Image and Video Processing
Matlab Assignment

Name: Amitesh Kumar Sah
NYU ID: N19714360

Question : Panaroma

```
im1 = imread('scene1.png');
im2 = imread('scene2.png');
fshowA=figure;
subA1=subplot(1,2,1); imshow(im1);
subA2=subplot(1,2,2); imshow(im2);
movegui(fshowA, 'northwest');

fshowH=figure;
subH1=subplot(1,2,1); imshow(im1);
subH2=subplot(1,2,2); imshow(im2);
movegui(fshowH, 'northeast');%% sift features
figure(fshowA); figure(fshowH);
[pts1 pts2] = SIFTmatch( im1, im2, 0, true );%% ransac affine
[im2_TA, best_ptsA] = ransac( pts2, pts1, 'aff_lsq', 3 );
showbestpts(subA2, subA1, best_ptsA);
figure(fshowA);%% ransac homography
figure(fshowA);
[im2_TH, best_ptsH] = ransac( pts2, pts1, 'proj_svd', 5 );
showbestpts(subH2, subH1, best_ptsH);
figure(fshowH);%% stitch affine
[im_stitchedA, stitched_maskA, im1TA, im2TA] = stitch(im1, im2, im2_TA);

fA=figure;
axis off;
movegui(fA, 'west');
imshow(im_stitchedA);%% stitch homography
[im_stitchedH, stitched_maskH, im1TH, im2TH] = stitch(im1, im2, im2_TH);


fH=figure;
axis off;
movegui(fH, 'east');
imshow(im_stitchedH);



function [image, descriptors, locs] = sift(imageFile)

% Load image
image = imread(imageFile);

% If you have the Image Processing Toolbox, you can uncomment the following
%   lines to allow input of color images, which will be converted to grayscale.
% if isrgb(image)
%    image = rgb2gray(image);
% end

[rows, cols] = size(image);

% Convert into PGM imagefile, readable by "keypoints" executable
f = fopen('tmp.pgm', 'w');
```

```
if f == -1
    error('Could not create file tmp.pgm.');
end
fprintf(f, 'P5\n%d\n%d\n255\n', cols, rows);
fwrite(f, image', 'uint8');
fclose(f);

% Call keypoints executable
if isunix
    command = '!./sift ';
else
    command = '!siftWin32 ';
end
command = [command ' <tmp.pgm >tmp.key'];
eval(command);

% Open tmp.key and check its header
g = fopen('tmp.key', 'r');
if g == -1
    error('Could not open file tmp.key.');
end
[header, count] = fscanf(g, '%d %d', [1 2]);
if count ~= 2
    error('Invalid keypoint file beginning.');
end
num = header(1);
len = header(2);
if len ~= 128
    error('Keypoint descriptor length invalid (should be 128).');
end

% Creates the two output matrices (use known size for efficiency)
locs = double(zeros(num, 4));
descriptors = double(zeros(num, 128));

% Parse tmp.key
for i = 1:num
    [vector, count] = fscanf(g, '%f %f %f %f', [1 4]); %row col scale ori
    if count ~= 4
        error('Invalid keypoint file format');
    end
    locs(i, :) = vector(1, :);

    [descrip, count] = fscanf(g, '%d', [1 len]);
    if (count ~= 128)
        error('Invalid keypoint file value.');
    end
    % Normalize each input vector to unit length
    descrip = descrip / sqrt(sum(descrip.^2));
    descriptors(i, :) = descrip(1, :);
end
fclose(g);

function [points1, points2] = SIFTmatch(im1, im2, mode, show)
if nargin < 4, show = false;end

if nargin >= 3 && mode == 1
    load previous_points points1 points2
    fprintf('using previous points (%i matches)\n',size(points1,1));
else
```

```
    % Get matching SIFT keypoints
    [loc1,loc2,matchidxs]=mymatch( rgb2gray(im1), rgb2gray(im2), show );

    % Initialize point vectors
    points1 = loc1(find(matchidxs>0),1:2);
    points2 = loc2(nonzeros(matchidxs),1:2);
    points1 = points1(:,[2 1]);
    points2 = points2(:,[2 1]);

    % Remove duplicate points
