clc

clear all

close all

%Image Acquisition [aa bb]=uigetfile('.jpg');

I=imread([bb aa]);

figure,imshow(I);

title('Input Image');

I1=imresize(I,[256,256]);

figure,imshow(I1);

title('Resized Image');

%face image

face\_de=vision.CascadeObjectDetector();

bb=step(face\_de,I1);

a1=insertObjectAnnotation(I1,'rectangle',bb,'face');

figure,imshow(a1);

title('Face Region');

%face normalisation

I1=imcrop(a1,bb);

I1=imresize(I1,[256,256]);

figure,imshow(I1);

title('Normalised Image');

%SPECULAR REFLECTANCE

bila=0.27;

w=32.9 sigma=35.9 I=double(a1)/255;

B = bfilter2(I,w,sigma);

figure,imshow(B);

title('Filtered Image');

BH=imhist(rgb2gray(B));

BHF=mean(mean(mean(BH)));

BM=mean(mean(mean(B)))/1000;

BS=sum(sum(sum(B)))/10;

BV=std(std(std(B)))/10;

SPF=[BHF BM BS BV];

% blurness detection

% bilateral filter

H=fspecial('motion',20,45);

BI=imfilter(I1,H,'replicate');

BIL=I1-BI; figure,imshow(BIL);

title('Bilateral Image');

BLF=sum(sum(sum(BIL)))/10000;

% Choromatic features

IH=rgb2hsv(I1);

figure,imshow(IH);

title('HSV Image');

H=IH(:,:,1);

HH=imhist(H);

HMA=max(max(HH));

HMI=min(min(HH));

S=IH(:,:,2);

SH=imhist(S);

SMA=max(max(SH));

SMI=min(min(SH));

V=IH(:,:,3);

VH=imhist(V);

VMA=max(max(VH));

VMI=min(min(VH));

HM=mean(mean(mean(IH)));

SM=mean(mean(mean(SH)));

VM=mean(mean(mean(VH)));

HV=std(std(std(HM)));

SV=std(std(std(SM)));

VV=std(std(std(VM)));

HS=skewness(skewness(H));

SS=skewness(skewness(S));

VS=skewness(skewness(V));

CF=[HMA SMA VMA HMI SMI VMI HM SM VM HV SV VV HS SS VS];

% color quantisation

[xx yy]=rgb2ind(I1,32);

figure,imshow(xx,yy);

CQ=sum(sum(xx))

% FEATURE CONCATENATION

FEATURE\_FACE=mean([SPF BLF CF CQ]);

% save F5 FEATURE5

FINGERPRINT DETECTION

%Read Input Image [aa,bb]=uigetfile('.tif');

I1=imread([bb aa]);

I2=imresize(I1,[256,256]);

I3=I2(:,:,1);

I3=I2(:,:,2);

I3=I2(:,:,3);

figure,imshow(I3);

title('Input Image');

binary\_image=im2bw(I3);

% binary\_image = binary\_image(120:400,20:250);

figure,imshow(binary\_image) title('Input image');

%Thinning thin\_image=~bwmorph(binary\_image,'thin',Inf); figure;

imshow(thin\_image);

title('Thinned Image');

%Minutiae extraction s=size(thin\_image);

N=3;

%window size n=(N-1)/2;

r=s(1)+2\*n;

c=s(2)+2\*n;

double temp(r,c);

temp=zeros(r,c);

bifurcation=zeros(r,c);

ridge=zeros(r,c);

temp((n+1):(end-n),(n+1):(end-n))=thin\_image(:,:);

outImg=zeros(r,c,3);%For Display outImg(:,:,1) = temp .\* 255; outImg(:,:,2) = temp .\* 255; outImg(:,:,3) = temp .\* 255; for x=(n+1+10):(s(1)+n-10) for y=(n+1+10):(s(2)+n-10) e=1; for k=x-n:x+n f=1; for l=y-n:y+n mat(e,f)=temp(k,l); f=f+1; end e=e+1; end; if(mat(2,2)==0) ridge(x,y)=sum(sum(~mat)); bifurcation(x,y)=sum(sum(~mat));

end

end;

end;

