDINATES
342 % [ X1 X2 ] = GETUSERCOORDINATES(IMG)
343 % Lets the user choose two points in an image and returns
344 % the coordinates of each point in the row vectors X1 and X2.
345
346 x1 = zeros(1,2);
347 x2 = zeros(1,2);
348
349 fig = figure;
350 imshow(img,[min(img(:)),max(img(:))]);axis image;colormap gray;
351 set(fig,'Name','Select area for estimation of standard deviation');
352
353 disp('Select upper left corner of preferred area with the mouse. ');
354 t = waitforbuttonpress;
355 [x1(1,1) x1(1,2)] = ginput(1);
356 disp('Select lower right corner of preferred area with the mouse. ');
357 t = waitforbuttonpress;
358 [x2(1,1) x2(1,2)] = ginput(1);
359
360 x1 = ceil(x1);
361 x2 = floor(x2);
362 close(fig);
363 % ----- %

```

B.3.2 mxSimpleDiff.c

Den här C-funktionen implementerades som en snabbare ersättare till `imfilter()` för ett visst filter. Anropet `mxSimpleDiff(X, 0)` returnerar samma resultat som anropet `imfilter(X, [0 1 -1], 'replicate', 'conv')`. Det andra argumentet till `mxSimpleDiff` anger hur många nollor som ska skjutas in mellan koeficienterna i filtret i anropet ovan.

```
1  /* =====
2  * EDGY - Speed optimizing substitute for imfilter when using a simple
3  *   differentiation filter gx = [0 1 -1].
4  * mxSimpleDiff.c
5  *
6  * Description: This is a MEX-file for MATLAB. Computational
7  * function that approximates the first order derivative along the second
8  * dimension of a vector or matrix. The following two calls return the same
9  * value:
10 *
11 * Y = mxSimpleDiff(X,0);
12 * Y = imfilter(X,[0 1 -1],'replicate','conv');
13 *
14 * The second argument to mxSimpleDiff should be an integer n > 0 and it
15 * does the same thing as inserting n zeros between the coefficients of the
16 * kernel in the call to 'imfilter' above.
17 *
18 * Note: The name of the function will be the name of this file.
19 * To compile run: mex mxSimpleDiff.c
20 * Then use in Matlab as: ret = mxSimpleDiff(matrix, numZeros);
21 *
22 * Warning: Most of the checks to catch faulty input arguments etcetera
23 * have been commented to optimize speed!
24 *
25 *
26 * Programmer: Alexander Tuttle (alexander.tuttle@gmail.com)
27 *
28 * Licensed under BSD as a part of EDGY project source code,
29 * see readme.txt file. For more information see project documentation.
30 *
31 * Revision: 1.0 (2007-05-08)
32 * =====
33 */
34
35 #include "mex.h"
36 #include "matrix.h"
37 #include "math.h"
38
39 void differentiate(float y[], float x[], int mrows, int ncols,
40                  int numElements, int shift)
41 {
42     int i,col,row = 0;
43
44     if (shift < ncols) {
45
46         /*The two for loops do this: Set y to x with the columns of x shifted
47         'shift' steps to the right, the first column of x is replicated*/
48
49         /*Set the first 0 to (shift-1) columns of y to the first column of x.*/
50         for (col=0; col<shift; col++){
51             for(row=0; row<mrows; row++) {
52                 y[row+col*mrows] = x[row];
53             }
54         }
55     }
```

```

54         }
55
56     /*Set the remaining columns of y to columns 1 through
57     ncols in x. Shift complete.*/
58     for(i=shift*mrows; i<numElements; i++) {
59         y[i] = x[i-shift*mrows];
60     }
61
62     for(i=0; i<numElements; i++) {
63         y[i] = x[i]-y[i];
64     }
65 }
66 else {
67
68     for(i=0; i<mrows; i++) {
69         y[i] = 0;
70     }
71
72     for(i=mrows; i<numElements; i++) {
73         y[i] = x[i-mrows];
74     }
75 }
76 }
77
78 void mexFunction(int nlhs, mxArray *plhs[], int nrhs,
79 const mxArray *prhs[])
80 {
81     float *x, *y;
82     int mrows, ncols, numElements, numZeros;
83
84     /* Check for proper number of arguments. */
85     /*
86     if (nrhs != 2) {
87         mexErrMsgTxt("Two inputs required.");
88     } else if (nlhs > 1) {
89         mexErrMsgTxt("Too many output arguments");
90     }
91     */
92     mrows = mxGetM(prhs[0]);
93     ncols = mxGetN(prhs[0]);
94     numElements = mxGetNumberOfElements(prhs[0]);
95
96     /* The input must be a noncomplex scalar double.*/
97     /*
98     if (!mxIsSingle(prhs[0]) || mxIsComplex(prhs[0])) {
99         mexErrMsgTxt("Input x must be a noncomplex single matrix.");
100     }
101
102     if (!(*prhs[1] < 0)) {
103         mexErrMsgTxt("Input n must be a noncomplex scalar >= 1.");
104     }
105     */
106
107     /* Create matrix for the return argument. */
108     plhs[0] = mxCreateNumericMatrix(mrows,ncols, mxSINGLE_CLASS, mxREAL);
109
110     /* Assign pointers to each input and output. */
111     x = (float *) mxGetPr(prhs[0]);
112     numZeros = (int) *mxGetPr(prhs[1]);
113     y = (float *) mxGetPr(plhs[0]);
114
115

```

```
116 //      printf("\nshift = %i\n",shift);
117      /* Call the differentiate subroutine. */
118      differentiate(y,x,mrows,ncols,numElements,numZeros+1);
119 }
```


B.4 Kod för delsystem 4: Adaptiv medianvärdesfiltrering

```

1 function [outPic time] = AWMF(inPic, c, K, wCenter, graphs)
2 %AWMF Adaptive Weighted Median Filter
3 % [OUTPIC TIME] = AWMF(INPIC, C, K, WCENTER, GRAPHS) A function which
4 % use an Adaptive Weighted Median Filter for reducing speckle noise. The
5 % result, OUTPIC, has the same size and class as INPIC. The running time
6 % for the function is returned in the variable TIME.
7 %
8 % The weights, [W], for a sequence, [X], extends the sequence [W] times.
9 % For example, if W = [w1=2 w2=0 w3=3] the weighted median of the
10 % sequence x = [x1 x2 x3] is [x1 x1 x3 x3 x3].
11 % The weights for a local neighbourhood is given by:
12 %
13 % W(i,j) = [WCENTER - distance * C * LocalMean / LocalVariance]
14 %
15 % For more information and details see project documentation.
16 %
17 % INPIC      Input grayscale picture, UINT8
18 % C          Scalar constant, C >= 0 (if 0, median filter)
19 % K          Size of 2D filter kernel (N*N), N = 2*K + 1
20 %           K <= min(size(INPIC))
21 % WCENTER    Kernel Centrum Weight, WCENTER >= 0 (integer)
22 % GRAPHS     if 1 shows extra graphs for development purpose
23 %
24 % Class support for inputs INPIC
25 %   uint8
26 % Class support for inputs C, K, WCENTER, GRAPHS:
27 %   double
28 %
29 % Class for output OUTPIC
30 %   uint8
31 % Class for output TIME
32 %   double
33 %
34 % Requirements: IMFILTER in Image Processing Toolbox
35 %
36 % See also MEDIAN.
37 %
38 % Licensed under BSD as a part of EDGY project source code,
39 % see readme.txt file. For more information see project documentation.
40 %
41 % Revision: 1.0
42 % Date: 2007/05/08
43 % Author: Per Thyr
44
45 if (~isnumeric(c) || numel(c) ~=1 || c < 0)
46     error('C must be a positive scalar.');
```

```

60     error('Input picture must be UINT8-class.')
61 end
62
63 % Starts calculating runningtime
64 tic
65
66 % Convert uint8 format to double. Add 1 to avoid div by zero
67 inPic = double(inPic) + 1;
68
69 % Local neighbourhood sidelength and area
70 N = 2*K+1;
71 sqrN = N^2;
72
73 % Calculate quota (c*var/mean)
74 kernelMean = ones(N,N)/(N*N);
75 inPicMean = imfilter(inPic, kernelMean, 'conv', 'replicate');
76 inPicMeanSq = imfilter(inPic.^2, kernelMean, 'conv', 'replicate');
77
78 quotaWeight = c*(inPicMeanSq./inPicMean - inPicMean);
79 clear inPicMean inPicMeanSq;
80
81 % Makes extended quotaWight
82 tquotaWeightExt = [repmat(quotaWeight(:,1),1,K) quotaWeight ...
83                    repmat(quotaWeight(:,end),1,K)];
84 quotaWeightExt = [repmat(tquotaWeightExt(1,:),K,1); tquotaWeightExt; ...
85                    repmat(tquotaWeightExt(end,:),K,1)];
86 clear tquotaWeightExt;
87
88 % Makes extended InPic
89 tmpInPicExt = [repmat(inPic(:,1),1,K) inPic ...
90                repmat(inPic(:,end),1,K)];
91 inPicExt = [repmat(tmpInPicExt(1,:),K,1); tmpInPicExt; ...
92             repmat(tmpInPicExt(end,:),K,1)];
93 clear tmpInPicExt;
94
95 % Calculate distancematrix
96 [dMeshCol, dMeshRow] = meshgrid(-K:K,-K:K);
97 distMap = sqrt((dMeshRow.^2) + (dMeshCol.^2));
98
99 % Initiation
100 outPic = zeros(size(inPic));
101 [numRowExt, numColExt] = size(inPicExt);
102
103 % Calculate new pixelevalue, pixel by pixel
104 for iRow = 1+K : numRowExt-K
105     for iCol = 1+K : numColExt - K
106
107         % Local Neighbourhood
108         inPicLocal = inPicExt(iRow-K : iRow+K, iCol-K : iCol+K);
109
110         % Calculate local weightmatrix, wLocal
111         wLocal = (wCenter - distMap*quotaWeightExt(iRow,iCol));
112         wLocal = round(max(wLocal,0)); % negative 2 zero
113         sumW = sum(wLocal(:));
114
115         % Sort
116         [sortInPicLocal index] = sort(inPicLocal(:), 1);
117
118         % Calculate median (output value)
119         newPxPos = (sumW+1)/2;
120         sumW = 0;
121         for iPos = 1:sqrN

```

```

122         sumW = sumW + wLocal(index(iPos));
123         if sumW >= newPxPos
124             break
125         end
126     end
127     yWM = sortInPicLocal(iPos);
128     outPic(iRow - K, iCol - K) = yWM;
129 end
130 end
131
132 % Development graphs
133 if graphs == 1
134     quota = quotaWeight./c;
135     figure('Name','AWMF: Development graphs, c = ' num2str(c) ...
136           ', K = ' num2str(K) ...
137           ' (' num2str(N) '*' num2str(N) ') '...
138           ', wCenter = ' num2str(wCenter) ], 'NumberTitle','off')
139     subplot(3,2,1), imagesc(inPic);
140     title('InPic'); colormap gray; axis image; colorbar;
141     subplot(3,2,2), imagesc(outPic, [min(inPic(:)) max(inPic(:))]);...
142     title('outPic'); colormap gray; axis image; colorbar;
143     subplot(3,2,3), imagesc(quota);
144     title('variance/mean quota'); colormap gray; axis image; colorbar;
145     subplot(3,2,4), imagesc(quotaWeight);
146     title('c*variance/mean quota'); colormap gray; axis image; colorbar;
147     subplot(3,2,5), imagesc(abs(inPic - outPic));
148     title('|inPic - outPic|'); colormap gray; axis image; colorbar;
149     subplot(3,2,6), imagesc(distMap);
150     title('distMap'); colormap gray; axis image; colorbar;
151 end
152
153 % Convert back to uint8
154 outPic = uint8(outPic - 1);
155
156 % Stops calculating runtime
157 time = toc;

```

C Tidsanalys

Varje algoritm har genomgått en tidsanalys för att undersöka vilka delar som tar tid. Syftet är att optimera algoritmerna för att minska exekveringstiden. Optimeringen genom omskrivningar samt konvertering av Matlab-kod till C-kod. De slutgiltiga tidsanalyserna, efter optimering, presenteras nedan.

Samtliga analyser har genomförst på samma referensdator i CYD-poolen (Datorsal i B-huset på Linköpings universitet), det vill säga en PC med 3 GHz Intel Pentium 4-processor, 2 GB RAM-minne samt operativsystemet Microsoft Windows XP Professional.

C.1 Tidsanalys för delsystem 2

AnisotropicDiffusion (1 call, 0.422 sec)

Generated 09-May-2007 23:20:39 using real time.







M-function in file <H:\edgy\Implementering\profilekoder\artikel1\AnisotropicDiffusion.m>

[\[Copy to new window for comparing multiple runs\]](#)




Refresh

- ☐ Show parent functions ☒ Show busy lines ☒ Show child functions
☐ Show M-Lint results ☒ Show file coverage ☒ Show function listing

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
100	[a, b, c, d] = fastD(J, alpha,...	2	0.156 s	37.0%	
96	J = imfilter(imfilter(nablaI,K...	2	0.155 s	36.7%	
104	I = I + divergenceC(a.*I_x + b...	2	0.032 s	7.6%	
86	I_y=imfilter(I,derivKernel', '...	2	0.031 s	7.3%	
87	I_x=imfilter(I,derivKernel, 'c...	2	0.016 s	3.8%	
Other lines & overhead			0.032 s	7.6%	
Totals			0.422 s	100%	

Children (called functions)

Function Name	Function Type	Calls	Total Time	% Time	Time Plot
imfilter	M-function	8	0.202 s	47.9%	
fastD	MEX-function	2	0.140 s	33.2%	
divergenceC	MEX-function	2	0.032 s	7.6%	
IsScalar	M-function	8	0 s	0%	
IsPositiveInteger	M-function	1	0 s	0%	
createDerivKernel	M-function	1	0 s	0%	

createGaussKernel	M-function	1	0 s	0%	
Self time (built-ins, overhead, etc.)			0.048 s	11.4%	■
Totals			0.422 s	100%	

Coverage results

[[Show coverage for parent directory](#)]

Total lines in file	138
Non-code lines (comments, blank lines)	62
Code lines (lines that can run)	76
Code lines that did run	32
Code lines that did not run	44
Coverage (did run/can run)	42.11 %

Function listing

Color highlight code according to

