Task7: Image Stitching & RANSAC

Name: Saeed Hatefi Ardakani
Degree: PH
ID: 20793159

Problem 1: LoG and DoG (10 points)

Please answer these questions in detail.

(a) What is the Laplacian of Gaussian (LoG)? When do we use LoG?

The Laplacian of Gaussian (LoG) is a second-order derivative filter of an image. If we want to discuss about some properties of the Laplacian, we can say that

- Laplacian gives better edge localization as compared to first-order.
- The edge orientation information is lost in Laplacian.
- Unlike first-order, Laplacian is an isotropic filter i.e. it produces a uniform edge magnitude for all directions.
- Similar to first-order, Laplacian is also very sensitive to noise.

Therefore, to reduce the noise effect, image is first smoothed (blurred) with a Gaussian filter and then we find the zero crossings using Laplacian. This two-step process is called the Laplacian of Gaussian (LoG) operation. It should be mentioned that this can also be performed in one step. Instead of first smoothing an image with a Gaussian kernel and then taking its Laplace, we can obtain the Laplacian of the Gaussian kernel and then convolve it with the image.

In addition, the Laplacian of an image highlights regions of rapid intensity change. Therefore, the Laplacian of Gaussian is useful for detecting edges that appear at various image scales or degrees of image focus.

(b) What is a difference of the Gaussian (DoG)? When do we use and what is DoG advantageous compared to LoG?

Difference of Gaussian (DoG) is an approximation of LoG. DoG is a feature enhancement algorithm that involves the subtraction of one blurred version of an original image from another, less blurred version of the original. In fact, the DOG performs edge detection by applying a Gaussian blur on an image at a certain standard deviation. Then, the amount of standard deviation will be changed and another Gaussian blur of the image will be considered. Finally, it is calculated the difference between the two images. Therefore, because it removes high-frequency spatial detail that can include random noise, the difference of gaussian algorithm is useful for enhancing edges in noisy digital images.

In fact, we can show that LoG is not separable while for DoG, we can use the Gaussian kernel's separability, accelerating the computational process and reducing computational costs in two or more dimensions. The separability of the Gaussian kernel means that the kernel can be represented as the multiplication of two vectors. So, one of the advantages of DoG is reducing the computational costs in comparison to the LoG.

Problem 2: Least Squares (20 points)

(a) Explain the approach 1 and approach 2 for least squares line fitting in your words. Please refer the course slide and tutorials.

Linear least squares (LLS) is the least squares approximation of linear functions to data. The least squares criterion is determined by minimizing the sum of squares created by a mathematical function. In fact, A square, as an error, is determined by squaring the distance between a data point and the regression line or mean value of the data set.

The approaches 1 and 2 are the same from the mathematical point of view. But, the difference between approach 1 and approach 2 is that the approach 1 is based on the summation notation while the approach 2 is based on the matrix notation.

Approach 1:

According to the approach 1, the square of error is calculated by:

$$E = \sum_{i=1}^{n} (y_i - mx_i - b)^2$$

As you can see, the error parameter *E* is defined by the summation of the square of the difference between *y* values of the data and their estimated values by the linear model. It should be mentioned that by using the square of the difference, we change the negative differences to the positive one to consider their effect on the total error.

After defining the error, we need to minimize the error in terms of *m* and *b*. So, we take the partial derivatives of the error respect to *m* and *b*:

$$\frac{\partial(E)}{\partial m} = -2 \sum_{i=1}^{n} (y_i - mx_i - b) x_i = 0$$

$$\frac{\partial(E)}{\partial b} = -2 \sum_{i=1}^{n} (y_i - mx_i - b) = 0$$

Finally, we can calculate the values of m and b by solving the system of equations as:

$$m = \frac{\sum_{i=1}^{n} x_i y_i - 1/n(\sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i)}{\sum_{i=1}^{n} {x_i}^2 - 1/n(\sum_{i=1}^{n} x_i)^2}$$

$$b = \frac{1/n(\sum_{i=1}^{n} y_i)(\sum_{i=1}^{n} x_i^2) - 1/n \sum_{i=1}^{n} x_i \sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} {x_i}^2 - 1/n(\sum_{i=1}^{n} x_i)^2}$$

Approach 2:

According to the approach 2, the square is defined as the Euclidian norm of the distance vector:

$$E = ||Y - XB||^2 = (Y - XB)^T (Y - XB) = Y^T Y - 2(XB)^T Y + (XB)^T (XB)$$

where

$$Y = \begin{bmatrix} y_1 \\ . \\ . \\ . \\ y_n \end{bmatrix}$$

$$X = \begin{bmatrix} x_1 & 1 \\ . & . \\ . & . \\ . & . \\ x_n & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} m \\ b \end{bmatrix}$$

After defining the error, we need to minimize the error respect to vector *B*. So, we take the partial derivative of the vector error *E* respect to *B*:

$$\frac{\mathrm{d}E}{\mathrm{d}B} = 2X^T XB - 2X^T Y = 0 \Rightarrow X^T XB = X^T Y$$

So, we have

$$B = (X^T X)^{-1} X^T Y$$

A response yi is measured from a linear system when an input is xi. The measurement data is provided (`prob2_data1.mat`). A model of your system can be approximated as yi = mxi + b. Please find m and b using the following methods.

(b) Least squares (approach 1)

(c) Least squares (approach 2)

We find *m* and *b* using the approaches 1 and 2:

m = 3.6540

b = 3.3197

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%---------- Problem 2 ----------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%% (b) Least squares (approach 1) %%%%%%%%%%
clc;
clear all;
close all;

load('prob2_data1.mat');
n = length(x);
sum_x = sum(x);
sum_y = sum(y);
sum_xy = sum(x.*y);
sumxx = sum(x.*x);

m = (sum_xy-(1/n)*(sum_x*sum_y))/(sumxx-(1/n)*(sum_x)^2);
```

```
b = ((1/n)*sum_y*sumxx-(1/n)*(sum_x*sum_xy))/(sumxx-(1/n)*(sum_x)^2);


%%%%%%%%%%% (c) Least squares (approach 2) %%%%%%%%%%
X = [x' ones(n,1)];
B = (X'*X)^-1*X'*y'; % B = [m b]
```

Here is another measurement data (`prob2_data2.mat`). Please find m and b using the following methods.

(d) Least squares (either approach 1 or approach 2)

We find *m* and *b* using the approaches 1 and 2:

m = 3.3348

b = 2.6458

```
%%%%%%%%%% (d) Least squares (either approach 1 or approach 2) %%%%%%%%%%
clc;
clear all;
close all;

load('prob2_data2.mat');
% approach 1
n = length(x);
sum_x = sum(x);
sum_y = sum(y);
sum_xy = sum(x.*y);
sumxx = sum(x.*x);

m = (sum_xy-(1/n)*(sum_x*sum_y))/(sumxx-(1/n)*(sum_x)^2);
b = ((1/n)*sum_y*sumxx-(1/n)*(sum_x*sum_xy))/(sumxx-(1/n)*(sum_x)^2);

% approach 2
X = [x' ones(n,1)];
B = (X'*X)^-1*X'*y'; % B = [m b]
```

(e) Use of RANSAC. You need to have your own RANSAC implementation without using existing functions in MATLAB (Do not use `ransac` or any other relevant functions)

We find *m* and *b* using RANSAC:

for Prob2-data1.mat:

m = 3.6544

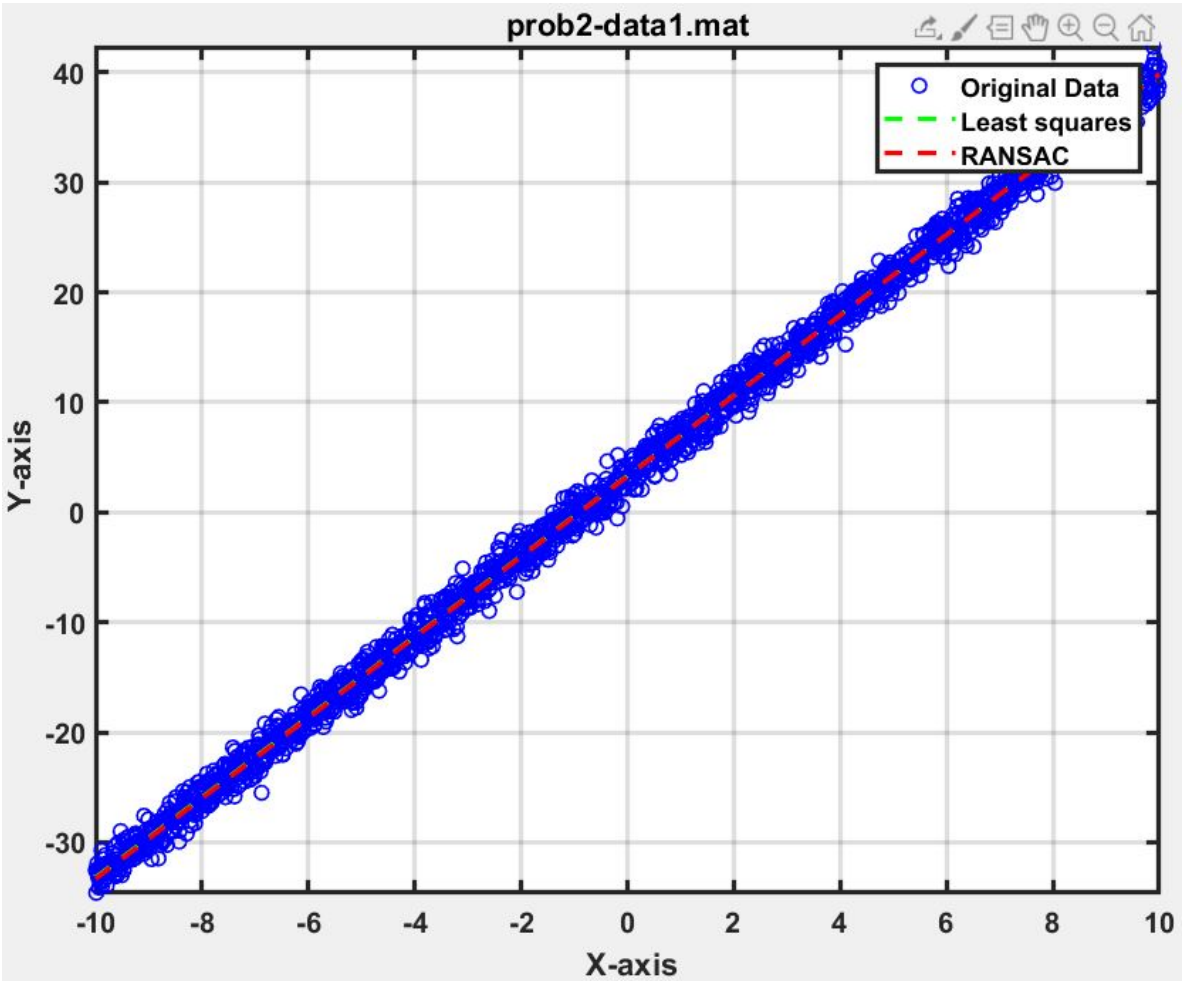
b = 3.2363

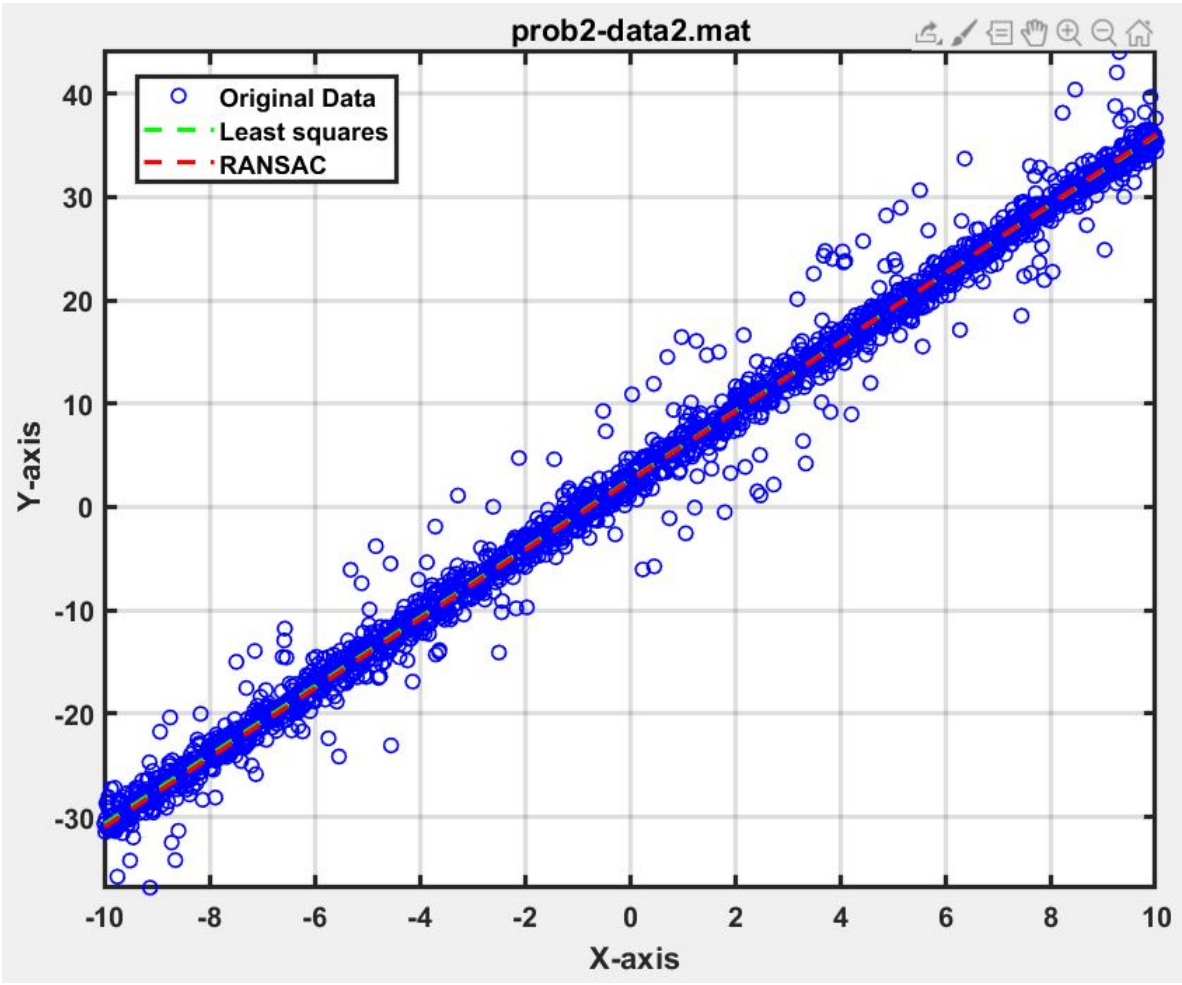
for Prob2-data2.mat:

m = 3.3454

b = 2.5982

In the below figure, we have plotted the original data and the lines related to the least square method and RANSAC for `prob2_data1.mat` and `prob2_data2.mat`:


prob2-data1.mat
Original Data
Least squares
RANSAC
Y-axis
X-axis
40
30
20
10
0
-10
-20
-30
-10
-8
-6
-4
-2
0
2
4
6
8
10


prob2-data2.mat
Original Data
Least squares
RANSAC
Y-axis
X-axis
40
30
20
10
0
-10
-20
-30
-10
-8
-6
-4
-2
0
2
4
6
8
10