% RIDGE END FINDING [ridge\_x ridge\_y]=find(ridge==2); len=length(ridge\_x);

%For Display for i=1:len outImg((ridge\_x(i)-3):(ridge\_x(i)+3),(ridge\_y(i)-3),2:3)=0; outImg((ridge\_x(i)-3):(ridge\_x(i)+3),(ridge\_y(i)+3),2:3)=0; outImg((ridge\_x(i)-3),(ridge\_y(i)-3):(ridge\_y(i)+3),2:3)=0; outImg((ridge\_x(i)+3),(ridge\_y(i)-3):(ridge\_y(i)+3),2:3)=0; outImg((ridge\_x(i)-3):(ridge\_x(i)+3),(ridge\_y(i)-3),1)=255; outImg((ridge\_x(i)-3):(ridge\_x(i)+3),(ridge\_y(i)+3),1)=255; outImg((ridge\_x(i)-3),(ridge\_y(i)-3):(ridge\_y(i)+3),1)=255;

outImg((ridge\_x(i)+3),(ridge\_y(i)-3):(ridge\_y(i)+3),1)=255;

End

%BIFURCATION FINDING [bifurcation\_x bifurcation\_y]=find(bifurcation==4); len=length(bifurcation\_x); %For Display for i=1:len outImg((bifurcation\_x(i)-3):(bifurcation\_x(i)+3),(bifurcation\_y(i)-3),1:2)=0; outImg((bifurcation\_x(i)-3):(bifurcation\_x(i)+3),(bifurcation\_y(i)+3),1:2)=0; outImg((bifurcation\_x(i)-3),(bifurcation\_y(i)-3):(bifurcation\_y(i)+3),1:2)=0; outImg((bifurcation\_x(i)+3),(bifurcation\_y(i)-3):(bifurcation\_y(i)+3),1:2)=0; outImg((bifurcation\_x(i)-3):(bifurcation\_x(i)+3),(bifurcation\_y(i)-3),3)=255; outImg((bifurcation\_x(i)-3):(bifurcation\_x(i)+3),(bifurcation\_y(i)+3),3)=255; outImg((bifurcation\_x(i)-3),(bifurcation\_y(i)-3):(bifurcation\_y(i)+3),3)=255; outImg((bifurcation\_x(i)+3),(bifurcation\_y(i)-3):(bifurcation\_y(i)+3),3)=255; end figure;imshow(outImg);title('Minutiae'); fingerprint\_out=outImg; Feat\_Finger=mean((mean(mean(outImg)))); Feat=mean([Feat\_Finger,FEATURE\_FACE])

%save F3 Feat3 % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%TESTING Y=Feat; [out ind]=matching1(Y); if(out~=0) msgbox('Authenticated'); else msgbox('Unauthenticated'); [y,Fs] = audioread('Wecker-sound.mp3'); sound(y(1:100000),Fs);