```

time calls line
1 function [I, timeSpent] = AnisotropicDiffusion( inIma
2     sigma2, delta2, s2, alpha, beta, tau, numIter, us
3 %AnisotropicDiffusion Implements Nonlinear Anisotropic
4 % [I, timeSpent] = AnisotropicDiffusion( inImage, s
5 %     sigma2, delta2, s2, alpha, beta, tau, useAOS, num
6 %
7 % Returns:
8 % I          Processed Image after selected iterat
9 % timeSpent  Time spent processing exclusive creat
10 %
11 % Parameters:
12 % inImage    Image to process. Must be gray scale
13 % sigma1     Sigma for derivating kernel
14 % delta1     Cut of tail of derivating kernel at t
15 % sigma2     Sigma for low pass kernel
16 % delta2     Cut of tail of low pass kernel at thi
17 % s2        Stop value for diffusion in gradient
18 % alpha     Isotropic diffusion number
19 % beta      Maximum amount of diffusion. Small be
20 % tau       Time step
21 % numIter   Maximum number
22 % useAOS    Choose to use an Additive Operatorspl
23 %           of a divergence operator
24 % graphs    1 - Show graphs, 0 - Show no graphs
25 %
26 % Requirements IMFILTER and IMSHOW in Image Process
27 % See also IMFILTER, IMSHOW.
28 %
29 % Licensed under BSD as a part of EDGY project sour
30 % see readme.txt file. For more information see prc
31 %

```

```

32 % Revision: 1.0
33 % Date: 2007/05/09
34 % Author: Henrik Wolkesson
35
1 36 if ~IsScalar(sigm1) || sigm1 < 0 || ...
37     ~IsScalar(delta1) || delta1 < 0 || ...
38     ~IsScalar(sigma2) || sigma2 < 0 || ...
39     ~IsScalar(delta2) || delta2 < 0
40     error('Sigm1, Sigma2, Delta1 and Delta2 must be
41 end
1 42 if ~IsScalar(s2) || s2 < 0 || ~IsScalar(alpha) || alp
43     ~IsScalar(beta) || beta < 0 || ~IsScalar(tau)
44     error('s2, alpha, beta and tau must be a positive
45 end
1 46 if ~IsPositiveInteger(numIter)
47     error('numIter must be positive integer');
48 end
1 49 if useAOS < 0 || useAOS > 1 || round(useAOS) ~= useAC
50     error('useAOS must be either 0 or 1');
51 end
1 52 if graphs < 0 || graphs > 1 || round(graphs) ~= graph
53     error('Graphs must be either 0 or 1');
54 end
55
56 % Section 1: Initize
1 57 inImage = double(inImage);
< 0.01 1 58 I=inImage;
< 0.01 1 59 N = size(I,1);
< 0.01 1 60 M = size(I,2);
0.02 1 61 nablaI=zeros(N,M,2);
1 62 if (useAOS == 1)
63     unitMatrixM = sparse(eye(M));
64     unitMatrixN = sparse(eye(N));
65 end
66
67 % Section 2: Create kernels
1 68 derivKernel = createDerivKernel(sigm1, delta1);
1 69 Kx = createGaussKernel(sigma2,delta2);
70
1 71 if (graphs == 1)
72     figure('Name','Kernels');
73     subplot(1,2,1);plot(Kx);
74     title(['Gauss Kernel (Size: ' num2str(size(Kx,2))
75     subplot(1,2,2);plot(derivKernel);
76     title(['Derivating Kernel (Size: ' num2str(size(d
77 end;
78
1 79 if (size(Kx,2)>size(I,2))
80     warning('Derivating kernel is bigger than image')
81 end
82
< 0.01 1 83 tic;
1 84 for k = 1:numIter
85     % Section 3: Calculate derivates
0.03 2 86 I_y=imfilter(I,derivKernel, 'conv', 'replicate')
0.02 2 87 I_x=imfilter(I,derivKernel, 'conv', 'replicate');
88

```

```

89      % Section 4: Create structure matrix
0.02  2  90      nablaI(:, :, 1) = I_x.*I_x;
      2  91      nablaI(:, :, 2) = I_x.*I_y;
      2  92      nablaI(:, :, 3) = I_y.*I_y;
      93
      94      % Section 5: LP filter structure matrix
0.16  2  95      Ky = Kx';
      2  96      J = imfilter(imfilter(nablaI, Kx, 'conv', 'replicate'),
      97      Ky, 'conv', 'replicate');
      98
      99      % Section 6-8: Eigenanalysis, modulation and constant
0.16  2  100     [a, b, c, d] = fastI(J, alpha, beta, s2, inImage, 1);
      101
      102      % Section 9: Solve diffusion equation
0.03  2  103     if (useAOS == 0)
      2  104     I = I + divergence(a.*I_x + b.*I_y, c.*I_x +
      105     else
      106         InextN = solveAOSN(I, N, M, a, b, c, d, tau,
      107         InextM = solveAOSM(I, N, M, a, b, c, d, tau,
      108         I = InextM + InextN;
      109     end
      110
      111      % Optional plotting
2  112     if (graphs == 1)
      113         figure;
      114         subplot(3,1,1); imshow(I, []); title(['Iteration
      115         title(['I of iteration ' num2str(k)]);
      116         subplot(3,1,2); imshow(I_x, []);
      117         title(['Derivate dI/dx of iteration ' num2str(k)]);
      118         subplot(3,1,3); imshow(I_y, []);
      119         title(['Derivate dI/dy of iteration ' num2str(k)]);
      120         figure('Name', ['LP-filtered Tensor J of iteration ' num2str(k)]);
      121         subplot(3,1,1); imshow(J(:, :, 1), []);
      122         title('dI/dx*dI/dx');
      123         subplot(3,1,2); imshow(J(:, :, 2), []);
      124         title('dI/dx*dI/dy');
      125         subplot(3,1,3); imshow(J(:, :, 3), []);
      126         title('dI/dy*dI/dy');
      127     end
2  128 end
      129
      130      % Optional Plotting
1  131     if (graphs == 1)
      132         figure; plot(inImage(floor(N/2), :), 'r'); hold on; plot(I, 'b');
      133         legend('Original with Speckle', 'Processed');
      134     end
      135
      136      % Section 10: Return time and processed image
< 0.01 1  137     timeSpent = toc;
      1  138     I = uint8(I);

```


C.2 Tidsanalys för delsystem 3

C.2.1 Med sampling

multiScaleEnh (1 call, 0.656 sec)

Generated 09-May-2007 23:21:32 using real time.







M-function in file <H:\edgy\implementering\profilekoder\artikel2\multiScaleEnh.m>

[\[Copy to new window for comparing multiple runs\]](#)







Refresh

- ☐ Show parent functions ☒ Show busy lines ☒ Show child functions
☐ Show M-Lint results ☒ Show file coverage ☒ Show function listing

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
127	outImg = exp(idwt(hpChannels, ...	1	0.359 s	54.7%	
98	[hpChannels lpChannels] = dwt(...	1	0.063 s	9.6%	
38	addpath common\;	1	0.063 s	9.6%	
40	addpath artikel1\;	1	0.047 s	7.2%	
39	addpath common\mxSimpleDiff;	1	0.031 s	4.7%	
Other lines & overhead			0.093 s	14.2%	
Totals			0.656 s	100%	

Children (called functions)

Function Name	Function Type	Calls	Total Time	% Time	Time Plot
multiScaleEnh>idwt	M-subfunction	1	0.344 s	52.4%	
addpath	M-function	3	0.141 s	21.5%	
multiScaleEnh>dwt	M-subfunction	1	0.063 s	9.6%	
multiScaleEnh>softThresh	M-subfunction	4	0.031 s	4.7%	
multiScaleEnh>egag	M-subfunction	1	0.016 s	2.4%	
mean	M-function	1	0.015 s	2.3%	

IsScalar	M-function	12	0 s	0%	
im2single	M-function	1	0 s	0%	
multiScaleEnh>lookUp	M-subfunction	6	0 s	0%	
Self time (built-ins, overhead, etc.)			0.046 s	7.0%	■
Totals			0.656 s	100%	

Coverage results

[[Show coverage for parent directory](#)]

Total lines in file	364
Non-code lines (comments, blank lines)	215
Code lines (lines that can run)	149
Code lines that did run	33
Code lines that did not run	116
Coverage (did run/can run)	22.15 %

Function listing

Color highlight code according to

```

time  calls  line
1 function [ outImg execTime ] = multiScaleEnh(inImg,j,
2                                     sigma,t1,
3                                     sampl)
4
5 %MULTISCALEENH Multi-scale enhancement of medical ult
6 % [ENHANCEDIMAGE EXECUTIONTIME] =
7 % MULTISCALENH( INIMG,J,ALPHA,TMIN,TMAX,SIGMA,T1,T2,
8 %
9 % Returns:
10 % outImg      Processed Image.
11 % execTime    Execution time for image enhancement
12 %
13 % Parameters:
14 % INIMG      - Input image.
15 % J          - Number of channels to be computed by
16 % ALPHA      - Decreasing factor for soft threshold
17 %             channels in DWT.
18 % TMIN       - Minimum scaling factor for soft thre
19 % TMAX       - Maximum scaling factor for soft thre
20 % SIGMA      - Estimate of the standard deviation c
21 % T1         - Hard threshold.
22 % T2         - Lower bound for edge enhancement thr
23 %             Adaptive Gain of DWT coefficients.
24 % T3         - Upper bound for edge enhancement thr
25 %             Adaptive Gain of DWT coefficients.
26 % C          - Gain in GAG-function.
27 % B          - Controls shape and sign of GAG-funct

```

```

28 %   GRAPHS    -   Controls whether graphs of thresholdi
29 %               functions should be plotted or not.
30 %
31 %   Licensed under BSD as a part of EDGY project sour
32 %   see readme.txt file. For more information see prc
33 %
34 %   Revision: 1.0
35 %   Date: 2007/05/08
36 %   Author: Alexander Tuttle, Erik Ringaby
37
0.06 1 38 addpath common\;
0.03 1 39 addpath common\mxSimpleDiff;
0.05 1 40 addpath artikkel1\;
41
1 42 if ~(IsScalar(j) && IsScalar(alpha) && IsScalar(tMax)
43     && IsScalar(t1) && IsScalar(t2) && IsScalar(t
44     && IsScalar(b) && IsScalar(sigma) && IsScalar
45     && IsScalar(sampl))
46     error('Parameters must be scalar.');
```

```

47 end
1 48 if ~(numel(j) == 1 && isnumeric(j) &&...
49     (j > 0) && (round(j) == j) && j < 6 );
50     error('The number of levels must be a positive in
51 end
1 52 if (tMin >= tMax)
53     error('t_min must be less than t_max');
54 end
1 55 if ~(t1 >= 0 && t2 >= t1 && t3 > t2 && t3 <= 1)
56     error('t1, t2 and t3 must satisfy 0 <= t1 <= t2 <
57 end
58
59
60 % Convert the image to gray scale.
0.02 1 61 inImg = im2single(mear(inImg,3));
62
1 63 if sampl && min(size(inImg)) <= 2^j
64     error('The input image is too small to be downsar
65 end
66
67 % If sigma is specified as 0 the user is prompted to
68 % image that is adequate for estimating the standard
69 % noise.
0.02 1 70 logImg = log(inImg+eps);
1 71 if sigma == 0
72     [upperLeft lowerRight] = getUserCoordinates(inImg
73     rows = min(upperLeft(1,2),lowerRight(1,2)):max(lc
74     cols = min(upperLeft(1,1),lowerRight(1,1)):max(lc
75     imgSelection = logImg(rows,cols);
76     sigma = std(imgSelection(:));
77 end
78
79 % Compute the function table for the enhancement func
< 0.01 1 80 res = 0.001; %table resolution
< 0.01 1 81 s = -1; %lower bound for table
< 0.01 1 82 e = 1; %upper bound for table
0.02 1 83 egagTable = egac(s:res:e, b, c, t1, t2, t3);
84

```

