Computer Vision Homework 3

Tianwen Chu
chutianwen123@gmail.com

December 17, 2013

1. Keypoint Detector

We will be implementing a simplified version of the DoG detector described in Section 3 of [6]. The parameters you will use for the following sections are $\sigma_0 = 1, k = \sqrt{2}$ *,levels=[-1 0 1 2 3 4], th_contrast = 0:03, and th_r = 12.*

1.1.1 The DoG Pyramid

The DoG pyramid is obtained by subtracting successive levels of the Gaussian pyramid.

$$D_l(x, y, \sigma_l) = (G(x, y, \sigma_l) - G(x, y, \sigma_{l-1})) * I(x, y)$$

The function should return DoGPyramid,a $R \times C \times (L\text{-}1)$ *matrix of the DoG pyramid created using GaussianPyramid. Note that you will have one less level than the Gaussian pyramid. DoG levels is an (L-1) vector specifying the corresponding levels of the DoG pyramid.*

```
function [DoGPyramid, DoG_levels] = createDoGPyramid(GaussianPyramid, levels)
for i=1:length(levels)-1
    DoGPyramid(:,:,i)=GaussianPyramid(:,:,i+1)-GaussianPyramid(:,:,i);
    DoG_levels(i)=levels(i+1);
end
```

1.1.2 Edge Suppression (10 pts)

The function takes in DoGPyramid generated in the previous section and returns PrincipalCurvature, a matrix of the same size where each point contains the curvature ratio R for the corresponding point in the DoG pyramid:

$$R = \frac{Tr(H)^2}{Det(H)} = \frac{(\lambda_{\min} + \lambda_{\max})^2}{\lambda_{\min}\lambda_{\max}}$$

Here, H is the Hessian of the Difference of Gaussian function (i.e. one level of the DoGpyramid) computed by using pixel differences as mentioned in Section 4.1 of [6]. (hint:Matlab function gradient). This is similar in spirit but different from the Harris matrix you saw in class. Both methods examine the eigen values of a matrix, but the method in [6] performs a test that does not require direct computation of the values.

According to reference, $= \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$, $Tr(H) = D_{xx} + D_{yy}$, $Det(H) = D_{xx}D_{yy} - {D_{xy}}^2$

```
function PrincipalCurvature = computePrincipalCurvature(DoGPyramid)
PrincipalCurvature=zeros(size(DoGPyramid));
L=size(DoGPyramid);
for i=1:L(3)
    [Dx,Dy]=gradient(DoGPyramid(:,:,i));
    Dxx=gradient(Dx);
    [Dyx,Dyy]=gradient(Dy);
    Tr=Dxx+Dyy;
    Det=Dxx.*Dyy-Dyx.*Dyx;
    PrincipalCurvature(:,:,i)=Tr.^2./Det;
end
```

1.1.3 Detecting Extrema (10 pts)

The function should return locs, an $N \times 3$ matrix where the DoG pyramid achieves a local extrema in both scale and space, and also satisfies the two thresholds.

I use 26 points surrounding the target points to determine whether it is an extrema. For the first and last layer, because the neighbor layer just exists one, so the number of surrounding points become 17.

```
function locs = getLocalExtrema(DoGPyramid, DoG_levels,
PrincipalCurvature,th_contrast, th_r)
[num_row,num_col]=size(DoGPyramid(:,:,1));
[col,row]=meshgrid(2:num_col-1,2:num_row-1);
index_center=sub2ind([num_row,num_col],row,col);
index_center=reshape(index_center,1,(num_row-2)*(num_col-2));
locs=[];
%% process
for lay=1:length(DoG_levels)
    index=[];       %   all the surrounding points
    for i=-1:1
        for j=-1:1
        temp=sub2ind([num_row,num_col],row+i,col+j);
        temp=reshape(temp,1,(num_row-2)*(num_col-2));
        index=[index;temp];
        end
    end
    % Three layers
    DoGPyramid_current=DoGPyramid(:,:,lay);
    Pixels_all=DoGPyramid_current(index);
    if (lay>1 & lay<length(DoG_levels))
    DoGPyramid_current_below=DoGPyramid(:,:,lay-1);
    DoGPyramid_current_above=DoGPyramid(:,:,lay+1);

Pixels_all=[Pixels_all;DoGPyramid_current_below(index);DoGPyramid_current_a
bove(index)];
    Pixels_all_max=max(Pixels_all);
    Pixels_all_min=min(Pixels_all);
    else if(lay==1)
           DoGPyramid_current_above=DoGPyramid(:,:,lay+1);
           Pixels_all=[Pixels_all;DoGPyramid_current_above(index)];
           Pixels_all_max=max(Pixels_all);
           Pixels_all_min=min(Pixels_all);
       else
           DoGPyramid_current_below=DoGPyramid(:,:,lay-1);
           Pixels_all=[Pixels_all;DoGPyramid_current_below(index)];
           Pixels_all_max=max(Pixels_all);
           Pixels_all_min=min(Pixels_all);
       end
    end
```

```
    % compare and get the extrema
    Pixels_center=DoGPyramid_current(index_center);
    PrincipalCurvature_current=PrincipalCurvature(:,:,lay);
    PrincipalCurvature_current=PrincipalCurvature_current(index_center);
    index_extrema=index_center(find( (Pixels_all_max==Pixels_center |
Pixels_all_min== Pixels_center )& abs(Pixels_center)>th_contrast &
PrincipalCurvature_current<th_r ));
    [x,y]=ind2sub([num_row,num_col],index_extrema);
    %% record the num of DOG_layer
    current_layer_num=zeros(1,length(index_extrema));
    current_layer_num(:)=DoG_levels(lay);
    temp=[x;y;current_layer_num];
    locs=[locs,temp];
end
end
```

1.2 Putting it together (5 pts)

Write the following function to combine the above parts into a DoG detector:

```
function [locs, GaussianPyramid] = DoGdetector(im, sigma0, k,
levels,th_contrast, th_r)
[GaussianPyramid] = createGaussianPyramid(im, sigma0, k, levels);
[DoGPyramid, DoG_levels] = createDoGPyramid(GaussianPyramid, levels);
PrincipalCurvature = computePrincipalCurvature(DoGPyramid);
locs = getLocalExtrema(DoGPyramid, DoG_levels, PrincipalCurvature,th_contrast,
th_r);
end
```

2. BRIEF Descriptor

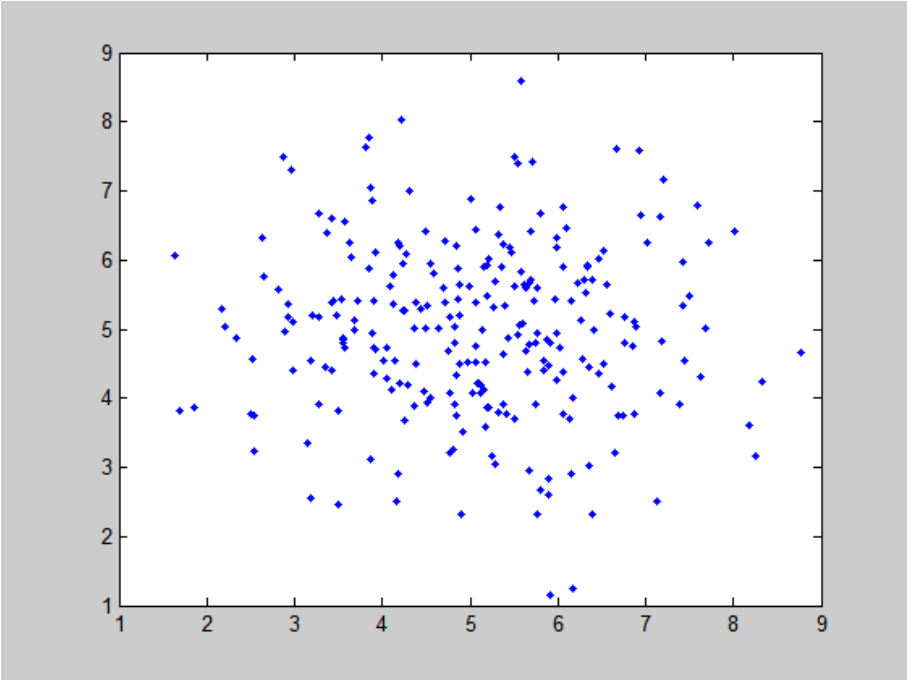
2.1 Creating a Set of BRIEF Tests (5 pts)

The descriptor itself is a vector that is n-bits long, where each bit is the result of the following simple test:

$$\tau(\mathrm{p};x,y):=\begin{cases}1 & p(x)<p(y)\\ 0 & otherwise\end{cases}$$

Set n to 256 bits. There is no need to encode the test results as actual bits. It is fine to encode them as a 256 element vector.Write the function:Where patchWidth is the width of the image patch (usually 9) and nbits is the number of tests in the BRIEF descriptor. compareX and compareY are linear indices into the patchWidth×patchWidth image patch and are each nbits ×1 vectors.

```
function [compareX, compareY] = makeTestPattern(patchWidth, nbits)
sigma=patchWidth/5;
limit_border=floor(patchWidth/2);
compareX = round(mvnrnd([0 0],[sigma,0;0,sigma], nbits));
compareX(compareX>limit_border)=limit_border;
compareX(compareX<-limit_border)=-limit_border;
compareY = round(mvnrnd([0 0],[sigma,0;0,sigma], nbits));
compareY(compareY>limit_border)=limit_border;
compareY(compareY<-limit_border)=-limit_border;
%%  normalize to positive index
compareX=compareX+limit_border+1;
compareY=compareY+limit_border+1;
compareX=sub2ind([patchWidth,patchWidth],compareX(:,1),compareX(:,2));
compareY=sub2ind([patchWidth,patchWidth],compareY(:,2),compareY(:,2));
save testPattern compareX compareY;
end
```