COLORSPACE function varargout = colorspace(Conversion,varargin) %%% Input parsing %%% if nargin < 2, error('Not enough input arguments.'); end [SrcSpace,DestSpace] = parse(Conversion); if nargin == 2 Image = varargin{1}; elseif nargin >= 3 Image = cat(3,varargin{:}); else error('Invalid number of input arguments.'); end FlipDims = (size(Image,3) == 1); if FlipDims, Image = permute(Image,[1,3,2]); end if ~isa(Image,'double'), Image = double(Image)/255; end if size(Image,3) ~= 3, error('Invalid input size.'); end SrcT = gettransform(SrcSpace); DestT = gettransform(DestSpace); if ~ischar(SrcT) & ~ischar(DestT) % Both source and destination transforms are affine, so they % can be composed into one affine operation T = [DestT(:,1:3)\*SrcT(:,1:3),DestT(:,1:3)\*SrcT(:,4)+DestT(:,4)]; Temp = zeros(size(Image)); Temp(:,:,1) = T(1)\*Image(:,:,1) + T(4)\*Image(:,:,2) + T(7)\*Image(:,:,3) + T(10); Temp(:,:,2) = T(2)\*Image(:,:,1) + T(5)\*Image(:,:,2) + T(8)\*Image(:,:,3) + T(11); Temp(:,:,3) = T(3)\*Image(:,:,1) + T(6)\*Image(:,:,2) + T(9)\*Image(:,:,3) + T(12); 54 Image = Temp; elseif ~ischar(DestT) Image = rgb(Image,SrcSpace); Temp = zeros(size(Image)); Temp(:,:,1) = DestT(1)\*Image(:,:,1) + DestT(4)\*Image(:,:,2) + DestT(7)\*Image(:,:,3) + DestT(10); Temp(:,:,2) = DestT(2)\*Image(:,:,1) + DestT(5)\*Image(:,:,2) + DestT(8)\*Image(:,:,3) + DestT(11); Temp(:,:,3) = DestT(3)\*Image(:,:,1) + DestT(6)\*Image(:,:,2) + DestT(9)\*Image(:,:,3) + DestT(12); Image = Temp; else Image = feval(DestT,Image,SrcSpace); end %%% Output format %%% if nargout > 1 varargout = {Image(:,:,1),Image(:,:,2),Image(:,:,3)}; else if FlipDims, Image = permute(Image,[1,3,2]); end varargout = {Image}; end return; function [SrcSpace,DestSpace] = parse(Str) % Parse conversion argument if isstr(Str) Str = lower(strrep(strrep(Str,'-',''),' ','')); k = find(Str == '>'); if length(k) == 1 % Interpret the form 'src->dest' SrcSpace = Str(1:k-1); DestSpace = Str(k+1:end); else 55 k = find(Str == '<'); if length(k) == 1 % Interpret the form 'dest<-src' DestSpace = Str(1:k-1); SrcSpace = Str(k+1:end); else error(['Invalid conversion, ''',Str,'''.']); end end SrcSpace = alias(SrcSpace); DestSpace = alias(DestSpace); else SrcSpace = 1; % No source pre-transform DestSpace = Conversion; if any(size(Conversion) ~= 3), error('Transformation matrix must be 3x3.'); end end return; function Space = alias(Space) Space = strrep(Space,'cie',''); if isempty(Space) Space = 'rgb'; end switch Space case {'ycbcr','ycc'} Space = 'ycbcr'; case {'hsv','hsb'} Space = 'hsv'; case {'hsl','hsi','hls'} Space = 'hsl'; case {'rgb','yuv','yiq','ydbdr','ycbcr','jpegycbcr','xyz','lab','luv','lch'} return; 56 end return; function T = gettransform(Space) % Get a colorspace transform: either a matrix describing an affine transform, % or a string referring to a conversion subroutine switch Space case 'ypbpr' T = [0.299,0.587,0.114,0;-0.1687367,-0.331264,0.5,0;0.5,-0.418688,- 