```
%%%%%%%%%%% (e) Use of RANSAC %%%%%%%%%%
clc;
clear all;
close all;
% -------- measurement data: prob2_data1.mat --------
load('prob2_data1.mat');

% Use of RANSAC
nData = numel(x);
xrange = [min(x) max(x)];
param_best = zeros(2,1);
max_n_trials = 1000;
num_inliner_best = 0;
dist_thr = 1;
for ii=1:max_n_trials
    pt_idx = randperm(nData,2);
    x_sub = x(pt_idx)';
    y_sub = y(pt_idx)';

    % https://en.wikipedia.org/wiki/Line_(geometry)
    mh = diff(y_sub)/diff(x_sub);
    bh = det([y_sub x_sub])/diff(x_sub);

    dist_pt = abs(mh*x+bh-y);

    num_inlier = sum(dist_pt<dist_thr);
    if num_inlier>num_inliner_best
        param_best(1) = mh;
        param_best(2) = bh;
        num_inliner_best = num_inlier;
    end
end

% Least squares
X = [x' ones(nData,1)];
B = (X'*X)^-1*X'*y'; % B = [m b]

figure(1);
plot(x, y, 'ob', 'linewidth', 1);hold on;
line(xrange,B(1)*xrange + B(2), 'Color', 'g', 'linewidth',  2, 'LineStyle', '--
');
line(xrange,param_best(1)*xrange + param_best(2), 'Color', 'r', 'linewidth',  2,
'LineStyle', '--');
title('prob2-data1.mat')
legend('Original Data', 'Least squares', 'RANSAC');
axis tight;grid on; hold off
xlabel('\bf X-axis');xlim(xrange);
ylabel('\bf Y-axis');
set(gca,'FontSize',12,'linewidth',2,'fontweight','bold')


% -------- measurement data: prob2_data2.mat --------
load('prob2_data2.mat');

% Use of RANSAC
```

```
nData = numel(x);
xrange = [min(x) max(x)];
param_best = zeros(2,1);
max_n_trials = 1000;
num_inliner_best = 0;
dist_thr = 1;
for ii=1:max_n_trials
    pt_idx = randperm(nData,2);
    x_sub = x(pt_idx)';
    y_sub = y(pt_idx)';

    % https://en.wikipedia.org/wiki/Line_(geometry)
    mh = diff(y_sub)/diff(x_sub);
    bh = det([y_sub x_sub])/diff(x_sub);

    dist_pt = abs(mh*x+bh-y);

    num_inlier = sum(dist_pt<dist_thr);
    if num_inlier>num_inliner_best
        param_best(1) = mh;
        param_best(2) = bh;
        num_inliner_best = num_inlier;
    end
end

% Least squares
X = [x' ones(nData,1)];
B = (X'*X)^-1*X'*y'; % B = [m b]

