%

% Copyright (C) 2016 Starsky Wong <sununs11@gmail.com>

%

% Note: The SIFT algorithm is patented in the United States and cannot be

% used in commercial products without a license from the University of

% British Columbia. For more information, refer to the file LICENSE

% that accompanied this distribution.

function [ descrs, locs ] = getFeatures( input\_img )

% Function: Get sift features and descriptors

global gauss\_pyr;

global dog\_pyr;

global init\_sigma;

global octvs;

global intvls;

global ddata\_array;

global features;

if(size(input\_img,3)==3)

input\_img = rgb2gray(input\_img);

end

input\_img = im2double(input\_img);

%% Build DoG Pyramid

% initial sigma

init\_sigma = 1.6;

% number of intervals per octave

intvls = 3;

s = intvls;

k = 2^(1/s);

sigma = ones(1,s+3);

sigma(1) = init\_sigma;

sigma(2) = init\_sigma\*sqrt(k\*k-1);

for i = 3:s+3

sigma(i) = sigma(i-1)\*k;

end

% default cubic method

input\_img = imresize(input\_img,2);

% assume the original image has a blur of sigma = 0.5

input\_img = gaussian(input\_img,sqrt(init\_sigma^2-0.5^2\*4));

% smallest dimension of top level is about 8 pixels

octvs = floor(log( min(size(input\_img)) )/log(2) - 2);

% gaussian pyramid

[img\_height,img\_width] = size(input\_img);

gauss\_pyr = cell(octvs,1);

% set image size

gimg\_size = zeros(octvs,2);

gimg\_size(1,:) = [img\_height,img\_width];

for i = 1:octvs

if (i~=1)

gimg\_size(i,:) = [round(size(gauss\_pyr{i-1},1)/2),round(size(gauss\_pyr{i-1},2)/2)];

end

gauss\_pyr{i} = zeros( gimg\_size(i,1),gimg\_size(i,2),s+3 );

end

for i = 1:octvs

for j = 1:s+3

if (i==1 && j==1)

gauss\_pyr{i}(:,:,j) = input\_img;

% downsample for the first image in an octave, from the s+1 image

% in previous octave.

elseif (j==1)

gauss\_pyr{i}(:,:,j) = imresize(gauss\_pyr{i-1}(:,:,s+1),0.5);

else

gauss\_pyr{i}(:,:,j) = gaussian(gauss\_pyr{i}(:,:,j-1),sigma(j));

end

end

end

% dog pyramid

dog\_pyr = cell(octvs,1);

for i = 1:octvs

dog\_pyr{i} = zeros(gimg\_size(i,1),gimg\_size(i,2),s+2);

for j = 1:s+2

dog\_pyr{i}(:,:,j) = gauss\_pyr{i}(:,:,j+1) - gauss\_pyr{i}(:,:,j);

end

end

% for i = 1:size(dog\_pyr,1)

% for j = 1:size(dog\_pyr{i},3)

% imwrite(im2bw(im2uint8(dog\_pyr{i}(:,:,j)),0),['dog\_pyr\dog\_pyr\_',num2str(i),num2str(j),'.png']);

% end

% end

%% Accurate Keypoint Localization

% width of border in which to ignore keypoints

img\_border = 5;

% maximum steps of keypoint interpolation

max\_interp\_steps = 5;

% low threshold on feature contrast

contr\_thr = 0.04;

% high threshold on feature ratio of principal curvatures

curv\_thr = 10;

prelim\_contr\_thr = 0.5\*contr\_thr/intvls;

ddata\_array = struct('x',0,'y',0,'octv',0,'intvl',0,'x\_hat',[0,0,0],'scl\_octv',0);

ddata\_index = 1;

for i = 1:octvs

[height, width] = size(dog\_pyr{i}(:,:,1));

% find extrema in middle intvls

for j = 2:s+1

dog\_imgs = dog\_pyr{i};

dog\_img = dog\_imgs(:,:,j);

for x = img\_border+1:height-img\_border

for y = img\_border+1:width-img\_border

% preliminary check on contrast

if(abs(dog\_img(x,y)) > prelim\_contr\_thr)

% check 26 neighboring pixels

if(isExtremum(j,x,y))

ddata = interpLocation(dog\_imgs,height,width,i,j,x,y,img\_border,contr\_thr,max\_interp\_steps);

if(~isempty(ddata))

if(~isEdgeLike(dog\_img,ddata.x,ddata.y,curv\_thr))

ddata\_array(ddata\_index) = ddata;

ddata\_index = ddata\_index + 1;

end

end

end

end

end

end

end

end

function [ flag ] = isExtremum( intvl, x, y)