0.081312,0]; case 'yuv' % R'G'B' to NTSC/PAL YUV % Wikipedia: http://en.wikipedia.org/wiki/YUV T = [0.299,0.587,0.114,0;-0.147,-0.289,0.436,0;0.615,-0.515,-0.100,0]; case 'ydbdr' % R'G'B' to SECAM YDbDr % Wikipedia: http://en.wikipedia.org/wiki/YDbDr T = [0.299,0.587,0.114,0;-0.450,-0.883,1.333,0;-1.333,1.116,0.217,0]; case 'yiq' % R'G'B' in [0,1] to NTSC YIQ in [0,1];[-0.595716,0.595716];[- 0.522591,0.522591]; % Wikipedia: http://en.wikipedia.org/wiki/YIQ T = [0.299,0.587,0.114,0;0.595716,-0.274453,-0.321263,0;0.211456,- 0.522591,0.311135,0]; case 'ycbcr' % R'G'B' (range [0,1]) to ITU-R BRT.601 (CCIR 601) Y'CbCr % Wikipedia: http://en.wikipedia.org/wiki/YCbCr % Poynton, Equation 3, scaling of R'G'B to Y'PbPr conversion T = [65.481,128.553,24.966,16;-37.797,-74.203,112.0,128;112.0,-93.786,- 18.214,128]; case 'jpegycbcr' % Wikipedia: http://en.wikipedia.org/wiki/YCbCr T = [0.299,0.587,0.114,0;-0.168736,-0.331264,0.5,0.5;0.5,-0.418688,- 0.081312,0.5]\*255; case {'rgb','xyz','hsv','hsl','lab','luv','lch'} T = Space; 57 otherwise error(['Unknown color space, ''',Space,'''.']); end return; function Image = rgb(Image,SrcSpace) % Convert to Rec. 709 R'G'B' from 'SrcSpace' switch SrcSpace case 'rgb' return; case 'hsv' % Convert HSV to R'G'B' Image = huetorgb((1 - Image(:,:,2)).\*Image(:,:,3),Image(:,:,3),Image(:,:,1)); case 'hsl' % Convert HSL to R'G'B' L = Image(:,:,3); Delta = Image(:,:,2).\*min(L,1-L); Image = huetorgb(L-Delta,L+Delta,Image(:,:,1)); case {'xyz','lab','luv','lch'} % Convert to CIE XYZ Image = xyz(Image,SrcSpace); % Convert XYZ to RGB T = [3.240479,-1.53715,-0.498535;-0.969256,1.875992,0.041556;0.055648,- 0.204043,1.057311]; R = T(1)\*Image(:,:,1) + T(4)\*Image(:,:,2) + T(7)\*Image(:,:,3); % R G = T(2)\*Image(:,:,1) + T(5)\*Image(:,:,2) + T(8)\*Image(:,:,3); % G B = T(3)\*Image(:,:,1) + T(6)\*Image(:,:,2) + T(9)\*Image(:,:,3); % B % Desaturate and rescale to constrain resulting RGB values to [0,1] AddWhite = -min(min(min(R,G),B),0); Scale = max(max(max(R,G),B)+AddWhite,1); R = (R + AddWhite)./Scale; G = (G + AddWhite)./Scale; B = (B + AddWhite)./Scale; % Apply gamma correction to convert RGB to Rec. 709 R'G'B' Image(:,:,1) = gammacorrection(R); % R' Image(:,:,2) = gammacorrection(G); % G' 58 Image(:,:,3) = gammacorrection(B); % B' otherwise % Conversion is through an affine transform T = gettransform(SrcSpace); temp = inv(T(:,1:3)); T = [temp,-temp\*T(:,4)]; R = T(1)\*Image(:,:,1) + T(4)\*Image(:,:,2) + T(7)\*Image(:,:,3) + T(10); G = T(2)\*Image(:,:,1) + T(5)\*Image(:,:,2) + T(8)\*Image(:,:,3) + T(11); B = T(3)\*Image(:,:,1) + T(6)\*Image(:,:,2) + T(9)\*Image(:,:,3) + T(12); AddWhite = -min(min(min(R,G),B),0); Scale = max(max(max(R,G),B)+AddWhite,1); R = (R + AddWhite)./Scale; G = (G + AddWhite)./Scale; B = (B + AddWhite)./Scale; Image(:,:,1) = R; Image(:,:,2) = G; Image(:,:,3) = B; end % Clip to [0,1] Image = min(max(Image,0),1); return; function Image = xyz(Image,SrcSpace) % Convert to CIE XYZ from 'SrcSpace' WhitePoint = [0.950456,1,1.088754]; switch SrcSpace case 'xyz' return; case 'luv' % Convert CIE L\*uv to XYZ WhitePointU = (4\*WhitePoint(1))./(WhitePoint(1) + 15\*WhitePoint(2) + 3\*WhitePoint(3)); WhitePointV = (9\*WhitePoint(2))./(WhitePoint(1) + 15\*WhitePoint(2) + 3\*WhitePoint(3)); L = Image(:,:,1); 59 Y = (L + 16)/116; Y = invf(Y)\*WhitePoint(2); U = Image(:,:,2)./(13\*L + 1e-6\*(L==0)) + WhitePointU; V = Image(:,:,3)./(13\*L + 1e-6\*(L==0)) + WhitePointV; Image(:,:,1) = -(9\*Y.\*U)./((U-4).\*V - U.\*V); % X Image(:,:,2) = Y; % Y Image(:,:,3) = (9\*Y - (15\*V.\*Y) - (V.\*Image(:,:,1)))./(3\*V); % Z case {'lab','lch'} Image = lab(Image,SrcSpace); % Convert CIE L\*ab to XYZ fY = (Image(:,:,1) + 16)/116; fX = fY + Image(:,:,2)/500; fZ = fY - Image(:,:,3)/200; Image(:,:,1) = WhitePoint(1)\*invf(fX); % X Image(:,:,2) = WhitePoint(2)\*invf(fY); % Y Image(:,:,3) = WhitePoint(3)\*invf(fZ); % Z otherwise % Convert from some gamma-corrected space % Convert to Rec. 701 R'G'B' Image = rgb(Image,SrcSpace); % Undo gamma correction R = invgammacorrection(Image(:,:,1)); G = invgammacorrection(Image(:,:,2)); B = invgammacorrection(Image(:,:,3)); % Convert RGB to XYZ T = inv([3.240479,-1.53715,-0.498535;- 0.969256,1.875992,0.041556;0.055648,-0.204043,1.057311]); Image(:,:,1) = T(1)\*R + T(4)\*G + T(7)\*B; % X Image(:,:,2) = T(2)\*R + T(5)\*G + T(8)\*B; % Y Image(:,:,3) = T(3)\*R + T(6)\*G + T(9)\*B; % Z end return; function Image = hsv(Image,SrcSpace) % Convert to HSV Image = rgb(Image,SrcSpace); V = max(Image,[],3); 60 S = (V - min(Image,[],3))./(V + (V == 0)); Image(:,:,1) = rgbtohue(Image); Image(:,:,2) = S; Image(:,:,3) = V; return; function Image = hsl(Image,SrcSpace) % Convert to HSL switch SrcSpace case 'hsv' % Convert HSV to HSL MaxVal = Image(:,:,3); MinVal = (1 - Image(:,:,2)).\*MaxVal; L = 0.5\*(MaxVal + MinVal); temp = min(L,1-L); Image(:,:,2) = 0.5\*(MaxVal - MinVal)./(temp + (temp == 0)); Image(:,:,3) = L; otherwise Image = rgb(Image,SrcSpace); % Convert to Rec. 701 R'G'B' % Convert R'G'B' to HSL MinVal = min(Image,[],3); MaxVal = max(Image,[],3); L = 0.5\*(MaxVal + MinVal); temp = min(L,1-L); S = 0.5\*(MaxVal - MinVal)./(temp + (temp == 0)); Image(:,:,1) = rgbtohue(Image); Image(:,:,2) = S; Image(:,:,3) = L; end return; function Image = lab(Image,SrcSpace) % Convert to CIE L\*a\*b\* (CIELAB) WhitePoint = [0.950456,1,1.088754]; 61 switch SrcSpace case 'lab' return; case 'lch' % Convert CIE L\*CH to CIE L\*ab C = Image(:,:,2); Image(:,:,2) = cos(Image(:,:,3)\*pi/180).\*C; % a\* Image(:,:,3) = sin(Image(:,:,3)\*pi/180).