2.2 Compute the BRIEF Descriptor (10 pts)

The function returns locs, an $m \times 3$ *vector, where the first two columns are the image coordinates of keypoints and the third column is the pyramid level of the keypoints, and desc, an* $m \times$ *nbits matrix of stacked BRIEF descriptors. m is the number of valid descriptors in the image and will vary.*

```
function [locs,desc] = computeBrief(im, locs, levels, compareX, compareY)
%%  output the deviation
```

```
[height,width]=size(im);       % height:row, width: col
[compareX_row,compareX_col]=ind2sub([9,9],compareX);
[compareY_row,compareY_col]=ind2sub([9,9],compareY);
compareX=[compareX_row-5,compareX_col-5];
compareY=[compareY_row-5,compareY_col-5];
%% wipe out the unvalid interesting points
points_interest=locs';     %   m*3
index_row_unvalid=find(points_interest(:,1)<5 |
points_interest(:,1)>height-4);
index_col_unvalid=find(points_interest(:,2)<5 |
points_interest(:,2)>width-4);
points_interest([index_row_unvalid;index_col_unvalid],:)=[];
m=length(points_interest(:,1));    %   number of valid interesting points
desc=zeros(m,256);
%%  outputs absolute coordinates interesting points
index_center_row=repmat(points_interest(:,1),1,256);
index_center_col=repmat(points_interest(:,2),1,256);
%   output X group
index_compareX_row=zeros(1,256);
index_compareX_col=zeros(1,256);
compareX=compareX';
index_compareX_row=compareX(1,:);
index_compareX_col=compareX(2,:);
index_compareX_row=repmat(index_compareX_row,m,1);
index_compareX_col=repmat(index_compareX_col,m,1);
index_points_X_row=index_compareX_row+index_center_row;
index_points_X_col=index_compareX_col+index_center_col;
index_points_X_linear=sub2ind(size(im),index_points_X_row,index_points_X_co
l);
points_X=im(index_points_X_linear);
%   output Y group
index_compareY_row=zeros(1,256);
index_compareY_col=zeros(1,256);
compareY=compareY';
index_compareY_row=compareY(1,:);
index_compareY_col=compareY(2,:);
index_compareY_row=repmat(index_compareY_row,m,1);
index_compareY_col=repmat(index_compareY_col,m,1);
index_points_Y_row=index_compareY_row+index_center_row;
index_points_Y_col=index_compareY_col+index_center_col;
index_points_Y_linear=sub2ind(size(im),index_points_Y_row,index_points_Y_co
l);
points_Y=im(index_points_Y_linear);
%%  output desc
```

```
result=points_Y-points_X;
desc(find(result>0))=1;
locs=points_interest;
end
```

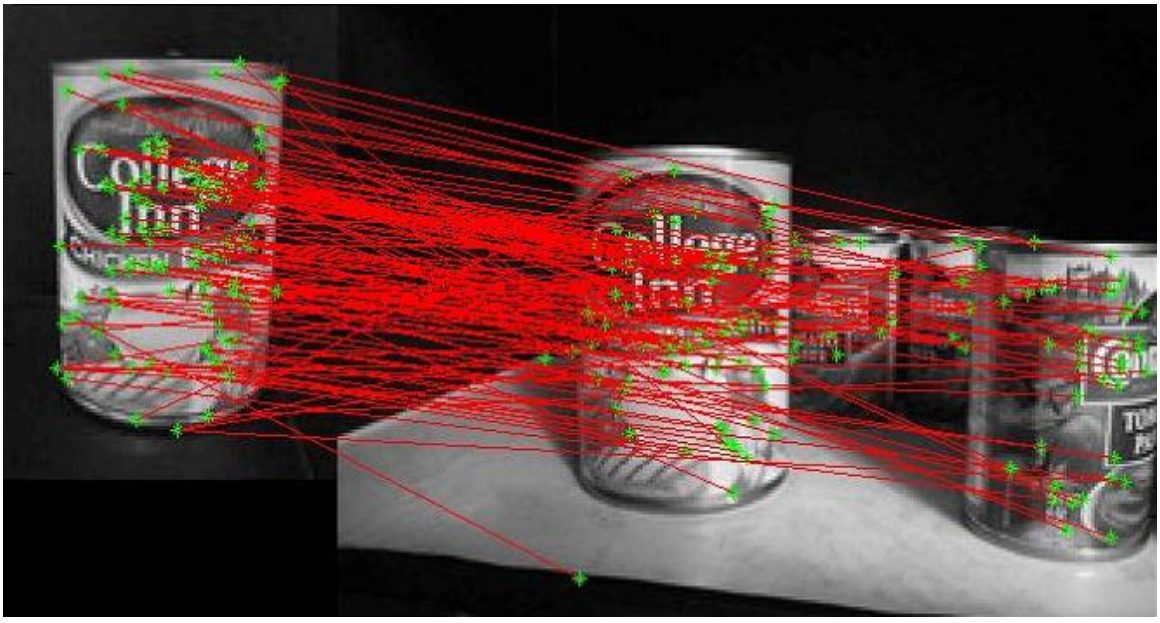
2.3 Putting it together (5 pts)

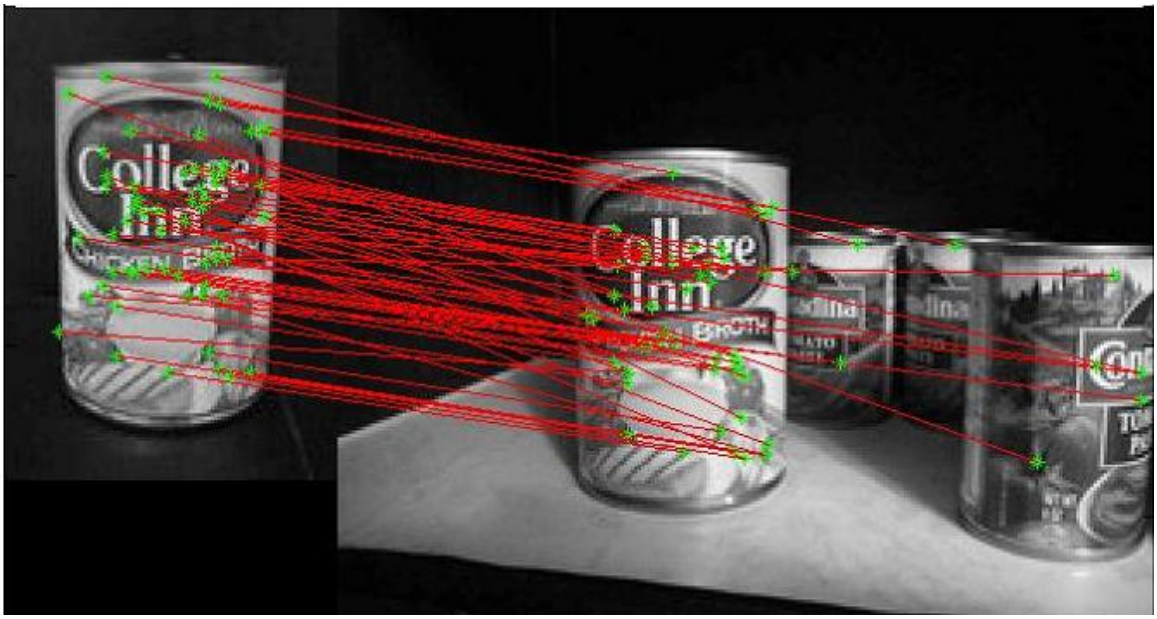
Write a function: Which accepts a grayscale image im with values between zero and one and returns locs, an m_3 vector, where the _rst two columns are the image coordinates of keypoints andthe third column is the pyramid level of the keypoints, and desc, an m _ nbits matrixof stacked BRIEF descriptors. m is the number of valid descriptors in the image andwill vary.

```
function [locs, desc] = brief(im)
load parameters.mat;
[locs, GaussianPyramid] = DoGdetector(im, sigma0, k, levels,th_contrast, th_r);
load testPattern1;
% h = fspecial('gaussian',9,2);
% im_filtered=imfilter(im,h);
[locs,desc] = computeBrief(im, locs, levels, compareX, compareY);
end
```

2.4 Descriptor Matching (10 pts)

A descriptor's strength is in its ability to match to other descriptors generated by the same world point, despite change of view, lighting, etc. The distance metric used to compute the similarity between two descriptors is critical. For BRIEF, this distance metric is the Hamming distance. The Hamming distance is simply the percentage of bits in two descriptors that di_er. (Note that the position of the bits matters.) Write the function





