ECE661 Computer Vision HW 3

Rih-Teng Wu

Email: wu952@purdue.edu

1. Remove Projective Distortion – Point to Point Correspondence

When we see the parallel lines in the world plane are not parallel in an image, there exists the projective distortion in the image. To remove the projective distortion, we can use the homography calculated by means of point to point correspondence. According to what we have learned from HW2, the mapping between an image plane and a world plane is given by:

$$X_i = HX_w$$

Where X_i is the points of image plane in homogeneous coordinates, X_w is the points of world plane in homogeneous coordinates, and H is the homography matrix. Now, let $X_i = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}, X_w = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$, and $H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$. Since we only care about ratios in homogeneous representation, the value of h_{33} can be set to 1. To solve the eight unknowns, we need to find at least four pairs of corresponding points in the image plane and the world plane. As a result, if now we have n pairs of corresponding points, and denote the i^{th} pair of corresponding points as $(x_i', y_i'), (x_i, y_i)$, we have the following relation:

$$x_i' = x_i h_{11} + y_i h_{12} + h_{13} - x_i' x_i h_{31} - x_i' y_i h_{32}$$
$$y_i' = x_i h_{21} + y_i h_{22} + h_{23} - y_i' x_i h_{31} - y_i' y_i h_{32}$$

Rewrite the above relation in matrix form, we have:

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1'x_1 & -x_1'y_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y_1'x_1 & -y_1'y_1 \\ & \vdots & & & \vdots & & & \vdots \\ x_n & y_n & 1 & 0 & 0 & 0 & -x_n'x_n & -x_n'y_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -y_n'x_n & -y_n'y_n \end{bmatrix}_{2nx8} \times \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix}_{8x1} = \begin{bmatrix} x_1' \\ y_1' \\ \vdots \\ x_n' \\ y_n' \end{bmatrix}_{2nx1}$$

Denote the above equation as $Ah = B$, then h can be solved using the least square estimate method just in case we have more than four pairs of corresponding points.

$$h = (A^T A)^{-1} A^T B$$

Once h is found, we can use the homography matrix H to perform the mapping we desire.

Some note for programming:

a. In this task, the homography matrix H is calculated using $X_w = HX_i$.
b. The information of world coordinates is given in unit of centimeter. Thus, I assume one pixel is equal to one centimeter.
c. Using four corresponding point pairs, the homography H is solved.
d. Similar to what we have done in HW2, at first the four corner points of image is mapped to the world plane using homography H, then the dimension of the output image is determined using the mapped points. Finally, the points in the output image are mapped back to the image using H^{-1} to find the corresponding pixel values.

2. Remove Projective Distortion – Vanishing Line

Another way to remove the projective distortion is to use the vanishing lines in the image. Since an affine homography maps l_∞ to l_∞, all we have to do is to find a homography that maps the vanishing line back to l_∞.

Suppose now we have two line pairs (L_1', L_2') and (M_1', M_2') in the image that are supposed to be two parallel line pairs in the world plane $(L_1 // L_2, M_1 // M_2)$, we can find the intersection of this two line pairs, which are called vanishing points:

$$P_1 = L_1' \times L_2'$$
$$P_2 = M_1' \times M_2'$$

Then, the vanishing line L' is given by:

$$L' = P_1 \times P_2$$

Therefore, if the vanishing line $L' = [l_1 \quad l_2 \quad l_3]^{\mathrm{T}}$, the homography that maps the vanishing line back to l_∞ is given by:

$$H_P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & l_3 \end{bmatrix}$$

As a result, apply this homography to our image can get rid of the projective distortion.

Some note for programming:

a. The homography is given by $X_w = H_\mathrm{p} X_i$. Be careful to the mapping direction.
b. The vanishing line needs to be normalized before putting into H_P.
c. The homography H_P can be easily obtained using the vanishing line.
d. Similar to what we have done befroe, at first the four corner points of image is mapped to the world plane using homography H_P, then the dimension of the output image is determined using the mapped points. Finally, the points in the output image are mapped back to the image using ${H_P}^{-1}$ to find the corresponding pixel values.

3. Remove Affine Distortion

When we see the angles are not preserved in the image, there exists the affine distortion in the image. Suppose now we have two orthogonal lines $\mathrm{L} = [l_1 \quad l_2 \quad l_3]^\mathrm{T}$ and $\mathrm{M} = [m_1 \quad m_2 \quad m_3]^\mathrm{T}$ in the world plane, the angle between this two lines is given by:

$$\cos\theta = \frac{\mathrm{L^T C_\infty^* M}}{\sqrt{(\mathrm{L^T C_\infty^* L})(\mathrm{M^T C_\infty^* M})}}, \mathrm{C}_\infty^* = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Suppose now there is a homography H applied to the world plane and result in an image plane, the relation between the transformed lines and the original lines is given by $\mathrm{L} = H^T L', \mathrm{M} = H^T M'$. In addition, the transformed conic is given by $C_\infty^{*\prime} = \mathrm{H} C_\infty^* \mathrm{H}$. Since $\cos\theta = 0$ for two orthogonal lines in the world plane, we have:

$$\mathrm{L'^T H_a C_\infty^* H_a^T M'} = 0$$

Where $\mathrm{H_a} = \begin{bmatrix} \mathrm{A} & 0 \\ 0 & 1 \end{bmatrix}$ is the homography that can remove the affine distortion. Expand the above equation we get:

$$\begin{bmatrix} l_1' & l_2' & l_3' \end{bmatrix} \begin{bmatrix} \mathrm{AA^T} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} m_1' \\ m_2' \\ m_3' \end{bmatrix} = 0$$

Denote S= AA$^\mathrm{T}$=$\begin{bmatrix} s_{11} & s_{12} \\ s_{12} & s_{22} \end{bmatrix}$, we have:

$$s_{11}m_1'l_1' + s_{12}\ (l_1'm_2'+l_2'm_1') + s_{22}l_2'm_2' = 0$$

Since we only care about ratios, we can set $s_{22} = 1$. As a result, we would need two orthogonal line pairs to solve for S. Once we have S, we can take the singular value decomposition (SVD) on S to get A :

$$\mathrm{S = AA^T = VD^2V^T, A = VDV^T}$$

Some note for programming:

a. The homography now is given by $X_i = H_\mathrm{a}X_w$. Be careful to the mapping direction.
b. When selecting the points to find the homography H_a , the coordinates of these points should correspond to the image plane where the projective distortion is removed.
c. The MATLAB eig function could output negative eigenvalues, we should use the SVD to get A.
d. To remove the projective distortion and the affine distortion, at first the four corner points of the original image is mapped to the world plane using homography ${H_\mathrm{a}}^{-1}H_P$, then the dimension of the output image is determined using the mapped points. Finally, the points in the output image are mapped back to the image using the inverse of ${H_\mathrm{a}}^{-1}H_P$ to find the corresponding pixel values.
e. Poor selection of points could lead to bad results. Select the points carefully.

4. One-Step Method

To remove the projective distortion and the affine distortion at one step, we need to find a general homography:

$$\mathrm{H} = \begin{bmatrix} \mathrm{A} & 0 \\ \mathrm{v^T} & 1 \end{bmatrix}$$

Since the dual conic in the image plane is given by:

$$\mathrm{C}_\infty^{*\prime} = \mathrm{H}\mathrm{C}_\infty^{*}\mathrm{H}^\mathrm{T} = \begin{bmatrix} \mathrm{AA}^\mathrm{T} & \mathrm{Av} \\ \mathrm{v}^\mathrm{T}\mathrm{A}^\mathrm{T} & \mathrm{v}^\mathrm{T}\mathrm{v} \end{bmatrix} = \begin{bmatrix} \mathrm{a} & \mathrm{b}/2 & \mathrm{d}/2 \\ \mathrm{b}/2 & \mathrm{c} & \mathrm{e}/2 \\ \mathrm{d}/2 & \mathrm{e}/2 & \mathrm{f} \end{bmatrix}$$

When there are two lines in the image plane $L' = [l_1' \quad l_2' \quad l_3']$ and $M' = [m_1' \quad m_2' \quad m_3']$ that is orthogonal to each other in the world plane, we have:

$$\mathrm{L'}^\mathrm{T}\mathrm{C}_\infty^{*\prime}\,\mathrm{M'} = 0 = [l_1' \quad l_2' \quad l_3'] \begin{bmatrix} \mathrm{a} & \mathrm{b}/2 & \mathrm{d}/2 \\ \mathrm{b}/2 & \mathrm{c} & \mathrm{e}/2 \\ \mathrm{d}/2 & \mathrm{e}/2 & \mathrm{f} \end{bmatrix} [m_1' \quad m_2' \quad m_3']^\mathrm{T}$$

Since we only care about the ratios, we set $\mathrm{f} = 1$. Therefore, we need five orthogonal line pairs to solve for five unknowns. Once the unknowns are solved, we can again do SVD on $\mathrm{S} = \mathrm{AA}^\mathrm{T} = \begin{bmatrix} \mathrm{a} & \mathrm{b}/2 \\ \mathrm{b}/2 & \mathrm{c} \end{bmatrix}$ to get A matrix, and then solve for matrix v using $\mathrm{v}^\mathrm{T}\mathrm{A}^\mathrm{T} = [\mathrm{d}/2 \quad \mathrm{e}/2]$. By doing this, we get the general homography that can get rid of the projective distortion and affine distortion.