figure(2);
plot(x, y, 'ob', 'linewidth', 1);hold on;
line(xrange,B(1)*xrange + B(2), 'Color', 'g', 'linewidth',  2, 'LineStyle', '--
');
line(xrange,param_best(1)*xrange + param_best(2), 'Color', 'r', 'linewidth',  2,
'LineStyle', '--');
title('prob2-data2.mat')
legend('Original Data', 'Least squares', 'RANSAC');
axis tight;grid on; hold off
xlabel('\bf X-axis');xlim(xrange);
ylabel('\bf Y-axis');
set(gca,'FontSize',12,'linewidth',2,'fontweight','bold')
```

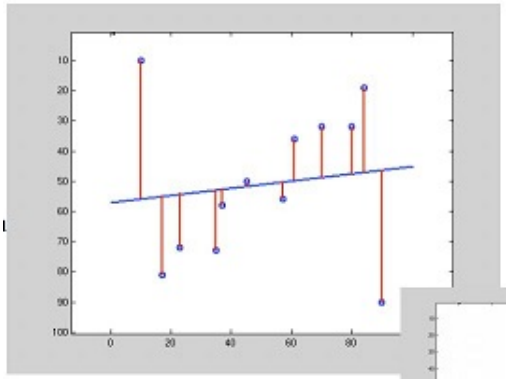
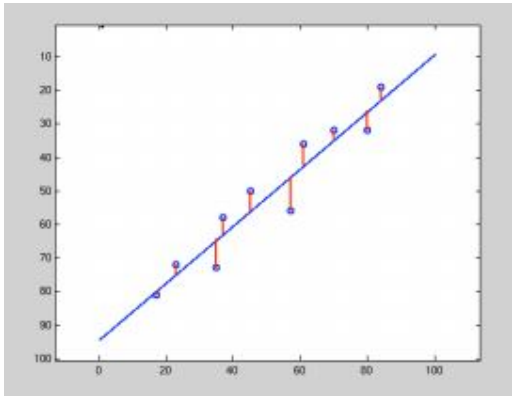
(f) When do we use either least squares or RANSAC? What are the pros and cons in each technique?

Advantages of least squares method:

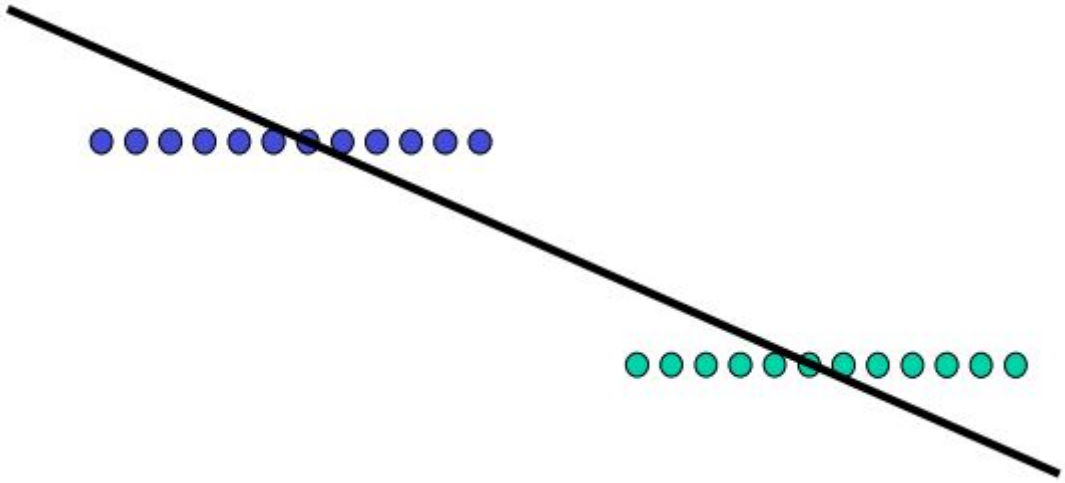
- Simplicity: It is very easy to explain and to understand
- Applicability: There are hardly any applications where least squares doesn't make sense

Disadvantages of least squares method:

- One of the limitation of Least squares method is for vertical lines. This method will fail for vertical lines.
- Sensitivity to outliers: Least squares estimation is sensitive to outliers, so that a few outliers can greatly skew the result.

example 1	example 2
	

- Test statistics might be unreliable when the data is not normally distributed (but with many data points that problem gets mitigated).
- Another problem about least squares is that multiple structures can also skew the results (the fit procedure implicitly assumes there is only one instance of the model in the data).



So, the solution for these disadvantages is estimation methods like RANSAC that are robust to outliers:

- Classify data points as outliers or inliers
- Fit model to inliers while ignoring outliers

An advantage of RANSAC is its ability to do robust estimation of the model parameters, i.e., it can estimate the parameters with a high degree of accuracy even when a significant number of outliers are present in the data set.

A disadvantage of RANSAC is that there is no upper bound on the time it takes to compute these parameters. When the number of iterations computed is limited, the solution obtained may not be optimal, and it may not even be one that fits the data in a good way.

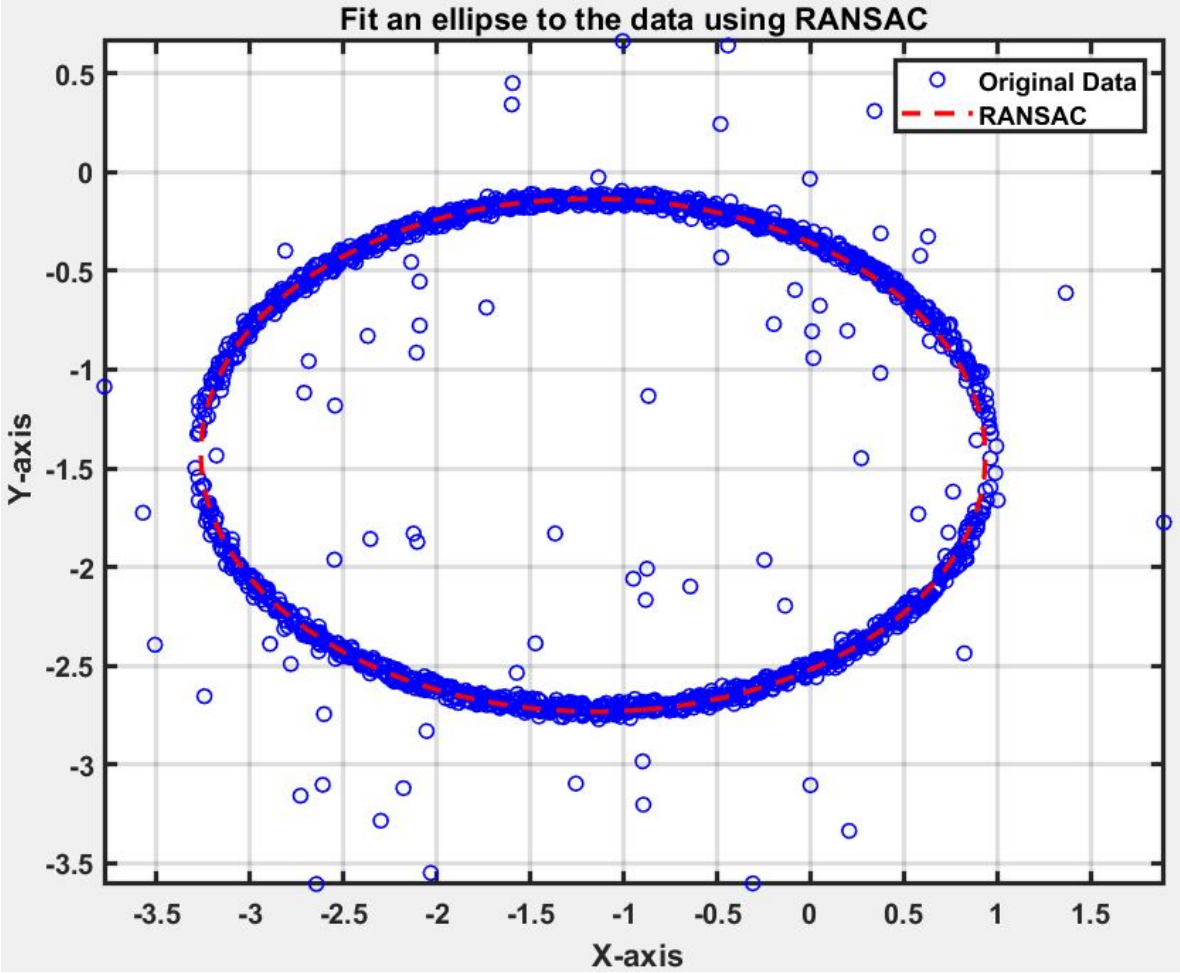
Also, RANSAC can only estimate one model for a particular data set. When two (or more) model instances exist, RANSAC may fail to find either one. The Hough transform is an alternative robust estimation technique that may be useful when more than one model instance is present.

Problem 3: Fitting using RANSAC (30 points)

Note that you need to write your own code for implementing RANSAC like Problem 2.

(a) Fit an ellipse to the given data (`prob3_ellipse.mat`) using RANSAC

In the below figure, the ellipse has been properly fitted to the given data using RANSAC.

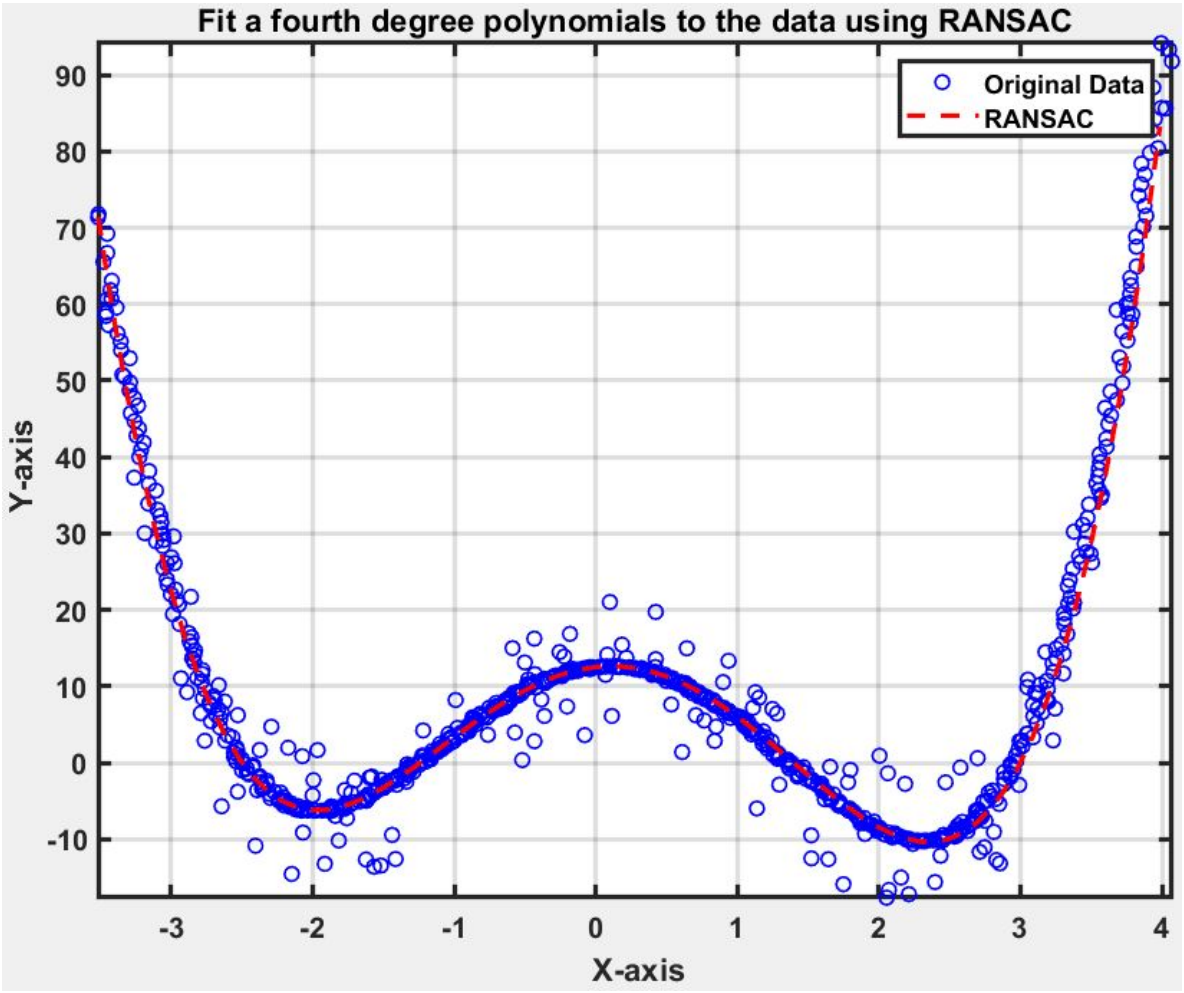



```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%---------- Problem 3 ----------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%% (a) Fit an ellipse to the given data using RANSAC %%%%%%%%%%
% the general equation for an ellipse is as follows:
% ax^2 + bxy + cy^2 + dx + ey + f = 0
clc;
clear all;
close all;

load('prob3_ellipse.mat');
nData = numel(x);
nSampLen = 5;
param_best = zeros(5,1);
max_n_trials = 1000;
num_inliner_best = 0;
dist_thr = 0.05;

for ii=1:max_n_trials

    pt_idx = ceil(nData .* rand(1, nSampLen));
    x_sub = x(pt_idx)';
    y_sub = y(pt_idx)';
```

```
    coef_matrix = [x_sub.^2 x_sub.*y_sub y_sub.^2 x_sub y_sub ones(5,1)];
    coef = null(coef_matrix); % calculating a, b, c, d, e, and f

    % finding outlier points
    % because it is hard to find the vertical distance of points to the
    % ellipse, we calculate abs(ax^2 + bxy + cy^2 + dx + ey + f)
    diff_pt = abs(coef(1)*x.^2 + coef(2)*x.*y + coef(3)*y.^2 + coef(4)*x +
coef(5)*y + coef(6));
    num_inlier = sum(diff_pt<dist_thr);

    if num_inlier>num_inliner_best
        param_best(1) = coef(1);
        param_best(2) = coef(2);
        param_best(3) = coef(3);
        param_best(4) = coef(4);
        param_best(5) = coef(5);
        param_best(6) = coef(6);
        num_inliner_best = num_inlier;
    end
end


figure(1);
plot(x, y, 'ob', 'linewidth', 1);hold on;
fimplicit(@(x,y) param_best(1)*x.^2 + param_best(2)*x.*y + param_best(3)*y.^2 +
param_best(4)*x + param_best(5)*y + param_best(6), 'r--','Linewidth',2);
title('Fit an ellipse to the data using RANSAC')
legend('Original Data', 'RANSAC');
axis tight;grid on; hold off
xlabel('\bf X-axis');
ylabel('\bf Y-axis');
set(gca,'FontSize',12,'linewidth',2,'fontweight','bold')
```

(b) Fit a fourth degree polynomials to the given data (`prob3_polynomial.mat`) using RANSAC

In the below figure, the fourth degree polynomials has been properly fitted to the given data using RANSAC.

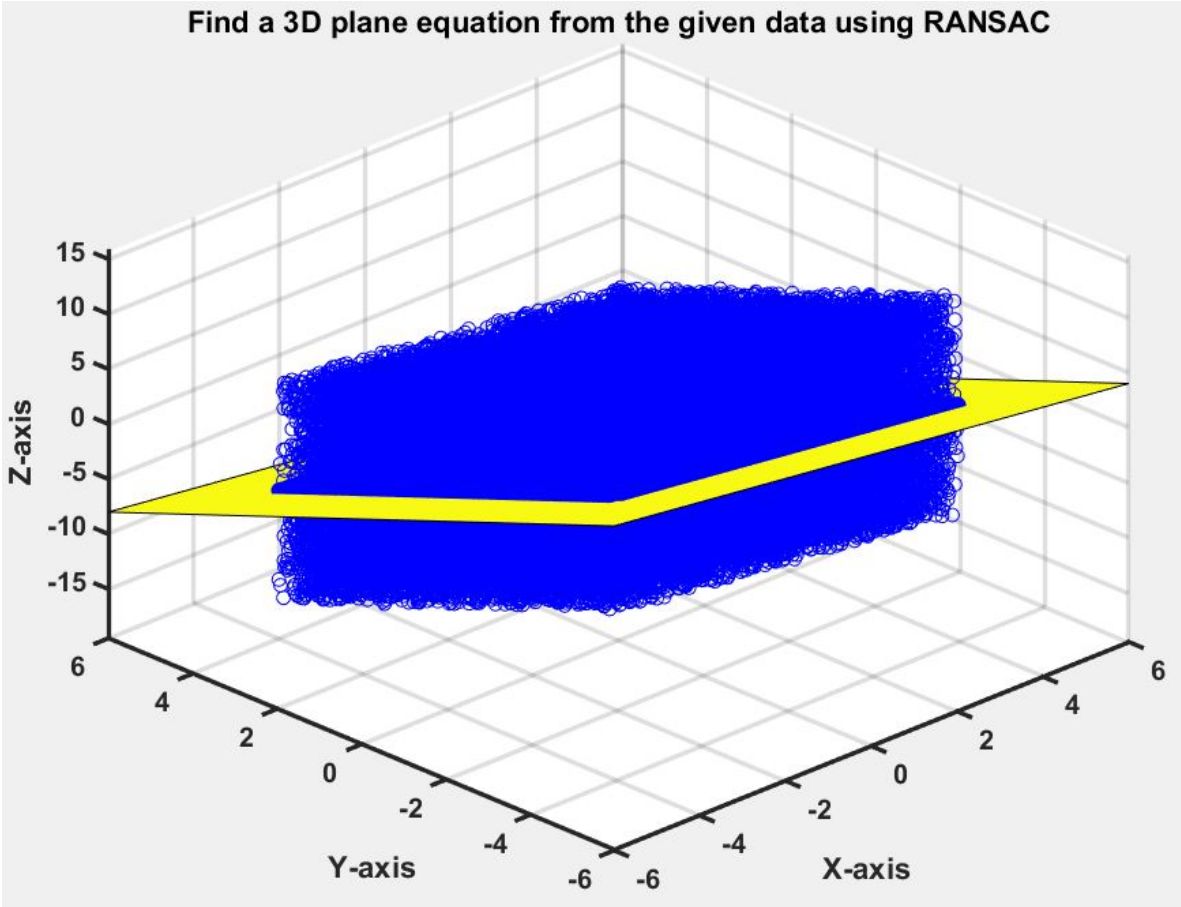