The first picture is processed after imfiltering Gaussian filter, and second one has not such process. Opposite to assumption, second one seems much better.

```
function [matches] = briefMatchfrom1to2(desc1, desc2, ratio)
num_row_1=length(desc1(:,1));      % m rows
num_row_2=length(desc2(:,1));      % n rows
%% creat a (m*n)*256 big computing matrix
% Hamming_distance=zeros(m,n);
desc2_big=reshape(desc2',1,num_row_2*256);
desc2_big=repmat(desc2_big,num_row_1,1);
desc1_big=repmat(desc1,1,num_row_2);
desc_difference=desc2_big-desc1_big;
desc_difference_cell=mat2cell(desc_difference,ones(1,num_row_1),repmat(256,
1,num_row_2));
%%  compute the distance and output result matrix
Distance_result= cellfun(@Hamming_distance, desc_difference_cell,
'UniformOutput', false);
Distance_result=cell2mat(Distance_result);
Distance_result=Distance_result';      % change from 1 to 2
[first_min, first_min_index]=min(Distance_result);
%%  get the second max
col_min=1:num_row_1;
index_min=sub2ind([num_row_2,num_row_1],first_min_index,col_min);
Distance_result(index_min)=1;
[second_min, second_min_index]=min(Distance_result);
Result_ratio=first_min./second_min;
index_out=find(Result_ratio>ratio);
points_test_chose=(1:num_row_1)';
points_test_chose(index_out)=[];
```

```
first_min_index(index_out)=[];
matches=[points_test_chose,first_min_index'];
end

function result_distance=Hamming_distance(data)
result_distance=numel(find(abs(data)==1))/256;
end
```

Another version which uses *Pdist* to compute, more convenient and suitable for large picture.

```
function [matches] = briefMatch_easyversion(desc1, desc2, ratio)
%% output 1st min and corresponding index
[min_1st, I] = min(pdist2(desc1, desc2, 'hamming')');

%% initiate matches.
matches = [[1:length(min_1st)]',I'];

%% prepare for find min_2nd
hammingDistance = sort(pdist2(desc1, desc2, 'hamming'), 2);

%% output the indeces of valid ratio between min_1st and min_2nd
% than ratio.
ratio_compared = min_1st./hammingDistance(:, 2)';

indece = find(ratio_compared > ratio);

%% Assign empty to non-matched points.
matches(indece, :) = [];

end
```

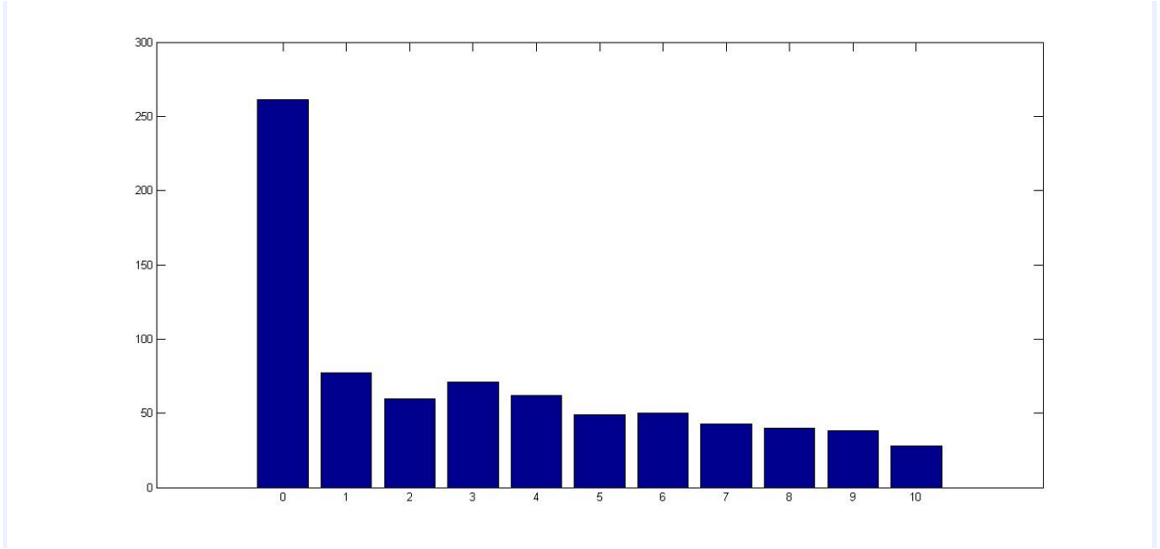
2.5 BRIEF and Rotations (10 pts)

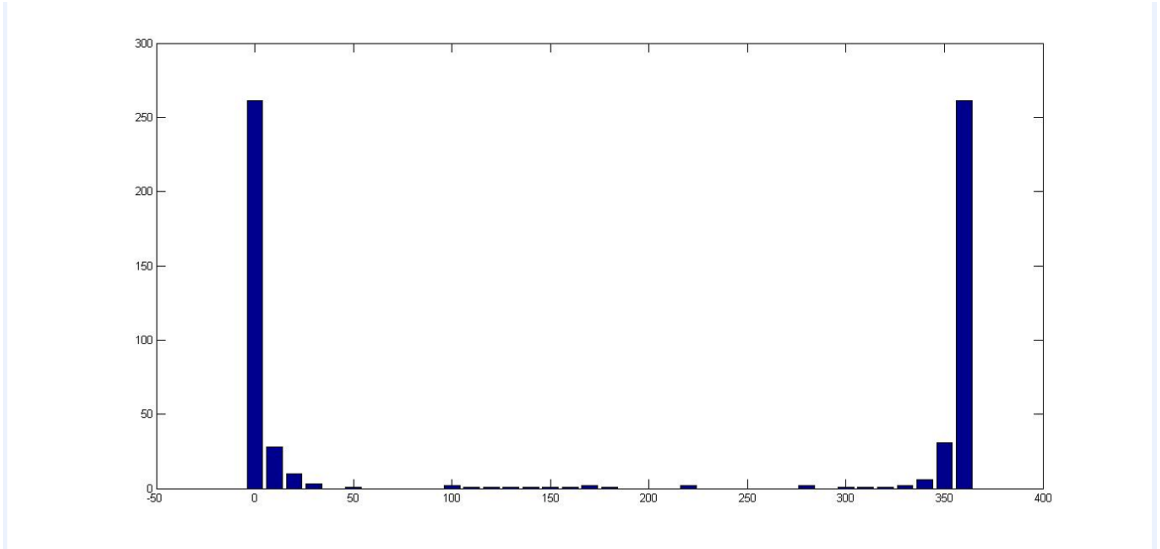
Take one of the test images and match it to itself while rotating the second imagein increments of 10 degrees. Count the number of correct matches at each rotationand construct a histogram. Include this in your PDF and explain why you think thedescriptor behaves this way. Create a script briefRotTest that performs this task.

I experiment the rotation within two periods, one is [0,10] degrees and another is [0,360] degrees.

```
tic;
% Read in two images.
I_RGB = imread('model_chickenbroth.jpg');
I_gray = im2double(rgb2gray(I_RGB));

% Define inline function to create an
% affine scaling matrix:
Scalef = @(s)([ s 0 0; 0 s 0; 0 0 1]);
% Same for translation
Transf = @(tx,ty)([1 0 tx; 0 1 ty; 0 0 1]);
% Same for rotation
Rotf = @(t)([cos(t) -sin(t) 0; sin(t) cos(t) 0; 0 0 1]);

% Output
out_size = [floor(size(I_gray,1)) floor(size(I_gray,2))];

% Pick a point around which to center
cx = size(I_gray,2)/2;
cy = size(I_gray,1)/2;

% Set fill value to black.
fill_value = 0;
j=1;
for i = 0:10:360

    % Center around cx,cy, rotate it a bit.
```

```
    H =
Transf(out_size(2)/2,out_size(1)/2)*Scalef(1)*Rotf(-i*pi/180)*Transf(-cx,-c
y);

    % Rotate the image.
    I_gray_warp = warpH(I_gray, H, out_size, fill_value);

    % Set the ratio to 0.8
    ratio = 0.8;

    % Calculate the location and descriptor of interested points.
    [locs1, desc1] = brief(I_gray);
    [locs2, desc2] = brief(I_gray_warp);

    % Find the matches.
    [matches] = briefMatchfrom1to2(desc1, desc2, ratio);

    % Plot the Match points.
%     plotMatches(I_gray, I_gray_warp, matches, locs1, locs2);

%% output two groups of mapping points
points_left_pic=locs1(matches(:,1),2:-1:1);       % into [x, y]
points_left_pic(:,3)=1;
points_left_pic=points_left_pic';
points_left_pic_tran=H*points_left_pic;
points_left_pic_tran(1:2,:)=points_left_pic_tran(1:2,:)./[points_left_pic_t
ran(3,:);points_left_pic_tran(3,:)];
points_left_pic_tran(1:2,:)=round(points_left_pic_tran(1:2,:));
points_right_pic=locs2(matches(:,2),2:-1:1);
points_right_pic(:,3)=1;
points_right_pic=points_right_pic';

%% compute num of right mapping points
points_difference=points_left_pic_tran-points_right_pic;
number_right(j)=numel(find(points_difference(1,:)==0 &
points_difference(2,:)==0 ));
j=j+1;
end
toc;
```





First histogram describes the change of number of correctly mapping points during 0 to 10 degrees. Because the brief descriptor is not rotational invariance, so the number drops swiftly from 0 to 1 degree, then gradually decreases. Once the angle is larger than 30 degrees, the number is almost zero until the angle larger than 340 degrees. Afterwards, the picture rotates back to starting moment, so the number increases. The brief descriptor is sensitive to the rotation, because the 256 bits information recorded by the spatial order.