Some note for programming:

a. The homography now is given by $X_i = \mathrm{H}X_w$. Be careful to the mapping direction.
b. When calculating the orthogonal line pairs, normalize these lines before solving for H.
c. When the matrix $\begin{bmatrix} \mathrm{a} & \mathrm{b}/2 & \mathrm{d}/2 \\ \mathrm{b}/2 & \mathrm{c} & \mathrm{e}/2 \\ \mathrm{d}/2 & \mathrm{e}/2 & \mathrm{f} \end{bmatrix}$ is solved, normalize it before applying singular value decomposition.
d. To remove the projective distortion and the affine distortion, at first the four corner points of the original image is mapped to the world plane using homography H^{-1}, then the dimension of the output image is determined using the mapped points. Finally, the points in the output image are mapped back to the image using H to find the corresponding pixel values.

e. It is possible that the dimension of output image is too large or too small. Thus, scale the dimension of the output image according to the original image size before finding the corresponding pixel values.
f. Poor selection of points could lead to bad results. Select the points carefully.

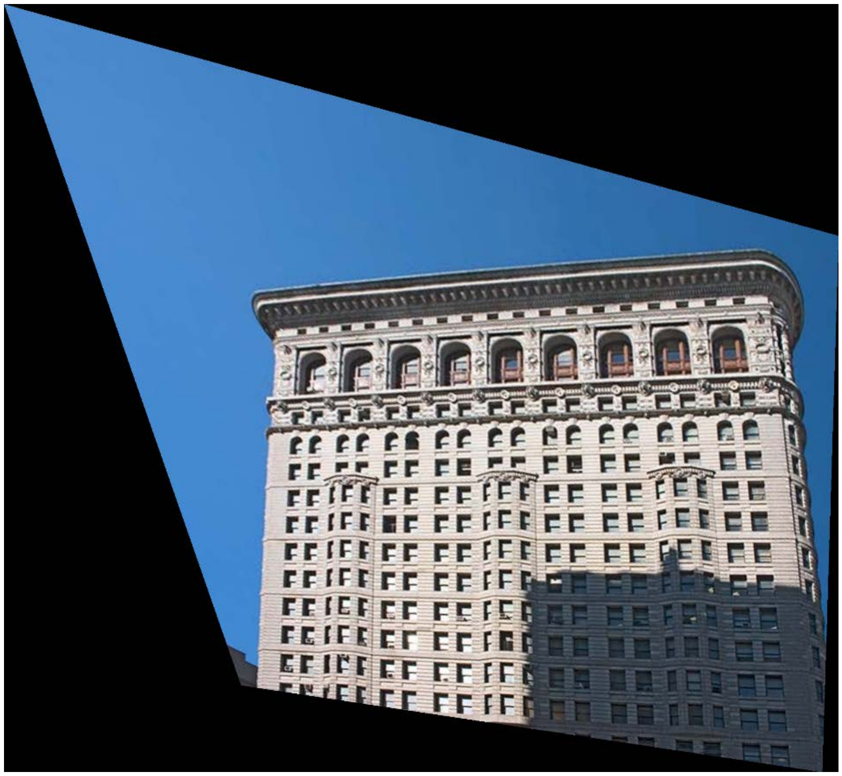
5. Comment

Although the two-steps method seems to be more complicated than the one-step method, it appears to me that the two-step method is more robust than the one-step method. First of all, although the two-step method requires more coding lines, it is more straightforward in concept than the one-step method. Second, when selecting the points to calculate the homography, the one-step method is very sensitive to the points being selected, which means it requires longer time to find the correct points to yield reasonable results. Therefore, I would say the two-steps method is more practical than the one-step approach when we try to remove the projective and affine distortion.

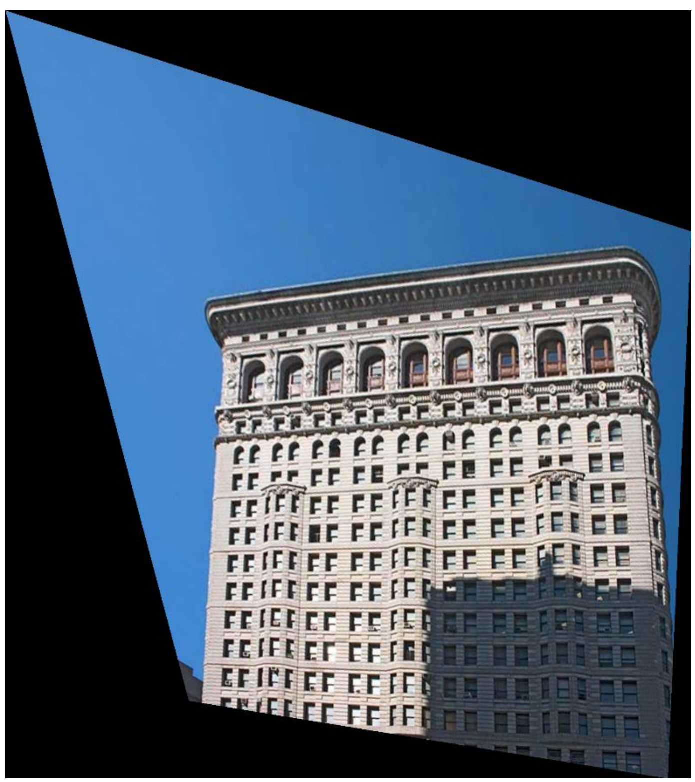
6. Results: Two-steps

Image a – original (green: point to point correspondence; red: parallel lines for vanishing line; yellow: two orthogonal line pairs PQ-QS and PS-QR for removing affine.)

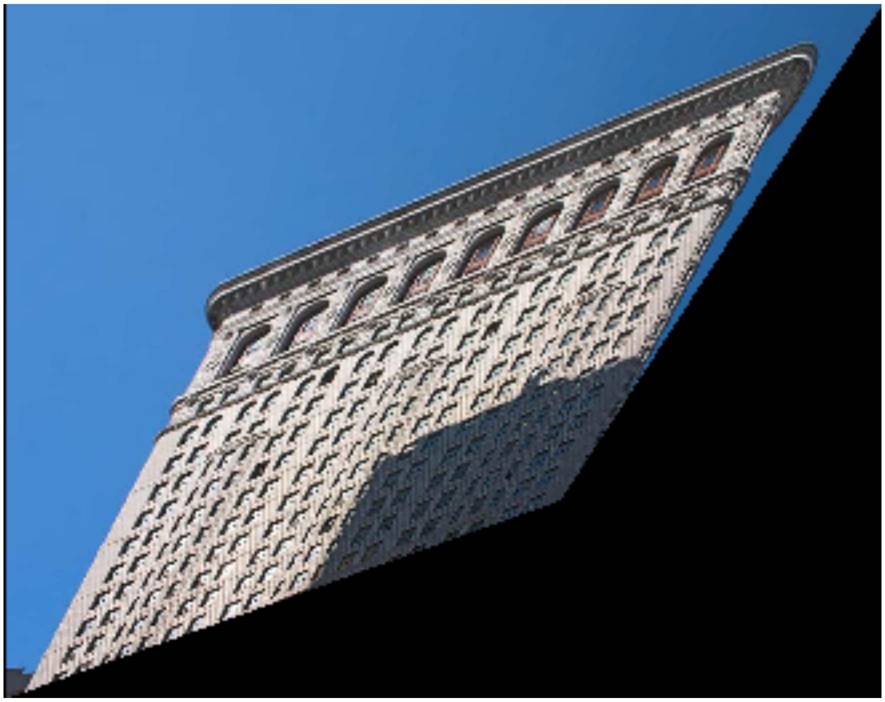
Remove projective (point to point correspondence):



Remove affine:

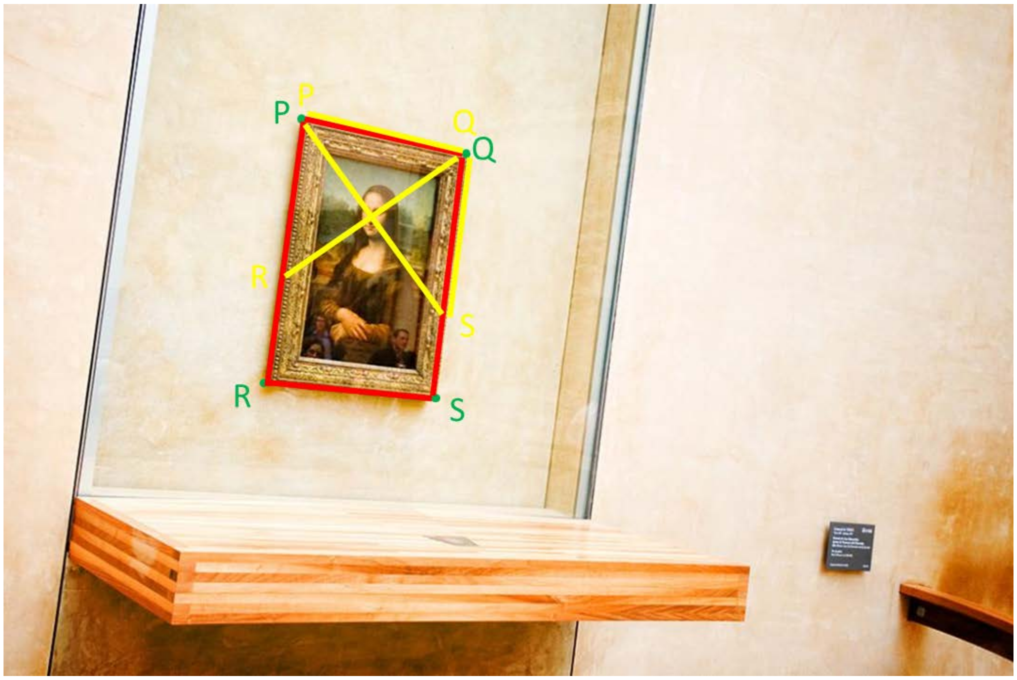


Remove projective (vanishing line):



Remove affine:

Image b - original (green: point to point correspondence; red: parallel lines for vanishing line; yellow: two orthogonal line pairs PQ-QS and PS-QR for removing affine.)