```

85 % Plot graphs of thresholding and enhancement function
1 86 if graphs
87     showGraphs(j,alpha,tMin,tMax,sigma,egagTable,res)
88 end
89
90 % Start timer for measuring execution time of image e
< 0.01 1 91 tic;
92
93 % Compute the natural logarithm of the image to separ
94 % noise.
0.02 1 95 inImg = log(inImg+eps);
96
97 % ----- DWT -----
0.06 1 98 [hpChannels lpChannels] = dwt(inImg,j,sampl);
99 % -----
100
101 % ----- Soft thresholding -----
102 % The coefficients below describe the relations betwe
103 % deviation of the noise in the image and the respect
< 0.01 1 104 sigmaxCoeff = [0.7122 0.2828 0.1907 0.1472 0.1217];
< 0.01 1 105 sigmayCoeff = [0.3416 0.1946 0.1468 0.1194 0.1012];
106
107 for level = 1:floor(j/2)
0.02 2 108     hpChannels{level}(:, :, 1) = softThresh(hpChannels{1
109         tMax, tMin, sigmaxCoeff(level)*sigma, alpha,
0.02 2 110     hpChannels{level}(:, :, 2) = softThresh(hpChannels{1
111         tMax, tMin, sigmayCoeff(level)*sigma, alpha,
2 112 end
113 % -----
114
115 % ----- Enhancement through Generalized Adaptiv
1 116 for iChannel = ceil(j/2):j
3 117     M1 = max(abs(reshape(hpChannels{iChannel}(:, :, 1),
3 118     M2 = max(abs(reshape(hpChannels{iChannel}(:, :, 2),
3 119     hpChannels{iChannel}(:, :, 1) = M1*lookUp(egagTable
120         hpChannels{iChannel}(:, :, 1)/M1);
3 121     hpChannels{iChannel}(:, :, 2) = M2*lookUp(egagTable
122         hpChannels{iChannel}(:, :, 2)/M2);
3 123 end
124 % -----
125
126 % ----- IDWT -----
0.36 1 127 outImg = exp(idwt(hpChannels, lpChannels, sampl));
128 % -----
< 0.01 1 129 execTime = toc;
130
131 % ##### Help functions #####
132
133 % ----- DWT -----
134 function [hpChannels lpChannel] = dwt(img,numChannels
135 %DWT Frequency decomposition of 2D signal.
136 % [HP LP] = DWT(S,N,D) decomposes the signal into f
137 % according to the follow illustration:
138 %
139 % |--HPx{1}
140 % |
141 % |--HPy{1}          |--HPx{N}

```

```

142 %      |      |
143 %  S---|      |--HPy{N}
144 %      |      |
145 %      |--LP--- ... ---|
146 %      |      |
147 %      |      |--LP
148 %
149 %  S - Signal to be decomposed.
150 %  N - Number of channels to compute.
151 %  D - Downsample LP-component between channels.
152
153 % Filter coefficients for analyzing filters.
154 hx = [0.0625 0.25 0.375 0.25 0.0625];
155 gx = [0 1 -1];
156
157 lpChannel = img;
158
159 if(sampl)
160     for iChannel = 1:numChannels
161         hpChannels{iChannel}(:, :, 1) = mxSimpleDiff(lp
162             hpChannels{iChannel}(:, :, 2) = mxSimpleDiff(lp
163
164         lpChannel = imfilter(imfilter(lpChannel, hx, 'r
165             hx', 'replicate', 'conv'));
166         lpChannel = lpChannel(1:2:end, 1:2:end);
167     end
168 else
169     for iChannel = 1:numChannels
170         hpChannels{iChannel}(:, :, 1) = mxSimpleDiff(lp
171             2^(iChannel-1)-1);
172         hpChannels{iChannel}(:, :, 2) = mxSimpleDiff(lp
173             2^(iChannel-1)-1)';
174
175         lpChannel = imfilter(imfilter(lpChannel, zeroF
176             'replicate', 'conv'), zeroPad(hx, iChannel)
177     end
178 end
179 % -----
180
181 % ----- IDWT -----
182 function img = idwt(hpChannels, lpChannel, sampl)
183 %IDWT Reconstructs a signal that has been decomposed
184 %  S = IDWT(HPCHANNELS, LPCHANNEL, SAMPL)
185 %  HPCHANNELS, LPCHANNEL - Output from DWT, write 'h
186 %  SAMPL - Boolean that indicates we
187 %  downsampling during decomp
188
189 % Filter coefficients for reconstructing filters.
190 hx = [0.0625 0.25 0.375 0.25 0.0625];
191 kx = [-0.00390625 -0.03515625 -0.14453125 -0.36328125
192     0.14453125 0.03515625 0.00390625 0];
193 lx = [0.001953125 0.015625 0.0546875 0.109375 0.63671
194     0.0546875 0.015625 0.001953125];
195
196 % Split signal into channels with or without downsamp
197 % channels.
198 iChannel = length(hpChannels);

```

```

199 if(sampl)
200     while iChannel > 0
201         hpChannelX = imfilter(imfilter(hpChannels{iCh
202             lx', 'replicate', 'conv'), kx, 'replicate'
203
204         hpChannelY = imfilter(imfilter(hpChannels{iCh
205             kx', 'replicate', 'conv'),lx, 'replicate'
206
207         lpChannel = interp2(lpChannel,'spline');
208         if mod(size(hpChannelX,1),2) == 0
209             lpChannel = [lpChannel;lpChannel(end,:)];
210         end
211         if mod(size(hpChannelX,2),2) == 0
212             lpChannel = [lpChannel lpChannel(:,end)];
213         end
214
215         lpChannel = imfilter(imfilter(lpChannel,hx','
216             'replicate');
217         lpChannel = hpChannelX + hpChannelY +lpChanne
218         iChannel = iChannel-1;
219     end
220 else
221     while iChannel > 0
222         hpChannelX = imfilter(imfilter(hpChannels{iCh
223             zeroPad(lx,iChannel)', 'replicate', 'conv
224             zeroPad(kx,iChannel), 'replicate', 'conv'
225
226         hpChannelY = imfilter(imfilter(hpChannels{iCh
227             zeroPad(kx,iChannel)', 'replicate', 'conv
228             zeroPad(lx,iChannel), 'replicate', 'conv'
229
230         lpChannel = imfilter(imfilter(lpChannel,zeroF
231             'replicate'),zeroPad(hx,iChannel),'repl
232
233         lpChannel = hpChannelX + hpChannelY +lpChanne
234         iChannel = iChannel-1;
235     end
236 end
237 img = lpChannel;
238 % -----
239
240 % ----- SOFTTHRESH -----
241 function U = softThresh( V, tMax, tMin, sigma, alpha,
242 %SOFTTHRESH performs soft thresholding of the values
243 % Y = softThresh( X, TMAX, TMIN, S, A, L ) performs
244 % thresholding of X with the threshold t computed a
245 %
246 % ( (TMAX-A*(L-1))*S if TMAX-A*(L-1)>TMIN
247 % t = <
248 % ( TMIN*S otherwise
249
250 % Calculate a threshold
251 c = tMax - alpha*(lev-1);
252
253 if c > tMin
254     t = c*sigma;
255 else

```

```

256     t = tMin*sigma;
257 end
258 % Apply soft thresholding
259 U = sign(V).*(abs(V)-t).*(abs(V) > t);
260 % -----
261
262 % ----- EGAG -----
263 function Y = egag( X, b, c, t1, t2, t3 )
264 %EGAG Signal enhancement of X through Generalized Ada
265 %   Y = EGAG( X, B, C, T1, T2, T3 ).
266 %   X is the signal that you want to enhance, it can
267 %   a matrix. Parameters B and C control the amount c
268 %   T2 and T3 are thresholds. Values in [0 T1] are ma
269 %   [T2 T3] are amplified and values in ]T1 T2[ and a
270 %   altered.
271
272 a = 1./(sigm(c*(1-b))-sigm(-c*(1+b)));
273 signx = sign(X);
274 U = signx.*(abs(X)-t2)./(t3-t2);
275 U_bar = a*(t3-t2)*(sigm(c*(U-b))-sigm(-c*(U+b)));
276
277 Y = (signx.*t2+U_bar).*((abs(X) <= t3).*((abs(X) >= t
278     X.*((abs(X) > t3) + (abs(X) < t2)).*(abs(X)>t1);
279
280 function Y = sigm( X )
281 %SIGM A sigmoid function
282 %   Y = SIGM(X) = 1./(1+exp(-X));
283
284 Y = 1./(1+exp(-X));
285 % -----
286
287 % ----- ZEROPAD -----
288 function out = zeroPad(in, n)
289 %ZEROPAD Insert zeros between elements of a vector or
290 %   Y = ZEROPAD(X,N) inserts 2^(N-1)-1 zeros between
291 %   vector or matrix X. If X is a matrix X will be ze
292 %   dimensions.
293
294 if (n <= 0)
295     error('n must be equal to or larger than one');
296 end
297 if (n == 1)
298     out = in;
299 else
300     n = 2^(n-1);
301     out = zeros(size(in,1),n*size(in,2)-n+1);
302     out(:,1:n:end)=in;
303 end
304 % -----
305
306 % ----- LOOKUP -----
307 function Y = lookUp(funTable,res,X)
308 %LOOKUP Look up values in a function table with NN-in
309 %   Y = LOOKUP(FUNTABLE, R, X)
310 %   FUNTABLE - table of function values.
311 %   R         - table resolution.
312 %   X         - values whos correspodng function valu

```

```

313 %               are to be looked up.
314
315 % Dummy variables intended to optimize speed
316 tVar1 = (1/res); % Multiplication is faster than divi
317 tVar2 = length(funTable)/2;
318
319 Y = funTable(ceil(X*tVar1+tVar2));
320 % -----
321
322 % ----- SHOWGRAPHS -----
323 function showGraphs(j,alpha,tMin,tMax,sigma,egagTable
324 %SHOWGRAPHS Displays graphs of thresholding and enhan
325 x = linspace(-1,1,1/res);
326 numC = floor(j/2);
327 % Plot softThresh for fine scale levels of DWT
328 fig1 = figure;
329 set(fig1,'Name','Thresholding and enhancement functio
330 for channel = 1:numC
331     subplot(1,numC+1,channel);
332     plot(x,softThresh(x, tMax, tMin, sigma, alpha, ch
333     axis square;title(sprintf('softThreshold in chann
334     xlabel x;ylabel('softThresh(x) ');
335 end
336 %Plot egag function
337 subplot(1,numC+1,numC+1);plot(x,lookUp(egagTable,res,
338 title('Generalized Adaptive Gain function');xlabel x;
339 % -----
340
341 function [ x1 x2 ] = getUserCoordinates(img)
342 %GETUSERCOORDINATES
343 % [ X1 X2 ] = GETUSERCOORDINATES(IMG)
344 % Lets the user choose two points in an image and ret
345 % the coordinates of each point in the row vectors X1
346
347 x1 = zeros(1,2);
348 x2 = zeros(1,2);
349
350 fig = figure;
351 imshow(img,[min(img(:)),max(img(:))]);axis image;colo
352 set(fig,'Name','Select area for estimation of standar
353
354 disp('Select upper left corner of preferred area with
355 t = waitforbuttonpress;
356 [x1(1,1) x1(1,2)] = ginput(1);
357 disp('Select lower right corner of preferred area wit
358 t = waitforbuttonpress;
359 [x2(1,1) x2(1,2)] = ginput(1);
360
361 x1 = ceil(x1);
362 x2 = floor(x2);
363 close(fig);
364 % -----

```








C.2.2 Utan sampling

multiScaleEnh (1 call, 0.733 sec)







Generated 10-May-2007 01:36:47 using real time.

M-function in file <H:\edgy\Implementering\profilekoder\artikel2\multiScaleEnh.m>

[\[Copy to new window for comparing multiple runs\]](#)

Refresh					
<input type="checkbox"/> Show parent functions <input checked="" type="checkbox"/> Show busy lines <input checked="" type="checkbox"/> Show child functions <input type="checkbox"/> Show M-Lint results <input checked="" type="checkbox"/> Show file coverage <input checked="" type="checkbox"/> Show function listing					
Lines where the most time was spent					
Line Number	Code	Calls	Total Time	% Time	Time Plot
127	outImg = exp(idwt(hpChannels, ...	1	0.250 s	34.1%	
98	[hpChannels lpChannels] = dwt(...	1	0.109 s	14.9%	
38	addpath common\;	1	0.062 s	8.5%	
40	addpath artikell\;	1	0.046 s	6.3%	
119	hpChannels{iChannel}(:, :, 1) = ...	3	0.032 s	4.4%	
Other lines & overhead			0.234 s	31.9%	
Totals			0.733 s	100%	

Children (called functions)

Function Name	Function Type	Calls	Total Time	% Time	Time Plot
multiScaleEnh>idwt	M-subfunction	1	0.234 s	31.9%	
addpath	M-function	3	0.140 s	19.1%	
multiScaleEnh>dwt	M-subfunction	1	0.109 s	14.9%	
multiScaleEnh>softThresh	M-subfunction	4	0.046 s	6.3%	
multiScaleEnh>lookUp	M-subfunction	6	0.032 s	4.4%	
mean	M-function	1	0.016 s	2.2%	

IsScalar	M-function	12	0 s	0%	
im2single	M-function	1	0 s	0%	
multiScaleEnh>egag	M-subfunction	1	0 s	0%	
Self time (built-ins, overhead, etc.)			0.156 s	21.3%	<div style="width: 21.3%; height: 10px; background-color: blue;"></div>
Totals			0.733 s	100%	

Coverage results

[[Show coverage for parent directory](#)]

Total lines in file	368
Non-code lines (comments, blank lines)	219
Code lines (lines that can run)	149
Code lines that did run	33
Code lines that did not run	116
Coverage (did run/can run)	22.15 %

Function listing

Color highlight code according to