    pts=unique([points1 points2], 'rows');
    disp(sprintf('Number of matches: %i (unique: %i)',size(points1,1),size(pts,1)))
    fprintf('\n')
    points1 = pts(:,[1 2]);
    points2 = pts(:,[3 4]);

    save previous_points points1 points2

end
end


function [stitched_image, stitched_mask, im1, im2] = stitch(im1, im2, T_im2, mask)

disp('Stitching');
time = tic();

% init masks
mask_im2 = uint8(ones(size(im2,1),size(im2,2)));

if nargin < 4
    mask_im1 = uint8(ones(size(im1, 1), size(im1, 2)));
else
    mask_im1 = mask;
end

% Transform image 2 so it fits on image 1
[im2, XDATA, YDATA] = imtransform(im2, T_im2);
mask_im2 = imtransform(mask_im2, T_im2);

% stitched image bounds
W=max( [size(im1,2) size(im1,2)-XDATA(1) size(im2,2) size(im2,2)+XDATA(1)] );
H=max( [size(im1,1) size(im1,1)-YDATA(1) size(im2,1) size(im2,1)+YDATA(1)] );

% Align image 1 bounds
im1_X = eye(3);
if XDATA(1) < 0, im1_X(3,1)= -XDATA(1); end
if YDATA(1) < 0, im1_X(3,2)= -YDATA(1); end
T_im1 = maketform('affine',im1_X);

[im1, XDATA2, YDATA2] = imtransform(im1, T_im1, 'XData', [1 W], 'YData', [1 H]);
mask_im1 = imtransform(mask_im1, T_im1, 'XData', [1 W], 'YData', [1 H]);

% Align image 2 bounds
im2_X = eye(3);
if XDATA(1) > 0, im2_X(3,1)= XDATA(1); end
if YDATA(1) > 0, im2_X(3,2)= YDATA(1); end
T_im2 = maketform('affine',im2_X);
```

```
[im2, XDATA, YDATA] = imtransform(im2, T_im2, 'XData', [1 W], 'YData', [1 H]);
mask_im2 = imtransform(mask_im2, T_im2, 'XData', [1 W], 'YData', [1 H]);

% Size check
if (size(im1,1) ~= size(im2,1)) || (size(im1,2) ~= size(im2,2))
    H = max( size(im1,1), size(im2,1) );
    W = max( size(im1,2), size(im2,2) );
    im1(H,W,:)=0;
    im2(H,W,:)=0;
    mask_im1(H,W)=0;
    mask_im2(H,W)=0;
end

% Combine both images
n_layers = max(max(mask_im1));
im1part = uint16(mask_im1 > (n_layers * mask_im2));
im2part = uint16(mask_im2 > mask_im1);

combpart = uint16(repmat(mask_im1 .* mask_im2,[1 1 3]));
combmask = uint16(combpart > 0);

stitched_image = repmat(im1part,[1 1 3]) .* uint16(im1) + repmat(im2part,[1 1 3]) .*
uint16(im2);
stitched_image = stitched_image + ( combpart .* uint16(im1) + combmask .* uint16(im2) )
./ (combpart + uint16(ones(size(combpart,1),size(combpart,2),3)));
stitched_image = uint8(stitched_image);
stitched_mask = mask_im1 + mask_im2;

disp('Done.');
toc(time);

end

function [loc1,loc2,matchidxs] = mymatch(image1, image2, show)

if nargin < 3
    show=true;
end

% Find SIFT keypoints for each image
[im1, des1, loc1] = sift(image1);
[im2, des2, loc2] = sift(image2);

% For efficiency in Matlab, it is cheaper to compute dot products between
%  unit vectors rather than Euclidean distances.  Note that the ratio of
%  angles (acos of dot products of unit vectors) is a close approximation
%  to the ratio of Euclidean distances for small angles.
%
% distRatio: Only keep matches in which the ratio of vector angles from the
%   nearest to second nearest neighbor is less than distRatio.
distRatio = 0.6;

% For each descriptor in the first image, select its match to second image.
des2t = des2';                          % Precompute matrix transpose
matchidxs=zeros(size(des1,1),1);
for i = 1 : size(des1,1)
   dotprods = des1(i,:) * des2t;        % Computes vector of dot products
   [vals,indx] = sort(acos(dotprods));  % Take inverse cosine and sort results
```

```
   % Check if nearest neighbor has angle less than distRatio times 2nd.