% Function: Find Extrema in 26 neighboring pixels

value = dog\_imgs(x,y,intvl);

block = dog\_imgs(x-1:x+1,y-1:y+1,intvl-1:intvl+1);

if ( value > 0 && value == max(block(:)) )

flag = 1;

elseif ( value == min(block(:)) )

flag = 1;

else

flag = 0;

end

end

%% Orientation Assignment

% number of detected points

n = size(ddata\_array,2);

% determines gaussian sigma for orientation assignment

ori\_sig\_factr = 1.5;

% number of bins in histogram

ori\_hist\_bins = 36;

% orientation magnitude relative to max that results in new feature

ori\_peak\_ratio = 0.8;

% array of feature

features = struct('ddata\_index',0,'x',0,'y',0,'scl',0,'ori',0,'descr',[]);

feat\_index = 1;

for i = 1:n

ddata = ddata\_array(i);

ori\_sigma = ori\_sig\_factr \* ddata.scl\_octv;

% generate a histogram for the gradient distribution around a keypoint

hist = oriHist(gauss\_pyr{ddata.octv}(:,:,ddata.intvl),ddata.x,ddata.y,ori\_hist\_bins,round(3\*ori\_sigma),ori\_sigma);

for j = 1:2

smoothOriHist(hist,ori\_hist\_bins);

end

% generate feature from ddata and orientation hist peak

% add orientations greater than or equal to 80% of the largest orientation magnitude

feat\_index = addOriFeatures(i,feat\_index,ddata,hist,ori\_hist\_bins,ori\_peak\_ratio);

end

%% Descriptor Generation

% number of features

n = size(features,2);

% width of 2d array of orientation histograms

descr\_hist\_d = 4;

% bins per orientation histogram

descr\_hist\_obins = 8;

% threshold on magnitude of elements of descriptor vector

descr\_mag\_thr = 0.2;

descr\_length = descr\_hist\_d\*descr\_hist\_d\*descr\_hist\_obins;

local\_features = features;

local\_ddata\_array = ddata\_array;

local\_gauss\_pyr = gauss\_pyr;

clear features;

clear ddata\_array;

clear gauss\_pyr;

clear dog\_pyr;

%par

parfor feat\_index = 1:n

feat = local\_features(feat\_index);

ddata = local\_ddata\_array(feat.ddata\_index);

gauss\_img = local\_gauss\_pyr{ddata.octv}(:,:,ddata.intvl);

% computes the 2D array of orientation histograms that form the feature descriptor

hist\_width = 3\*ddata.scl\_octv;

radius = round( hist\_width \* (descr\_hist\_d + 1) \* sqrt(2) / 2 );

feat\_ori = feat.ori;

ddata\_x = ddata.x;

ddata\_y = ddata.y;

hist = zeros(1,descr\_length);

for i = -radius:radius

for j = -radius:radius

j\_rot = j\*cos(feat\_ori) - i\*sin(feat\_ori);

i\_rot = j\*sin(feat\_ori) + i\*cos(feat\_ori);

r\_bin = i\_rot/hist\_width + descr\_hist\_d/2 - 0.5;

c\_bin = j\_rot/hist\_width + descr\_hist\_d/2 - 0.5;

if (r\_bin > -1 && r\_bin < descr\_hist\_d && c\_bin > -1 && c\_bin < descr\_hist\_d)

mag\_ori = calcGrad(gauss\_img,ddata\_x+i,ddata\_y+j);

if (mag\_ori(1) ~= -1)

ori = mag\_ori(2);

ori = ori - feat\_ori;

while (ori < 0)

ori = ori + 2\*pi;

end

% i think it's theoretically impossible

while (ori >= 2\*pi)

ori = ori - 2\*pi;

disp('###################what the fuck?###################');

end

o\_bin = ori \* descr\_hist\_obins / (2\*pi);

w = exp( -(j\_rot\*j\_rot+i\_rot\*i\_rot) / (2\*(0.5\*descr\_hist\_d\*hist\_width)^2) );

hist = interpHistEntry(hist,r\_bin,c\_bin,o\_bin,mag\_ori(1)\*w,descr\_hist\_d,descr\_hist\_obins);

end

end

end

end

local\_features(feat\_index) = hist2Descr(feat,hist,descr\_mag\_thr);

end

% sort the descriptors by descending scale order

features\_scl = [local\_features.scl];

[~,features\_order] = sort(features\_scl,'descend');

% return descriptors and locations

descrs = zeros(n,descr\_length);

locs = zeros(n,2);

for i = 1:n

descrs(i,:) = local\_features(features\_order(i)).descr;

locs(i,1) = local\_features(features\_order(i)).x;

locs(i,2) = local\_features(features\_order(i)).y;

end

end