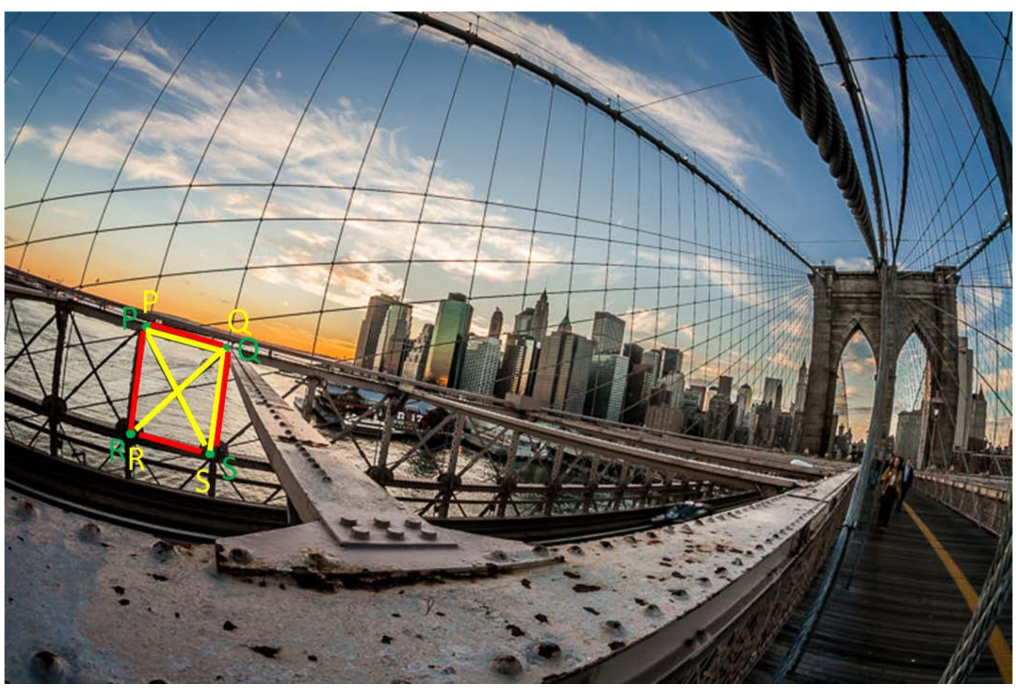
Remove projective (point to point correspondence):

Remove affine:

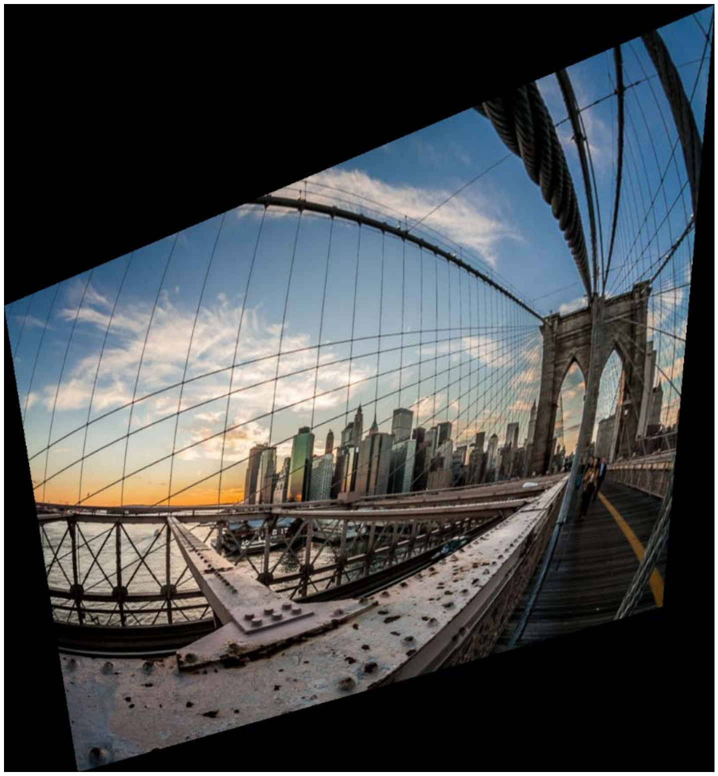
Remove projective (vanishing line):

Remove affine:

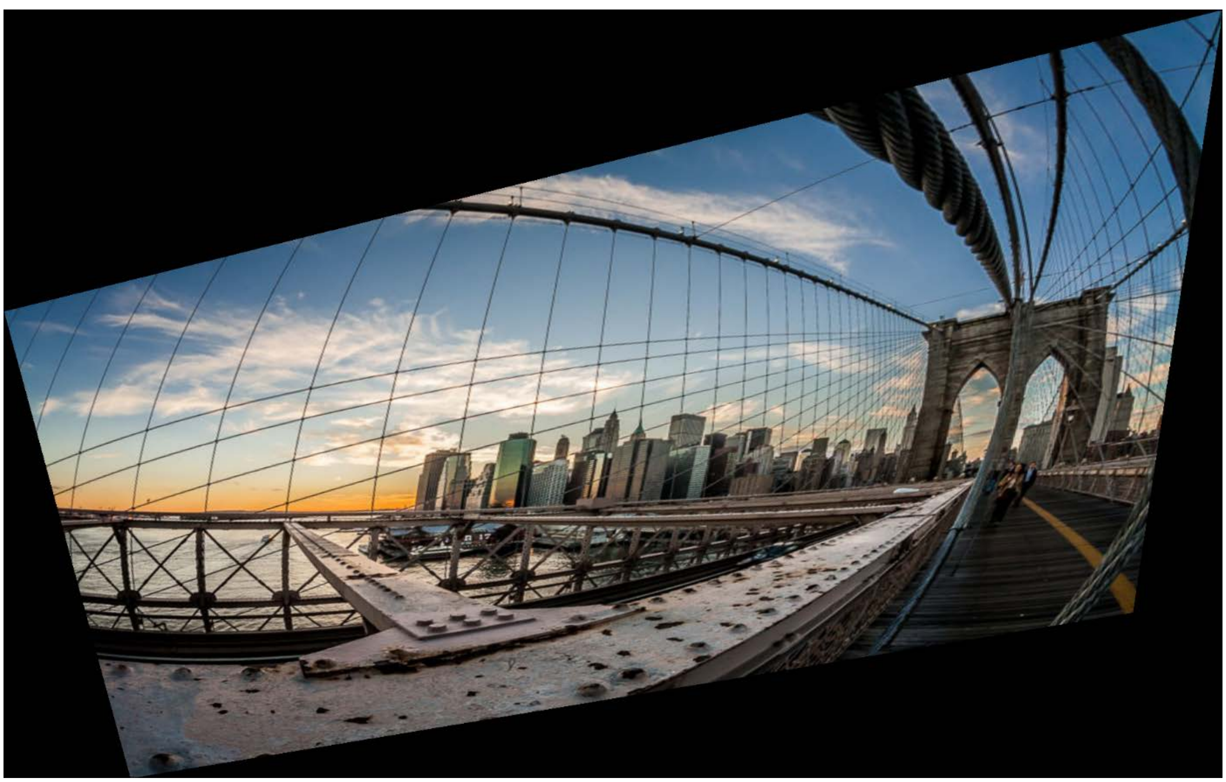
Image c - original (green: point to point correspondence; red: parallel lines for vanishing line; yellow: two orthogonal line pairs PQ-QS and PS-QR for removing affine.)