3 Fun With Homography

3.1 RANSAC (10 pts)

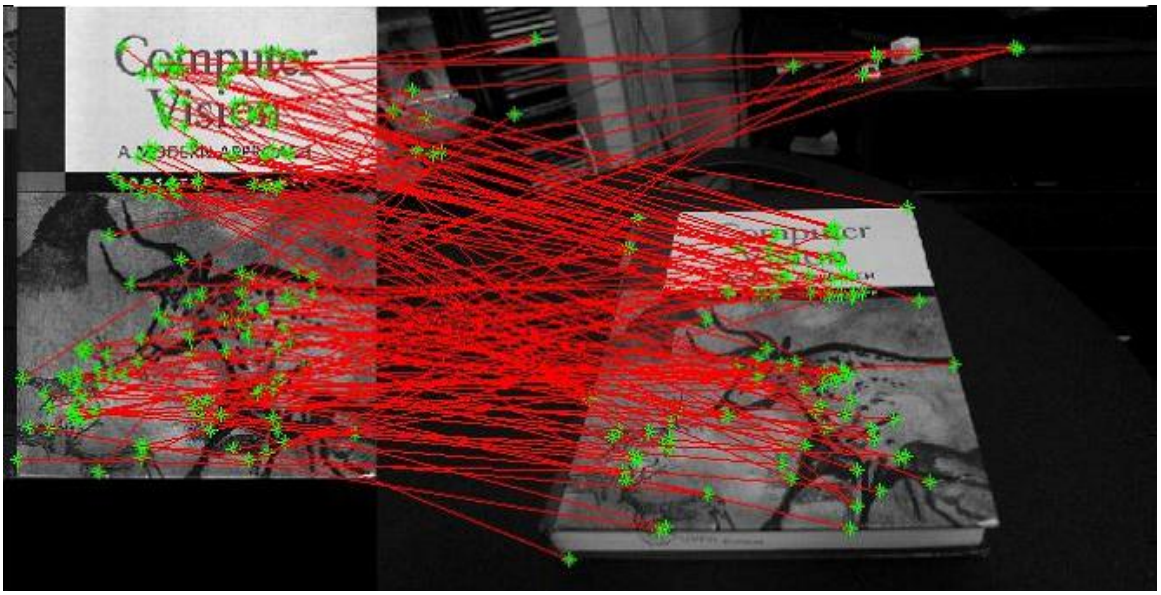
The RANSAC algorithm [5] can generally to _t any model to data. You will implement it for (planar) homographies between images. Remember that 4 point-pairs are required at a minimum to compute a homography.

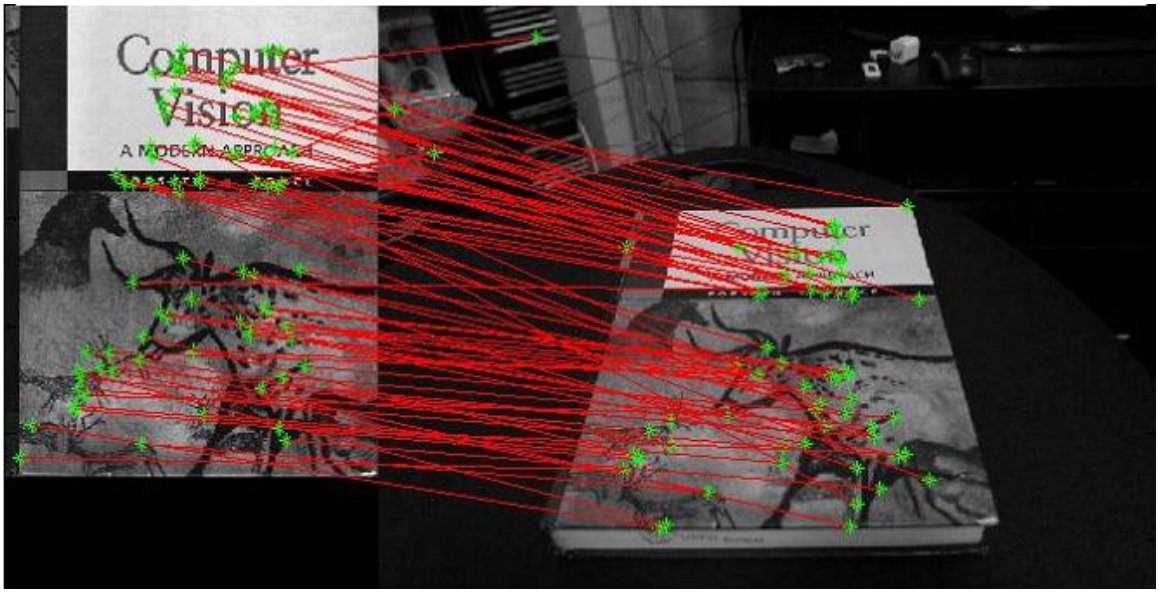
Before doing Ransac, I preprocess the matching points using codes as below. The process can mainly be described as:

for ith (I = 1:N) estimation

(a) randomly choose 4 correspondences
(b) check whether these points are colinear, if so, redo the above step
(c) compute the homography by *ComputeNormH* from the 4 points pairs
(d) for each putative correspondence, calculate current number *m* of correctly matching points
(e) if current *m* is larger than previous max one, we re-estimate the iteration N and update H
(f) $N = \frac{\log(1-p)}{\log(1-(1-\epsilon)^4)}$, $\epsilon = 1 - \frac{m}{n}$, n is the number of sample we choose four points.

```
% preprocess
points_left_pic=locs1(matches(:,1),2:-1:1)';        % into [x, y]  locs1
points_right_pic=locs2(matches(:,2),2:-1:1)';       % locs2
k_feature_all=(points_right_pic(2,:)-points_left_pic(2,:))./(points_right_p
ic(1,:)-points_left_pic(1,:));
matches=matches(find(k_feature_all>0 & k_feature_all<1.5),:);
```






```
function [bestH2to1, bestError, inliers,num_correct]=ransacH2to1(matches,
locs1, locs2)
inliers=zeros(1,size(matches,1));
%%  produce homography coordinates
points_left_pic=locs1(matches(:,1),2:-1:1)';        % into [x, y]  locs1
points_right_pic=locs2(matches(:,2),2:-1:1)';       % locs2

%%  initiate the parameters
i=1;
num_iteration=1000;
p=0.99;
max_number=0;
while(i<num_iteration)
    num_mode=3;num_mode_r=3;
    while ( (num_mode==3 | num_mode==6) | ( num_mode_r==3 | num_mode_r==6) )
    %%  randperm 4 points
    index_rand_four_points=randperm(size(points_left_pic,2),4);
    points_left_four=points_left_pic(:,index_rand_four_points);
    points_right_four=points_right_pic(:,index_rand_four_points);

    %%  whether three points colinear

    k_1_234=(points_left_four(2,2:4)-points_left_four(2,1))./(points_left_fo
    ur(1,2:4)-points_left_four(1,1));

    k_2_34=(points_left_four(2,3:4)-points_left_four(2,2))./(points_left_fou
    r(1,3:4)-points_left_four(1,2));

    k_3_4=(points_left_four(2,4)-points_left_four(2,3))./(points_left_four(1
```

```
    ,4)-points_left_four(1,3));
    k_all=[k_1_234,k_2_34,k_3_4];
    %   right points also not colinear

    k_1_234_r=(points_right_four(2,2:4)-points_right_four(2,1))./(points_rig
    ht_four(1,2:4)-points_right_four(1,1));

    k_2_34_r=(points_right_four(2,3:4)-points_right_four(2,2))./(points_righ
    t_four(1,3:4)-points_right_four(1,2));

    k_3_4_r=(points_right_four(2,4)-points_right_four(2,3))./(points_right_f
    our(1,4)-points_right_four(1,3));
    k_all_r=[k_1_234_r,k_2_34_r,k_3_4_r];
    [useless, num_mode]=mode(k_all);
    [useless, num_mode_r]=mode(k_all_r);
    check_nan=isnan(k_all);
    check_nan_2=isnan(k_all_r);
    if ( length(find(check_nan==1))~=0 | length(find(check_nan_2==1))~=0 )
        num_mode=3;num_mode_r=3;
    end

    end

%%  get the  H
    [H,A] = computeH_norm( points_left_four,points_right_four);

%%  calculate the right matching numbers
    points_left_pic(3,:)=1;
    points_right_pic(3,:)=1;
    points_left_pic_transform=H*points_left_pic;

points_left_pic_transform(1:2,:)=round(points_left_pic_transform(1:2,:)./[p
oints_left_pic_transform(3,:);points_left_pic_transform(3,:)]);

%%  calculate the distance
    points_difference=points_left_pic_transform-points_right_pic;
    index_correct_mapping=find(points_difference(1,:)==0 &
    points_difference(2,:)==0);
    num_correct(i)=length(index_correct_mapping);

%%  iterate the number of loops
    if num_correct(i)>max_number
        num_iteration= log(1-p)/log(1-(num_correct(i)/size(matches, 1))^4);
        bestH2to1=H;
```

```
        max_number= num_correct(i);
        index_correct_mapping_result=index_correct_mapping;

    bestError=sum(sqrt( (points_left_pic_transform(1,:)-points_right_pic(1,:
    )).^2+(points_left_pic_transform(2,:)-points_right_pic(2,:)).^2 ));
    end
    points_right_pic(3,:)=[];
    points_left_pic(3,:)=[];
    i=i+1;
end
inliers(index_correct_mapping_result)=1;

end
```