```
%%%% (b) Fit a fourth degree polynomials to the given data using RANSAC %%%
% the general equation for a fourth degree polynomial is as follows:
% ax^4 + bx^3 + cx^2 + dx + e = 0
clc;
clear all;
close all;

load('prob3_polynomial.mat');
nData = numel(x);
nSampLen = 5;
param_best = zeros(5,1);
max_n_trials = 1000;
num_inliner_best = 0;
dist_thr = 0.1;

for ii=1:max_n_trials

    pt_idx = ceil(nData .* rand(1, nSampLen));
    x_sub = x(pt_idx)';
    y_sub = y(pt_idx)';

    A = [x_sub.^4 x_sub.^3 x_sub.^2 x_sub ones(5,1)];
    coef = A\y_sub; % calculating a, b, c, d, and e

    % finding outlier points
    dist_pt = abs(y - (coef(1)*x.^4 + coef(2)*x.^3 + coef(3)*x.^2 + coef(4)*x +
coef(5)));

    num_inlier = sum(dist_pt<dist_thr);
```

```
    if num_inlier>num_inliner_best
        param_best(1) = coef(1);
        param_best(2) = coef(2);
        param_best(3) = coef(3);
        param_best(4) = coef(4);
        param_best(5) = coef(5);
        num_inliner_best = num_inlier;
    end
end

xrange = min(x):0.1:max(x);
y_fit = param_best(1)*xrange.^4 + param_best(2)*xrange.^3 +
param_best(3)*xrange.^2 + param_best(4)*xrange + param_best(5);
figure(2);
plot(x, y, 'ob', 'linewidth', 1);hold on;
plot(xrange, y_fit, 'r--', 'linewidth', 2);
title('Fit a fourth degree polynomials to the data using RANSAC')
legend('Original Data', 'RANSAC');
axis tight;grid on; hold off
xlabel('\bf X-axis');
ylabel('\bf Y-axis');
set(gca,'FontSize',12,'linewidth',2,'fontweight','bold')
```

(c) Find a 3D plane equation from the given data(`prob3_plane.mat`) using RANSAC

In the below figure, a 3D plane has been properly fitted to the given data using RANSAC.




```
%%%%%% (c) Find a 3D plane equation from the given data using RANSAC %%%%%%
% the general equation for a 3D plane is as follows:
% ax + by + cz + d = 0
```

```
clc;
clear all;
close all;

load('prob3_plane.mat');
nData_x = size(x,1);
nData_y = size(y,1);
nSampLen = 3;
param_best = zeros(3,1);
max_n_trials = 1000;
num_inliner_best = 0;
dist_thr = 0.05;

for ii=1:max_n_trials

    pt_idx = ceil(nData_x .* rand(1, nSampLen));
    pt_idy = ceil(nData_y .* rand(1, nSampLen));
    x_sub =
[x(pt_idx(1),pt_idy(1));x(pt_idx(2),pt_idy(2));x(pt_idx(3),pt_idy(3))];
    y_sub =
[y(pt_idx(1),pt_idy(1));y(pt_idx(2),pt_idy(2));y(pt_idx(3),pt_idy(3))];
    z_sub =
[z(pt_idx(1),pt_idy(1));z(pt_idx(2),pt_idy(2));z(pt_idx(3),pt_idy(3))];

    coef_matrix = [x_sub y_sub z_sub ones(3,1)];
    coef = null(coef_matrix); % calculating a, b, c, and d

    % finding outlier points
    % find the vertical distance of points to the plane
    diff_pt = abs(coef(1)*x + coef(2)*y + coef(3)*z + coef(4))/sqrt(coef(1)^2 +
coef(2)^2 + coef(3)^2);

    num_inlier = sum(sum(diff_pt<dist_thr));

    if num_inlier>num_inliner_best
        param_best(1) = coef(1);
        param_best(2) = coef(2);
        param_best(3) = coef(3);
        param_best(4) = coef(4);
        num_inliner_best = num_inlier;
    end
end

[x_range,y_range] = meshgrid(-6:12:6);
z_fit = -1/param_best(3)*(param_best(1)*x_range + param_best(2)*y_range +
param_best(4)); % Solve for z data

figure(3);
plot3(x, y, z, 'ob', 'linewidth', 0.5);hold on;
surf(x_range,y_range,z_fit) %Plot the surface
title('Find a 3D plane equation from the given data using RANSAC')
axis tight;grid on; hold off
xlabel('\bf X-axis');
ylabel('\bf Y-axis');
zlabel('\bf Z-axis');
set(gca,'FontSize',12,'linewidth',2,'fontweight','bold')
```

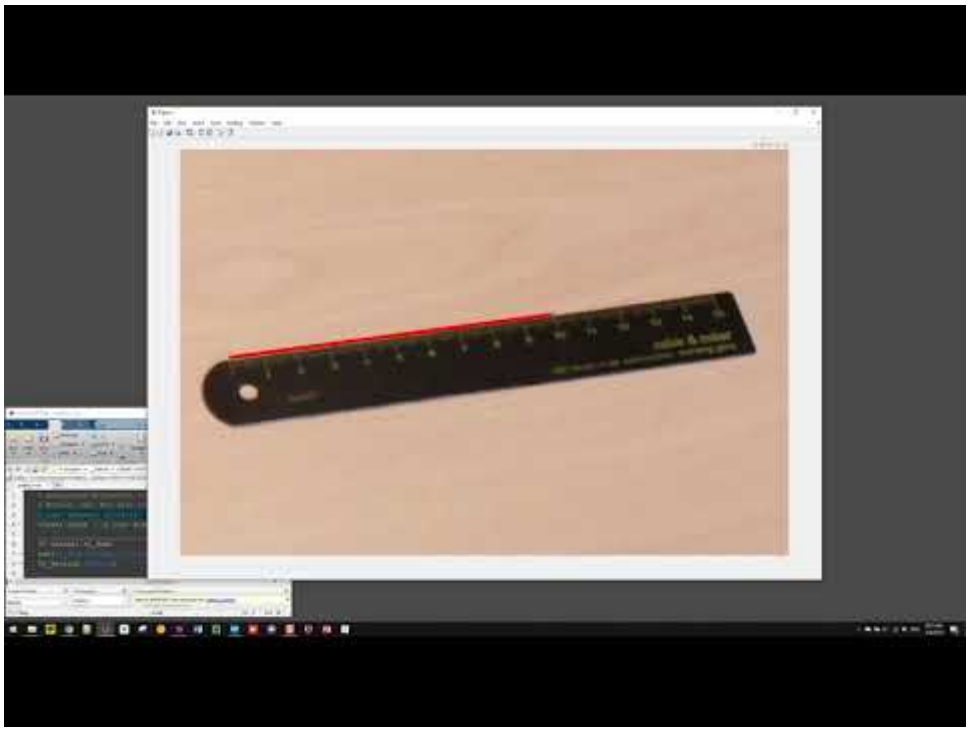
☆☆ **You need to install vl_feat to solve the following problems 4, 5, and 6 for MATLAB users**

☆☆ **For Python users, please install OpenCV to solve the following problems 4, 5, and 6**

Problem 4: Improved 3D Planar Measurement Tool (30 points)

You are going to build an improved 3D planar measurement tool. This application is designed to measure a 3D distance on a plane without picking the corners of the booklet in advance. Users will take a photo in a planar surface where a calibrate paper (here, booklet) is placed and simply click two points on the image for measurement. This reduces a step for picking the corners of your calibration paper.

Here is a demo video:



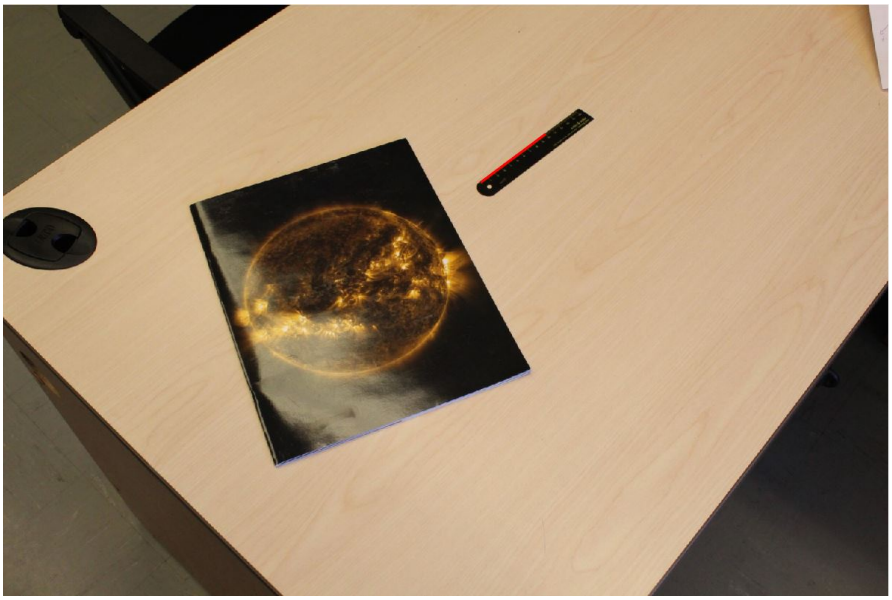
(a) Build your own measurement tool and evaluate your measurement using the images provided (see the folder of `prob4_img`). You may need to estimate homography based on SIFT feature matching. The exact size of the booklet is 24 cm x 31.5 cm and use `cover.jpg` to solve this problem.

Note that you should not do hard cording. This means the corner points of the booklet on your test image is completely unknown and must be detected or measured with your code.

In this problem, the measurement is evaluated from the 0 mark to the 10 cm on the ruler. In the below table, I have shown the measurement evaluated for each images. It should be mentioned that there is a little error for evaluating each measurement, stemming from the fact that we cannot select the exact points on the rulers in images, and we have some errors in the selection of the points on the rulers.

Image	measurement (exact value is 10 cm)
	9.77
	9.86
	9.79

Image	measurement (exact value is 10 cm)
	9.85
	9.77
	9.70

Image	measurement (exact value is 10 cm)
	9.73

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%---------- Problem 4a ----------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc;
clear all;
close all;
run('C:\vlfeat-0.9.21-bin\vlfeat-0.9.21\toolbox\vl_setup')

plot_fig = 'OFF'; % Display figures: ON-->YES    OFF-->NO

coverRGB = imread('cover.JPG');
coverRGB = imresize(coverRGB, [1000, NaN]);%Resize to have 1000 rows
cover = single(rgb2gray(coverRGB));

imgRGB = imread('IMG_0086.JPEG');
imgRGB = imresize(imgRGB, [1000, NaN]);%Resize to have 1000 rows
img = single(rgb2gray(imgRGB));

% detect cover features
[f_cover,d_cover] = vl_sift(cover);

% show the features of cover
if ( strcmp(plot_fig,'ON') )
    figure;imshow(uint8(cover));
    h2 = vl_plotframe(f_cover) ;
    set(h2,'color','y','linewidth',2) ;
end

% detect img features
[f_img,d_img] = vl_sift(img);

% show the features of img
if ( strcmp(plot_fig,'ON') )
    figure;imshow(uint8(img));
    h2 = vl_plotframe(f_img) ;
```