\*C; % b\* otherwise Image = xyz(Image,SrcSpace); % Convert to XYZ % Convert XYZ to CIE L\*a\*b\* X = Image(:,:,1)/WhitePoint(1); Y = Image(:,:,2)/WhitePoint(2); Z = Image(:,:,3)/WhitePoint(3); fX = f(X); fY = f(Y); fZ = f(Z); Image(:,:,1) = 116\*fY - 16; % L\* Image(:,:,2) = 500\*(fX - fY); % a\* Image(:,:,3) = 200\*(fY - fZ); % b\* end return; function Image = luv(Image,SrcSpace) % Convert to CIE L\*u\*v\* (CIELUV) WhitePoint = [0.950456,1,1.088754]; WhitePointU = (4\*WhitePoint(1))./(WhitePoint(1) + 15\*WhitePoint(2) + 3\*WhitePoint(3)); WhitePointV = (9\*WhitePoint(2))./(WhitePoint(1) + 15\*WhitePoint(2) + 3\*WhitePoint(3)); Image = xyz(Image,SrcSpace); % Convert to XYZ U = (4\*Image(:,:,1))./(Image(:,:,1) + 15\*Image(:,:,2) + 3\*Image(:,:,3)); V = (9\*Image(:,:,2))./(Image(:,:,1) + 15\*Image(:,:,2) + 3\*Image(:,:,3)); Y = Image(:,:,2)/WhitePoint(2); L = 116\*f(Y) - 16; 62 Image(:,:,1) = L; % L\* Image(:,:,2) = 13\*L.\*(U - WhitePointU); % u\* Image(:,:,3) = 13\*L.\*(V - WhitePointV); % v\* return; function Image = lch(Image,SrcSpace) % Convert to CIE L\*ch Image = lab(Image,SrcSpace); % Convert to CIE L\*ab H = atan2(Image(:,:,3),Image(:,:,2)); H = H\*180/pi + 360\*(H < 0); Image(:,:,2) = sqrt(Image(:,:,2).^2 + Image(:,:,3).^2); % C Image(:,:,3) = H; % H return; function Image = huetorgb(m0,m2,H) % Convert HSV or HSL hue to RGB N = size(H); H = min(max(H(:),0),360)/60; m0 = m0(:); m2 = m2(:); F = H - round(H/2)\*2; M = [m0, m0 + (m2-m0).\*abs(F), m2]; Num = length(m0); j = [2 1 0;1 2 0;0 2 1;0 1 2;1 0 2;2 0 1;2 1 0]\*Num; k = floor(H) + 1; Image = reshape([M(j(k,1)+(1:Num).'),M(j(k,2)+(1:Num).'),M(j(k,3)+(1:Num).')],[N,3]); return; function H = rgbtohue(Image) % Convert RGB to HSV or HSL hue [M,i] = sort(Image,3); i = i(:,:,3); Delta = M(:,:,3) - M(:,:,1); 63 Delta = Delta + (Delta == 0); R = Image(:,:,1); G = Image(:,:,2); B = Image(:,:,3); H = zeros(size(R)); k = (i == 1); H(k) = (G(k) - B(k))./Delta(k); k = (i == 2); H(k) = 2 + (B(k) - R(k))./Delta(k); k = (i == 3); H(k) = 4 + (R(k) - G(k))./Delta(k); H = 60\*H + 360\*(H < 0); H(Delta == 0) = nan; return; function Rp = gammacorrection(R) Rp = real(1.099\*R.^0.45 - 0.099); i = (R < 0.018); Rp(i) = 4.5138\*R(i); return; function R = invgammacorrection(Rp) R = real(((Rp + 0.099)/1.099).^(1/0.45)); i = (R < 0.018); R(i) = Rp(i)/4.5138; return; function fY = f(Y) fY = real(Y.^(1/3)); i = (Y < 0.008856); fY(i) = Y(i)\*(841/108) + (4/29); return; function Y = invf(fY) Y = fY.^3; 64 i = (Y < 0.008856); Y(i) = (fY(i) - 4/29)\*(108/841); return; %%%%%%%% eucdistme function d=eucdistme(X,Y) if~exist('Y','var')||isempty(Y) U=zeros(size(X,1),1); d=abs(X'.^2\*U).';return end V=~isnan(X);X(~V)=0; U=~isnan(Y);Y(~U)=0; d=abs(X'.^2\*U-2\*X'\*Y+V'\*Y.^2); %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%%%%%%%%%% % Pre-process input and select appropriate filter. function B = bfilter2(A,w,sigma) % Verify that the input image exists and is valid. if ~exist('A','var') || isempty(A) error('Input image A is undefined or invalid.'); end if ~isfloat(A) || ~sum([1,3] == size(A,3)) || ... min(A(:)) < 0 || max(A(:)) > 1 error(['Input image A must be a double precision ',... 'matrix of size NxMx1 or NxMx3 on the closed ',... 