3.2 Putting it together (5 pts)

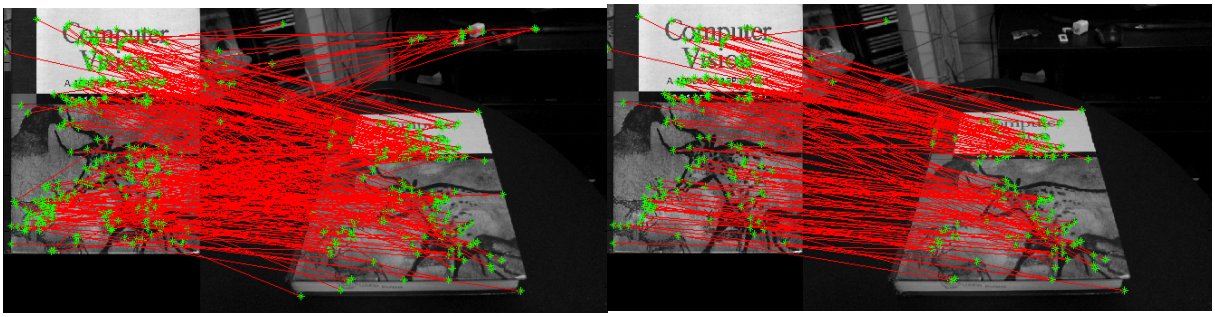
Write a script HarryPotterize to stick HarryPotter picture to CV books cover

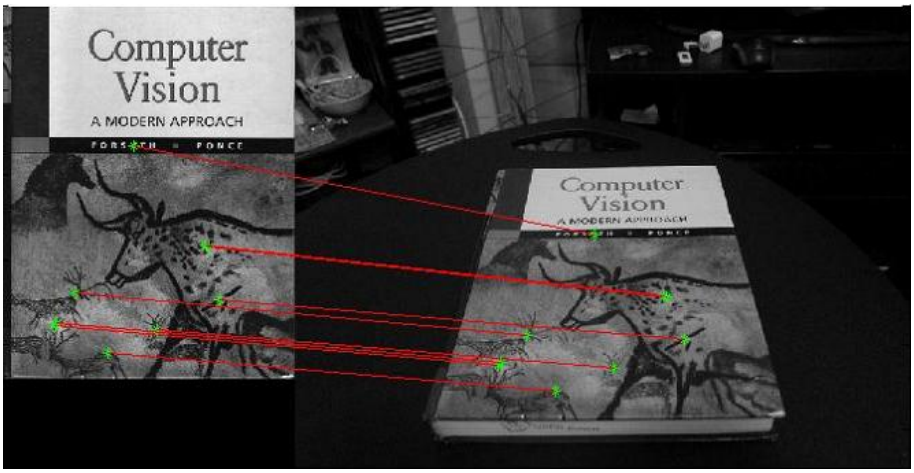
This case I will give my script covering all processes above.
Before using Ransac to get *H*, I preprocess each case through different approaches from three aspects: ratio, distance between interesting points, slopes between interesting points.

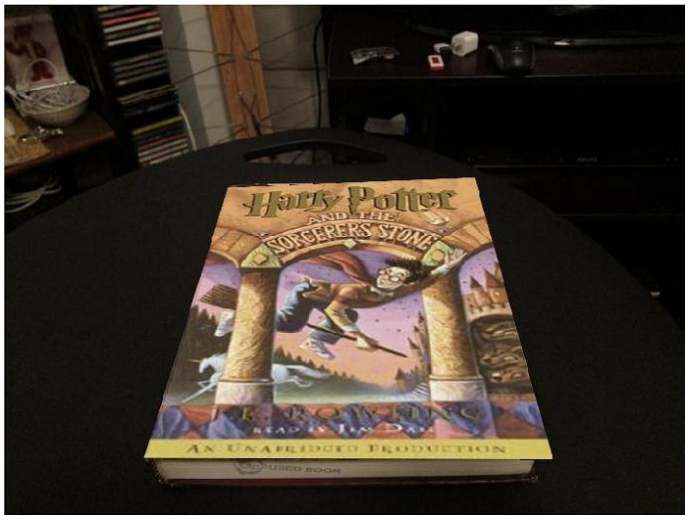
Case 1. Desk

Using slopes selection between [0, 1.5], the incorrect matching points will be eliminated.

```
points_left_pic=locs1(matches(:,1),2:-1:1)';        % into [x, y]  locs1
points_right_pic=locs2(matches(:,2),2:-1:1)';       % locs2
k_feature_all=(points_right_pic(2,:)-points_left_pic(2,:))./(points_right_p
ic(1,:)-points_left_pic(1,:));
matches=matches(find(k_feature_all>0 & k_feature_all<1.5),:);   % desk
```



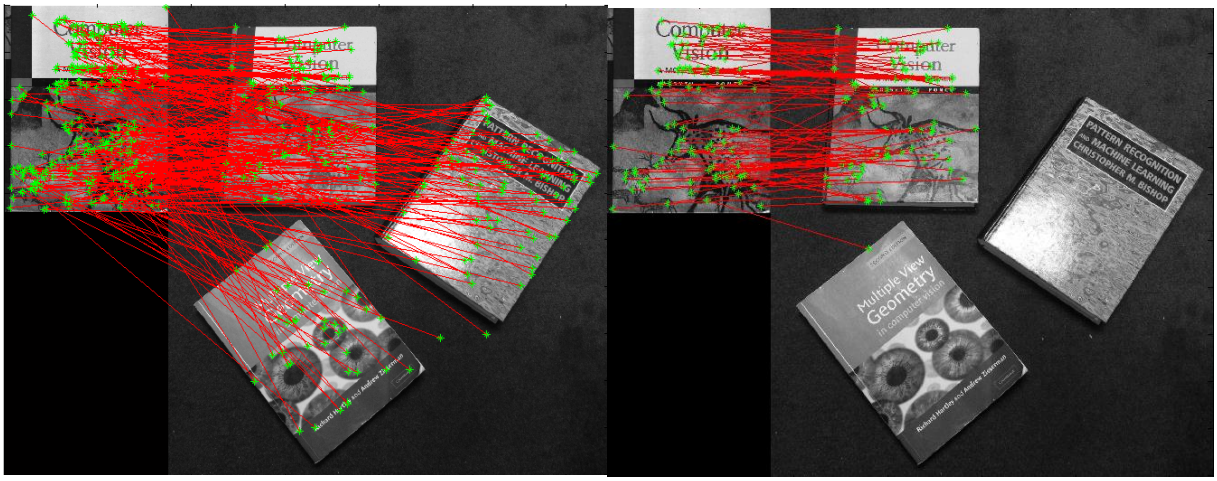


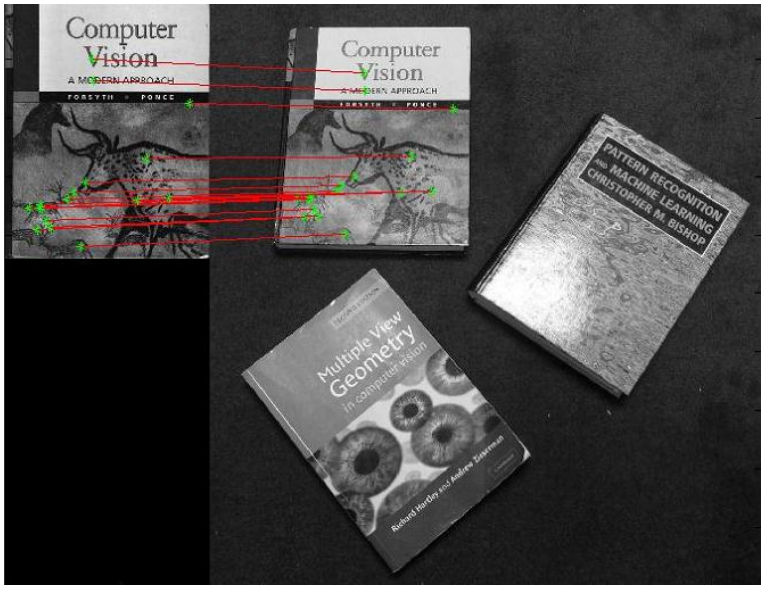


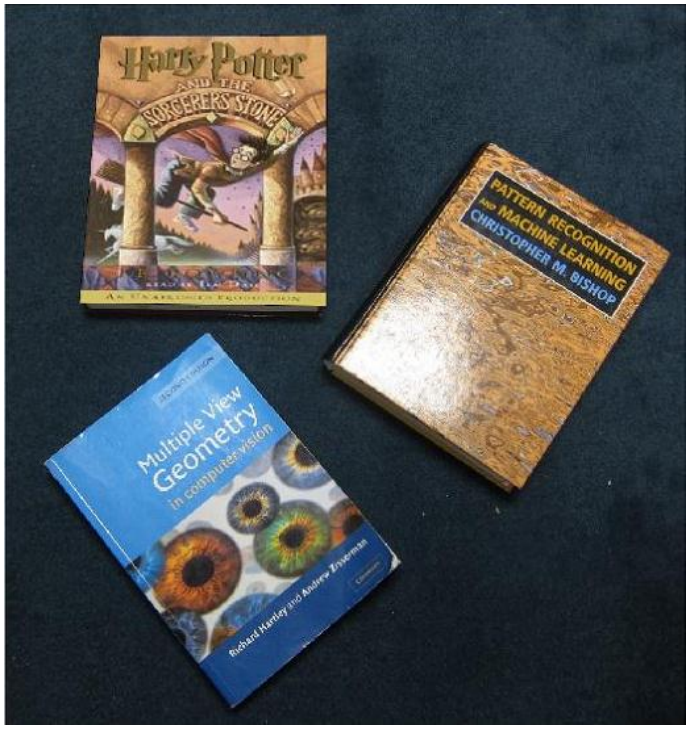
Case 2. Floor

Using geometric distance (smaller than 200) between matching points, the incorrect matching points will be eliminated.

```
points_left_pic=locs1(matches(:,1),2:-1:1)';       % into [x, y]  locs1
points_right_pic=locs2(matches(:,2),2:-1:1)';      % locs2
distance=sqrt( (points_right_pic(2,:)-points_left_pic(2,:)).^2+(points_righ
t_pic(1,:)-points_left_pic(1,:)).^2);
% matches=matches(find(k_feature_all>0 & k_feature_all<1.5),:);   % desk
matches=matches(find(distance<200),:);     % floor
```