Remove projective (point to point correspondence):



Remove affine:

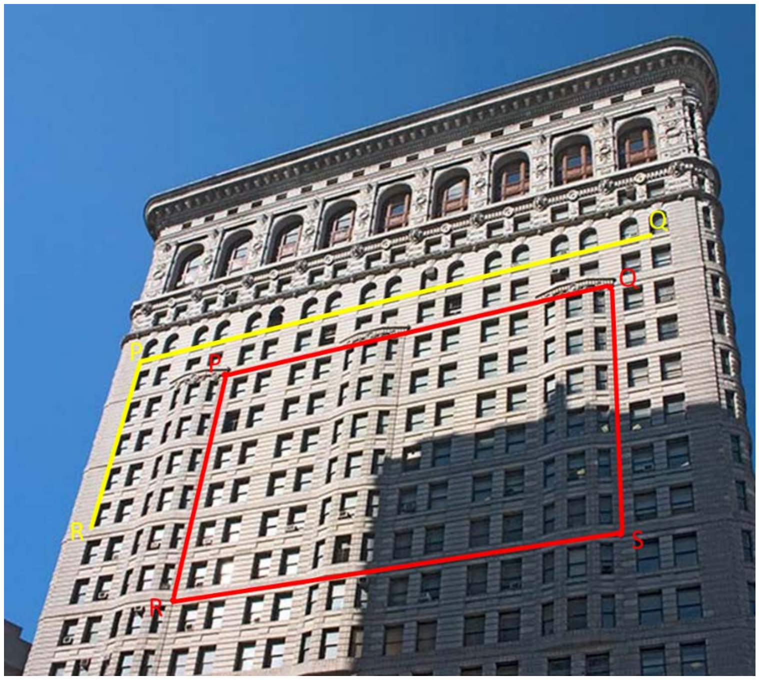


Remove projective (vanishing line):

Remove affine:

7. Results: One-step

Image a - Original (five orthogonal line pairs, red: PQ-PR, PR-RS, RS-SQ, SQ-QP; yellow: PQ-PR)



Remove projective and affine:

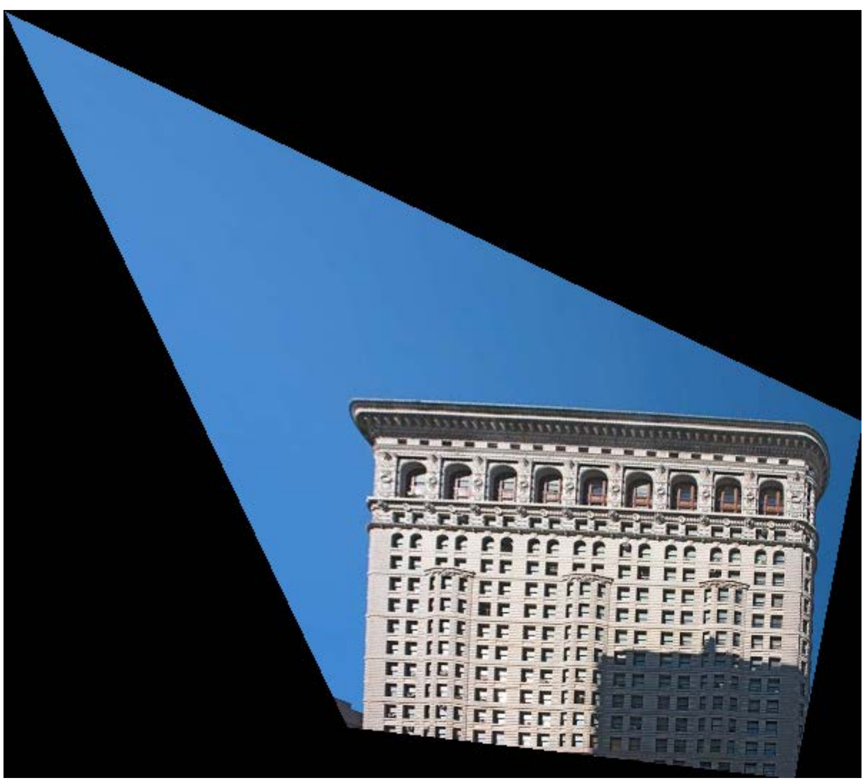
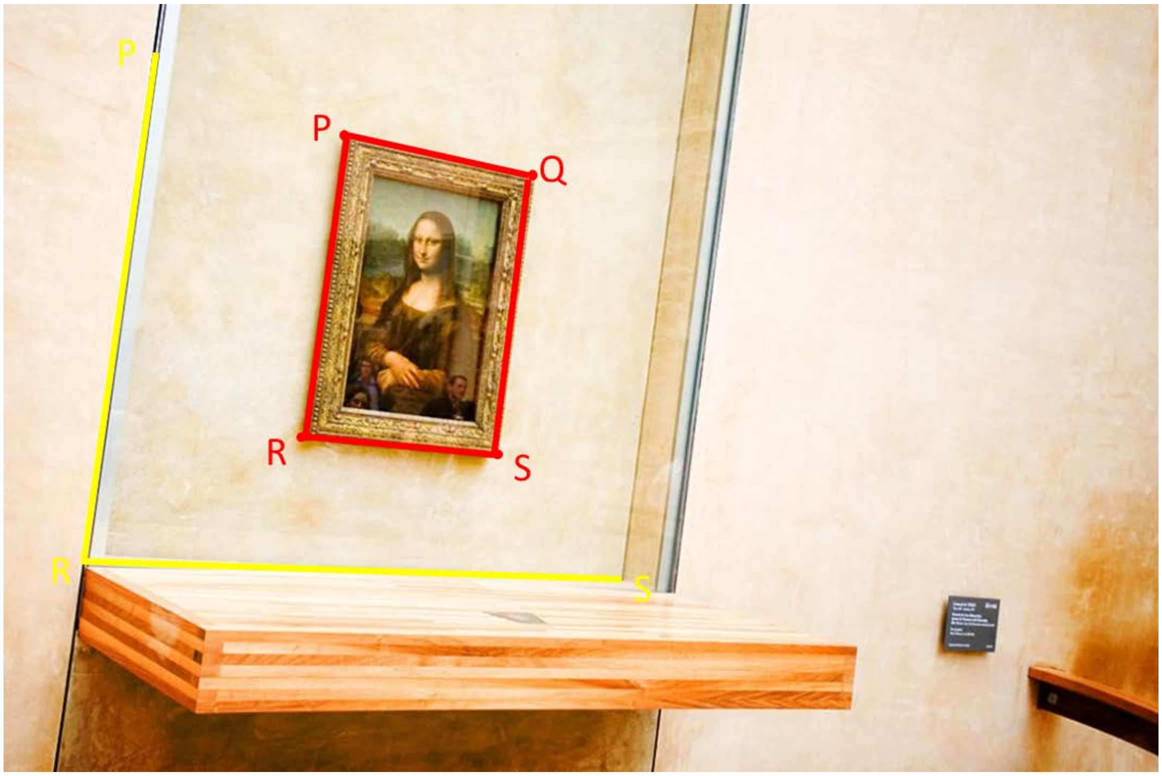
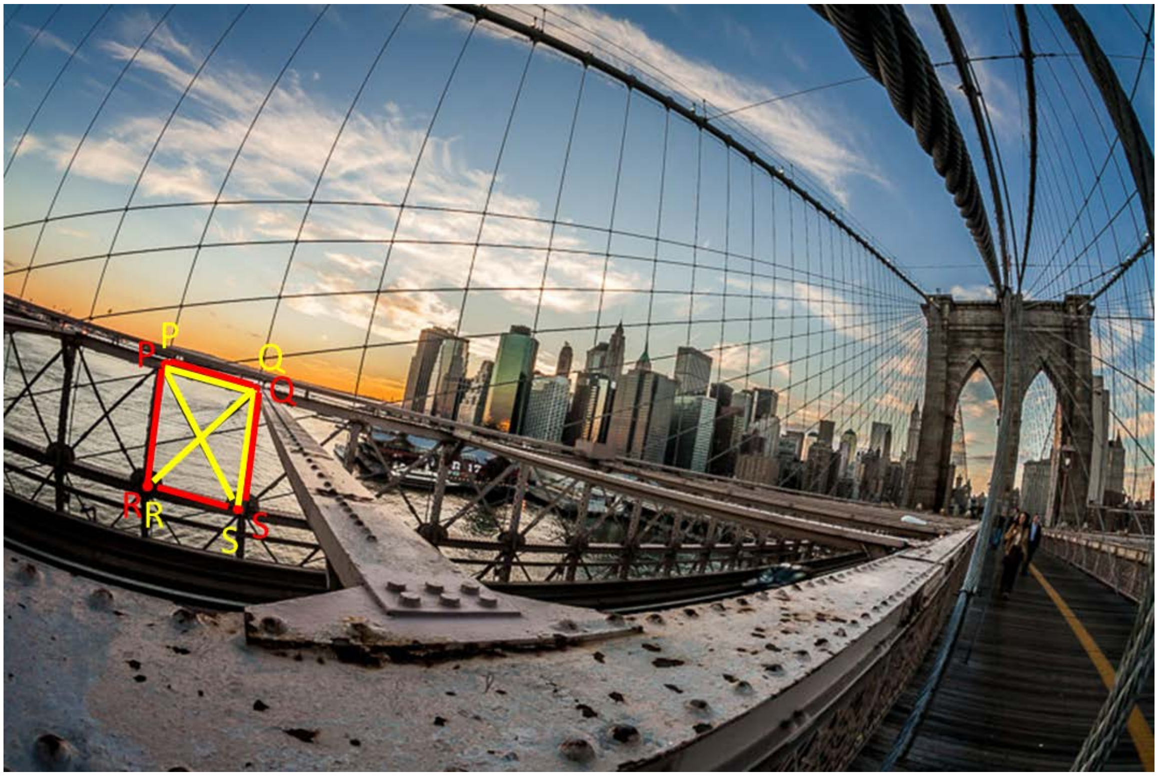


Image b – Original (five orthogonal line pairs, red: PQ-PR, PR-RS, RS-SQ, SQ-QP; yellow: PR-RS)



Remove projective and affine:

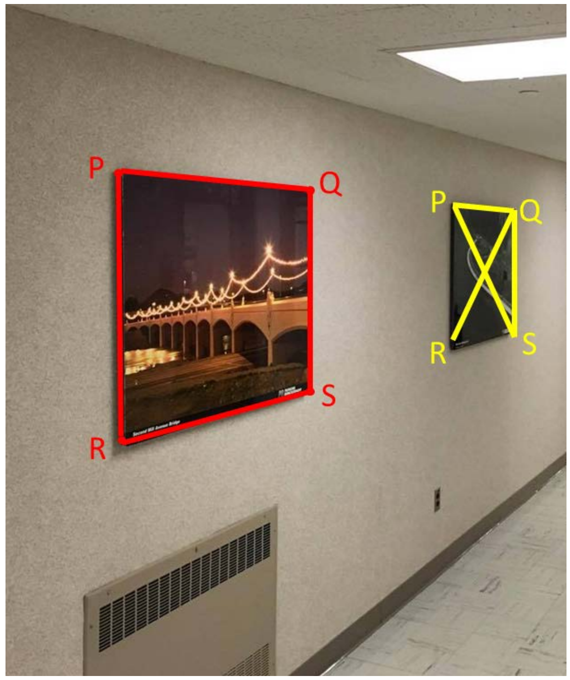
Image c – Original (five orthogonal line pairs, red: PQ-QS, QS-SR, SR-RP, RP-PQ; yellow: PS-QR)



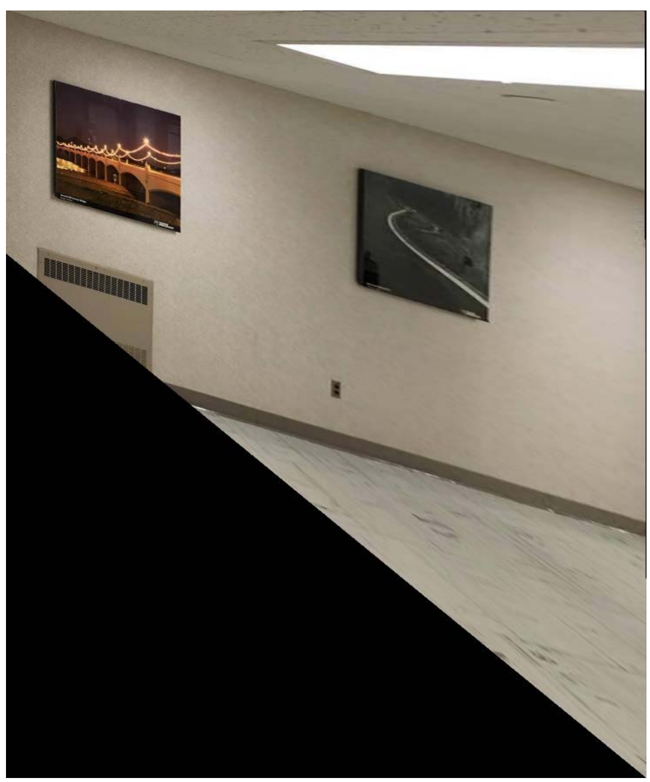
Remove projective and affine:

8. Results: My Own Images (Two-Steps)

Image a – Original (red: parallel lines for vanishing line; yellow: two orthogonal line pairs PQ-QS and PS-QR for removing affine.)



Remove projective (vanishing line):



Remove affine:

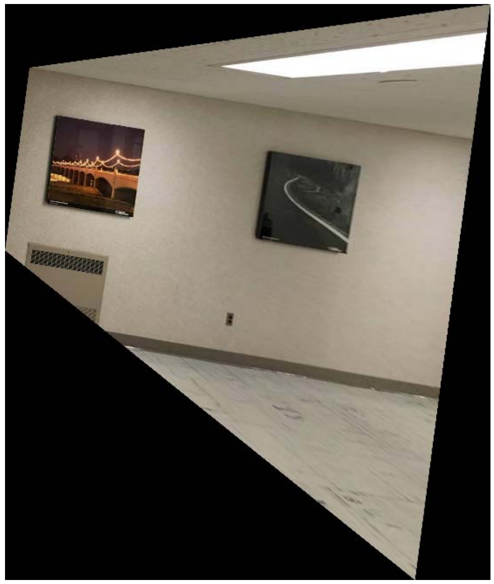


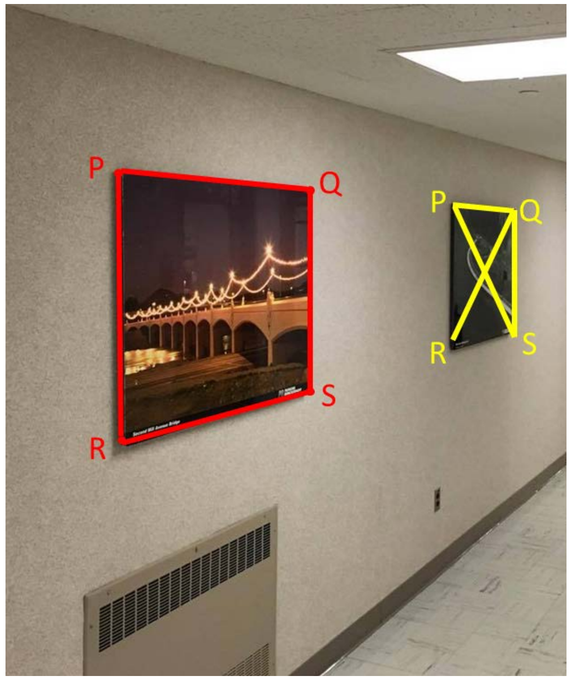
Image b – Original (red: parallel lines for vanishing line; yellow: two orthogonal line pairs PQ-QS and PS-QR for removing affine.)

Remove projective (vanishing line):

Remove affine:

9. Results: My Own Images (One-Step)

Image a – Original (five orthogonal line pairs, red: PQ-QS, QS-SR, SR-RP; yellow: PQ-QS, PS-QR)



Remove projective and affine:

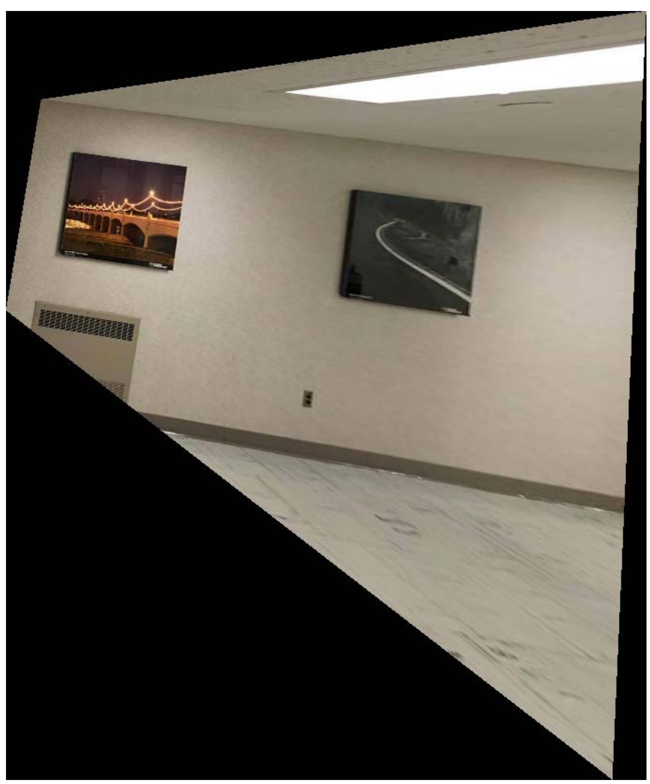


Image b – Original (five orthogonal line pairs, red: PQ-QS, QS-SR, SR-RP; yellow: PQ-QS, PS-QR)

Remove projective and affine:

0.1 Code

0.1.1 Bilinear.m

```
function [ pix_val ] = Bilinear(x,y,image)
% Author: Rih-Teng Wu
% This function output the pixel-value based on bilinear
    interpolation
% x and y: x and y coordinates that may not be intergers
% image: contain the pixel values we want

% ==== first check if x and y exceed the coordinates in
    the mapped image
if (x < 1) || (x > size(image,2))
     pix_val = zeros(1,1,3);
elseif (y < 1) || (y > size(image,1))
     pix_val = zeros(1,1,3);
else
     x1 = floor(x);
     x2 = ceil(x);
     y1 = floor(y);
     y2 = ceil(y);
     deno = (x2-x1)*(y2-y1);% denominator

     pix_val = (x2-x)*(y2-y)/deno*image(y1,x1,:) +...
          (x-x1)*(y2-y)/deno*image(y1,x2,:) +...
          (x2-x)*(y-y1)/deno*image(y2,x1,:) +...
          (x-x1)*(y-y1)/deno*image(y2,x2,:);
end
end
```