```

time  calls  line
      1 function [ outImg execTime ] = multiScaleEnh(inImg,j,
      2                                     sigma,t1,
      3                                     sampl)
      4
      5 %MULTISCALEENH Multi-scale enhancement of medical ult
      6 %   [ENHANCEDIMAGE EXECUTIONTIME] =
      7 %   MULTISCALENH( INIMG,J,ALPHA,TMIN,TMAX,SIGMA,T1,T2,
      8 %
      9 %   Returns:
     10 %   outImg      Processed Image.
     11 %   execTime    Execution time for image enhancement
     12 %
     13 %   Parameters:
     14 %   INIMG      -   Input image.
     15 %   J          -   Number of channels to be computed by
     16 %   ALPHA      -   Decreasing factor for soft threshold
     17 %                  channels in DWT.
     18 %   TMIN       -   Minimum scaling factor for soft thre
     19 %   TMAX       -   Maximum scaling factor for soft thre
     20 %   SIGMA      -   Estimate of the standard deviation c
     21 %   T1         -   Hard threshold.
     22 %   T2         -   Lower bound for edge enhancement thr
     23 %                  Adaptive Gain of DWT coefficients.
     24 %   T3         -   Upper bound for edge enhancement thr
     25 %                  Adaptive Gain of DWT coefficients.
     26 %   C          -   Gain in GAG-function.
     27 %   B          -   Controls shape and sign of GAG-funct

```

```

28 %   GRAPHS    -   Controls whether graphs of thresholdi
29 %               functions should be plotted or not.
30 %
31 %   Licensed under BSD as a part of EDGY project sour
32 %   see readme.txt file. For more information see prc
33 %
34 %   Revision: 1.0
35 %   Date: 2007/05/08
36 %   Author: Alexander Tuttle, Erik Ringaby
37
0.06 1 38 addpath common\;
0.03 1 39 addpath common\mxSimpleDiff;
0.05 1 40 addpath artikell\;
41
1 42 if ~(IsScalar(j) && IsScalar(alpha) && IsScalar(tMax)
43     && IsScalar(t1) && IsScalar(t2) && IsScalar(t
44     && IsScalar(b) && IsScalar(sigma) && IsScalar
45     && IsScalar(sampl))
46     error('Parameters must be scalar.');
```

47 end

```

1 48 if ~(numel(j) == 1 && isnumeric(j) &&...
49     (j > 0) && (round(j) == j) && j < 6 );
50     error('The number of levels must be a positive in
51 end
1 52 if (tMin >= tMax)
53     error('t_min must be less than t_max');
54 end
1 55 if ~(t1 >= 0 && t2 >= t1 && t3 > t2 && t3 <= 1)
56     error('t1, t2 and t3 must satisfy 0 <= t1 <= t2 <
57 end
58
59
60 % Convert the image to gray scale.
0.02 1 61 inImg = im2single(mear(inImg,3));
62
1 63 if sampl && min(size(inImg)) <= 2^j
64     error('The input image is too small to be downsar
65 end
66
67 % If sigma is specified as 0 the user is prompted to
68 % image that is adequate for estimating the standard
69 % noise.
0.02 1 70 logImg = log(inImg+eps);
1 71 if sigma == 0
72     [upperLeft lowerRight] = getUserCoordinates(inImg
73     rows = min(upperLeft(1,2),lowerRight(1,2)):max(lc
74     cols = min(upperLeft(1,1),lowerRight(1,1)):max(lc
75     imgSelection = logImg(rows,cols);
76     sigma = std(imgSelection(:));
77 end
78
79 % Compute the function table for the enhancement func
< 0.01 1 80 res = 0.001; %table resolution
1 81 s = -1; %lower bound for table
< 0.01 1 82 e = 1; %upper bound for table
1 83 egagTable = egag(s:res:e, b, c, t1, t2, t3);
84
```

```

85 % Plot graphs of thresholding and enhancement function
1 86 if graphs
87     showGraphs(j,alpha,tMin,tMax,sigma,egagTable,res)
88 end
89
90 % Start timer for measuring execution time of image e
< 0.01 1 91 tic;
92
93 % Compute the natural logarithm of the image to separ
94 % noise.
0.03 1 95 inImg = log(inImg+eps);
96
97 % ----- DWT -----
0.11 1 98 [hpChannels lpChannels] = dwt(inImg,j,sampl);
99 % -----
100
101 % ----- Soft thresholding -----
102 % The coefficients below describe the relations betwe
103 % deviation of the noise in the image and the respect
< 0.01 1 104 sigmaxCoeff = [0.7122 0.2828 0.1907 0.1472 0.1217];
< 0.01 1 105 sigmayCoeff = [0.3416 0.1946 0.1468 0.1194 0.1012];
106
107 for level = 1:floor(j/2)
0.03 2 108     hpChannels{level}(:, :, 1) = softThresh(hpChannels{1
109         tMax, tMin, sigmaxCoeff(level)*sigma, alpha,
0.03 2 110     hpChannels{level}(:, :, 2) = softThresh(hpChannels{1
111         tMax, tMin, sigmayCoeff(level)*sigma, alpha,
2 112 end
113 % -----
114
115 % ----- Enhancement through Generalized Adaptiv
1 116 for iChannel = ceil(j/2):j
0.03 3 117     M1 = max(abs(reshape(hpChannels{iChannel}(:, :, 1),
0.02 3 118     M2 = max(abs(reshape(hpChannels{iChannel}(:, :, 2),
0.03 3 119     hpChannels{iChannel}(:, :, 1) = M1*lookUp(egagTable,
120         hpChannels{iChannel}(:, :, 1)/M1);
0.03 3 121     hpChannels{iChannel}(:, :, 2) = M2*lookUp(egagTable,
122         hpChannels{iChannel}(:, :, 2)/M2);
3 123 end
124 % -----
125
126 % ----- IDWT -----
0.25 1 127 outImg = exp(idwt(hpChannels, lpChannels, sampl));
128 % -----
< 0.01 1 129 execTime = toc;
130
131 % ##### Help functions #####
132
133 % ----- DWT -----
134 function [hpChannels lpChannel] = dwt(img,numChannels
135 %DWT Frequency decomposition of 2D signal.
136 % [HP LP] = DWT(S,N,D) decomposes the signal into f
137 % according to the follow illustration:
138 %
139 % |--HPx{1}
140 % |
141 % |--HPy{1}          |--HPx{N}

```

```

142 %      |      |
143 %  S---|      |--HPy{N}
144 %      |      |
145 %      |--LP--- ... ---|
146 %      |      |
147 %      |      |--LP
148 %
149 %  S - Signal to be decomposed.
150 %  N - Number of channels to compute.
151 %  D - Downsample LP-component between channels.
152
153 % Filter coefficients for analyzing filters.
154 hx = [0.0625 0.25 0.375 0.25 0.0625];
155 gx = [0 1 -1];
156
157 lpChannel = img;
158
159 if(sampl)
160     for iChannel = 1:numChannels
161         hpChannels{iChannel}(:, :, 1) = mxSimpleDiff(lp
162             hpChannels{iChannel}(:, :, 2) = mxSimpleDiff(lp
163 %         hpChannels{iChannel}(:, :, 1) = imfilter(lpCh
164 %         hpChannels{iChannel}(:, :, 2) = imfilter(lpCh
165
166         lpChannel = imfilter(imfilter(lpChannel, hx, 'r
167             hx', 'replicate', 'conv'));
168         lpChannel = lpChannel(1:2:end, 1:2:end);
169     end
170 else
171     for iChannel = 1:numChannels
172         hpChannels{iChannel}(:, :, 1) = mxSimpleDiff(lp
173             2^(iChannel-1)-1);
174         hpChannels{iChannel}(:, :, 2) = mxSimpleDiff(lp
175             2^(iChannel-1)-1)';
176 %         hpChannels{iChannel}(:, :, 1) = imfilter(lpCh
177 %         hpChannels{iChannel}(:, :, 2) = imfilter(lpCh
178
179         lpChannel = imfilter(imfilter(lpChannel, zeroF
180             'replicate', 'conv'), zeroPad(hx, iChannel)
181     end
182 end
183 % -----
184
185 % ----- IDWT -----
186 function img = idwt(hpChannels, lpChannel, sampl)
187 %IDWT Reconstructs a signal that has been decomposed
188 %  S = IDWT(HPCHANNELS, LPCHANNEL, SAMPL)
189 %  HPCHANNELS, LPCHANNEL - Output from DWT, write 'h
190 %  SAMPL - Boolean that indicates we
191 %          downsampling during decomp
192
193 % Filter coefficients for reconstructing filters.
194 hx = [0.0625 0.25 0.375 0.25 0.0625];
195 kx = [-0.00390625 -0.03515625 -0.14453125 -0.36328125
196       0.14453125 0.03515625 0.00390625 0];
197 lx = [0.001953125 0.015625 0.0546875 0.109375 0.63671
198       0.0546875 0.015625 0.001953125];

```

```

199
200 % Split signal into channels with or without downsamp
201 % channels.
202 iChannel = length(hpChannels);
203 if(sampl)
204     while iChannel > 0
205         hpChannelX = imfilter(imfilter(hpChannels{iCh
206             lx', 'replicate', 'conv'), kx, 'replicate
207
208         hpChannelY = imfilter(imfilter(hpChannels{iCh
209             kx', 'replicate', 'conv'),lx, 'replicate'
210
211         lpChannel = interp2(lpChannel,'spline');
212         if mod(size(hpChannelX,1),2) == 0
213             lpChannel = [lpChannel;lpChannel(end,:)];
214         end
215         if mod(size(hpChannelX,2),2) == 0
216             lpChannel = [lpChannel lpChannel(:,end)];
217         end
218
219         lpChannel = imfilter(imfilter(lpChannel,hx','
220             'replicate');
221         lpChannel = hpChannelX + hpChannelY +lpChanne
222         iChannel = iChannel-1;
223     end
224 else
225     while iChannel > 0
226         hpChannelX = imfilter(imfilter(hpChannels{iCh
227             zeroPad(lx,iChannel)', 'replicate', 'conv
228             zeroPad(kx,iChannel), 'replicate', 'conv'
229
230         hpChannelY = imfilter(imfilter(hpChannels{iCh
231             zeroPad(kx,iChannel)', 'replicate', 'conv
232             zeroPad(lx,iChannel), 'replicate', 'conv'
233
234         lpChannel = imfilter(imfilter(lpChannel,zeroF
235             'replicate'),zeroPad(hx,iChannel),'repl
236
237         lpChannel = hpChannelX + hpChannelY +lpChanne
238         iChannel = iChannel-1;
239     end
240 end
241 img = lpChannel;
242 % -----
243
244 % ----- SOFTTHRESH -----
245 function U = softThresh( V, tMax, tMin, sigma, alpha,
246 %SOFTTHRESH performs soft thresholding of the values
247 % Y = softThresh( X, TMAX, TMIN, S, A, L ) performs
248 % thresholding of X with the threshold t computed a
249 %
250 % ( (TMAX-A*(L-1))*S if TMAX-A*(L-1)>TMIN
251 % t = <
252 % ( TMIN*S otherwise
253
254 % Calculate a threshold
255 c = tMax - alpha*(lev-1);

```

```

256
257 if c > tMin
258     t = c*sigma;
259 else
260     t = tMin*sigma;
261 end
262 % Apply soft thresholding
263 U = sign(V).*(abs(V)-t).*(abs(V) > t);
264 % -----
265
266 % ----- EGAG -----
267 function Y = egag( X, b, c, t1, t2, t3 )
268 %EGAG Signal enhancement of X through Generalized Ada
269 % Y = EGAG( X, B, C, T1, T2, T3 ).
270 % X is the signal that you want to enhance, it can
271 % a matrix. Parameters B and C control the amount c
272 % T2 and T3 are thresholds. Values in [0 T1] are ma
273 % [T2 T3] are amplified and values in ]T1 T2[ and a
274 % altered.
275
276 a = 1./(sigm(c*(1-b))-sigm(-c*(1+b)));
277 signx = sign(X);
278 U = signx.*(abs(X)-t2)./(t3-t2);
279 U_bar = a*(t3-t2)*(sigm(c*(U-b))-sigm(-c*(U+b)));
280
281 Y = (signx.*t2+U_bar).*((abs(X) <= t3).*((abs(X) >= t
282     X.*((abs(X) > t3) + (abs(X) < t2)).*(abs(X)>t1);
283
284 function Y = sigm( X )
285 %SIGM A sigmoid function
286 % Y = SIGM(X) = 1./(1+exp(-X));
287
288 Y = 1./(1+exp(-X));
289 % -----
290
291 % ----- ZEROPAD -----
292 function out = zeroPad(in, n)
293 %ZEROPAD Insert zeros between elements of a vector or
294 % Y = ZEROPAD(X,N) inserts 2^(N-1)-1 zeros between
295 % vector or matrix X. If X is a matrix X will be ze
296 % dimensions.
297
298 if (n <= 0)
299     error('n must be equal to or larger than one');
300 end
301 if (n == 1)
302     out = in;
303 else
304     n = 2^(n-1);
305     out = zeros(size(in,1),n*size(in,2)-n+1);
306     out(:,1:n:end)=in;
307 end
308 % -----
309
310 % ----- LOOKUP -----
311 function Y = lookUp(funTable,res,X)
312 %LOOKUP Look up values in a function table with NN-in

```