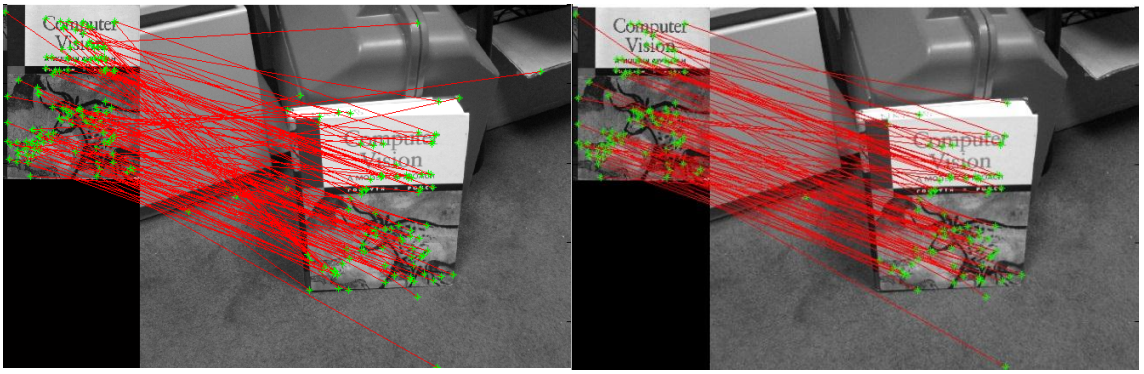

Computer Vision
A MODERN APPROACH
FORSYTH • PONCE
Computer Vision
PATTERN RECOGNITION AND MACHINE LEARNING
CHRISTOPHER M. BISHOP
Multiple View Geometry
in computer vision
Richard Hartley and Andrew Zisserman

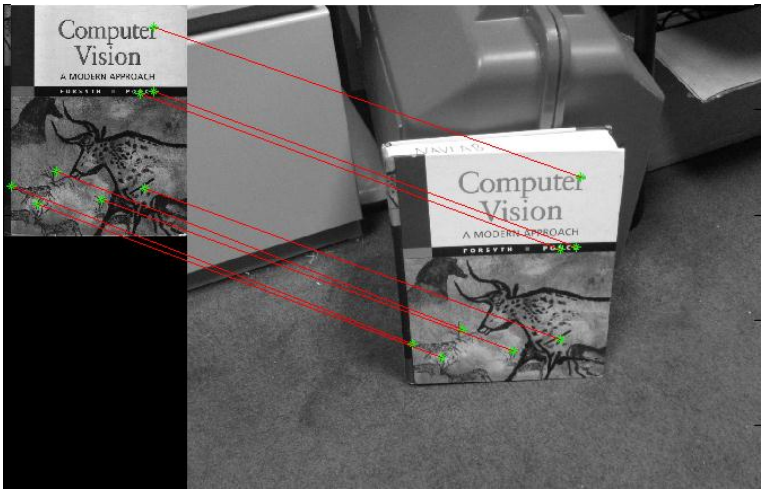

Harry Potter
AND THE
SORCERER'S STONE
PATTERN RECOGNITION
AND MACHINE LEARNING
CHRISTOPHER M. BISHOP
Multiple View
Geometry
in computer vision
Richard Hartley and Andrew Zisserman

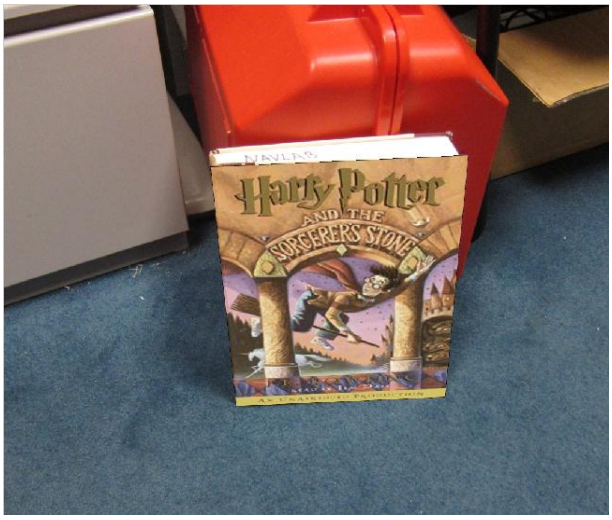
Case 3. Stand

Decreasing ratio=0.7, and using slope selection between [0 , 1.1] to eliminate incorrect points.

```
points_left_pic=locs1(matches(:,1),2:-1:1)';        % into [x, y]  locs1
points_right_pic=locs2(matches(:,2),2:-1:1)';       % locs2
k_feature_all=(points_right_pic(2,:)-points_left_pic(2,:))./(points_right_p
ic(1,:)-points_left_pic(1,:));
matches=matches(find(k_feature_all>0 & k_feature_all<1.1),:);
```







```
tic;
format long;
%% initial inputs
% im1=im2double(rgb2gray(imread('model_chickenbroth.jpg')));
% im2=im2double(rgb2gray(imread('chickenbroth_01.jpg')));
im1=im2double(imread('pf_scan_scaled.jpg'));
im2=im2double(rgb2gray(imread('pf_desk.jpg')));
sigma0=1;
k=sqrt(2);
th_contrast=0.03;
th_r=12;
levels=[-1,0,1,2,3,4];
ratio=0.74;

% save parameters sigma0 k levels th_contrast th_r;
%% Q1.2 get extremas
[locs1, GaussianPyramid] = DoGdetector(im1, sigma0, k, levels,th_contrast,
th_r);
[locs2, GaussianPyramid] = DoGdetector(im2, sigma0, k, levels,th_contrast,
th_r);
% imshow(im2);
% hold on;
% plot(locs2(2,:), locs2(1,:), 'o', 'MarkerEdgeColor', 'g', 'MarkerFaceColor',
'g', 'MarkerSize', 5);

%% Q2.3 change to 256bit expression
[locs1, desc1] = brief(im1);
[locs2, desc2] = brief(im2);

%% Q2.4 get matching points index
% [matches] = briefMatch(desc1, desc2, ratio);
% [matches] = briefMatchfrom1to2(desc1, desc2, ratio);
[matches] = briefMatch_easyversion(desc1, desc2, ratio);
%% plot matching result
plotMatches(im1, im2, matches, locs1, locs2);
%%  Q3.1 get the bestH
% preprocess
points_left_pic=locs1(matches(:,1),2:-1:1)';       % into [x, y]  locs1
points_right_pic=locs2(matches(:,2),2:-1:1)';      % locs2
k_feature_all=(points_right_pic(2,:)-points_left_pic(2,:))./(points_right_p
ic(1,:)-points_left_pic(1,:));
matches=matches(find(k_feature_all>0 & k_feature_all<1.5),:);
figure(2);
plotMatches(im1, im2, matches, locs1, locs2);
```

```
[bestH2to1, bestError, inliers,num_correct]=ransacH2to1(matches, locs1,
locs2);
%%  Q3.2 melting
im_book=im2double(imread('pf_scan_scaled.jpg'));
im_desk=im2double(imread('pf_desk.jpg'));
im_harry=im2double(imread('harrypotter.jpg'));
im_harry_enlarged=imresize(im_harry,size(im_book));
im_harry_enlarged_transform= warpH(im_harry_enlarged, bestH2to1,
size(im_desk), 0);
index_content=find(im_harry_enlarged_transform~=0);
im_desk(index_content)=0;
im_melt=im_desk+im_harry_enlarged_transform;
figure(3);
imshow(im_melt);
%%  show the inliners
figure(4);
plotMatches(im1, im2, matches(find(inliers==1),:), locs1, locs2);
toc;
```

[Notes] When I process the code, not all the time the results run well, because although applied with Ransack sampling approach, relative small number of iteration may sometimes get bad luck. However, after preprocessing section, the probability of getting satisfying result increases a lot, almost always right for case1, and 5/6 for *Stand* case. The ***besterror*** in the code is from sum all the matching points' geometric distances. In my experiment, in most cases after preprocessing, the number of matching points are controlled around 100. Because the criterion of judging mapping correctly is:

$$x' = H * x,\ x' = x_2$$

Sometimes due to *round* function, the number is not very large, besides, although the *H* is best, left pair of matching points will get large distances.

3.3 Theory

Suppose you wished to recover the pose of the camera from the homography matrix. Assume there is a plane with a known equation in 3D space.