   if (vals(1) < distRatio * vals(2))
      matchidxs(i) = indx(1);
   end
end

if show
    % Create a new image showing the two images side by side.
    im3 = appendimages(im1,im2);

    % Show a figure with lines joining the accepted matches.
    figure('Position', [100 100 size(im3,2) size(im3,1)]);
    colormap('gray');
    imagesc(im3);
    hold on;
    cols1 = size(im1,2);
    for i = 1: size(des1,1)
      if (matchidxs(i) > 0)
        line([loc1(i,2) loc2(matchidxs(i),2)+cols1], ...
             [loc1(i,1) loc2(matchidxs(i),1)], 'Color', 'c');
      end
    end
hold off;
end
num = sum(matchidxs > 0);
fprintf('Found %d matches.\n', num);


function [ T_im1, best_pts ] = ransac( points1, points2, transMode, n_pts )

disp('Performing RANSAC');
t = tic();

best_n_inliers = -1;
idx_best = zeros(n_pts,1);
it_improv={}; % saves the amount of inliers per improvment
n_iters = 2^n_pts * 10; % Iteration rule of thumb :)

fprintf('RANSAC Progress (%i iterations): <*- 0%%',n_iters)
for i = 1:n_iters

    % progressbar
    if mod(i,floor(n_iters/10))==0
        fprintf('\b\b\b\b*-%i%%',floor(100.0*i/n_iters))
    end

    % Create a set of 'n_pts' unique point indices
    idxset = 1:length(points2(:,1));
    idxs=zeros(n_pts,1);
    for j = 1:n_pts
        idx=randi(length(idxset)-j+1);
        idxs(j)=idxset(idx);
        idxset(idx:end-1)=idxset(idx+1:end);
    end

    % Calculate the transformation
    warning off all
    switch transMode
        case 'aff_lsq'
```

```
            T = affine_leastsquare(points1(idxs,:), points2(idxs,:));
        case 'proj_svd'
            T = homography_svd(points1(idxs,:), points2(idxs,:));
        otherwise
            disp('transmode not known')
            return
    end
    warning on all
    % transformation check
    if max(max(isnan(T.tdata.T)))==1
        disp('nan');
        continue
    end

    % Apply the transformation ...
    [A_X A_Y] = tformfwd(T,points1(:,1),points1(:,2));
    dXsq = (A_X - points2(:,1)).^2;
    dYsq = (A_Y - points2(:,2)).^2;

    % .. and count the amount of inliers
    n_inliers=0;
    for i=1:length(dXsq)
        e=sqrt(dXsq(i)+dYsq(i));
        if e <= 2 % inlier radius in px
            n_inliers=n_inliers+1;
        end
    end

    % improvment check
    if n_inliers > best_n_inliers
        best_n_inliers = n_inliers;
        T_im1 = T;
        it_improv{end+1}=best_n_inliers;
        idx_best = idxs;
    end
end
fprintf('>\n')

disp('Inliers:');
disp(it_improv);

best_pts=zeros(length(idx_best),4);
best_pts(:,[1 2])=points1(idx_best,:);
best_pts(:,[3 4])=points2(idx_best,:);

disp('done.')
toc(t)

end


function void = showbestpts(subim1, subim2, best_pts)

h=impoint(subim1,best_pts(1,[1 2]));setColor(h,'r');
h=impoint(subim2,best_pts(1,[3 4]));setColor(h,'r');
h=impoint(subim1,best_pts(2,[1 2]));setColor(h,'g');
h=impoint(subim2,best_pts(2,[3 4]));setColor(h,'g');
h=impoint(subim1,best_pts(3,[1 2]));setColor(h,'y');
h=impoint(subim2,best_pts(3,[3 4]));setColor(h,'y');
if size(best_pts,1)>=4
```

```
    h=impoint(subim1,best_pts(4,[1 2]));setColor(h,'b');
    h=impoint(subim2,best_pts(4,[3 4]));setColor(h,'b');
end
if size(best_pts,1)>=5
    h=impoint(subim1,best_pts(5,[1 2]));setColor(h,'black');
    h=impoint(subim2,best_pts(5,[3 4]));setColor(h,'black');
end

end
```

Output

Left Image

Right Image