```

313 %   Y = LOOKUP(FUNTABLE, R, X)
314 %   FUNTABLE - table of function values.
315 %   R         - table resolution.
316 %   X         - values whos correspoding function valu
317 %               are to be looked up.
318
319 % Dummy variables intended to optimize speed
320 tVar1 = (1/res); % Multiplication is faster than divi
321 tVar2 = length(funTable)/2;
322
323 Y = funTable(ceil(X*tVar1+tVar2));
324 % -----
325
326 % ----- SHOWGRAPHS -----
327 function showGraphs(j,alpha,tMin,tMax,sigma,egagTable
328 %SHOWGRAPHS Displays graphs of thresholding and enhan
329 x = linspace(-1,1,1/res);
330 numC = floor(j/2);
331 % Plot softThresh for fine scale levels of DWT
332 fig1 = figure;
333 set(fig1,'Name','Thresholding and enhancement functio
334 for channel = 1:numC
335     subplot(1,numC+1,channel);
336     plot(x,softThresh(x, tMax, tMin, sigma, alpha, ch
337     axis square;title(sprintf('softThreshold in chann
338     xlabel x;ylabel('softThresh(x)');
339 end
340 %Plot egag function
341 subplot(1,numC+1,numC+1);plot(x,lookUp(egagTable,res,
342 title('Generalized Adaptive Gain function');xlabel x;
343 % -----
344
345 function [ x1 x2 ] = getUserCoordinates(img)
346 %GETUSERCOORDINATES
347 %   [ X1 X2 ] = GETUSERCOORDINATES(IMG)
348 % Lets the user choose two points in an image and ret
349 % the coordinates of each point in the row vectors X1
350
351 x1 = zeros(1,2);
352 x2 = zeros(1,2);
353
354 fig = figure;
355 imshow(img,[min(img(:)),max(img(:))]);axis image;colo
356 set(fig,'Name','Select area for estimation of standar
357
358 disp('Select upper left corner of preferred area with
359 t = waitforbuttonpress;
360 [x1(1,1) x1(1,2)] = ginput(1);
361 disp('Select lower right corner of preferred area wit
362 t = waitforbuttonpress;
363 [x2(1,1) x2(1,2)] = ginput(1);
364
365 x1 = ceil(x1);
366 x2 = floor(x2);
367 close(fig);
368 % -----

```


C.3 Tidsanalys för delsystem 4

AWMF (1 call, 8.312 sec)

Generated 09-May-2007 23:14:34 using real time.







M-function in file <H:\edgy\Implementering\profilekoder\artikel3\AWMF.m>

[\[Copy to new window for comparing multiple runs\]](#)



Refresh

☐ Show parent functions ☒ Show busy lines ☒ Show child functions
☐ Show M-Lint results ☒ Show file coverage ☒ Show function listing

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
116	[sortInPicLocal index] = sort(...	126024	3.097 s	37.3%	
112	wLocal = round(max(wLocal,0));...	126024	1.016 s	12.2%	
113	sumW = sum(wLocal(:));	126024	0.933 s	11.2%	
108	inPicLocal = inPicExt(iRow-K :...	126024	0.734 s	8.8%	
111	wLocal = (wCenter - distMap*qu...	126024	0.578 s	7.0%	
Other lines & overhead			1.954 s	23.5%	
Totals			8.312 s	100%	

Children (called functions)

Function Name	Function Type	Calls	Total Time	% Time	Time Plot
imfilter	M-function	2	0.062 s	0.7%	
repmat	M-function	8	0.015 s	0.2%	
meshgrid	M-function	1	0 s	0%	
Self time (built-ins, overhead, etc.)			8.235 s	99.1%	
Totals			8.312 s	100%	

Coverage results

[[Show coverage for parent directory](#)]

Total lines in file	157
Non-code lines (comments, blank lines)	86
Code lines (lines that can run)	71
Code lines that did run	47
Code lines that did not run	24
Coverage (did run/can run)	66.20 %

Function listing

Color highlight code according to

```

time  calls  line
1 function [outPic time] = AWMF(inPic, c, K, wCenter, g
2 %AWMF Adaptive Weighted Median Filter
3 % [OUTPIC TIME] = AWMF(INPIC, C, K, WCENTER, GRAPHS
4 % use an Adaptive Weighted Median Filter for reduci
5 % result, OUTPIC, has the same size and class as IN
6 % for the function is returned in the variable TIME
7 %
8 % The weights, [W], for a sequence, [X], extends th
9 % For example, if W = [w1=2 w2=0 w3=3] the weighted
10 % sequence x = [x1 x2 x3] is [x1 x1 x3 x3 x3].
11 % The weights for a local neighbourhood is given by
12 %
13 % W(i,j) = [WCENTER - distance * C * LocalMean / Lo
14 %
15 % For more information and details see project docu
16 %
17 % INPIC      Input grayscale picture, UINT8
18 % C          Scalar constant, C >= 0 (if 0, me
19 % K          Size of 2D filter kernel (N*N), N
20 %           K <= min(size(INPIC))
21 % WCENTER   Kernel Centrum Weight, WCENTER >=
22 % GRAPHS     if 1 shows extra graphs for devel
23 %
24 % Class support for inputs INPIC
25 %      uint8
26 % Class support for inputs C, K, WCENTER, GRAPHS:
27 %      double
28 %
29 % Class for output OUTPIC
30 %      uint8
31 % Class for output TIME
32 %      double
33 %
34 % Requirements: IMFILTER in Image Processing Toolbo
35 %
36 % See also MEDIAN.
37 %
38 % Licensed under BSD as a part of EDGY project sour

```

```

39 % see readme.txt file. For more information see pro
40 %
41 % Revision: 1.0
42 % Date: 2007/05/08
43 % Author: [Per Thyr]
44
1 45 if (~isnumeric(c) || numel(c) ~=1 || c < 0)
46     error('C must be a positive scalar.');
```

< 0.01 1 47 elseif (~isnumeric(K) || numel(K) ~=1 ||...

```

48     K < 0 || round(K) ~= K)
49     error('K must be a positive integer scalar.');
```

0.02 1 50 elseif ndims(inPic) > 2

```

51     error('Input picture must be in 2 dimensions')
1 52 elseif (K > min(size(inPic)))
53     error('K <= min(size(INPIC)).');
```

< 0.01 1 54 elseif (~isnumeric(wCenter) || numel(wCenter) ~=1 ||

```

55     error('WCENTER must be a positive scalar.');
```

< 0.01 1 56 elseif (~isnumeric(graphs) || numel(graphs) ~=1 ||...

```

57     graphs > 2 || graphs < 0)
58     error('GRAPHS must be 0 or 1.');
```

< 0.01 1 59 elseif (~isa(inPic, 'uint8'))

```

60     error('Input picture must be UINT8-class.')
61 end
62
63 % Starts calculating runningtime
< 0.01 1 64 tic
65
66 % Convert uint8 format to double. Add 1 to avoid div
0.02 1 67 inPic = double(inPic) + 1;
68
69 % Local neighbourhood sidelength and area
< 0.01 1 70 N = 2*K+1;
< 0.01 1 71 sqrN = N^2;
72
73 % Calculate quota (c*var/mean)
0.05 1 74 kernelMean = ones(N,N)/(N*N);
0.02 1 75 inPicMean = imfilter(inPic, kernelMean, 'conv', 'repl:
0.02 1 76 inPicMeanSq = imfilter(inPic.^2, kernelMean, 'conv',
77
0.02 1 78 quotaWeight = c*(inPicMeanSq./inPicMean - inPicMean);
1 79 clear inPicMean inPicMeanSq;
80
81 % Makes extended quotaWight
0.02 1 82 tquotaWeightExt = repmat(quotaWeight(:,1),1,K) quotaW
83     repmat(quotaWeight(:,end),1,K)];
1 84 quotaWeightExt = [repmat(tquotaWeightExt(1,:),K,1); t
85     repmat(tquotaWeightExt(end,:),K,1
1 86 clear tquotaWeightExt;
87
88 % Makes extended InPic
0.02 1 89 tmpInPicExt = [repmat(inPic(:,1),1,K) inPic ...
90     repmat(inPic(:,end),1,K)];
1 91 inPicExt = [repmat(tmpInPicExt(1,:),K,1); tmpInPicExt
92     repmat(tmpInPicExt(end,:),K,1)];
0.02 1 93 clear tmpInPicExt;
94
95 % Calculate distancematrix
```

```

1  96 [dMeshCol, dMeshRow] = meshgrid(-K:K,-K:K);
1  97 distMap = sqrt((dMeshRow.^2) + (dMeshCol.^2));
    98
    99 % Initiation
1  100 outPic = zeros(size(inPic));
< 0.01 1  101 [numRowExt, numColExt] = size(inPicExt);
    102
    103 % Calculate new pixelevalue, pixel by pixel
1  104 for iRow = 1+K : numRowExt-K
177 105     for iCol = 1+K : numColExt - K
    106
    107         % Local Neighbourhood
0.73 126024 108         inPicLocal = inPicExt(iRow-K : iRow+K, iCol-K
    109
    110         % Calculate local weightmatrix, wLocal
0.58 126024 111         wLocal = (wCenter - distMap*quotaWeightExt(iR
1.02 126024 112         wLocal = round(max(wLocal,0)); % negative 2 z
0.93 126024 113         sumW = sum(wLocal(:));
    114
    115         % Sort
3.10 126024 116         [sortInPicLocal index] = sort(inPicLocal(:),
    117
    118         % Calculate median (output value)
< 0.01 126024 119         newPxPos = (sumW+1)/2;
< 0.01 126024 120         sumW = 0;
0.11 126024 121         for iPos = 1:sqrN
0.03 5169838 122             sumW = sumW + wLocal(index(iPos));
0.18 5169838 123             if sumW >= newPxPos
< 0.01 126024 124                 break
    125             end
0.19 5043814 126         end
< 0.01 126024 127         yWM = sortInPicLocal(iPos);
< 0.01 126024 128         outPic(iRow - K, iCol - K) = yWM;
0.34 126024 129     end
177 130 end
    131
    132 % Development graphs
1  133 if graphs == 1
    134     quota = quotaWeight./c;
    135     figure('Name', ['AWMF: Development graphs, c = ' n
    136         ', K = ' num2str(K) ...
    137         ' ( ' num2str(N) '*' num2str(N) ' )' ...
    138         ', wCenter = ' num2str(wCenter) ], 'NumberTitl
    139     subplot(3,2,1), imagesc(inPic);
    140         title('InPic'); colormap gray; axis image; co
    141     subplot(3,2,2), imagesc(outPic, [min(inPic(:)) max
    142         title('outPic'); colormap gray; axis image; co
    143     subplot(3,2,3), imagesc(quota);
    144         title('variance/mean quota'); colormap gray; a
    145     subplot(3,2,4), imagesc(quotaWeight);
    146         title('c*variance/mean quota'); colormap gray;
    147     subplot(3,2,5), imagesc(abs(inPic - outPic));
    148         title('|inPic - outPic|'); colormap gray; axi
    149     subplot(3,2,6), imagesc(distMap);
    150         title('distMap'); colormap gray; axis image;
    151 end
    152

```

```
153 % Convert back to uint8
1 154 outPic = uint8(outPic - 1);
155
156 % Stops calculating runningtime
< 0.01 1 157 time = toc;
```

D Tidsanalys utan C-implementering

D.1 Tidsanalys för delsystem 2 utan C-implementering

anisotropicDiffusion (1 call, 7.527 sec)

Generated 10-May-2007 01:40:14 using real time.

M-function in file H:\edgy_ext\artikel1\anisotropicDiffusion.m

[\[Copy to new window for comparing multiple runs\]](#)

Refresh

☐ Show parent functions ☒ Show busy lines ☒ Show child functions
☐ Show M-Lint results ☒ Show file coverage ☒ Show function listing

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time
110	<code>[w(y,x,:), mu(y,x,:)) = ei...</code>	252048	6.732 s	89.4%
99	<code>J = imfilter(imfilter(nablaI,K...</code>	2	0.156 s	2.1%
202	<code>I_t = divergence(a.*I_x + b.*I...</code>	2	0.156 s	2.1%
125	<code>D(:,:,1,2) = lambda(:,:,1).*w(...</code>	2	0.078 s	1.0%
111	<code>end</code>	252048	0.078 s	1.0%
Other lines & overhead			0.327 s	4.3%
Totals			7.527 s	100%

Children (called functions)

Function Name	Function Type	Calls	Total Time	% Time	Time Plo
imfilter	M-function	8	0.217 s	2.9%	<div></div>
divergence	M-function	2	0.141 s	1.9%	<div></div>
addpath	M-function	1	0.032 s	0.4%	
createGaussKernel	M-function	1	0.016 s	0.2%	
IsScalar	M-function	8	0 s	0%	
IsPositiveInteger	M-function	1	0 s	0%	

createDerivKernel	M-function	1	0 s	0%	
squeeze	M-function	8	0 s	0%	
Self time (built-ins, overhead, etc.)			7.121 s	94.6%	<div style="width: 94.6%; height: 10px; background-color: blue;"></div>
Totals			7.527 s	100%	

Coverage results

[[Show coverage for parent directory](#)]

Total lines in file	212
Non-code lines (comments, blank lines)	134
Code lines (lines that can run)	78
Code lines that did run	45
Code lines that did not run	33
Coverage (did run/can run)	57.69 %

Function listing

Color highlight code according to

```

time  calls  line
1 function I = AnisotropicDiffusion( inImage, sigmal, d
2     delta2, s2, alpha, beta, tau, numIter, graphs )
3 %AnisotropicDiffusion Implements Nonlinear Anisotropic
4 %   I_t = AnisotropicDiffusion( path, sigmal, delta1,
5 %   alpha, beta, tau, numIter )
6 %
7 % Returns:
8 %   I_t      Processed Image after t iterations
9 %
10 % Parameters:
11 % inImage   Image to process. Must be gray scale
12 % sigmal    Sigma for derivating kernel
13 % delta1    Cut of tail of derivating kernel at t
14 % sigma2    Sigma for low pass kernel
15 % delta2    Cut of tail of low pass kernel at thi
16 % s2       Stop value for diffusion in gradient
17 % alpha     Isotropic diffusion number
18 % beta      Maximum amount of diffusion. Small be
19 % tau       Time step
20 % numIter   Maximum number
21 % graphs    1 - Show graphs, 0 - Show no graphs
22 %
23 % Requirements IMFILTER and IMSHOW in Image Process
24 % See also IMFILTER, IMSHOW.
25 %
26 % Licensed under BSD as a part of EDGY project sour
27 % see readme.txt file. For more information see prc
28 %
29 % Revision: 0.3

```

