|  |
| --- |
| function imgProb = exDetect( rgbImgOrig, removeON, onY, onX ) |
|  | %exDetect: detect exudates |
|  | % V. 0.2 - 2010-02-01 |
|  | % make compatible with Matlab2008 |
|  | % V. 0.1 - 2010-02-01 |
|  | % source: /mnt/data/ornl/lesions/exudatesCpp2/matlab/exudatesCpp3 |
|  |  |
|  | addpath('misc'); |
|  |  |
|  | %-- Parameters |
|  | showRes = 0; % show lesions in image |
|  | %-- |
|  |  |
|  | % if no parameters are given use the test image |
|  | if( nargin == 0 ) |
|  | rgbImgOrig = imread( 'misc/img\_ex\_test.jpg' ); |
|  | removeON = 1; |
|  | onY = 905; |
|  | onX = 290; |
|  | showRes = 1; |
|  | end |
|  | % |
|  |  |
|  | imgProb = getLesions( rgbImgOrig, showRes, removeON, onY, onX ); |
|  | end |
|  |  |
|  | function [lesCandImg] = getLesions( rgbImgOrig, showRes, removeON, onY, onX ) |
|  | % Parameters |
|  | winOnRatio = [1/8,1/8]; |
|  | % |
|  | % resize |
|  | origSize = size( rgbImgOrig ); |
|  | newSize = [750 round( 750\*(origSize(2)/origSize(1)) ) ]; |
|  | %newSize = newSize-mod(newSize,2); % force the size to be even |
|  | newSize = findGoodResolutionForWavelet(newSize); |
|  | imgRGB = imresize(rgbImgOrig, newSize); |
|  | imgG = imgRGB(:,:,2); |
|  | % change colour plane |
|  | imgHSV = rgb2hsv( imgRGB ); |
|  | imgV = imgHSV(:,:,3); |
|  | imgV8 = uint8(imgV.\*255); |
|  |  |
|  | % %--- normalise |
|  | % imgV = []; |
|  | % if( isempty( forBgImg ) ) |
|  | % [imgVfor, imgVnorm, forN, forTrimSize] = getForacchiaBg2( imgV, 10, 1 ); |
|  | % %create an image with the original size |
|  | % imgVforOs = zeros(newSize); |
|  | % imgVforOs(forTrimSize:newSize(1)-forTrimSize,forTrimSize:newSize(2)-forTrimSize) = imgVfor; |
|  | % else |
|  | % imgVforOs = imresize(forBgImg, newSize); |
|  | % end |
|  | % %--- |
|  |  |
|  | %--- Remove OD region |
|  | if( removeON ) |
|  | % get ON window |
|  | onY = onY \* newSize(1)/origSize(1); |
|  | onX = onX \* newSize(2)/origSize(2); |
|  | onX = round(onX); |
|  | onY = round(onY); |
|  | winOnSize = round(winOnRatio .\* newSize); |
|  | % remove ON window from imgTh |
|  | winOnCoordY = [onY-winOnSize(1),onY+winOnSize(1)]; |
|  | winOnCoordX = [onX-winOnSize(2),onX+winOnSize(2)]; |
|  | if(winOnCoordY(1) < 1), winOnCoordY(1) = 1; end |
|  | if(winOnCoordX(1) < 1), winOnCoordX(1) = 1; end |
|  | if(winOnCoordY(2) > newSize(1)), winOnCoordY(2) = newSize(1); end |
|  | if(winOnCoordX(2) > newSize(2)), winOnCoordX(2) = newSize(2); end |
|  | % imgThNoOD = imgTh; |
|  | % imgThNoOD(winOnCoordY(1):winOnCoordY(2), winOnCoordX(1):winOnCoordX(2)) = 0; |
|  | end |
|  | %--- |
|  |  |
|  | % Create FOV mask |
|  | imgFovMask = getFovMask( imgV8, 1, 30 ); |
|  | imgFovMask(winOnCoordY(1):winOnCoordY(2), winOnCoordX(1):winOnCoordX(2)) = 0; |