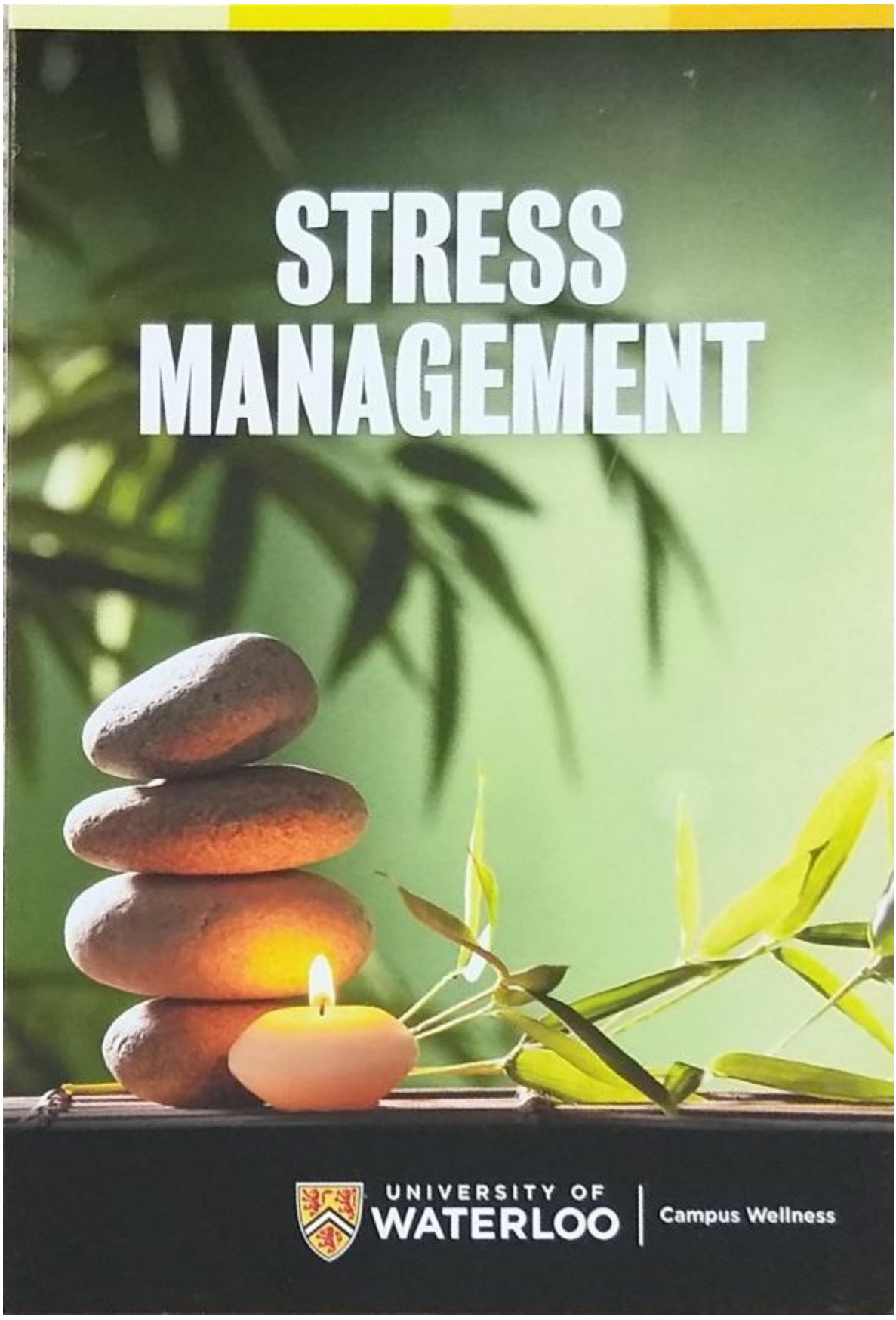
```
    set(h2,'color','y','linewidth',2) ;
end

% match features between cover and img
[matches, scores] = vl_ubcmatch(d_cover, d_img);
% number of matches between two images
num_matches = numel(scores);

% show matches between two images
if ( strcmp(plot_fig,'ON') )
    figure; ax = axes;
    showMatchedFeatures(cover, img,
f_cover(1:2,matches(1,:))',f_img(1:2,matches(2,:))','montage','Parent',ax);
end

% determine x and y values of matching points in two images in HC
% coordinate
points_cover = [f_cover(1:2,matches(1,:));ones(1,num_matches)];
points_img = [f_img(1:2,matches(2,:));ones(1,num_matches)];

% ------ use of RANSAC for estimating the homography ------
nSampLen = 4;
max_n_trials = 1000;
dist_thr = 2; % pixels: define a threshold to determine outliers
H_best =
RANSAC_H(nSampLen,max_n_trials,dist_thr,num_matches,points_cover,points_img);
%-------------------------------------------------------------

% use of homography matrix estimated by RANSAC to find the 3D measurement
% exact size of the booklet is 24 cm x 31.5 cm
bookletSize = [24 31.5]; % cm
% scale of cover picture in x and y directions
scale_x = bookletSize(1)/size(coverRGB,2);
scale_y = bookletSize(2)/size(coverRGB,1);

% %Please click on the points to create polyline (hit enter after choosing
% the polyline)
figure
imshow(imgRGB)
hold on
line = getline();
plot(line(:,1),line(:,2),'r', 'linewidth',  2, 'LineStyle', '-')

line_HC = [line(:,1)';line(:,2)';ones(1,size(line,1))]; % line in HC
H_inv = inv(H_best');
line_r = H_inv*line_HC; % tranform the start and end points of the line
% from img plane to cover plane
line_cr = line_r(1:2,:)./line_r(3,:); % calculate in cartesian coordinate

% calculate the length of lines in polyline
for ii = 1 : (size(line,1)-1)
    dist(ii) = norm((line_cr(:,ii+1)-line_cr(:,ii)).*[scale_x;scale_y]);
    fprintf('Length of line %1u is %2.3f cm\n',ii,dist(ii));
end


% RANSAC function for estimating the homography
```

```
function H_best =
RANSAC_H(nSampLen,max_n_trials,dist_thr,num_matches,points_cover,points_img)

num_inliner_best = 0;
for ii=1:max_n_trials
    % randomly select four points to estimate homography
    pt_idx = ceil(num_matches .* rand(1, nSampLen));
    x = points_cover(1,pt_idx)';
    y = points_cover(2,pt_idx)';
    xp = points_img(1,pt_idx)';
    yp = points_img(2,pt_idx)';

    % --- compute homography matrix ---
    x_xp = x.*xp;
    x_yp = x.*yp;
    y_xp = y.*xp;
    y_yp = y.*yp;
    A = zeros(size(x,1)*2,9);
    A(1:2:end,3) = 1;
    A(2:2:end,6) = 1;
    A(1:2:end,1:2) = [x y];
    A(2:2:end,4:5) = [x y];
    A(1:2:end,7) = -x_xp;
    A(1:2:end,8) = -y_xp;
    A(2:2:end,7) = -x_yp;
    A(2:2:end,8) = -y_yp;
    A(1:2:end,9) = -xp;
    A(2:2:end,9) = -yp;

    T = null(A);
    T = T(:,1);
    T = T/T(end);
    H = (reshape(T,3,3));
    % ---------------------------------

    % transform cover points to img using homography projection
    points_cover_H = H'*points_cover;
    % calculate distance between projected points and exact points in img
    dx = points_cover_H(1,:)./points_cover_H(3,:)-points_img(1,:);
    dy = points_cover_H(2,:)./points_cover_H(3,:)-points_img(2,:);

    % finding outlier points
    dist_pt = sqrt(dx.^2+dy.^2);
    num_inlier = sum(dist_pt<dist_thr);
    if num_inlier>num_inliner_best
        H_best = H;
        num_inliner_best = num_inlier;
    end
end
end
```

(b) Prepare your own calibration paper and take photos similar to the ones in (a). Then, evaluate your tool.

For this section, my calibration paper is a stress management catalogue from UW's campus wellness. The dimension of the catalogue is **11.8 x 17.4 cm**. Also, the measurement is evaluated from the 0 mark to the 10 cm on the ruler.

First, using the code written in problem#4 Task#6, I extracted the catalogue image from the original image and remove its projective distortion in an automated way:



Than, I used the catalogue image to measure a 3D distance. In the below table, I have shown the measurement evaluated for each images.

Image	measurement (exact value is 10 cm)
	9.73

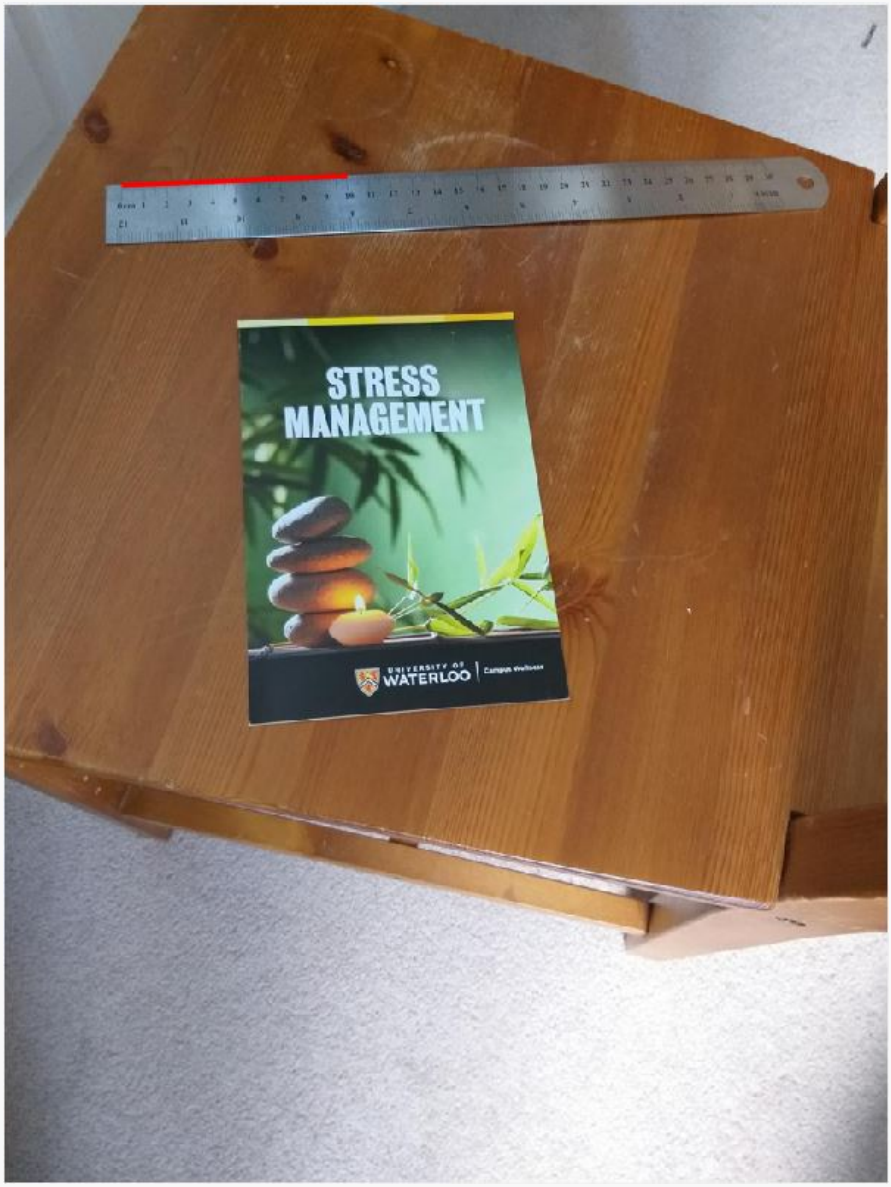
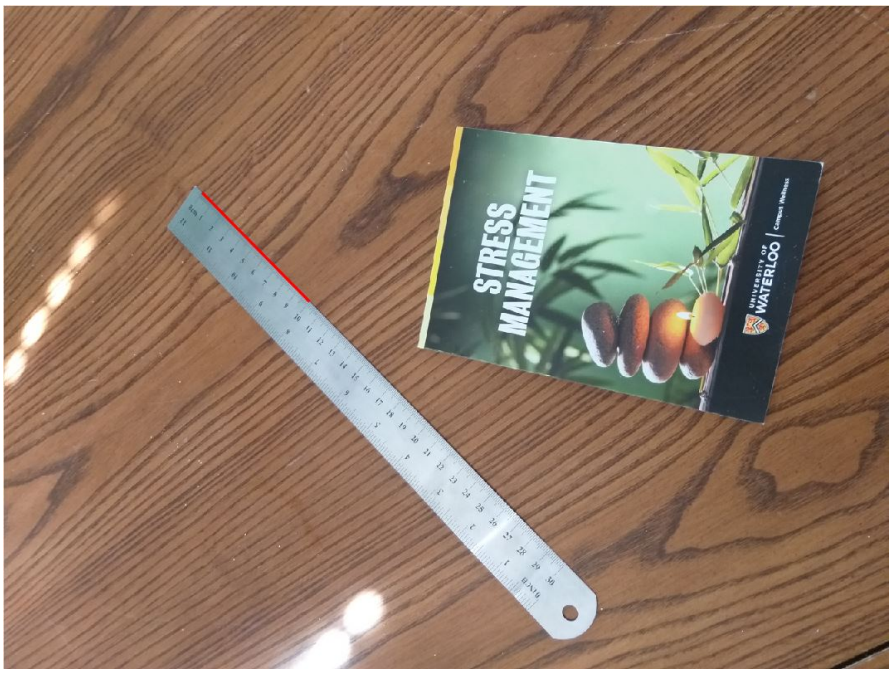
Image	measurement (exact value is 10 cm)
	9.73
	9.71