0.1.2 calculatehomography.m

```
function [ H ] = calculate_homography(X_i, X_w)
% Author: Rih-Teng Wu
% H: 8x8 homography matrix
% X_i: image coordinates; X_w: world plane coordinates;
    They have the same
% dimensions.
% A*h = B, A and B are given, h is estimated.
% This function return H based on least square estimates

% ==== initial
n = size(X_i,1); % number or corresponding point pairs
```

```
h = zeros(8,1);   % h is of 8x1 unknowns
H = zeros(3,3);
A = zeros(2*n,8);% A is of 2nx8, given
B = zeros(2*n,1);   % B is of 2nx1, given

% ==== fill out B
for z = 1:n
    B(2*z-1) = X_i(z,1);
    B(2*z) = X_i(z,2);
end

% ==== fill out A
for z = 1:n
    A(2*z-1,:) = [X_w(z,:) 1 0 0 0 -X_i(z,1)*X_w(z,1) -
        X_i(z,1)*X_w(z,2)];
    A(2*z,:) = [0 0 0 X_w(z,:) 1 -X_i(z,2)*X_w(z,1) -X_i(
        z,2)*X_w(z,2)];
end

% ==== calculate h using least square estimates just in
    case there is more
% than 4 corresponding point pairs
h = (A'*A)^(-1)*A'*B;

% ==== fill out H
H(1,1) = h(1);
H(1,2) = h(2);
H(1,3) = h(3);
H(2,1) = h(4);
H(2,2) = h(5);
H(2,3) = h(6);
H(3,1) = h(7);
H(3,2) = h(8);
H(3,3) = 1; % since we care only about ratios in
    homogeneous representation
end
```

0.1.3 calculatehomographyaffine.m

```
function [ H ] = calculate_homography_affine(P,Q,Z,R)
% This function output the homography using two
    orthogonal lines in the world plane
% Author: Rih-Teng Wu
% H: 3x3 homography matrix
% H_p: homography that removes projective
```

```
% P,Q,Z,R: 4 points in image that form two set of
    orthogonal line pairs in the world plane
% This function return H based on least square estimates
% C*x = B, C and B are given, x is estimated.

% ==== homogeneous representation
P = [P';1];
Q = [Q';1];
Z = [Z';1];
R = [R';1];

% ==== two orthogonal lines in world plane
l1 = cross(P,Q);
m1 = cross(Q,Z);
l2 = cross(Z,P);
m2 = cross(R,Q);


% ==== fill out A and B
C = [l1(1)*m1(1)  l1(1)*m1(2)+l1(2)*m1(1);l2(1)*m2(1)  l2
    (1)*m2(2)+l2(2)*m2(1)];
B = [-l1(2)*m1(2);-l2(2)*m2(2)];

% ==== calculate x using least square estimates
x = (C'*C)^(-1)*C'*B;

% ==== fill out S
S = [x(1)  x(2);x(2)  1];

% ==== perform SVD on S
[U,D,V] = svd(S);

% ==== calculate A
A = V*sqrt(D)*V';

% ==== obtain H
H = [A  [0;0];0  0  1];
end
```

0.1.4 calculatehomographyonestep.m

```
function [ H ] = calculate_homography_onestep(P)
% This function output the homography using 5 pairs of
    orthogonal lines in the world plane
% Author: Rih-Teng Wu
```

```
% H: 3x3 homography matrix
% P: 11 points in image that form 5 pairs of orthogonal lines in the world plane
% C*x = B, C and B are given, x is estimated.

% ==== homogeneous representation
P2 = zeros(size(P,1),3);
for i = 1:size(P,1)
    P2(i,:) = [P(i,:) 1];
end
% ==== find 5 pairs of orthogonal lines in the world plane
L1 = cross(P2(1,:)',P2(2,:)'); L1 = L1./max(L1);
M1 = cross(P2(3,:)',P2(4,:)'); M1 = M1./max(M1);
L2 = cross(P2(5,:)',P2(6,:)'); L2 = L2./max(L2);
M2 = cross(P2(7,:)',P2(8,:)'); M2 = M2./max(M2);
L3 = cross(P2(9,:)',P2(10,:)'); L3 = L3./max(L3);
M3 = cross(P2(11,:)',P2(12,:)'); M3 = M3./max(M3);
L4 = cross(P2(13,:)',P2(14,:)'); L4 = L4./max(L4);
M4 = cross(P2(15,:)',P2(16,:)'); M4 = M4./max(M4);
L5 = cross(P2(17,:)',P2(18,:)'); L5 = L5./max(L5);
M5 = cross(P2(19,:)',P2(20,:)'); M5 = M5./max(M5);

% ==== fill out C and B
C = [L1(1)*M1(1) (L1(1)*M1(2)+L1(2)*M1(1))/2 L1(2)*M1(2) (L1(1)*M1(3)+L1(3)*M1(1))/2 (L1(2)*M1(3)+L1(3)*M1(2))/2;
    L2(1)*M2(1) (L2(1)*M2(2)+L2(2)*M2(1))/2 L2(2)*M2(2) (L2(1)*M2(3)+L2(3)*M2(1))/2 (L2(2)*M2(3)+L2(3)*M2(2))/2;
    L3(1)*M3(1) (L3(1)*M3(2)+L3(2)*M3(1))/2 L3(2)*M3(2) (L3(1)*M3(3)+L3(3)*M3(1))/2 (L3(2)*M3(3)+L3(3)*M3(2))/2;
    L4(1)*M4(1) (L4(1)*M4(2)+L4(2)*M4(1))/2 L4(2)*M4(2) (L4(1)*M4(3)+L4(3)*M4(1))/2 (L4(2)*M4(3)+L4(3)*M4(2))/2;
    L5(1)*M5(1) (L5(1)*M5(2)+L5(2)*M5(1))/2 L5(2)*M5(2) (L5(1)*M5(3)+L5(3)*M5(1))/2 (L5(2)*M5(3)+L5(3)*M5(2))/2];

B = [-L1(3)*M1(3);
    -L2(3)*M2(3);
    -L3(3)*M3(3);
    -L4(3)*M4(3);
    -L5(3)*M5(3)];
```

```
% ==== calculate x
x = C^(-1)*B;
x = x./max(x(:)); % Normalize x, very important!

% ==== fill out S and solve for A using SVD of S = AA'
S = [x(1) x(2)/2;x(2)/2 x(3)];
[U,D,V] = svd(S);

% ==== calculate A
A = V*sqrt(D)*V';

% ==== solve for v',
v = [x(4)/2 x(5)/2]*(A')^(-1);

% ==== obtain H
H = [A [0;0];v 1];
end
```

0.1.5 calculatehomographyvanish.m

```
function [ H ] = calculate_homography_vanish(P,Q,S,R)
% This function output the homography using vanishing
    line approach
% Author: Rih-Teng Wu
% H: homography
% P,Q,S,R: image coordinates (x_i, y_i)
% L1 = P X Q; L2 = Q X S; L3 = S X R; L4= R X P;

P = [P';1]; % homogeneous representation
Q = [Q';1];
S = [S';1];
R = [R';1];

L1 = cross(P,Q);
L2 = cross(Q,S);
L3 = cross(S,R);
L4 = cross(R,P);

VP1 = cross(L1,L3); % Vanishing point 1
VP2 = cross(L2,L4); % Vanishing point 2
VL = cross(VP1,VP2);% Vanishing line
% ====== normalize
VL = VL./max(VL);

H = [1 0 0; 0 1 0; VL']; % homography
```

```
end
```