```

30 % Date: 2007/04/09
31 % Author: Henrik Wolkesson
32
0.03 1 33 addpath('.../common');
34
35 if 0
36     error('inImage must be ???');
37 end
0.02 1 38 if ~IsScalar(sigm1) || sigm1 < 0 || ...
39     ~IsScalar(delta1) || delta1 < 0 || ...
40     ~IsScalar(sigma2) || sigma2 < 0 || ...
41     ~IsScalar(delta2) || delta2 < 0
42     error('Sigma1, Sigma2, Delta1 and Delta2 must be
43 end
1 44 if ~IsScalar(s2) || s2 < 0 || ~IsScalar(alpha) || alp
45     ~IsScalar(beta) || beta < 0 || ~IsScalar(tau)
46     error('s2, alpha, beta and tau must be a positive
47 end
1 48 if ~IsPositiveInteger(numIter)
49     error('numIter must be positive integer');
50 end
1 51 if graphs < 0 || graphs > 1 || round(graphs) ~= graph
52     error('Graphs must be either 0 or 1');
53 end
54
55 % Section 1
1 56 inImage = double(inImage);
< 0.01 1 57 I=inImage;
58 %figure('Name', 'InputImage');imshow(inImage,[]);
59
60 % Section 2
1 61 derivKernel = createDerivKernel(sigm1, delta1);
0.02 1 62 Kx = createGaussKerne(sigma2,delta2);
63
1 64 if (graphs == 1)
65     figure('Name','Kernels');
66     subplot(1,2,1);plot(Kx);
67     title(['Gauss Kernel (Size: ' num2str(size(Kx,2))
68     subplot(1,2,2);plot(derivKernel);
69     title(['Derivating Kernel (Size: ' num2str(size(d
70 end;
71
1 72 if (size(Kx,2)>size(I,2))
73     warning('Derivating kernel is bigger than image')
74 end
75
1 76 for k = 1:numIter
77 % Section 3
0.05 2 78 I_x=imfilter(I,derivKernel, 'conv', 'replicate');
0.02 2 79 I_y=imfilter(I,derivKernel, 'conv', 'replicate');
80
2 81 if (graphs == 1)
82     figure(32);
83     title('Derivates');
84     subplot(1,2,1);imshow(I_x, []);
85     subplot(1,2,2);imshow(I_y, []);
86 end;

```



```

87
88 % Section 4
2 89 nablaI(:,:,1)=I_x.*I_x;
0.02 2 90 nablaI(:,:,2)=I_x.*I_y;
0.02 2 91 nablaI(:,:,3)=I_y.*I_y;
92
2 93 if (graphs == 1)
94     figure(33);imshow([nablaI(:,:,1) nablaI(:,:,2) na
95 end
96
97 % Section 5
2 98 Ky = Kx';
0.16 2 99 J = imfilter(imfilter(nablaI,Kx, 'conv', 'replicate'),
100 Ky, 'conv', 'replicate');
101
2 102 if (graphs == 1)
103     figure(34);imshow([J(:,:,1) J(:,:,2) J(:,:,3)],[])
104 end
105
106 % Section 6
107 % [w1, mul] = fastEig(J);
2 108 for y=1:size(J,1)
354 109     for x=1:size(J,2)
6.73 252048 110         [w(y,x,:), mu(y,x,:)] = eig([J(y,x,1) J(y
0.08 252048 111     end
354 112 end
113
114 %Section 7
2 115 delta = 1 - beta*(abs(inImage - I));
2 116 delta = delta .* (delta>=0);
117
0.06 2 118 lambda(:,:,1) = alpha.*delta.*((mu(:,:,1,1)-mu(:,:,2,
119     (1-(mu(:,:,1,1)-mu(:,:,2,2)).^2./s2)));
2 120 lambda(:,:,2) = alpha.*delta;
121
122 %lambda(:,:,) = [mu(:,:,1,1) mu(:,:,2,2)];
123 %Section 8
0.03 2 124 D(:,:,1,1) = lambda(:,:,1).*w(:,:,1,1).^2 + lambda(:,
0.08 2 125 D(:,:,1,2) = lambda(:,:,1).*w(:,:,1,1).*w(:,:,2,1) +
2 126 D(:,:,2,1) = D(:,:,1,2);
0.03 2 127 D(:,:,2,2) = lambda(:,:,1).*w(:,:,2,1).^2 + lambda(:,
128
129 % if (graphs == 1)
130 %     figure('Name','D');imshow([D(:,:,1,1) D(:,:,1,2
131 % end
132
2 133 a = squeeze(D(:,:,1,1));
2 134 b = squeeze(D(:,:,1,2));
0.03 2 135 c = squeeze(D(:,:,2,1));
2 136 d = squeeze(D(:,:,2,2));
137
138 % === KLART HIT === %
139 % A1 = zeros(size(I));
140 % A1(1,2) = (a(1,1)+a(1,2)) / 2;
141 % A1(1,1) = a(1,2)*2; % ???
142 % for i = 2:(size(A1, 1)-1)
143 %     for j = 2:(size(A1,2)-1)

```

D.2 Tidsanalys för delsystem 3 utan C-implementering

D.2.1 Med sampling

multiScaleEnh (1 call, 0.654 sec)

Generated 10-May-2007 01:35:40 using real time.







M-function in file <H:\edgy\Implementering\profilekoder\artikel2\multiScaleEnh.m>

[\[Copy to new window for comparing multiple runs\]](#)







Refresh

☐ Show parent functions ☒ Show busy lines ☒ Show child functions
☐ Show M-Lint results ☒ Show file coverage ☒ Show function listing

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
127	outImg = exp(idwt(hpChannels, ...	1	0.343 s	52.4%	
98	[hpChannels lpChannels] = dwt(...	1	0.063 s	9.6%	
38	addpath common\;	1	0.062 s	9.5%	
40	addpath artikell\;	1	0.047 s	7.2%	
95	inImg = log(inImg+eps);	1	0.031 s	4.7%	
Other lines & overhead			0.108 s	16.5%	
Totals			0.654 s	100%	

Children (called functions)

Function Name	Function Type	Calls	Total Time	% Time	Time Plot
multiScaleEnh>idwt	M-subfunction	1	0.328 s	50.2%	
addpath	M-function	3	0.140 s	21.4%	
multiScaleEnh>dwt	M-subfunction	1	0.063 s	9.6%	
multiScaleEnh>softThresh	M-subfunction	4	0.031 s	4.7%	
im2single	M-function	1	0.016 s	2.4%	
multiScaleEnh>lookUp	M-subfunction	6	0.015 s	2.3%	

IsScalar	M-function	12	0 s	0%	
mean	M-function	1	0 s	0%	
multiScaleEnh>egag	M-subfunction	1	0 s	0%	
Self time (built-ins, overhead, etc.)			0.061 s	9.3%	■
Totals			0.654 s	100%	

Coverage results

[[Show coverage for parent directory](#)]

Total lines in file	368
Non-code lines (comments, blank lines)	219
Code lines (lines that can run)	149
Code lines that did run	33
Code lines that did not run	116
Coverage (did run/can run)	22.15 %

Function listing

Color highlight code according to

```

time  calls  line
1 function [ outImg execTime ] = multiScaleEnh(inImg,j,
2                                     sigma,t1,
3                                     sampl)
4
5 %MULTISCALEENH Multi-scale enhancement of medical ult
6 % [ENHANCEDIMAGE EXECUTIONTIME] =
7 % MULTISCALENH( INIMG,J,ALPHA,TMIN,TMAX,SIGMA,T1,T2,
8 %
9 % Returns:
10 % outImg      Processed Image.
11 % execTime    Execution time for image enhancement
12 %
13 % Parameters:
14 % INIMG      - Input image.
15 % J          - Number of channels to be computed by
16 % ALPHA      - Decreasing factor for soft threshold
17 %             channels in DWT.
18 % TMIN       - Minimum scaling factor for soft thre
19 % TMAX       - Maximum scaling factor for soft thre
20 % SIGMA      - Estimate of the standard deviation c
21 % T1         - Hard threshold.
22 % T2         - Lower bound for edge enhancement thr
23 %             Adaptive Gain of DWT coefficients.
24 % T3         - Upper bound for edge enhancement thr
25 %             Adaptive Gain of DWT coefficients.
26 % C          - Gain in GAG-function.
27 % B          - Controls shape and sign of GAG-funct

```

```

28 %   GRAPHS    -   Controls whether graphs of thresholdi
29 %               functions should be plotted or not.
30 %
31 %   Licensed under BSD as a part of EDGY project sour
32 %   see readme.txt file. For more information see prc
33 %
34 %   Revision: 1.0
35 %   Date: 2007/05/08
36 %   Author: Alexander Tuttle, Erik Ringaby
37
0.06 1 38 addpath common\;
0.03 1 39 addpath common\mxSimpleDiff;
0.05 1 40 addpath artikkel1\;
41
1 42 if ~(IsScalar(j) && IsScalar(alpha) && IsScalar(tMax)
43     && IsScalar(t1) && IsScalar(t2) && IsScalar(t
44     && IsScalar(b) && IsScalar(sigma) && IsScalar
45     && IsScalar(sampl))
46     error('Parameters must be scalar.');
```

```

47 end
1 48 if ~(numel(j) == 1 && isnumeric(j) &&...
49     (j > 0) && (round(j) == j) && j < 6 );
50     error('The number of levels must be a positive in
51 end
1 52 if (tMin >= tMax)
53     error('t_min must be less than t_max');
54 end
1 55 if ~(t1 >= 0 && t2 >= t1 && t3 > t2 && t3 <= 1)
56     error('t1, t2 and t3 must satisfy 0 <= t1 <= t2 <
57 end
58
59
60 % Convert the image to gray scale.
0.02 1 61 inImg = im2single(mear(inImg,3));
62
1 63 if sampl && min(size(inImg)) <= 2^j
64     error('The input image is too small to be downsar
65 end
66
67 % If sigma is specified as 0 the user is prompted to
68 % image that is adequate for estimating the standard
69 % noise.
0.02 1 70 logImg = log(inImg+eps);
1 71 if sigma == 0
72     [upperLeft lowerRight] = getUserCoordinates(inImg
73     rows = min(upperLeft(1,2),lowerRight(1,2)):max(lc
74     cols = min(upperLeft(1,1),lowerRight(1,1)):max(lc
75     imgSelection = logImg(rows,cols);
76     sigma = std(imgSelection(:));
77 end
78
79 % Compute the function table for the enhancement func
1 80 res = 0.001; %table resolution
< 0.01 1 81 s = -1; %lower bound for table
< 0.01 1 82 e = 1; %upper bound for table
1 83 egagTable = egag(s:res:e, b, c, t1, t2, t3);
84

```

```

85 % Plot graphs of thresholding and enhancement function
1 86 if graphs
87     showGraphs(j,alpha,tMin,tMax,sigma,egagTable,res)
88 end
89
90 % Start timer for measuring execution time of image e
< 0.01 1 91 tic;
92
93 % Compute the natural logarithm of the image to separ
94 % noise.
0.03 1 95 inImg = log(inImg+eps);
96
97 % ----- DWT -----
0.06 1 98 [hpChannels lpChannels] = dwt(inImg,j,sampl);
99 % -----
100
101 % ----- Soft thresholding -----
102 % The coefficients below describe the relations betwe
103 % deviation of the noise in the image and the respect
< 0.01 1 104 sigmaxCoeff = [0.7122 0.2828 0.1907 0.1472 0.1217];
< 0.01 1 105 sigmayCoeff = [0.3416 0.1946 0.1468 0.1194 0.1012];
106
107 for level = 1:floor(j/2)
0.02 2 108     hpChannels{level}(:, :, 1) = softThresh(hpChannels{1
109         tMax, tMin, sigmaxCoeff(level)*sigma, alpha,
0.02 2 110     hpChannels{level}(:, :, 2) = softThresh(hpChannels{1
111         tMax, tMin, sigmayCoeff(level)*sigma, alpha,
2 112 end
113 % -----
114
115 % ----- Enhancement through Generalized Adaptiv
1 116 for iChannel = ceil(j/2):j
3 117     M1 = max(abs(reshape(hpChannels{iChannel}(:, :, 1),
3 118     M2 = max(abs(reshape(hpChannels{iChannel}(:, :, 2),
0.02 3 119     hpChannels{iChannel}(:, :, 1) = M1*lookUp(egagTable,
120         hpChannels{iChannel}(:, :, 1)/M1);
3 121     hpChannels{iChannel}(:, :, 2) = M2*lookUp(egagTable
122         hpChannels{iChannel}(:, :, 2)/M2);
3 123 end
124 % -----
125
126 % ----- IDWT -----
0.34 1 127 outImg = exp(idwt(hpChannels, lpChannels, sampl));
128 % -----
< 0.01 1 129 execTime = toc;
130
131 % ##### Help functions #####
132
133 % ----- DWT -----
134 function [hpChannels lpChannel] = dwt(img,numChannels
135 %DWT Frequency decomposition of 2D signal.
136 % [HP LP] = DWT(S,N,D) decomposes the signal into f
137 % according to the follow illustration:
138 %
139 % |--HPx{1}
140 % |
141 % |--HPy{1}          |--HPx{N}

```