3.3.1

Write the relation between Pi and the imaged points in our target image, (ui; vi; 1)T , in terms of the camera matrix K, and the camera rotation R and translation t. Use homogeneous coordinates and projective equivalence.

P_i is described by 4×1 vector in homogeneous coordinate. And the p_i is the corresponding point in image plane.

$$p_i = MP_i \qquad P_i = \begin{pmatrix} P_i' \\ 1 \end{pmatrix}_{4*1} \quad p_i = \begin{pmatrix} u_i \\ v_i \\ 1 \end{pmatrix} \quad M = K[\,R\,|t]$$

The plane in which P_i stands can be described as

$$\pi = \begin{pmatrix} \hat{n}_x \\ \hat{n}_y \\ \hat{n}_z \\ d \end{pmatrix} = \begin{pmatrix} \hat{n}_\pi \\ 1 \end{pmatrix}_{4*1} \quad \text{and } \| \hat{n}_\pi \| = 1$$

$$\pi^T P_i = 0 \rightarrow \ \widehat{n_\pi}^T P_i' + d = 0$$

$$\frac{-\hat{n}_\pi^{\ T} P_i'}{d} = 1$$

$$P_i = \begin{pmatrix} P_i' \\ 1 \end{pmatrix} = \begin{pmatrix} P_i' \\ \frac{-\widehat{n_\pi}^T P_i'}{d} \end{pmatrix} = \begin{pmatrix} I_{3*3} \\ \frac{-\widehat{n_\pi}^T}{d} \end{pmatrix}_{4*3} P_i'$$

$$p_i = MP_i = K[\,R\,|t] \begin{pmatrix} I_{3*3} \\ \frac{-\widehat{n_\pi}^T}{d} \end{pmatrix}_{4*3} P_i' = [KR|Kt] \begin{pmatrix} I_{3*3} \\ \frac{-\widehat{n_\pi}^T}{d} \end{pmatrix}_{4*3} P_i' = [KR - Kt\frac{\widehat{n_\pi}^T}{d}]_{3*3} P_i'$$

3.3.2

Write an expression for the columns of a matrix A such that H ≡KA. This should be in terms of the rows and/or columns of R and t and the plane equation.

A homography can map points between two images of a plane, or simply from one plane to another, as in this case. That is why there is only one camera matrix K.

$$p_1 = [KR_1 - Kt_1\frac{\widehat{n_\pi}^T}{d}]_{3*3} P_i' \quad p_2 = [KR_2 - Kt_2\frac{\widehat{n_\pi}^T}{d}]_{3*3} P_i'$$

$$H = \left[KR_1 - Kt_1\frac{\widehat{n_\pi}^T}{d}\right]_{3*3} \left[KR_2 - Kt_2\frac{\widehat{n_\pi}^T}{d}\right]_{3*3}^{-1} = KA$$

$$H = K\left[R_1 - t_1\frac{\widehat{n_\pi}^T}{d}\right] * [R_2 - t_2\frac{\widehat{n_\pi}^T}{d}]^{-1}K^{-1}$$

So, matrix A is

$$A = \left[R_1 - t_1\frac{\widehat{n_\pi}^T}{d}\right] * [R_2 - t_2\frac{\widehat{n_\pi}^T}{d}]^{-1}K^{-1}$$

3.3.3

Devise a way to recover R and t from H given K and the plane equation. There may be more than one solution that satisfies the relationship. Explain the ambiguities and how they arise.

We can select six pairs of points in the world coordinate system and image plane to calculate the transform matrix M. After we get M, based on the knowledge of K (intrinsic matrix,) according to M=K[R |t], we can get [R |t].

When the image plane and world plane are coplanar, then there will be more than one solution which satisfies the transformation. Because the solution of homography matrix H is not unique, the R and t calculated by H will have many results.

4. Extra

4.1

As we have seen, BRIEF is not rotationally invariant. Design a simple _x to solve this problem using the tools you have developed so far. Explain in your PDF your design decisions and how you selected any parameters that you use. Demonstrate the effectiveness of your algorithm on image pairs related by large rotation.

Both rotation and scaling will cause the incorrect results due to the spatial transformation causes the 256bits location losing correct information.

- Rotational invariant

To solve the rotational problem, we should use one invariant feature of image that is the gradient direction of each interesting points. When we random choosing 256 points surrounding interesting points, the coordinates of these 256 points should multiply a transforming matrix H. For each interesting point, there is a gradient direction, and before choosing points we should rotate the image towards this direction, so the information recorded is invariant. However, we need not to rotate the image each time, just use gradient direction angle to get a transformation matrix *H*. When we choose 256 points as sections before, then we multiply these points and the *H*. In turn, we can get the same effects.

Notice that, when computing the *H*, we need to firstly multiply a translating matrix to move the original (0,0) to interesting point(x_1,y_1), and then multiply a rotational matrix , then multiply a translating matrix to move the interesting point back to origin(x_0,y_0).

Due to many reasons, I just give the program to get the transformed coordinates, steps after haven't been proceeded.

- Scale invariant

To solve the scaling invariant, as the way to multiply H with points, the difference between is just around H. This time, we don't care the gradient direction but the ratio of scaling. We can find this ratio from the variance of the [x,y] coordinates of interesting points. If the object is scaled, then the 'x' of interesting points in scaled object will become proportionally larger than the variance in original object. We can also find the change in 'y', then calculate the coefficient of x and y direction. Then the scaling coefficient c1 and c2 will know. As such, the H will be produced.