Image	measurement (exact value is 10 cm)
	10.08
	9.88

Image	measurement (exact value is 10 cm)
	10.25

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%--------------- Problem 4b --------------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc;
clear all;
close all;
run('C:\vlfeat-0.9.21-bin\vlfeat-0.9.21\toolbox\vl_setup')

plot_fig = 'OFF'; % Display figures: ON-->YES    OFF-->NO

coverRGB = imread('cover.JPG');
coverRGB = imresize(coverRGB, [1000, NaN]);%Resize to have 1000 rows
cover = single(rgb2gray(coverRGB));

imgRGB = imread('IMG1.JPG');
imgRGB = imresize(imgRGB, [1000, NaN]);%Resize to have 1000 rows
img = single(rgb2gray(imgRGB));

% detect cover features
[f_cover,d_cover] = vl_sift(cover);

% show the features of cover
if ( strcmp(plot_fig,'ON') )
    figure;imshow(uint8(cover));
    h2 = vl_plotframe(f_cover) ;
    set(h2,'color','y','linewidth',2) ;
end

% detect img features
[f_img,d_img] = vl_sift(img);

% show the features of img
```

```
if ( strcmp(plot_fig,'ON') )
    figure;imshow(uint8(img));
    h2 = vl_plotframe(f_img) ;
    set(h2,'color','y','linewidth',2) ;
end

% match features between cover and img
[matches, scores] = vl_ubcmatch(d_cover, d_img);
% number of matches between two images
num_matches = numel(scores);

% show matches between two images
if ( strcmp(plot_fig,'ON') )
    figure; ax = axes;
    showMatchedFeatures(cover, img,
f_cover(1:2,matches(1,:))',f_img(1:2,matches(2,:))','montage','Parent',ax)
end

% determine x and y values of matching points in two images in HC
% coordinate
points_cover = [f_cover(1:2,matches(1,:));ones(1,num_matches)];
points_img = [f_img(1:2,matches(2,:));ones(1,num_matches)];

% ------ use of RANSAC for estimating the homography ------
nSampLen = 4;
max_n_trials = 1000;
dist_thr = 2; % pixels: define a threshold to determine outliers
H_best =
RANSAC_H(nSampLen,max_n_trials,dist_thr,num_matches,points_cover,points_img);
%-------------------------------------------------------------

% use of homography matrix estimated by RANSAC to find the 3D measurement
% exact size of the booklet is 11.8 cm x 17.4 cm
bookletSize = [11.8 17.4]; % cm
% scale of cover picture in x and y directions
scale_x = bookletSize(1)/size(coverRGB,2);
scale_y = bookletSize(2)/size(coverRGB,1);

% %Please click on the points to create polyline (hit enter after choosing
% the polyline)
figure
imshow(imgRGB)
hold on
line = getline();
plot(line(:,1),line(:,2),'r', 'linewidth',  2, 'LineStyle', '-')

line_HC = [line(:,1)';line(:,2)';ones(1,size(line,1))]; % line in HC
H_inv = inv(H_best');
line_r = H_inv*line_HC; % tranform the start and end points of the line
% from img plane to cover plane
line_cr = line_r(1:2,:)./line_r(3,:); % calculate in cartesian coordinate

% calculate the length of lines in polyline
for ii = 1 : (size(line,1)-1)
    dist(ii) = norm((line_cr(:,ii+1)-line_cr(:,ii)).*[scale_x;scale_y]);
    fprintf('Length of line %1u is %2.3f cm\n',ii,dist(ii));
end
```

```
% RANSAC function for estimating the homography
function H_best =
RANSAC_H(nSampLen,max_n_trials,dist_thr,num_matches,points_cover,points_img)

num_inliner_best = 0;
for ii=1:max_n_trials
    % randomly select four points to estimate homography
    pt_idx = ceil(num_matches .* rand(1, nSampLen));
    x = points_cover(1,pt_idx)';
    y = points_cover(2,pt_idx)';
    xp = points_img(1,pt_idx)';
    yp = points_img(2,pt_idx)';

    % --- compute homography matrix ---
    x_xp = x.*xp;
    x_yp = x.*yp;
    y_xp = y.*xp;
    y_yp = y.*yp;
    A = zeros(size(x,1)*2,9);
    A(1:2:end,3) = 1;
    A(2:2:end,6) = 1;
    A(1:2:end,1:2) = [x y];
    A(2:2:end,4:5) = [x y];
    A(1:2:end,7) = -x_xp;
    A(1:2:end,8) = -y_xp;
    A(2:2:end,7) = -x_yp;
    A(2:2:end,8) = -y_yp;
    A(1:2:end,9) = -xp;
    A(2:2:end,9) = -yp;

    T = null(A);
    T = T(:,1);
    T = T/T(end);
    H = (reshape(T,3,3));
    % ----------------------------------

    % transform cover points to img using homography projection
    points_cover_H = H'*points_cover;
    % calculate distance between projected points and exact points in img
    dx = points_cover_H(1,:)./points_cover_H(3,:)-points_img(1,:);
    dy = points_cover_H(2,:)./points_cover_H(3,:)-points_img(2,:);