```

142 %      |      |
143 %  S---|      |--HPy{N}
144 %      |      |
145 %      |--LP--- ... ---|
146 %      |      |
147 %      |      |--LP
148 %
149 %  S - Signal to be decomposed.
150 %  N - Number of channels to compute.
151 %  D - Downsample LP-component between channels.
152
153 % Filter coefficients for analyzing filters.
154 hx = [0.0625 0.25 0.375 0.25 0.0625];
155 gx = [0 1 -1];
156
157 lpChannel = img;
158
159 if(sampl)
160     for iChannel = 1:numChannels
161 %         hpChannels{iChannel}(:, :, 1) = mxSimpleDiff(
162 %         hpChannels{iChannel}(:, :, 2) = mxSimpleDiff(
163         hpChannels{iChannel}(:, :, 1) = imfilter(lpChan
164         hpChannels{iChannel}(:, :, 2) = imfilter(lpChan
165
166         lpChannel = imfilter(imfilter(lpChannel, hx, 'r
167         hx', 'replicate', 'conv'));
168         lpChannel = lpChannel(1:2:end, 1:2:end);
169     end
170 else
171     for iChannel = 1:numChannels
172 %         hpChannels{iChannel}(:, :, 1) = mxSimpleDiff(
173 %         2^(iChannel-1)-1);
174 %         hpChannels{iChannel}(:, :, 2) = mxSimpleDiff(
175 %         2^(iChannel-1)-1)';
176         hpChannels{iChannel}(:, :, 1) = imfilter(lpChan
177         hpChannels{iChannel}(:, :, 2) = imfilter(lpChan
178
179         lpChannel = imfilter(imfilter(lpChannel, zeroF
180         'replicate', 'conv'), zeroPad(hx, iChannel)
181     end
182 end
183 % -----
184
185 % ----- IDWT -----
186 function img = idwt(hpChannels, lpChannel, sampl)
187 %IDWT Reconstructs a signal that has been decomposed
188 %  S = IDWT(HPCHANNELS, LPCHANNEL, SAMPL)
189 %  HPCHANNELS, LPCHANNEL - Output from DWT, write 'h
190 %  SAMPL - Boolean that indicates we
191 %          downsampld during decomp
192
193 % Filter coefficients for reconstructing filters.
194 hx = [0.0625 0.25 0.375 0.25 0.0625];
195 kx = [-0.00390625 -0.03515625 -0.14453125 -0.36328125
196       0.14453125 0.03515625 0.00390625 0];
197 lx = [0.001953125 0.015625 0.0546875 0.109375 0.63671
198       0.0546875 0.015625 0.001953125];

```

```

199
200 % Split signal into channels with or without downsamp
201 % channels.
202 iChannel = length(hpChannels);
203 if(sampl)
204     while iChannel > 0
205         hpChannelX = imfilter(imfilter(hpChannels{iCh
206             lx', 'replicate', 'conv'), kx, 'replicate
207
208         hpChannelY = imfilter(imfilter(hpChannels{iCh
209             kx', 'replicate', 'conv'),lx, 'replicate'
210
211         lpChannel = interp2(lpChannel,'spline');
212         if mod(size(hpChannelX,1),2) == 0
213             lpChannel = [lpChannel;lpChannel(end,:)];
214         end
215         if mod(size(hpChannelX,2),2) == 0
216             lpChannel = [lpChannel lpChannel(:,end)];
217         end
218
219         lpChannel = imfilter(imfilter(lpChannel,hx','
220             'replicate');
221         lpChannel = hpChannelX + hpChannelY +lpChanne
222         iChannel = iChannel-1;
223     end
224 else
225     while iChannel > 0
226         hpChannelX = imfilter(imfilter(hpChannels{iCh
227             zeroPad(lx,iChannel)', 'replicate', 'conv
228             zeroPad(kx,iChannel), 'replicate', 'conv'
229
230         hpChannelY = imfilter(imfilter(hpChannels{iCh
231             zeroPad(kx,iChannel)', 'replicate', 'conv
232             zeroPad(lx,iChannel), 'replicate', 'conv'
233
234         lpChannel = imfilter(imfilter(lpChannel,zeroF
235             'replicate'),zeroPad(hx,iChannel),'repl
236
237         lpChannel = hpChannelX + hpChannelY +lpChanne
238         iChannel = iChannel-1;
239     end
240 end
241 img = lpChannel;
242 % -----
243
244 % ----- SOFTTHRESH -----
245 function U = softThresh( V, tMax, tMin, sigma, alpha,
246 %SOFTTHRESH performs soft thresholding of the values
247 % Y = softThresh( X, TMAX, TMIN, S, A, L ) performs
248 % thresholding of X with the threshold t computed a
249 %
250 %         ( (TMAX-A*(L-1))*S if TMAX-A*(L-1)>TMIN
251 % t = <
252 %         ( TMIN*S           otherwise
253
254 % Calculate a threshold
255 c = tMax - alpha*(lev-1);

```

```

256
257 if c > tMin
258     t = c*sigma;
259 else
260     t = tMin*sigma;
261 end
262 % Apply soft thresholding
263 U = sign(V).*(abs(V)-t).*(abs(V) > t);
264 % -----
265
266 % ----- EGAG -----
267 function Y = egag( X, b, c, t1, t2, t3 )
268 %EGAG Signal enhancement of X through Generalized Ada
269 %   Y = EGAG( X, B, C, T1, T2, T3 ).
270 %   X is the signal that you want to enhance, it can
271 %   a matrix. Parameters B and C control the amount c
272 %   T2 and T3 are thresholds. Values in [0 T1] are ma
273 %   [T2 T3] are amplified and values in ]T1 T2[ and a
274 %   altered.
275
276 a = 1./(sigm(c*(1-b))-sigm(-c*(1+b)));
277 signx = sign(X);
278 U = signx.*(abs(X)-t2)./(t3-t2);
279 U_bar = a*(t3-t2)*(sigm(c*(U-b))-sigm(-c*(U+b)));
280
281 Y = (signx.*t2+U_bar).*((abs(X) <= t3).*((abs(X) >= t
282     X.*((abs(X) > t3) + (abs(X) < t2)).*(abs(X)>t1);
283
284 function Y = sigm( X )
285 %SIGM A sigmoid function
286 %   Y = SIGM(X) = 1./(1+exp(-X));
287
288 Y = 1./(1+exp(-X));
289 % -----
290
291 % ----- ZEROPAD -----
292 function out = zeroPad(in, n)
293 %ZEROPAD Insert zeros between elements of a vector or
294 %   Y = ZEROPAD(X,N) inserts 2^(N-1)-1 zeros between
295 %   vector or matrix X. If X is a matrix X will be ze
296 %   dimensions.
297
298 if (n <= 0)
299     error('n must be equal to or larger than one');
300 end
301 if (n == 1)
302     out = in;
303 else
304     n = 2^(n-1);
305     out = zeros(size(in,1),n*size(in,2)-n+1);
306     out(:,1:n:end)=in;
307 end
308 % -----
309
310 % ----- LOOKUP -----
311 function Y = lookUp(funTable,res,X)
312 %LOOKUP Look up values in a function table with NN-in

```



```

313 %   Y = LOOKUP(FUNTABLE, R, X)
314 %   FUNTABLE - table of function values.
315 %   R         - table resolution.
316 %   X         - values whos correspodng function valu
317 %               are to be looked up.
318
319 % Dummy variables intended to optimize speed
320 tVar1 = (1/res); % Multiplication is faster than divi
321 tVar2 = length(funTable)/2;
322
323 Y = funTable(ceil(X*tVar1+tVar2));
324 % -----
325
326 % ----- SHOWGRAPHS -----
327 function showGraphs(j,alpha,tMin,tMax,sigma,egagTable
328 %SHOWGRAPHS Displays graphs of thresholding and enhan
329 x = linspace(-1,1,1/res);
330 numC = floor(j/2);
331 % Plot softThresh for fine scale levels of DWT
332 fig1 = figure;
333 set(fig1,'Name','Thresholding and enhancement functio
334 for channel = 1:numC
335     subplot(1,numC+1,channel);
336     plot(x,softThresh(x, tMax, tMin, sigma, alpha, ch
337     axis square;title(sprintf('softThreshold in chann
338     xlabel x;ylabel('softThresh(x)');
339 end
340 %Plot egag function
341 subplot(1,numC+1,numC+1);plot(x,lookUp(egagTable,res,
342 title('Generalized Adaptive Gain function');xlabel x;
343 % -----
344
345 function [ x1 x2 ] = getUserCoordinates(img)
346 %GETUSERCOORDINATES
347 %   [ X1 X2 ] = GETUSERCOORDINATES(IMG)
348 % Lets the user choose two points in an image and ret
349 % the coordinates of each point in the row vectors X1
350
351 x1 = zeros(1,2);
352 x2 = zeros(1,2);
353
354 fig = figure;
355 imshow(img,[min(img(:)),max(img(:))]);axis image;colo
356 set(fig,'Name','Select area for estimation of standar
357
358 disp('Select upper left corner of preferred area with
359 t = waitforbuttonpress;
360 [x1(1,1) x1(1,2)] = ginput(1);
361 disp('Select lower right corner of preferred area wit
362 t = waitforbuttonpress;
363 [x2(1,1) x2(1,2)] = ginput(1);
364
365 x1 = ceil(x1);
366 x2 = floor(x2);
367 close(fig);
368 % -----

```







D.2.2 Utan sampling

multiScaleEnh (1 call, 0.779 sec)







Generated 10-May-2007 01:33:57 using real time.

M-function in file <H:\edgy\Implementering\profilekoder\artikel2\multiScaleEnh.m>

[\[Copy to new window for comparing multiple runs\]](#)

Refresh					
<input type="checkbox"/> Show parent functions <input checked="" type="checkbox"/> Show busy lines <input checked="" type="checkbox"/> Show child functions <input type="checkbox"/> Show M-Lint results <input checked="" type="checkbox"/> Show file coverage <input checked="" type="checkbox"/> Show function listing					
Lines where the most time was spent					
Line Number	Code	Calls	Total Time	% Time	Time Plot
127	outImg = exp(idwt(hpChannels, ...	1	0.264 s	33.9%	
98	[hpChannels lpChannels] = dwt(...	1	0.141 s	18.1%	
119	hpChannels{iChannel}(:, :, 1) = ...	3	0.048 s	6.2%	
40	addpath artikell\;	1	0.047 s	6.0%	
38	addpath common\;	1	0.047 s	6.0%	
Other lines & overhead			0.232 s	29.8%	
Totals			0.779 s	100%	

Children (called functions)

Function Name	Function Type	Calls	Total Time	% Time	Time Plot
multiScaleEnh>idwt	M-subfunction	1	0.233 s	29.9%	
multiScaleEnh>dwt	M-subfunction	1	0.141 s	18.1%	
addpath	M-function	3	0.140 s	18.0%	
multiScaleEnh>lookUp	M-subfunction	6	0.078 s	10.0%	
multiScaleEnh>softThresh	M-subfunction	4	0.062 s	8.0%	
im2single	M-function	1	0.016 s	2.1%	

IsScalar	M-function	12	0 s	0%	
mean	M-function	1	0 s	0%	
multiScaleEnh>egag	M-subfunction	1	0 s	0%	
Self time (built-ins, overhead, etc.)			0.109 s	14.0%	■
Totals			0.779 s	100%	

Coverage results

[[Show coverage for parent directory](#)]

Total lines in file	368
Non-code lines (comments, blank lines)	219
Code lines (lines that can run)	149
Code lines that did run	33
Code lines that did not run	116
Coverage (did run/can run)	22.15 %

Function listing

Color highlight code according to

```

time  calls  line
1 function [ outImg execTime ] = multiScaleEnh(inImg,j,
2                                     sigma,t1,
3                                     sampl)
4
5 %MULTISCALEENH Multi-scale enhancement of medical ult
6 % [ENHANCEDIMAGE EXECUTIONTIME] =
7 % MULTISCALENH( INIMG,J,ALPHA,TMIN,TMAX,SIGMA,T1,T2,
8 %
9 % Returns:
10 % outImg      Processed Image.
11 % execTime    Execution time for image enhancement
12 %
13 % Parameters:
14 % INIMG      - Input image.
15 % J          - Number of channels to be computed by
16 % ALPHA      - Decreasing factor for soft threshold
17 %             channels in DWT.
18 % TMIN       - Minimum scaling factor for soft thre
19 % TMAX       - Maximum scaling factor for soft thre
20 % SIGMA      - Estimate of the standard deviation c
21 % T1         - Hard threshold.
22 % T2         - Lower bound for edge enhancement thr
23 %             Adaptive Gain of DWT coefficients.
24 % T3         - Upper bound for edge enhancement thr
25 %             Adaptive Gain of DWT coefficients.
26 % C          - Gain in GAG-function.
27 % B          - Controls shape and sign of GAG-funct

```

```

28 %   GRAPHS    -   Controls whether graphs of thresholdi
29 %               functions should be plotted or not.
30 %
31 %   Licensed under BSD as a part of EDGY project sour
32 %   see readme.txt file. For more information see prc
33 %
34 %   Revision: 1.0
35 %   Date: 2007/05/08
36 %   Author: Alexander Tuttle, Erik Ringaby
37
0.05 1 38 addpath common\;
0.05 1 39 addpath common\mxSimpleDiff;
0.05 1 40 addpath artikell\;
41
1 42 if ~(IsScalar(j) && IsScalar(alpha) && IsScalar(tMax)
43     && IsScalar(t1) && IsScalar(t2) && IsScalar(t
44     && IsScalar(b) && IsScalar(sigma) && IsScalar
45     && IsScalar(sampl))
46     error('Parameters must be scalar.');
```