- Below are from the files 'testrotation.m', 'notransacH2to1.m', 'getdesc_rotational.m'

```
im1=im2double(imread('pf_scan_scaled.jpg'));
im2=im2double(rgb2gray(imread('pf_pile.jpg')));
load parameters.mat;
[locs1, GaussianPyramid] = DoGdetector(im1, sigma0, k, levels,th_contrast,
th_r);
[locs2, GaussianPyramid] = DoGdetector(im2, sigma0, k, levels,th_contrast,
th_r);
load testPattern1;
%%  im1 direction of gradient, go opposite direction in direction_gradient to
normalize
index_interesting_points=sub2ind(size(im1),locs1(1,:),locs1(2,:));
[Gmag,direction_gradient] = imgradient(im1);
%
direction_gradient(find(direction_gradient<0))=direction_gradient(find(dire
ction_gradient<0))+360;
angle_interesting_gradient = direction_gradient(index_interesting_points);
%%  im2 direction of gradient
index_interesting_points_2=sub2ind(size(im2),locs2(1,:),locs2(2,:));
[Gmag_2,direction_gradient_2] = imgradient(im2);
%
direction_gradient_2(find(direction_gradient_2<0))=direction_gradient_2(fin
d(direction_gradient_2<0))+360;
angle_interesting_gradient_2 =
direction_gradient_2(index_interesting_points_2);

%%  compute rotational H1
% Scalef = @(s)([ s 0 0; 0 s 0; 0 0 1]);
Transf = @(tx,ty)([1 0 tx; 0 1 ty; 0 0 1]);
Rotf = @(t)([cos(t) -sin(t) 0; sin(t) cos(t) 0; 0 0 1]);
tx = locs1(2,:);
ty = locs1(1,:);
for i=1:length(angle_interesting_gradient)
H1{i} =
Transf(tx(i),ty(i))*Rotf(-angle_interesting_gradient(i)*pi/180)*Transf(-tx(
i),-ty(i));
end
%%  compute rotational H2
tx = locs2(2,:);
ty = locs2(1,:);
for i=1:length(angle_interesting_gradient_2)
H2{i} =
Transf(tx(i),ty(i))*Rotf(-angle_interesting_gradient_2(i)*pi/180)*Transf(-t
x(i),--ty(i));
```

```
end

%%  get locs and desc and matches
ratio=0.8;
[locs1,desc1]= getdesc_rotational(im1,locs1,H1,compareX,compareY);
[locs2,desc2]= getdesc_rotational(im2,locs2,H2,compareX,compareY);
[matches] = briefMatch_easyversion(desc1, desc2, ratio);
figure(2);
plotMatches(im1, im2, matches, locs1, locs2);

%% preprocess
points_left_pic=locs1(matches(:,1),2:-1:1)';       % into [x, y]  locs1
points_right_pic=locs2(matches(:,2),2:-1:1)';      % locs2
k_feature_all=(points_right_pic(2,:)-points_left_pic(2,:))./(points_right_p
ic(1,:)-points_left_pic(1,:));
distance=sqrt( (points_right_pic(2,:)-points_left_pic(2,:)).^2+(points_righ
t_pic(1,:)-points_left_pic(1,:)).^2);
index_preprocess=unique([find(k_feature_all>0),find(distance<300)]);
% index_preprocess=find(k_feature_all>0);
matches(index_preprocess,:)=[];  % desk
% matches=matches(find(distance>200),:);     % floor
figure(3);
plotMatches(im1, im2, matches, locs1, locs2);
[bestH2to1, bestError, inliers,num_correct]=notransacH2to1(matches, locs1,
locs2);
%%  output result, I haven't changed the variables names, but if the H is right,
it works!
im_book=im2double(imread('pf_scan_scaled.jpg'));
im_desk=im2double(imread('pf_pile.jpg'));
im_harry=im2double(imread('harrypotter.jpg'));
im_harry_enlarged=imresize(im_harry,size(im_book));
im_harry_enlarged_transform= warpH(im_harry_enlarged, bestH2to1,
size(im_desk), 0);
index_content=find(im_harry_enlarged_transform~=0);
im_desk(index_content)=0;
im_melt=im_desk+im_harry_enlarged_transform;
figure(4);
imshow(im_melt);
```

```
function [locs,desc]= getdesc_rotational(im,locs,H,compareX,compareY)
[height,width]=size(im);       % height:row, width: col
[compareX_row,compareX_col]=ind2sub([9,9],compareX);
[compareY_row,compareY_col]=ind2sub([9,9],compareY);
compareX=[compareX_row-5,compareX_col-5];
```

```
compareY=[compareY_row-5,compareY_col-5];

%% get absolute [row,col]
points_interest=locs';
m=length(points_interest(:,1));     %   number of valid interesting points
%%  outputs absolute coordinates interesting points
index_center_row=repmat(points_interest(:,1),1,256);
index_center_col=repmat(points_interest(:,2),1,256);
%   output X group
index_compareX_row=zeros(1,256);
index_compareX_col=zeros(1,256);
compareX=compareX';
index_compareX_row=compareX(1,:);
index_compareX_col=compareX(2,:);
index_compareX_row=repmat(index_compareX_row,m,1);
index_compareX_col=repmat(index_compareX_col,m,1);
index_points_X_row=index_compareX_row+index_center_row;
index_points_X_col=index_compareX_col+index_center_col;
index_points_X_row_line=reshape(index_points_X_row',1,256*size(index_points
_X_row,1));
index_points_X_col_line=reshape(index_points_X_col',1,256*size(index_points
_X_row,1));
X_homogeneous=[index_points_X_col_line;index_points_X_row_line];
X_homogeneous(3,:)=1;
i=1;
while(i<=m)     % num of interesting points
    for j=1:256
    X_transformed(:,(i-1)*256+j)=  H{i}*X_homogeneous(:,(i-1)*256+j);
    end
i=i+1;
end

%%  output Y group
index_compareY_row=zeros(1,256);
index_compareY_col=zeros(1,256);
compareY=compareY';
index_compareY_row=compareY(1,:);
index_compareY_col=compareY(2,:);
index_compareY_row=repmat(index_compareY_row,m,1);
index_compareY_col=repmat(index_compareY_col,m,1);
index_points_Y_row=index_compareY_row+index_center_row;
index_points_Y_col=index_compareY_col+index_center_col;
index_points_Y_row_line=reshape(index_points_Y_row',1,256*size(index_points
_Y_row,1));
```

```
index_points_Y_col_line=reshape(index_points_Y_col',1,256*size(index_points
_Y_row,1));
Y_homogeneous=[index_points_Y_col_line;index_points_Y_row_line];
Y_homogeneous(3,:)=1;
i=1;
while(i<=m)     % num of interesting points
    for j=1:256
    Y_transformed(:,(i-1)*256+j)=  H{i}*Y_homogeneous(:,(i-1)*256+j);
    end
i=i+1;
end
X_homogeneous_row=round(reshape(X_transformed(2,:),256,m)');
X_homogeneous_col=round(reshape(X_transformed(1,:),256,m)');
Y_homogeneous_row=round(reshape(Y_transformed(2,:),256,m)');
Y_homogeneous_col=round(reshape(Y_transformed(1,:),256,m)');

%%  wipe out the outside points and corresponding index in locs
%%  X group
outside_row_min_x=min(X_homogeneous_row,[],2);    % find part<1
outside_row_max_x=max(X_homogeneous_row,[],2);    % find part>height
outside_col_min_x=min(X_homogeneous_col,[],2);    % find part<1
outside_col_max_x=max(X_homogeneous_col,[],2);    % find part>width
index_outside_x=find(outside_row_min_x<1 | outside_row_max_x>height |
outside_col_min_x<1 | outside_col_max_x>width );
%%  Y group
outside_row_min_y=min(Y_homogeneous_row,[],2);    % find part<1
outside_row_max_y=max(Y_homogeneous_row,[],2);    % find part>height
outside_col_min_y=min(Y_homogeneous_col,[],2);    % find part<1
outside_col_max_y=max(Y_homogeneous_col,[],2);    % find part>width
index_outside_y=find(outside_row_min_y<1 | outside_row_max_y>height |
outside_col_min_y<1 | outside_col_max_y>width );
%%  combine the unvalid index
index_outside=unique([index_outside_x;index_outside_y]);
X_homogeneous_row(index_outside,:)=[];
X_homogeneous_col(index_outside,:)=[];
Y_homogeneous_row(index_outside,:)=[];
Y_homogeneous_col(index_outside,:)=[];
index_X_group=sub2ind(size(im),X_homogeneous_row,X_homogeneous_col);
index_Y_group=sub2ind(size(im),Y_homogeneous_row,Y_homogeneous_col);
length_points=size(index_X_group,1);
desc=zeros(length_points,256);
points_difference=im(index_Y_group)-im(index_X_group);
desc(find(points_difference>0))=1;
locs=locs';
```

```
locs(index_outside,:)=[];
end
```

```
function [bestH2to1, bestError, inliers,max_number]=notransacH2to1(matches,
locs1, locs2)
inliers=zeros(1,size(matches,1));
%%  produce homography coordinates
points_left_pic=locs1(matches(:,1),2:-1:1)';       % into [x, y]  locs1
points_right_pic=locs2(matches(:,2),2:-1:1)';      % locs2

%%  initiate the parameters
i=1;
num_iteration=10000;
p=0.99;
max_number=0;
while(i<num_iteration)
    num_mode=3;num_mode_r=3;
    while ( (num_mode==3 | num_mode==6) | ( num_mode_r==3 | num_mode_r==6) )
    %%  randperm 4 points
    index_rand_four_points=randperm(size(points_left_pic,2),4);
    points_left_four=points_left_pic(:,index_rand_four_points);
    points_right_four=points_right_pic(:,index_rand_four_points);

    %%  whether three points colinear

k_1_234=(points_left_four(2,2:4)-points_left_four(2,1))./(points_left_four(
1,2:4)-points_left_four(1,1));

k_2_34=(points_left_four(2,3:4)-points_left_four(2,2))./(points_left_four(1
,3:4)-points_left_four(1,2));

k_3_4=(points_left_four(2,4)-points_left_four(2,3))./(points_left_four(1,4)
-points_left_four(1,3));
    k_all=[k_1_234,k_2_34,k_3_4];

    %   right points also not colinear

k_1_234_r=(points_right_four(2,2:4)-points_right_four(2,1))./(points_right_
four(1,2:4)-points_right_four(1,1));

k_2_34_r=(points_right_four(2,3:4)-points_right_four(2,2))./(points_right_f
our(1,3:4)-points_right_four(1,2));

k_3_4_r=(points_right_four(2,4)-points_right_four(2,3))./(points_right_four
```

```
(1,4)-points_right_four(1,3));
    k_all_r=[k_1_234_r,k_2_34_r,k_3_4_r];
    [useless, num_mode]=mode(k_all);
    [useless, num_mode_r]=mode(k_all_r);
    check_nan=isnan(k_all);
    check_nan_2=isnan(k_all_r);
    if ( length(find(check_nan==1))~=0 | length(find(check_nan_2==1))~=0 )
        num_mode=3;num_mode_r=3;
    end
    end
%%  get the  H
    [H,A] = computeH_norm( points_left_four,points_right_four);

%%  calculate the right matching numbers
    points_left_pic(3,:)=1;
    points_right_pic(3,:)=1;
    points_left_pic_transform=H*points_left_pic;

points_left_pic_transform(1:2,:)=round(points_left_pic_transform(1:2,:)./[p
oints_left_pic_transform(3,:);points_left_pic_transform(3,:)]);

%%  calculate the distance
    points_difference=points_left_pic_transform-points_right_pic;
    index_correct_mapping=find(points_difference(1,:)==0 &
points_difference(2,:)==0);
    num_correct(i)=length(index_correct_mapping);

%%  iterate the number of loops
    if num_correct(i)>max_number
%         num_iteration= log(1-p)/log(1-(num_correct(i)/size(matches, 1))^4);
        bestH2to1=H;
        max_number= num_correct(i);
        index_correct_mapping_result=index_correct_mapping;

bestError=sum(sqrt( (points_left_pic_transform(1,:)-points_right_pic(1,:)).
^2+(points_left_pic_transform(2,:)-points_right_pic(2,:)).^2 ));
    end
    points_right_pic(3,:)=[];
    points_left_pic(3,:)=[];
    i=i+1;
end
inliers(index_correct_mapping_result)=1;

end
```

Reference

[1] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded up robust features. In In ECCV, pages 404{417, 2006.

[2] Peter J. Burt, Edward, and Edward H. Adelson. The laplacian pyramid as a compact image code. IEEE Transactions on Communications, 31:532{540, 1983.

[3] M. Calonder, V. Lepetit, M. Ozuysal, T. Trzcinski, C. Strecha, and P. Fua. BRIEF: Computing a Local Binary Descriptor Very Fast. IEEE Transactions on Pattern Analysis and Machine Intelligence, 34(7):1281{1298, 2012.

[4] Michael Calonder, Vincent Lepetit, and Pascal Fua. Brief: Binary robust indepen- dent elementary features, 2010.

[5] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model _tting with applications to image analysis and automated cartography. Commun. ACM, 24(6):381{395, June 1981.

[6] David G. Lowe. Distinctive image features from scale-invariant keypoints. Int. J. Comput. Vision, 60(2):91{110, November 2004.