|  |  |
|  | % %--- Calculate threshold using median Background |
|  | % x=0:255; |
|  | % offset=4; |
|  | % subImg = double(imgVforOs) - double(medfilt2(imgVforOs, [round(newSize(1)/30) round(newSize(1)/30)] )); |
|  | % subImg = subImg .\* double(imgFovMask); |
|  | % subImg(subImg < 0) = 0; |
|  | % histImg=hist(subImg(:),x); |
|  | % histImg2 = histImg(offset:end); |
|  | % xPos = x(offset:end); |
|  | % pp = splinefit( xPos, histImg2 ); |
|  | % splineHist = ppval( pp, xPos ); |
|  | % % figure;plot(xPos,splineHist); |
|  | % splineHistDD = [diff(diff(splineHist)) 0 0]; |
|  | % zcList = crossing(splineHistDD); |
|  | % th = xPos(zcList(1)); |
|  | % imgThNoOD = subImg >= th; |
|  | % %--- |
|  |  |
|  | % %--- fixed threshold using median Background (normal) |
|  | % subImg = double(imgV8) - double(medfilt2(imgV8, [round(newSize(1)/30) round(newSize(1)/30)] )); |
|  | % subImg = subImg .\* double(imgFovMask); |
|  | % subImg(subImg < 0) = 0; |
|  | % imgThNoOD = uint8(subImg) > 10; |
|  | % %--- |
|  |  |
|  | %--- fixed threshold using median Background (with reconstruction) |
|  | medBg = double(medfilt2(imgV8, [round(newSize(1)/30) round(newSize(1)/30)] )); |
|  | %reconstruct bg |
|  | maskImg = double(imgV8); |
|  | pxLbl = maskImg < medBg; |
|  | maskImg(pxLbl) = medBg(pxLbl); |
|  | medRestored = imreconstruct( medBg, maskImg ); |
|  | % subtract, remove fovMask and threshold |
|  | subImg = double(imgV8) - double(medRestored); |
|  | subImg = subImg .\* double(imgFovMask); |
|  | subImg(subImg < 0) = 0; |
|  | imgThNoOD = uint8(subImg) > 0; |
|  | %--- |
|  |  |
|  |  |
|  | % %--- create mask to remove fov, on and vessels, hence enhance lesions |
|  | % se = strel('disk', 5); |
|  | % imgVess = imdilate(imgVess,se); |
|  | % imgMask = imgFovMask & ~imgVess; |
|  | % %--- |
|  |  |
|  | %--- Calculate wavelet background |
|  | % imgWav = preprocessWavelet( imgV8, imgMask ); |
|  | % imgWav = preprocessWavelet( imgVforOs, imgMask ); |
|  | %--- |
|  |  |
|  | %--- Calculate edge strength of lesions |
|  | imgKirsch = kirschEdges( imgG ); |
|  | img0 = imgG .\* uint8(imgThNoOD == 0); |
|  | img0recon = imreconstruct(img0, imgG); |
|  | img0Kirsch = kirschEdges(img0recon); |
|  | imgEdgeNoMask = imgKirsch - img0Kirsch; % edge strength map |
|  | %--- |
|  | % remove mask and ON (leave vessels) |
|  | imgEdge = double(imgFovMask) .\* imgEdgeNoMask; |
|  |  |
|  | % %--- Calculate edge strength for each lesion candidate (Matlab2009) |
|  | % lesCandImg = zeros( newSize ); |
|  | % lesCand = bwconncomp(imgThNoOD,8); |
|  | % for idxLes=1:lesCand.NumObjects |
|  | % pxIdxList = lesCand.PixelIdxList{idxLes}; |
|  | % lesCandImg(pxIdxList) = sum(imgEdge(pxIdxList)) / length(pxIdxList); |