0.1.6 $\mathbf{ECE}_661_HW3_1step_method_modified.m$

```
% ECE661 HW3 Task One-step method
% Rih-Teng Wu
clc; clear all; close all;

Case = 'c';% a: image is flatiron.jpg; b: image is
    monalisa.jpg; c: image is wideangle.jpg;
switch Case
    case 'a'
        image = imread('flatiron.jpg');
        figure; imshow(image);
    case 'b'
        image = imread('monalisa.jpg');
        figure; imshow(image);
    case 'c'
        image = imread('wideangle.jpg');
        figure; imshow(image);
end

% ====== image plane coordinates (x_i, y_i) of 5 pairs of
     orthogonal lines (Fig.a, Fig.b, Fig.c)
switch Case
    case 'a'
        P_1 = [177          297]; P_2 = [484         225];
        P_3 = [177          297]; P_4 = [134.78261
              476.45963];
        P_5 = [177          297]; P_6 = [134.78261
              476.45963];
        P_7 = [134.78261              476.45963]; P_8 =
            [492.07453     425.09938];
        P_9 = [134.78261              476.45963]; P_10 =
            [492.07453              425.09938];
        P_11 = [492.07453             425.09938]; P_12 =
            [484           225];
        P_13 = [492.07453             425.09938];P_14 = [484
                   225];
        P_15 = [484          225];P_16 = [177          297];
        P_17 = [109.57143             284.32298];P_18 =
            [515.91304     184.45342];
        P_19 = [109.57143             284.32298];P_20 =
            [69.53416      417.18012];
```

```
        P_i = [P_1;P_2;P_3;P_4;P_5;P_6;P_7;P_8;P_9;P_10;
            P_11;P_12;P_13;P_14;P_15;P_16;P_17;P_18;P_19;
               P_20];

    case 'b'
        P_1 = [289          108];  P_2 = [447          144];
        P_3 = [289          108];  P_4 = [254          367];
        P_5 = [289          108];  P_6 = [254          367];
        P_7 = [254          367];  P_8 = [417          383];
        P_9 = [254          367];  P_10 = [417         383];
        P_11 = [417         383];  P_12 = [447         144];
        P_13 = [417         383];  P_14 = [447         144];
        P_15 = [447         144];  P_16 = [289         108];
        P_17 = [126          47];  P_18 = [67          475];
        P_19 = [67          475];  P_20 = [523         487];

        P_i = [P_1;P_2;P_3;P_4;P_5;P_6;P_7;P_8;P_9;P_10;
            P_11;P_12;P_13;P_14;P_15;P_16;P_17;P_18;P_19;
               P_20];

    case 'c'
        P_1 = [107 231]; P_2 = [165 247];
        P_3 = [165 247]; P_4 = [154 328];
        P_5 = [165 247]; P_6 = [154 328];
        P_7 = [154 328]; P_8 = [94 312];
        P_9 = [154 328]; P_10 = [94 312];
        P_11 = [94 312]; P_12 = [107 231];
        P_13 = [94 312]; P_14 = [107 231];
        P_15 = [107 231]; P_16 = [165 247];
        P_17 = [107 231]; P_18 = [154 328];
        P_19 = [165 247]; P_20 = [94 312];

        P_i = [P_1;P_2;P_3;P_4;P_5;P_6;P_7;P_8;P_9;P_10;
            P_11;P_12;P_13;P_14;P_15;P_16;P_17;P_18;P_19;
               P_20];

end

% ====== calculate homography to remove both projective
    and affine distortion, using 5 pairs of orthogonal
    lines in world plane
H = calculate_homography_onestep(P_i); % X_i = H*X_w
H_inv = H^(-1); % H^(-1)*X_i = X_w
```

```
% ====== find the corner points in image, and map these
    points to world plane
CP_i = [1  1];
CQ_i = [size(image,2)  1];
CS_i = [size(image,2)  size(image,1)];
CR_i = [1  size(image,1)];

CP_w = H_inv*[CP_i';1]; CP_w(1) = round(CP_w(1)/CP_w(3));
     CP_w(2) = round(CP_w(2)/CP_w(3));
CQ_w = H_inv*[CQ_i';1]; CQ_w(1) = round(CQ_w(1)/CQ_w(3));
     CQ_w(2) = round(CQ_w(2)/CQ_w(3));
CS_w = H_inv*[CS_i';1]; CS_w(1) = round(CS_w(1)/CS_w(3));
     CS_w(2) = round(CS_w(2)/CS_w(3));
CR_w = H_inv*[CR_i';1]; CR_w(1) = round(CR_w(1)/CR_w(3));
     CR_w(2) = round(CR_w(2)/CR_w(3));

% ====== find the boundary in the output where we are
    going to find the pixel values in our image
xmin = min([CP_w(1) CQ_w(1) CS_w(1) CR_w(1)]);
xmax = max([CP_w(1) CQ_w(1) CS_w(1) CR_w(1)]);
ymin = min([CP_w(2) CQ_w(2) CS_w(2) CR_w(2)]);
ymax = max([CP_w(2) CQ_w(2) CS_w(2) CR_w(2)]);
width = xmax-xmin;
height = ymax-ymin;

% ====== find the proper scale for output image
scale_x = size(image,2)/width;
scale_y = size(image,1)/height;
width_scale = round(width*scale_x);
height_scale = round(height*scale_y);
% ====== initial output image
output = zeros(height_scale,width_scale,3);

% ====== map the points in the output frame to our image
    using H_p, then use bilinear
% interpolation to determine the corresponding pixel
    value
image_pixel = double(image);    % pixel values we want

for h = 1:height_scale
    for w = 1:width_scale
        homo_temp = H*[(w/scale_x+xmin-1);(h/scale_y+ymin
            -1);1]; % The coordinates must be scaled back
        x_i_homo = homo_temp(1)/homo_temp(3); %
            homogeneous representation
        y_i_homo = homo_temp(2)/homo_temp(3);
```

```
            pixel_value = Bilinear(x_i_homo,y_i_homo,
                image_pixel);
            output(h,w,:) = pixel_value;
        end
end
output = uint8(output);
figure; imshow(output);
imwrite(output, ['task2_OneStep_' Case '.tif']);
```

0.1.7 ECE$_6 61_H W3_1 step_m ethod_m odified_o wn.m$

```
% ECE661 HW3 Task One-step method - Own Images
% Rih-Teng Wu
clc; clear all; close all;

Case = 'b';% a: image a ; b: image b
switch Case
    case 'a'
        image = imread('own_a_cropped.tif');
        figure; imshow(image);
    case 'b'
        image = imread('own_b_cropped.tif');
        figure; imshow(image);
end

% ====== image plane coordinates (x_i, y_i) of 5 pairs of
    orthogonal lines (Fig.a, Fig.b)
switch Case
    case 'a'
        P_1 = [143 203]; P_2 = [369 223];
        P_3 = [369 223]; P_4 = [370 470];
        P_5 = [369 223]; P_6 = [370 470];
        P_7 = [370 470]; P_8 = [144 537];
        P_9 = [370 470]; P_10 = [144 537];
        P_11 = [144 537]; P_12 = [143 203];
        P_13 = [546 242];P_14 = [619 249];
        P_15 = [619 249];P_16 = [617 399];
        P_17 = [546 242];P_18 = [617 399];
        P_19 = [619 249];P_20 = [544 419];

        P_i = [P_1;P_2;P_3;P_4;P_5;P_6;P_7;P_8;P_9;P_10;
            P_11;P_12;P_13;P_14;P_15;P_16;P_17;P_18;P_19;
                P_20];

    case 'b'
```

```
        P_1 = [308 93]; P_2 = [563 43];
        P_3 = [563 43]; P_4 = [529 938];
        P_5 = [563 43]; P_6 = [529 938];
        P_7 = [529 938]; P_8 = [310 783];
        P_9 = [529 938]; P_10 = [310 783];
        P_11 = [310 783];P_12 = [308 93];
        P_13 = [51 144];P_14 = [262 103];
        P_15 = [262 103];P_16 = [269 488];
        P_17 = [51 144];P_18 = [269 488];
        P_19 = [262 103];P_20 = [61 420];

        P_i = [P_1;P_2;P_3;P_4;P_5;P_6;P_7;P_8;P_9;P_10;
            P_11;P_12;P_13;P_14;P_15;P_16;P_17;P_18;P_19;
               P_20];
end

% ====== calculate homography to remove both projective
    and affine distortion , using 5 pairs of orthogonal
    lines in world plane
H = calculate_homography_onestep(P_i); % X_i = H*X_w
H_inv = H^(-1); % H^(-1)*X_i = X_w

% ====== find the corner points in image, and map these
    points to world plane
CP_i = [1 1];
CQ_i = [size(image,2) 1];
CS_i = [size(image,2) size(image,1)];
CR_i = [1 size(image,1)];

CP_w = H_inv*[CP_i';1]; CP_w(1) = round(CP_w(1)/CP_w(3));
     CP_w(2) = round(CP_w(2)/CP_w(3));
CQ_w = H_inv*[CQ_i';1]; CQ_w(1) = round(CQ_w(1)/CQ_w(3));
     CQ_w(2) = round(CQ_w(2)/CQ_w(3));
CS_w = H_inv*[CS_i';1]; CS_w(1) = round(CS_w(1)/CS_w(3));
     CS_w(2) = round(CS_w(2)/CS_w(3));
CR_w = H_inv*[CR_i';1]; CR_w(1) = round(CR_w(1)/CR_w(3));
     CR_w(2) = round(CR_w(2)/CR_w(3));

% ====== find the boundary in the output where we are
    going to find the pixel values in our image
xmin = min([CP_w(1) CQ_w(1) CS_w(1) CR_w(1)]);
xmax = max([CP_w(1) CQ_w(1) CS_w(1) CR_w(1)]);
ymin = min([CP_w(2) CQ_w(2) CS_w(2) CR_w(2)]);
ymax = max([CP_w(2) CQ_w(2) CS_w(2) CR_w(2)]);
width = xmax-xmin;
height = ymax-ymin;
```

```
% ====== find the proper scale for output image
scale_x = size(image,2)/width;
scale_y = size(image,1)/height;
width_scale = round(width*scale_x);
height_scale = round(height*scale_y);
% ====== initial output image
output = zeros(height_scale,width_scale,3);

% ====== map the points in the output frame to our image
    using H_p, then use bilinear
% interpolation to determine the corresponding pixel
    value
image_pixel = double(image);    % pixel values we want

for h = 1:height_scale
    for w = 1:width_scale
        homo_temp = H*[(w/scale_x+xmin-1);(h/scale_y+ymin
            -1);1]; % The coordinates must be scaled back
        x_i_homo = homo_temp(1)/homo_temp(3); %
            homogeneous representation
        y_i_homo = homo_temp(2)/homo_temp(3);
        pixel_value = Bilinear(x_i_homo,y_i_homo,
            image_pixel);
        output(h,w,:) = pixel_value;
    end
end
output = uint8(output);
figure; imshow(output);
imwrite(output, ['Own_OneStep_' Case '.tif']);
```

