

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	[w(y,x, :, :), mu(y,x, :, :)] = ei...	252048	6.732 s	89.4%
99	J = imfilter(imfilter(nablaI,K...	2	0.156 s	2.1%
202	I_t = divergence(a.*I_x + b.*I...	2	0.156 s	2.1%
125	D(:, :, 1, 2) = lambda(:, :, 1).*w(...	2	0.078 s	1.0%
111	end	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%	

Time Plot



t

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></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

time calls line

```
1 function I = AnisotropicDiffusion( inImage, signal, d
2     delta2, s2, alpha, beta, tau, numIter, graphs )
3 %AnisotropicDiffsion Implements Nonlinear Anisotropic
4 %   I_t = AnisotropicDiffusion( path, signal, 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 %   signal        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 pro
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(sigma1) || sigma1 < 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(sigma1, 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, mu1] = 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)

```

```

144 %         if j==i-1
145 %             A1(i,j) = (a(i,j)+a(i,j-1))/2;
146 %         end
147 %         if j==i+1
148 %             A1(i,j) = (a(i,j)+a(i,j+1))/2;
149 %         end
150 %         if j==i
151 %             A1(i,j) = A1(i,i-1)+A1(i,i+1);
152 %         end
153 %     end
154 % end
155 % A2 = zeros(size(I));
156 % A2(1,2) = (d(1,1)+d(1,2)) / 2;
157 % A2(1,1) = d(1,2)*2; % ???
158 % for i = 2:(size(A2, 1)-1)
159 %     for j = 2:(size(A2,2)-1)
160 %         if j==i-1
161 %             A2(i,j) = (d(i,j)+d(i,j-1))/2;
162 %         end
163 %         if j==i+1
164 %             A2(i,j) = (d(i,j)+d(i,j+1))/2;
165 %         end
166 %         if j==i
167 %             A2(i,j) = A2(i,i-1)+A2(i,i+1);
168 %         end
169 %     end
170 % end
171 %
172 % % A12 = zeros(size(I));
173 % % A12(1,2) = (a(1,1)+a(1,2)) / 2;
174 % % A12(1,1) = a(1,2)*2; % ???
175 % % for i = 2:(size(A12,1)-1)
176 % %     A12(i,i+1) = (a(i,i)+a(i,i+1)) / 2;
177 % %     A12(i,i-1) = (a(i,i)+a(i,i-1)) / 2;
178 % %     A12(i,i) = A12(i,i-1) + A12(i,i+1);
179 % % end
180 %
181 % % A2 = zeros(size(I));
182 % % A2(1,2) = d(1,1)+a(1,2) / 2;
183 % % A2(1,1) = d(1,2)*2; % ???
184 % % for i = 2:(size(A2, 1)-1)
185 % %     A2(i,i+1) = d(i,i)+d(i,i+1) / 2;
186 % %     A2(i,i-1) = d(i,i)+d(i,i-1) / 2;
187 % %     A2(i,i) = A2(i,i-1) + A2(i,i+1);
188 % % end
189 %
190 % % Edges!!!
191 % J = zeros(size(I));
192 % for j = 2:(size(J,1)-1)
193 %     for l = 2:(size(J,2)-1)
194 %         J(j,l) = I_t(j,l) + ...
195 %             tau/4.*(c(j+1,l).*(I_t(j+1,l+1)-I_t(j+1,l))
196 %             + tau/4.*(b(j,l+1).*(I_t(j+1,l+1)-I_t(j,l+1))
197 %     end
198 % end
199 %
200 % I_t = 1/2 * ((1-2*tau*A1).^(-1)).*J + (1-2*tau*A2).^(-1)).*J

```

```
0.16      201
2 202 I_t = divergence (a.*I_x + b.*I_y, c.*I_x + d.*I_y);
2 203 I = I + I_t;
2 204 if (graphs == 1)
205     figure(10);imshow(I,[]);title(['Iteration ' num2s
206 end
2 207 end
1 208 if (graphs == 1)
209     figure;plot(inImage(145,:), 'r');hold on; plot(I(1
210     legend('Original+Speckle', 'Processed');
211 end
1 212 I = uint8(I);
```