|  | % end |
|  | % %--- |
|  | %--- Calculate edge strength for each lesion candidate (Matlab2008) |
|  | lesCandImg = zeros( newSize ); |
|  | lblImg = bwlabel(imgThNoOD,8); |
|  | lesCand = regionprops(lblImg, 'PixelIdxList'); |
|  | for idxLes=1:length(lesCand) |
|  | pxIdxList = lesCand(idxLes).PixelIdxList; |
|  | lesCandImg(pxIdxList) = sum(imgEdge(pxIdxList)) / length(pxIdxList); |
|  | end |
|  | %--- |
|  |  |
|  | % %--- Calculate edge strength for each lesion candidate (for wavelet) |
|  | % lesCandImg = zeros( newSize ); |
|  | % lesCandImg2 = zeros( newSize ); |
|  | % lesCand = bwconncomp(imgThNoOD,8); |
|  | % for idxLes=1:lesCand.NumObjects |
|  | % pxIdxList = lesCand.PixelIdxList{idxLes}; |
|  | % if( length(pxIdxList) > 4 ) |
|  | % % lesCandImg(pxIdxList) = sum(imgWav(pxIdxList)) / length(pxIdxList); %mean |
|  | % lesCandImg(pxIdxList) = std(double(imgWav(pxIdxList))); %std |
|  | % lesCandImg2(pxIdxList) = max(imgWav(pxIdxList))-min(imgWav(pxIdxList)); |
|  | % end |
|  | % end |
|  | % %--- |
|  |  |
|  | % resize back |
|  | lesCandImg = imresize( lesCandImg, origSize(1:2), 'nearest' ); |
|  |  |
|  | if( showRes ) |
|  | figure(442); |
|  | imagesc( rgbImgOrig ); |
|  | figure(446); |
|  | imagesc( lesCandImg ); |
|  | end |
|  | end |
|  |  |
|  | function sizeOut = findGoodResolutionForWavelet( sizeIn ) |
|  | % Parameters |
|  | maxWavDecom = 2; |
|  | % |
|  |  |
|  | pxToAddC = 2^maxWavDecom - mod(sizeIn(2),2^maxWavDecom); |
|  | pxToAddR = 2^maxWavDecom - mod(sizeIn(1),2^maxWavDecom); |
|  |  |
|  | sizeOut = sizeIn + [pxToAddR, pxToAddC]; |
|  | end |
|  |  |
|  | function imgOut = preprocessWavelet( imgIn, fovMask ) |
|  | % Parameters |
|  | maxWavDecom = 2; |
|  | % |
|  |  |
|  | % % add pixel to allow wavelet decomposition |
|  | % pxToAddC = 2^maxWavDecom - mod(size(imgIn,2),2^maxWavDecom); |
|  | % pxToAddR = 2^maxWavDecom - mod(size(imgIn,1),2^maxWavDecom); |
|  | % if(pxToAddC > 0 && pxToAddC <= 2^maxWavDecom) |
|  | % imgIn( :,end+1:end+pxToAddC ) = 0; |
|  | % fovMask( :,end+1:end+pxToAddC ) = 0; |
|  | % end |
|  | % if(pxToAddR > 0 && pxToAddR <= 2^maxWavDecom) |
|  | % imgIn( end+1:end+pxToAddR,: ) = 0; |
|  | % fovMask( end+1:end+pxToAddR,: ) = 0; |
|  | % end |
|  |  |
|  | [imgA,imgH,imgV,imgD] = swt2( imgIn, maxWavDecom, 'haar' ); |
|  | imgRecon = iswt2( zeros(size(imgA(:,:,2))),imgH(:,:,2),imgV(:,:,2),imgD(:,:,2), 'haar' ); |
|  |  |
|  | imgRecon(imgRecon < 0) = 0; |
|  | imgRecon = uint8( imgRecon ); |
|  |  |
|  | imgRecon = imgRecon .\* uint8(fovMask); |
|  | imgOut = imgRecon \* (255 / max(imgRecon(:))); |
|  |  |
|  | end |
|  | function f = gauss1d( x, mu, sigma ) |
|  | f = exp( -(x-mu).^2 / (2\*sigma^2) ) / (sigma \* sqrt(2\*pi) ); |
|  | end |