```

47 end
1 48 if ~(numel(j) == 1 && isnumeric(j) &&...
49     (j > 0) && (round(j) == j) && j < 6 );
50     error('The number of levels must be a positive in
51 end
1 52 if (tMin >= tMax)
53     error('t_min must be less than t_max');
54 end
1 55 if ~(t1 >= 0 && t2 >= t1 && t3 > t2 && t3 <= 1)
56     error('t1, t2 and t3 must satisfy 0 <= t1 <= t2 <
57 end
58
59
60 % Convert the image to gray scale.
0.02 1 61 inImg = im2single(mear(inImg,3));
62
1 63 if sampl && min(size(inImg)) <= 2^j
64     error('The input image is too small to be downsar
65 end
66
67 % If sigma is specified as 0 the user is prompted to
68 % image that is adequate for estimating the standard
69 % noise.
0.01 1 70 logImg = log(inImg+eps);
1 71 if sigma == 0
72     [upperLeft lowerRight] = getUserCoordinates(inImg
73     rows = min(upperLeft(1,2),lowerRight(1,2)):max(lc
74     cols = min(upperLeft(1,1),lowerRight(1,1)):max(lc
75     imgSelection = logImg(rows,cols);
76     sigma = std(imgSelection(:));
77 end
78
79 % Compute the function table for the enhancement func
1 80 res = 0.001; %table resolution
< 0.01 1 81 s = -1; %lower bound for table
< 0.01 1 82 e = 1; %upper bound for table
1 83 egagTable = egag(s:res:e, b, c, t1, t2, t3);
84

```

```

85 % Plot graphs of thresholding and enhancement function
1 86 if graphs
87     showGraphs(j,alpha,tMin,tMax,sigma,egagTable,res)
88 end
89
90 % Start timer for measuring execution time of image e
< 0.01 1 91 tic;
92
93 % Compute the natural logarithm of the image to separ
94 % noise.
0.03 1 95 inImg = log(inImg+eps);
96
97 % ----- DWT -----
0.14 1 98 [hpChannels lpChannels] = dwt(inImg,j,sampl);
99 % -----
100
101 % ----- Soft thresholding -----
102 % The coefficients below describe the relations betwe
103 % deviation of the noise in the image and the respect
< 0.01 1 104 sigmaxCoeff = [0.7122 0.2828 0.1907 0.1472 0.1217];
< 0.01 1 105 sigmayCoeff = [0.3416 0.1946 0.1468 0.1194 0.1012];
106
107 for level = 1:floor(j/2)
0.03 2 108     hpChannels{level}(:, :, 1) = softThresh(hpChannels{1
109         tMax, tMin, sigmaxCoeff(level)*sigma, alpha,
0.03 2 110     hpChannels{level}(:, :, 2) = softThresh(hpChannels{1
111         tMax, tMin, sigmayCoeff(level)*sigma, alpha,
2 112 end
113 % -----
114
115 % ----- Enhancement through Generalized Adaptiv
1 116 for iChannel = ceil(j/2):j
3 117     M1 = max(abs(reshape(hpChannels{iChannel}(:, :, 1),
3 118         M2 = max(abs(reshape(hpChannels{iChannel}(:, :, 2),
0.05 3 119     hpChannels{iChannel}(:, :, 1) = M1*lookUp(egagTable,
120         hpChannels{iChannel}(:, :, 1)/M1);
0.05 3 121     hpChannels{iChannel}(:, :, 2) = M2*lookUp(egagTable,
122         hpChannels{iChannel}(:, :, 2)/M2);
3 123 end
124 % -----
125
126 % ----- IDWT -----
0.26 1 127 outImg = exp(idwt(hpChannels, lpChannels, sampl));
128 % -----
< 0.01 1 129 execTime = toc;
130
131 % ##### Help functions #####
132
133 % ----- DWT -----
134 function [hpChannels lpChannel] = dwt(img,numChannels
135 %DWT Frequency decomposition of 2D signal.
136 % [HP LP] = DWT(S,N,D) decomposes the signal into f
137 % according to the follow illustration:
138 %
139 % |--HPx{1}
140 % |
141 % |--HPy{1}          |--HPx{N}

```

```

142 %      |      |
143 %  S---|      |--HPy{N}
144 %      |      |
145 %      |--LP--- ... ---|
146 %      |      |
147 %      |      |--LP
148 %
149 %  S - Signal to be decomposed.
150 %  N - Number of channels to compute.
151 %  D - Downsample LP-component between channels.
152
153 % Filter coefficients for analyzing filters.
154 hx = [0.0625 0.25 0.375 0.25 0.0625];
155 gx = [0 1 -1];
156
157 lpChannel = img;
158
159 if(sampl)
160     for iChannel = 1:numChannels
161 %         hpChannels{iChannel}(:, :, 1) = mxSimpleDiff(
162 %         hpChannels{iChannel}(:, :, 2) = mxSimpleDiff(
163         hpChannels{iChannel}(:, :, 1) = imfilter(lpChan
164         hpChannels{iChannel}(:, :, 2) = imfilter(lpChan
165
166         lpChannel = imfilter(imfilter(lpChannel, hx, 'r
167         hx', 'replicate', 'conv'));
168         lpChannel = lpChannel(1:2:end, 1:2:end);
169     end
170 else
171     for iChannel = 1:numChannels
172 %         hpChannels{iChannel}(:, :, 1) = mxSimpleDiff(
173 %         2^(iChannel-1)-1);
174 %         hpChannels{iChannel}(:, :, 2) = mxSimpleDiff(
175 %         2^(iChannel-1)-1)';
176         hpChannels{iChannel}(:, :, 1) = imfilter(lpChan
177         hpChannels{iChannel}(:, :, 2) = imfilter(lpChan
178
179         lpChannel = imfilter(imfilter(lpChannel, zeroF
180         'replicate', 'conv'), zeroPad(hx, iChannel)
181     end
182 end
183 % -----
184
185 % ----- IDWT -----
186 function img = idwt(hpChannels, lpChannel, sampl)
187 %IDWT Reconstructs a signal that has been decomposed
188 %  S = IDWT(HPCHANNELS, LPCHANNEL, SAMPL)
189 %  HPCHANNELS, LPCHANNEL - Output from DWT, write 'h
190 %  SAMPL - Boolean that indicates we
191 %          downsampld during decomp
192
193 % Filter coefficients for reconstructing filters.
194 hx = [0.0625 0.25 0.375 0.25 0.0625];
195 kx = [-0.00390625 -0.03515625 -0.14453125 -0.36328125
196       0.14453125 0.03515625 0.00390625 0];
197 lx = [0.001953125 0.015625 0.0546875 0.109375 0.63671
198       0.0546875 0.015625 0.001953125];

```

```

199
200 % Split signal into channels with or without downsamp
201 % channels.
202 iChannel = length(hpChannels);
203 if(sampl)
204     while iChannel > 0
205         hpChannelX = imfilter(imfilter(hpChannels{iCh
206             lx', 'replicate', 'conv'), kx, 'replicate
207
208         hpChannelY = imfilter(imfilter(hpChannels{iCh
209             kx', 'replicate', 'conv'),lx, 'replicate'
210
211         lpChannel = interp2(lpChannel,'spline');
212         if mod(size(hpChannelX,1),2) == 0
213             lpChannel = [lpChannel;lpChannel(end,:)];
214         end
215         if mod(size(hpChannelX,2),2) == 0
216             lpChannel = [lpChannel lpChannel(:,end)];
217         end
218
219         lpChannel = imfilter(imfilter(lpChannel,hx','
220             'replicate');
221         lpChannel = hpChannelX + hpChannelY +lpChanne
222         iChannel = iChannel-1;
223     end
224 else
225     while iChannel > 0
226         hpChannelX = imfilter(imfilter(hpChannels{iCh
227             zeroPad(lx,iChannel)', 'replicate', 'conv
228             zeroPad(kx,iChannel), 'replicate', 'conv'
229
230         hpChannelY = imfilter(imfilter(hpChannels{iCh
231             zeroPad(kx,iChannel)', 'replicate', 'conv
232             zeroPad(lx,iChannel), 'replicate', 'conv'
233
234         lpChannel = imfilter(imfilter(lpChannel,zeroF
235             'replicate'),zeroPad(hx,iChannel),'repl
236
237         lpChannel = hpChannelX + hpChannelY +lpChanne
238         iChannel = iChannel-1;
239     end
240 end
241 img = lpChannel;
242 % -----
243
244 % ----- SOFTTHRESH -----
245 function U = softThresh( V, tMax, tMin, sigma, alpha,
246 %SOFTTHRESH performs soft thresholding of the values
247 % Y = softThresh( X, TMAX, TMIN, S, A, L ) performs
248 % thresholding of X with the threshold t computed a
249 %
250 %         ( (TMAX-A*(L-1))*S if TMAX-A*(L-1)>TMIN
251 % t = <
252 %         ( TMIN*S           otherwise
253
254 % Calculate a threshold
255 c = tMax - alpha*(lev-1);

```

```

256
257 if c > tMin
258     t = c*sigma;
259 else
260     t = tMin*sigma;
261 end
262 % Apply soft thresholding
263 U = sign(V).*(abs(V)-t).*(abs(V) > t);
264 % -----
265
266 % ----- EGAG -----
267 function Y = egag( X, b, c, t1, t2, t3 )
268 %EGAG Signal enhancement of X through Generalized Ada
269 % Y = EGAG( X, B, C, T1, T2, T3 ).
270 % X is the signal that you want to enhance, it can
271 % a matrix. Parameters B and C control the amount c
272 % T2 and T3 are thresholds. Values in [0 T1] are ma
273 % [T2 T3] are amplified and values in ]T1 T2[ and a
274 % altered.
275
276 a = 1./(sigm(c*(1-b))-sigm(-c*(1+b)));
277 signx = sign(X);
278 U = signx.*(abs(X)-t2)./(t3-t2);
279 U_bar = a*(t3-t2)*(sigm(c*(U-b))-sigm(-c*(U+b)));
280
281 Y = (signx.*t2+U_bar).*((abs(X) <= t3).*((abs(X) >= t
282     X.*((abs(X) > t3) + (abs(X) < t2)).*(abs(X)>t1);
283
284 function Y = sigm( X )
285 %SIGM A sigmoid function
286 % Y = SIGM(X) = 1./(1+exp(-X));
287
288 Y = 1./(1+exp(-X));
289 % -----
290
291 % ----- ZEROPAD -----
292 function out = zeroPad(in, n)
293 %ZEROPAD Insert zeros between elements of a vector or
294 % Y = ZEROPAD(X,N) inserts 2^(N-1)-1 zeros between
295 % vector or matrix X. If X is a matrix X will be ze
296 % dimensions.
297
298 if (n <= 0)
299     error('n must be equal to or larger than one');
300 end
301 if (n == 1)
302     out = in;
303 else
304     n = 2^(n-1);
305     out = zeros(size(in,1),n*size(in,2)-n+1);
306     out(:,1:n:end)=in;
307 end
308 % -----
309
310 % ----- LOOKUP -----
311 function Y = lookUp(funTable,res,X)
312 %LOOKUP Look up values in a function table with NN-in

```



```

313 %   Y = LOOKUP(FUNTABLE, R, X)
314 %   FUNTABLE - table of function values.
315 %   R         - table resolution.
316 %   X         - values whos correspodng function valu
317 %               are to be looked up.
318
319 % Dummy variables intended to optimize speed
320 tVar1 = (1/res); % Multiplication is faster than divi
321 tVar2 = length(funTable)/2;
322
323 Y = funTable(ceil(X*tVar1+tVar2));
324 % -----
325
326 % ----- SHOWGRAPHS -----
327 function showGraphs(j,alpha,tMin,tMax,sigma,egagTable
328 %SHOWGRAPHS Displays graphs of thresholding and enhan
329 x = linspace(-1,1,1/res);
330 numC = floor(j/2);
331 % Plot softThresh for fine scale levels of DWT
332 fig1 = figure;
333 set(fig1,'Name','Thresholding and enhancement functio
334 for channel = 1:numC
335     subplot(1,numC+1,channel);
336     plot(x,softThresh(x, tMax, tMin, sigma, alpha, ch
337     axis square;title(sprintf('softThreshold in chann
338     xlabel x;ylabel('softThresh(x)');
339 end
340 %Plot egag function
341 subplot(1,numC+1,numC+1);plot(x,lookUp(egagTable,res,
342 title('Generalized Adaptive Gain function');xlabel x;
343 % -----
344
345 function [ x1 x2 ] = getUserCoordinates(img)
346 %GETUSERCOORDINATES
347 %   [ X1 X2 ] = GETUSERCOORDINATES(IMG)
348 % Lets the user choose two points in an image and ret
349 % the coordinates of each point in the row vectors X1
350
351 x1 = zeros(1,2);
352 x2 = zeros(1,2);
353
354 fig = figure;
355 imshow(img,[min(img(:)),max(img(:))]);axis image;colo
356 set(fig,'Name','Select area for estimation of standar
357
358 disp('Select upper left corner of preferred area with
359 t = waitforbuttonpress;
360 [x1(1,1) x1(1,2)] = ginput(1);
361 disp('Select lower right corner of preferred area wit
362 t = waitforbuttonpress;
363 [x2(1,1) x2(1,2)] = ginput(1);
364
365 x1 = ceil(x1);
366 x2 = floor(x2);
367 close(fig);
368 % -----

```