    % finding outlier points
    dist_pt = sqrt(dx.^2+dy.^2);
    num_inlier = sum(dist_pt<dist_thr);
    if num_inlier>num_inliner_best
        H_best = H;
        num_inliner_best = num_inlier;
    end
end
end
```

(c) Compare measurements using this new tool and the one developed in Task 3.

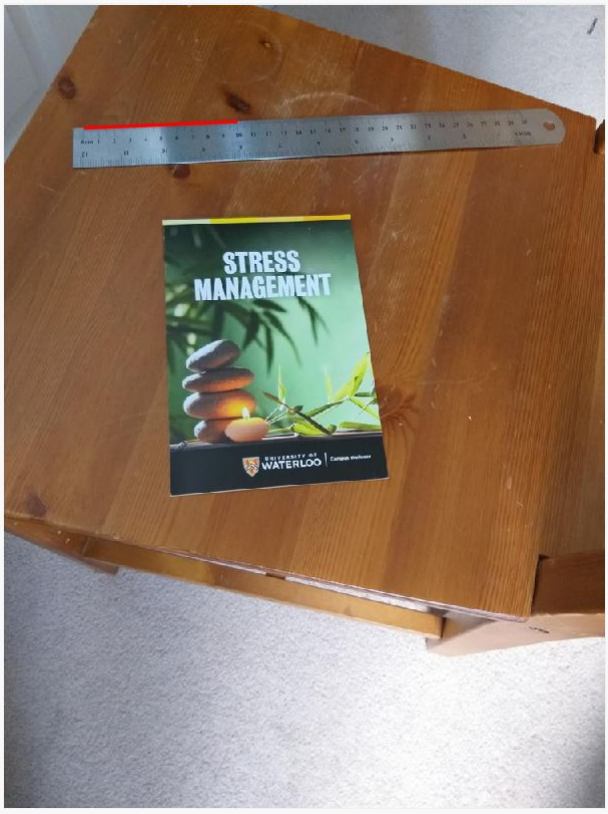
Image	measurement using SIFT feature matching (exact value is 10 cm)	measurement using homography (Task 3)
	9.73	9.89
	9.73	9.92

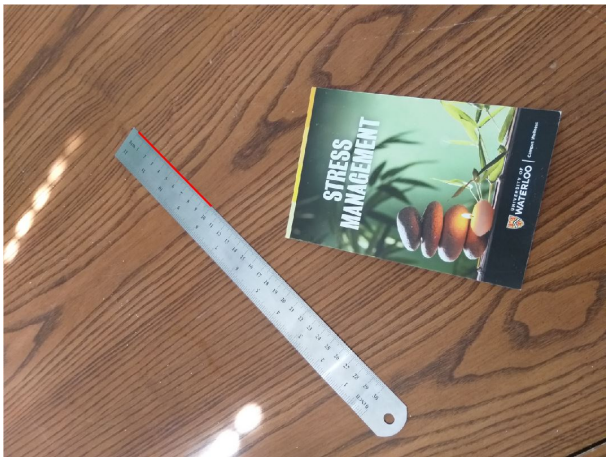
Image	measurement using SIFT feature matching (exact value is 10 cm)	measurement using homography (Task 3)
	9.71	10.13
	10.08	9.86
	9.88	9.96

Image	measurement using SIFT feature matching (exact value is 10 cm)	measurement using homography (Task 3)
	10.25	10.09

As you can see, measurements using homography are more accurate than the measurements using SIFT feature. In my view, the reason is that in Task 3, four corners of the catalogue have large distances from each other, probably resulting in reduction of some errors for measurements. On the other hand, in this new tool, features may be be very close to each other, making a potential for small errors that can be magnified after applying homography for the whole image.

Problem 5: Book Classification using SIFT (30 points)

You are going to categorize books on images. Here are the input images and expected outcomes.

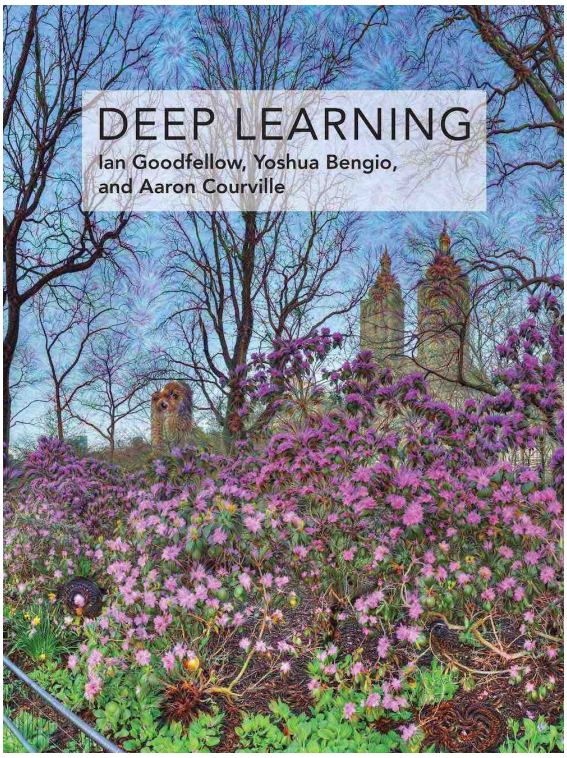
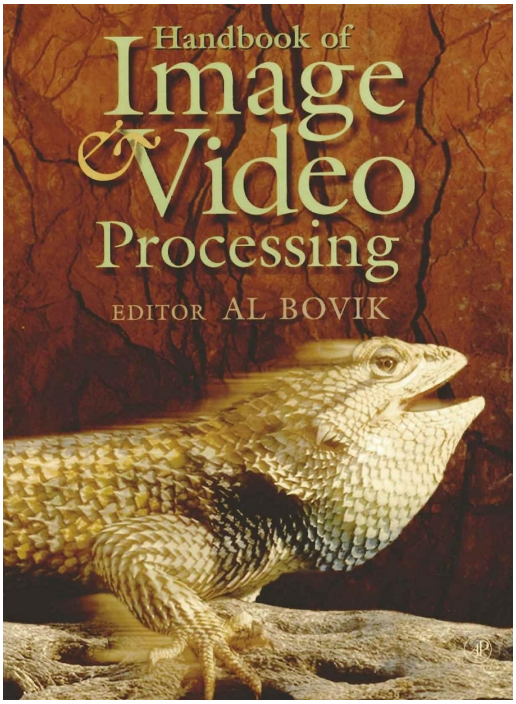
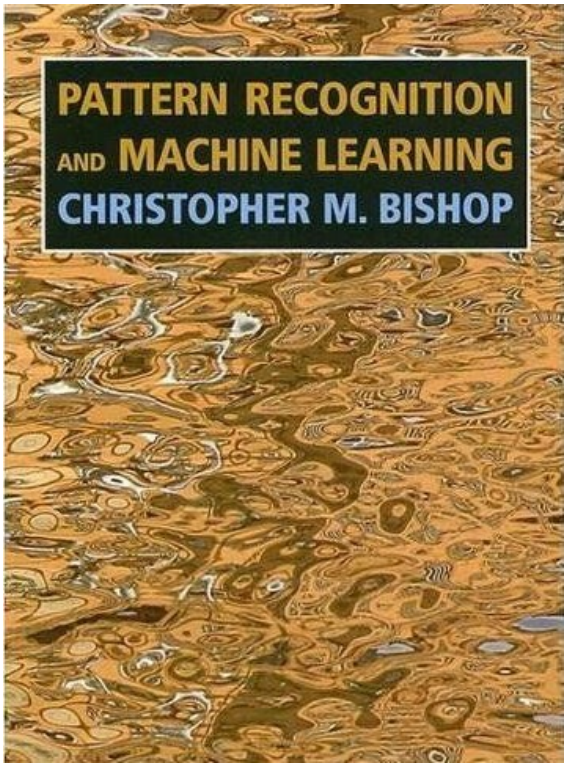
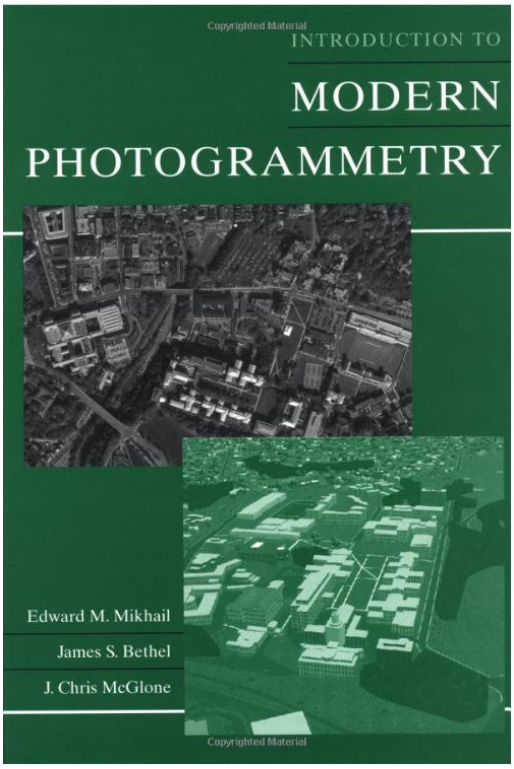
Test image	Result
	
	

Your code needs to automatically compute the outlines (boundary) of each book and its identity (book name). There are 17 images (see the folder of `prob5_img`) and in each image, four books are placed on a desk. Your code should not fail to identify a book or estimate its boundary more than 5 books among all books (85 = 17 images x 5 books). Note that you should not do hard cording. This is not a problem using edge detection!

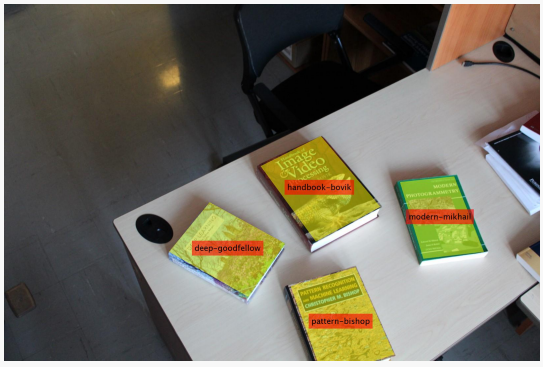
Hint: You can reuse the code that you are developing in Problem 4. Like `cover.jpg`, you need to prepare cover images and name them. Note this problem cannot be solved using Hough transform.

Hint: You should download the book covers from web.

I have downloaded the book covers from web and have named them as:

Cover	cover
	
deep_goodfellow	handbook_bovik
	
pattern_bishop	modern_mikhail

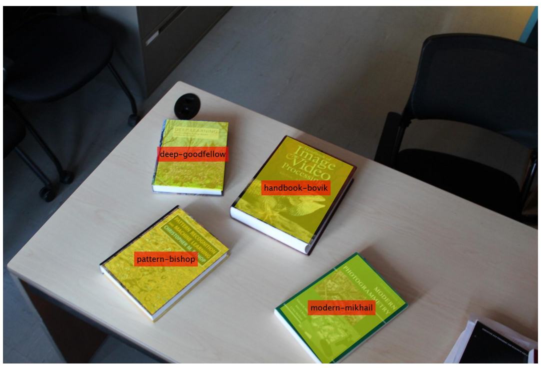
Using SIFT, my code automatically compute the outlines of each book and its identity (book name). In the below table, I have shown all the images along with all the books identified.

Test image	Result
	
	
	
	

Test image	Result
	
	
	
	

Test image	Result
	
	
	
	

Test image	Result
	
	
	
	

Test image	Result
	

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%---------- Problem 5 ----------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc;
clear all;
close all;
run('C:\vlfeat-0.9.21-bin\vlfeat-0.9.21\toolbox\vl_setup')

plot_fig = 'OFF'; % Display figures: ON-->YES    OFF-->NO

coversRGB{1} = imread('deep_goodfellow.jpg');
coversRGB{2} = imread('handbook_bovik.jpg');
coversRGB{3} = imread('pattern_bishop.jpg');
coversRGB{4} = imread('modern_mikhail.jpg');

names_of_books = {'deep-goodfellow', 'handbook-bovik', 'pattern-bishop',
'modern-mikhail'};

% detect and show cover features
for ii = 1 : numel(coversRGB)
    % detect cover features
    covers{ii} = single(rgb2gray(coversRGB{ii}));
    [f_covers{ii},d_covers{ii}] = vl_sift(covers{ii});

    % show the features of cover
    if ( strcmp(plot_fig,'ON') )
        figure;imshow(uint8(covers{ii}));
        h2 = vl_plotframe(f_covers{ii}) ;
        set(h2,'color','y','linewidth',2) ;
    end
end

% loop for automatically compute the outlines (boundary) of each book and
% its identity (book name)
for ii = 12 : 28
    fileName = char(strcat(num2str(ii,'IMG_02%2.f'),".JPEG"));
    imgRGB = imread(fileName);
    img = single(rgb2gray(imgRGB));
    % detect img features
    [f_img,d_img] = vl_sift(img);

    % show the features of img
    if ( strcmp(plot_fig,'ON') )
        figure;imshow(uint8(img));
        h2 = vl_plotframe(f_img) ;
        set(h2,'color','y','linewidth',2) ;
    end

    % --- find corners of four covers in each image using RANSAC and
    % homography
    for jj = 1 : numel(covers)
        % match features between cover and img
        [matches, scores] = vl_ubcmatch(d_covers{jj}, d_img);
        % number of matches between two images
        num_matches = numel(scores);
```

```
        % show matches between two images
        if ( strcmp(plot_fig,'ON') )
            figure; ax = axes;
            showMatchedFeatures(covers{jj}, img, f_covers{jj}
(1:2,matches(1,:))',f_img(1:2,matches(2,:))','montage','Parent',ax)
        end

        % determine x and y values of matching points in two images in HC
        % coordinate
        points_cover = [f_covers{jj}(1:2,matches(1,:));ones(1,num_matches)];
        points_img = [f_img(1:2,matches(2,:));ones(1,num_matches)];

        % ------ use of RANSAC for estimating the homography ------
        nSampLen = 4;
        max_n_trials = 1000;
        dist_thr = 2; % pixels: define a threshold to determine outliers
        H_best =
RANSAC_H(nSampLen,max_n_trials,dist_thr,num_matches,points_cover,points_img);
        %-------------------------------------------------------------

        % prjective transformation of corners from covers to images
        % (using best homography matrix resulted from RANSAC function)
        corner_c =
[1,1,1;size(covers{jj},2),1,1;size(covers{jj},2),size(covers{jj},1),1;1,size(cov
ers{jj},1),1]';
        corner_c_H = H_best'*corner_c;
        corner_c_Cr = corner_c_H(1:2,:)./corner_c_H(3,:); % calculate in
cartesian coordinate
        cover{jj} = [corner_c_Cr(:,1)' corner_c_Cr(:,2)' corner_c_Cr(:,3)'
corner_c_Cr(:,4)'];
        text_center(jj,:) = mean(corner_c_Cr');

    end
    %-------------------------------------------------------------

    % show edges and texts related to each book
    imgAnnotated = imgRGB;
    for kk = 1:length(cover)
        imgAnnotated =
insertShape(imgAnnotated,'FilledPolygon',cover{kk},'Opacity',0.5);
        imgAnnotated =
insertText(imgAnnotated,text_center(kk,:),names_of_books{kk},'FontSize',24,'Anch
orPoint','Center','BoxColor','red');
    end

    if ( strcmp(plot_fig,'ON') )
        figure
        imshow(imgAnnotated);
    end

    fileNameEx = char(strcat(num2str(ii,'final-IMG_02%2.f'),".jpg"));
    imwrite(imgAnnotated,fileNameEx)
end


% RANSAC function for estimating the homography
```

```
function H_best =
RANSAC_H(nSampLen,max_n_trials,dist_thr,num_matches,points_cover,points_img)

num_inliner_best = 0;
for ii=1:max_n_trials
    % randomly select four points to estimate homography
    pt_idx = ceil(num_matches .* rand(1, nSampLen));
    x = points_cover(1,pt_idx)';
    y = points_cover(2,pt_idx)';
    xp = points_img(1,pt_idx)';
    yp = points_img(2,pt_idx)';

    % --- compute homography matrix ---
    x_xp = x.*xp;
    x_yp = x.*yp;
    y_xp = y.*xp;
    y_yp = y.*yp;
    A = zeros(size(x,1)*2,9);
    A(1:2:end,3) = 1;
    A(2:2:end,6) = 1;
    A(1:2:end,1:2) = [x y];
    A(2:2:end,4:5) = [x y];
    A(1:2:end,7) = -x_xp;
    A(1:2:end,8) = -y_xp;
    A(2:2:end,7) = -x_yp;
    A(2:2:end,8) = -y_yp;
    A(1:2:end,9) = -xp;
    A(2:2:end,9) = -yp;

    T = null(A);
    T = T(:,1);
    T = T/T(end);
    H = (reshape(T,3,3));
    % ---------------------------------

    % transform cover points to img using homography projection
    points_cover_H = H'*points_cover;
    % calculate distance between projected points and exact points in img
    dx = points_cover_H(1,:)./points_cover_H(3,:)-points_img(1,:);
    dy = points_cover_H(2,:)./points_cover_H(3,:)-points_img(2,:);

    % finding outlier points
    dist_pt = sqrt(dx.^2+dy.^2);
    num_inlier = sum(dist_pt<dist_thr);
    if num_inlier>num_inliner_best
        H_best = H;
        num_inliner_best = num_inlier;
    end
end
end
```

Problem 6: Image Stitching (30 points)

First, go to an image stitching tutorial and thoroughly review the code. You have to fully understand its process.