'interval [0,1].']); end % Verify bilateral filter window size. if ~exist('w','var') || isempty(w) || ... numel(w) ~= 1 || w < 1 w = 5; end w = ceil(w); 65 % Verify bilateral filter standard deviations. if ~exist('sigma','var') || isempty(sigma) || ... numel(sigma) ~= 2 || sigma(1) <= 0 || sigma(2) <= 0 sigma = [3 0.1]; end % Apply either grayscale or color bilateral filtering. if size(A,3) == 1 B = bfltGray(A,w,sigma(1),sigma(2)); else B = bfltColor(A,w,sigma(1),sigma(2)); end %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%%%%%%%%%% % Implements bilateral filtering for grayscale images. function B = bfltGray(A,w,sigma\_d,sigma\_r) % Pre-compute Gaussian distance weights. [X,Y] = meshgrid(-w:w,-w:w); G = exp(-(X.^2+Y.^2)/(2\*sigma\_d^2)); % Create waitbar. h = waitbar(0,'Applying bilateral filter...'); set(h,'Name','Bilateral Filter Progress'); % Apply bilateral filter. dim = size(A); B = zeros(dim); for i = 1:dim(1) for j = 1:dim(2) % Extract local region. iMin = max(i-w,1); iMax = min(i+w,dim(1)); 66 jMin = max(j-w,1); jMax = min(j+w,dim(2)); I = A(iMin:iMax,jMin:jMax); % Compute Gaussian intensity weights. H = exp(-(I-A(i,j)).^2/(2\*sigma\_r^2)); % Calculate bilateral filter response. F = H.\*G((iMin:iMax)-i+w+1,(jMin:jMax)-j+w+1); B(i,j) = sum(F(:).\*I(:))/sum(F(:)); end waitbar(i/dim(1)); end % Close waitbar. close(h); %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %%%%%%%%%%%%%%%%%%%%%%% % Implements bilateral filter for color images. function B = bfltColor(A,w,sigma\_d,sigma\_r) % Convert input sRGB image to CIELab color space. if exist('applycform','file') A = applycform(A,makecform('srgb2lab')); else A = colorspace('Lab<-RGB',A); end % Pre-compute Gaussian domain weights. [X,Y] = meshgrid(-w:w,-w:w); G = exp(-(X.^2+Y.^2)/(2\*sigma\_d^2)); % Rescale range variance (using maximum luminance). sigma\_r = 100\*sigma\_r; 67 % Create waitbar. h = waitbar(0,'Applying bilateral filter...'); set(h,'Name','Bilateral Filter Progress'); % Apply bilateral filter. dim = size(A); B = zeros(dim); for i = 1:dim(1) for j = 1:dim(2) % Extract local region. iMin = max(i-w,1); iMax = min(i+w,dim(1)); jMin = max(j-w,1); jMax = min(j+w,dim(2)); I = A(iMin:iMax,jMin:jMax,:); % Compute Gaussian range weights. dL = I(:,:,1)-A(i,j,1); da = I(:,:,2)-A(i,j,2); db = I(:,:,3)-A(i,j,3); H = exp(-(dL.^2+da.^2+db.^2)/(2\*sigma\_r^2)); % Calculate bilateral filter response. F = H.\*G((iMin:iMax)-i+w+1,(jMin:jMax)-j+w+1); norm\_F = sum(F(:)); B(i,j,1) = sum(sum(F.\*I(:,:,1)))/norm\_F; B(i,j,2) = sum(sum(F.\*I(:,:,2)))/norm\_F; B(i,j,3) = sum(sum(F.\*I(:,:,3)))/norm\_F; end waitbar(i/dim(1)); end % Convert filtered image back to sRGB color space. if exist('applycform','file') 68 B = applycform(B,makecform('lab2srgb')); else B = colorspace('RGB<-Lab',B); end % Close waitbar. close(h); MATCHING function [out ind]=matching1(Y) load F1 load F2 load F3 X=[Feat1 Feat2 Feat3]; d=eucdistme(X,Y); [d1 ind]=find(d==0); out=d1;