Second, you are going to update this code. Your code must satisfy the following conditions:

- **Do not use `imageDatastore` and related functions that call this object in MATLAB.**
- **Use SIFT feature extractor and descriptor rather than `SURF`. You can use `vl_sift` for MATLAB and opencv for Python.**
- **Do not use `estimateGeometricTransform` and replace it using your own homography estimator that you may developed in problem 4. You need to use RANSAC for robust matching.**

Third, test your code using the building dataset (the image set used in the tutorial) in MATLAB. For Python users, you can find those images `prob6_building_img.zip`.

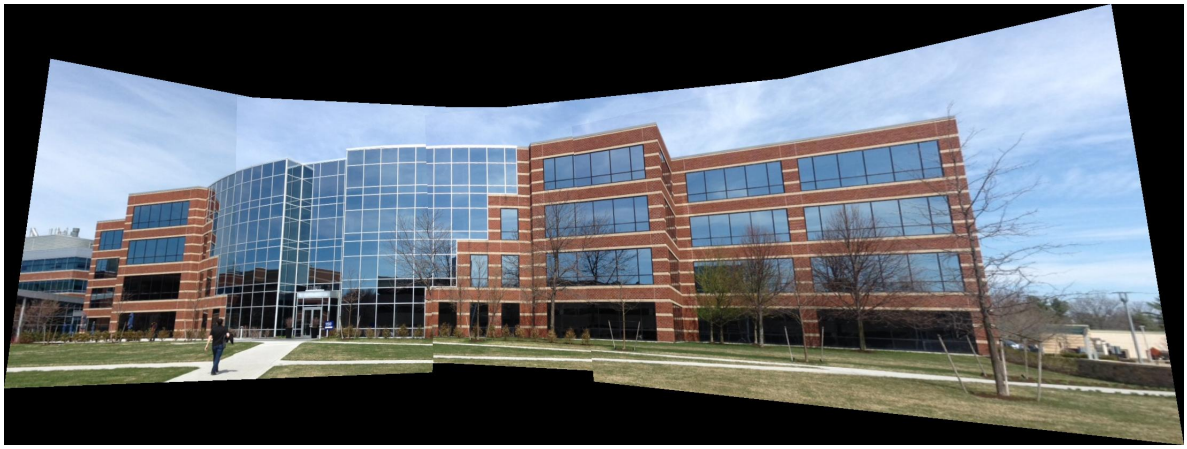
Fourth, take your own image set (more than 3 images) from a scenic place and stitch them to create a nice panorama image. You should not download images on the web or use my images and should not share your images with your colleagues. Please take your images using your cameras.

If your resulting panorama is not a good quality, you need to explain what factors make dropping its quality.

I have thoroughly reviewed the MATLAB tutorial for image stitching and update that code in such a way that satisfies the following conditions:

- I did not use `imageDatastore` and related functions that call this object in MATLAB.
- Using `vl_sift`, I applied SIFT feature extractor and descriptor rather than `SURF`.
- I tested my code using the building dataset.
- Finally, I took my own image set from the outside of my home, and then, I stitched them to create a nice panorama image.

In the below figure, I have tested my code using the building dataset:



Here, I have shown my own image set from the outside of my home, and then, I have stitched them to create a nice panorama image:

Test image	Result
	
	
	
	

My resulting panorama is a great quality:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%---------- Problem 6 ----------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clc;
clear all;
close all;
run('C:\vlfeat-0.9.21-bin\vlfeat-0.9.21\toolbox\vl_setup')

%% load images
ImageFiles = fullfile('building','*.JPG');
listImages = dir(ImageFiles);
for i=1:numel(listImages)
    imgRGB{i} = imread(fullfile(listImages(i).folder, listImages(i).name));
    imgRGB{i} = imresize(imgRGB{i},[1000 NaN]);
end

% display images to be stitched
montage(imgRGB);

%% Register Image Pairs
num_img = numel(imgRGB); % total number of images
I = imgRGB{1};% read the first image from the image set.

% initialize features for I(1)
img = single(rgb2gray(I));
[feature, discriptor] = vl_sift(img);

% initialize variable to hold image sizes.
imageSize = zeros(num_img,2);
imageSize(1,:) = size(img);
H_imgs = zeros(3,3,num_img); % create 3D matrix to calculate the
% transformation matrix of the nth image from the transformation matrix of
% previous images: Compute the transformation that maps I(n) into the
% panorama image as T(n)?T(n?1)?...?T(1)
H_imgs(:,:,1) = eye(3,3); % make the first transformation matrix as the identity
% matrix because the original image doesn't change

% Iterate over remaining image pairs
```

```
for ii = 2:num_img
    % store points and features for I(n-1).
    feature_pre = feature;
    discriptor_pre = discriptor;

    % read I(n).
    I = imgRGB{ii};
    % convert image to grayscale.
    img = single(rgb2gray(I));
    % save image size.
    imageSize(ii,:) = size(img);

    % detect features for I(n).
    [feature, discriptor] = vl_sift(img);
    % find matches between I(n) and I(n-1).
    [matches, scores] = vl_ubcmatch(discriptor_pre, discriptor);
    num_matches = numel(scores);

    % determine x and y values of matching points in two images in HC
    % coordinate
    points_pre = [feature_pre(1:2,matches(1,:));ones(1,num_matches)];
    points_I = [feature(1:2,matches(2,:));ones(1,num_matches)];

    % --use of RANSAC to estimate transformation between I(n) and I(n-1)--
    nSampLen = 4;
    max_n_trials = 50000;
    dist_thr = 2; % pixels: define a threshold to determine outliers
    H_best =
RANSAC_H(nSampLen,max_n_trials,dist_thr,num_matches,points_I,points_pre);
    %-------------------------------------------------------------

    % Compute T(n) * T(n-1) * ... * T(1)
    H_imgs(:,:,ii) = H_best * H_imgs(:,:,ii-1);
end

% --------- modify the transformations such that the center of the scene is
% the least distorted ---------
%Compute the output limits for each transform
for i = 1:num_img
    [xlim(i,:), ylim(i,:)] = outputLimits(projective2d(H_imgs(:,:,i)), [1
imageSize(i,2)], [1 imageSize(i,1)]);
end

% compute the average X limits for each transforms and find the image that
% is in the center
avgXLim = mean(xlim,2);
[~, idx] = sort(avgXLim);
centerIdx = floor((num_img+1)/2);
centerImageIdx = idx(centerIdx);

% apply the center image's inverse transform to all the others
Hinv = inv(H_imgs(:,:,centerImageIdx));
for i = 1:num_img
    H_imgs(:,:,i) = H_imgs(:,:,i) * Hinv;
end
%-------------------------------------------------------------
```

```
%% Initialize the Panorama
% stitch our images together for the panorama
% automatically compute the size of the panorama by finding the minimum and
% maximum output limits over all transformations
for i = 1:num_img
    [xlim(i,:), ylim(i,:)] = outputLimits(projective2d(H_imgs(:,:,i)), [1
imageSize(i,2)], [1 imageSize(i,1)]);
end
maxImageSize = max(imageSize);

% find the minimum and maximum output limits
xMin = min([1; xlim(:)]);
xMax = max([maxImageSize(2); xlim(:)]);
yMin = min([1; ylim(:)]);
yMax = max([maxImageSize(1); ylim(:)]);

% width and height of panorama.
width  = round(xMax - xMin);
height = round(yMax - yMin);

% initialize the "empty" panorama.
panorama = zeros([height width 3], 'like', I);

%% Create the Panorama
% Use imwarp to map images into the panorama and use vision.AlphaBlender to
% overlay the images together.
blender = vision.AlphaBlender('Operation', 'Binary mask','MaskSource', 'Input
port');

% Create a 2-D spatial reference object defining the size of the panorama.
xLimits = [xMin xMax];
yLimits = [yMin yMax];
panoramaView = imref2d([height width], xLimits, yLimits);

% Create the panorama.
for ii = 1 : num_img
    I = imgRGB{ii};

    % Transform I into the panorama.
    warpedImage = imwarp(I, projective2d(H_imgs(:,:,ii)), 'OutputView',
panoramaView);

    %Generate a binary mask.
    mask = imwarp(true(size(I,1),size(I,2)),
projective2d(H_imgs(:,:,ii)),'OutputView', panoramaView);

    %Overlay the warpedImage onto the panorama.
    panorama = step(blender, panorama, warpedImage, mask);
end

figure;
imshow(panorama);
imwrite(panorama,'panorama.jpg');


%% RANSAC function for estimating the homography
function H_best =
RANSAC_H(nSampLen,max_n_trials,dist_thr,num_matches,points_cover,points_img)
```

```
num_inliner_best = 0;
for ii=1:max_n_trials
    % randomly select four points to estimate homography
    pt_idx = ceil(num_matches .* rand(1, nSampLen));
    x = points_cover(1,pt_idx)';
    y = points_cover(2,pt_idx)';
    xp = points_img(1,pt_idx)';
    yp = points_img(2,pt_idx)';

    % --- compute homography matrix ---
    x_xp = x.*xp;
    x_yp = x.*yp;
    y_xp = y.*xp;
    y_yp = y.*yp;
    A = zeros(size(x,1)*2,9);
    A(1:2:end,3) = 1;
    A(2:2:end,6) = 1;
    A(1:2:end,1:2) = [x y];
    A(2:2:end,4:5) = [x y];
    A(1:2:end,7) = -x_xp;
    A(1:2:end,8) = -y_xp;
    A(2:2:end,7) = -x_yp;
    A(2:2:end,8) = -y_yp;
    A(1:2:end,9) = -xp;
    A(2:2:end,9) = -yp;

    T = null(A);
    T = T(:,1);
    T = T/T(end);
    H = (reshape(T,3,3));
    % ---------------------------------

    % transform cover points to img using homography projection
    points_cover_H = H'*points_cover;
    % calculate distance between projected points and exact points in img
    dx = points_cover_H(1,:)./points_cover_H(3,:)-points_img(1,:);
    dy = points_cover_H(2,:)./points_cover_H(3,:)-points_img(2,:);

    % finding outlier points
    dist_pt = sqrt(dx.^2+dy.^2);
    num_inlier = sum(dist_pt<dist_thr);
    if num_inlier>num_inliner_best
        H_best = H;
        num_inliner_best = num_inlier;
    end
end
end
```