0.1.8 $\mathbf{ECE}_{6}61_{H}W3_{2}steps_{m}ethod.m$

```
% ECE661 HW3 Task 2-step method
% Rih-Teng Wu
clc; clear all; close all;

Remove_projective = 'vanish'; % 'point_point': estimate
   homography using point to point correspondence
                                      % 'vanish': estimate
                                          homography using
                                          vanishing line

Case = 'c';% a: image is flatiron.jpg; b: image is
    monalisa.jpg; c: image is wideangle.jpg;
```

```
Case_2 = '2'; % '1': remove projective distortion only;
    '2': remove both projective and affine distortion
switch Case
    case 'a'
        image = imread('flatiron.jpg');
        figure; imshow(image);
    case 'b'
        image = imread('monalisa.jpg');
        figure; imshow(image);
    case 'c'
        image = imread('wideangle.jpg');
        figure; imshow(image);
end

% ====== world plane coordinates (x_w, y_w) (Fig.a, Fig.b
    , Fig.c)
switch Case
    case 'a'
        P_w = [0 0];Q_w = [100 0];S_w = [100 150];R_w =
            [0 150]; % coordinate obtained from real world
             coordinate (assume 1 pixel = 1 cm)
    case 'b'
        P_w = [0 0];Q_w = [60 0];S_w = [60 100];R_w = [0
            100];
    case 'c'
        P_w = [0 0];Q_w = [70 0];S_w = [70 120];R_w = [0
            120];
end
% ====== image plane coordinates (x_i, y_i) (Fig.a, Fig.b
    , Fig.c)
switch Case
    case 'a'
        P_i = [230 285];Q_i = [263 278];S_i = [254 332];
            R_i = [220 339]; % used for point-point
            correspondence, and for affine
        P_v = [125 180];Q_v = [554 58];S_v = [594 341];
            R_v = [48 428]; % used for vanishing line
        P_i2 = [329 265];Q_i2 = [418 245];S_i2 = [417
            325];R_i2 = [320 342];% used for affine
    case 'b'
        P_i = [290 109];Q_i = [451 144];S_i = [418 383];
            R_i = [255 366]; % used for point-point
            correspondence
        P_v = [290 109];Q_v = [451 144];S_v = [418 383];
            R_v = [255 366]; % used for vanishing line
```

```
        P_i2 = [290 109];Q_i2 = [451 144];S_i2 = [430
            295];R_i2 = [267 273];% used for affine
    case 'c'
        P_i = [107 231];Q_i = [165 247];S_i = [154 328];
            R_i = [94 312]; % used for point-point
            correspondence
        P_v = [107 231];Q_v = [165 247];S_v = [154 328];
            R_v = [94 312]; % used for vanishing line
        P_i2 = [107 231];Q_i2 = [165 247];S_i2 = [154
            328];R_i2 = [94 312];% used for affine
end

% ====== calculate homography to remove projective
    distortion, using X_w = H*X_i
switch Remove_projective
    case 'point_point'
        X_i = [P_i;Q_i;S_i;R_i];
        X_w = [P_w;Q_w;S_w;R_w];
        H_p = calculate_homography(X_w, X_i); %
            homoghraphy to remove the projective
            distortion
        H_p_inv = H_p^(-1); % calculate inverse of H_p
    case 'vanish'
        H_p = calculate_homography_vanish(P_v,Q_v,S_v,R_v
            ); % homoghraphy to remove the projective
            distortion
        H_p_inv = H_p^(-1); % calculate inverse of H_p

end

% ====== calculate the point coordinates after projective
     distortion
CP_i = [1 1]; % four corner points in the original image
CQ_i = [size(image,2) 1];
CS_i = [size(image,2) size(image,1)];
CR_i = [1 size(image,1)];

if Case_2 == '2'
    % ==== calculate xmin and ymin
    CP_h = H_p*[CP_i';1]; CP_h(1) = round(CP_h(1)/CP_h(3)
        ); CP_h(2) = round(CP_h(2)/CP_h(3));
    CQ_h = H_p*[CQ_i';1]; CQ_h(1) = round(CQ_h(1)/CQ_h(3)
        ); CQ_h(2) = round(CQ_h(2)/CQ_h(3));
    CS_h = H_p*[CS_i';1]; CS_h(1) = round(CS_h(1)/CS_h(3)
        ); CS_h(2) = round(CS_h(2)/CS_h(3));
```

```
    CR_h = H_p*[CR_i';1]; CR_h(1) = round(CR_h(1)/CR_h(3)
        ); CR_h(2) = round(CR_h(2)/CR_h(3));

    xmin_h = min([CP_h(1) CQ_h(1) CS_h(1) CR_h(1)]);
    ymin_h = min([CP_h(2) CQ_h(2) CS_h(2) CR_h(2)]);

    % ==== point coordinates after removing affine, x_h =
        H_p*x - xmin
    temp = H_p*[P_i2 1]'; P_o = double(int64([temp(1)
        temp(2)]./temp(3) - [xmin_h ymin_h]));
    temp = H_p*[Q_i2 1]'; Q_o = double(int64([temp(1)
        temp(2)]./temp(3) - [xmin_h ymin_h]));
    temp = H_p*[S_i2 1]'; S_o = double(int64([temp(1)
        temp(2)]./temp(3) - [xmin_h ymin_h]));
    temp = H_p*[R_i2 1]'; R_o = double(int64([temp(1)
        temp(2)]./temp(3) - [xmin_h ymin_h]));

    % ==== calculate homography to remove affine
        distortion, using two orthogonal lines in world
        plane
    H_a = calculate_homography_affine(P_o,Q_o,S_o,R_o); %
        X_i = H_a*X_w
    H_a_inv = H_a^(-1); % H_a^(-1)*X_i = X_w

end

% ====== determine the homography that we would like to
    use
switch Case_2
    case '1' % remove projective only
        H = H_p;
        H_inv = H^(-1);
    case '2' % remove both projective and affine
        H = H_a_inv*H_p;
        H_inv = H^(-1);
end

% ====== find the corner points in image, and map these
    points to world plane
CP_w = H*[CP_i';1]; CP_w(1) = round(CP_w(1)/CP_w(3));
    CP_w(2) = round(CP_w(2)/CP_w(3));
CQ_w = H*[CQ_i';1]; CQ_w(1) = round(CQ_w(1)/CQ_w(3));
    CQ_w(2) = round(CQ_w(2)/CQ_w(3));
CS_w = H*[CS_i';1]; CS_w(1) = round(CS_w(1)/CS_w(3));
    CS_w(2) = round(CS_w(2)/CS_w(3));
```

```
CR_w = H*[CR_i';1]; CR_w(1) = round(CR_w(1)/CR_w(3));
    CR_w(2) = round(CR_w(2)/CR_w(3));

% ====== find the boundary in the output where we are
    going to find the pixel values in our image
xmin = min([CP_w(1) CQ_w(1) CS_w(1) CR_w(1)]);
xmax = max([CP_w(1) CQ_w(1) CS_w(1) CR_w(1)]);
ymin = min([CP_w(2) CQ_w(2) CS_w(2) CR_w(2)]);
ymax = max([CP_w(2) CQ_w(2) CS_w(2) CR_w(2)]);
width = xmax-xmin;
height = ymax-ymin;

% ====== initial output image
output = zeros(height,width,3);

% ====== map the points in the output frame to our image
    using H_p, then use bilinear
% interpolation to determine the corresponding pixel
    value
image_pixel = double(image);    % pixel values we want

for h = 1:height
    for w = 1:width
        homo_temp = H_inv*[w+xmin-1;h+ymin-1;1];
        x_i_homo = homo_temp(1)/homo_temp(3); %
            homogeneous representation
        y_i_homo = homo_temp(2)/homo_temp(3);
        pixel_value = Bilinear(x_i_homo,y_i_homo,
            image_pixel);
        output(h,w,:) = pixel_value;
    end
end
output = uint8(output);
figure; imshow(output);
imwrite(output, ['task2_TwoStep_' Remove_projective '_'
    Case '_' Case_2 '.tif']);
```

0.1.9 ECE$_{6}61_{H}W3_{2}steps_{m}ethod_{o}wn.m$

```
% ECE661 HW3 Task 2-step method
% Remove projective error based on vanishing line method
% Rih-Teng Wu
clc; clear all; close all;
```

```
Case = 'b';% a: image is flatiron.jpg; b: image is
    monalisa.jpg;
Case_2 = '2'; % '1': remove projective distortion only;
    '2': remove both projective and affine distortion
switch Case
    case 'a'
        image = imread('own_a_cropped.tif'); %
            160:975,220:900
        figure; imshow(image);
    case 'b'
        image = imread('own_b_cropped.tif');
        figure; imshow(image);

end

% ====== image plane coordinates (x_i, y_i) (Fig.a, Fig.b
    )
switch Case
    case 'a'
        P_v = [142 201];Q_v = [369 223];S_v = [370 470];
            R_v = [144 537]; % used for vanishing line
        P_i2 = [546 242];Q_i2 = [619 249];S_i2 = [617
            399];R_i2 = [544 419];% used for affine
    case 'b'
        P_v = [310 93];Q_v = [564 40];S_v = [530 935];R_v
            = [310 782]; % used for vanishing line
        P_i2 = [52 141];Q_i2 = [262 101];S_i2 = [271
            485];R_i2 = [64 421];% used for affine
end

% ====== calculate homography to remove projective
    distortion, using X_w = H*X_i
H_p = calculate_homography_vanish(P_v,Q_v,S_v,R_v); %
    homoghraphy to remove the projective distortion
H_p_inv = H_p^(-1); % calculate inverse of H_p

% ====== calculate the point coordinates after projective
     distortion
CP_i = [1 1]; % four corner points in the original image
CQ_i = [size(image,2) 1];
CS_i = [size(image,2) size(image,1)];
CR_i = [1 size(image,1)];

if Case_2 == '2'
    % ==== calculate xmin and ymin
```

```
    CP_h = H_p*[CP_i';1]; CP_h(1) = round(CP_h(1)/CP_h(3)
        ); CP_h(2) = round(CP_h(2)/CP_h(3));
    CQ_h = H_p*[CQ_i';1]; CQ_h(1) = round(CQ_h(1)/CQ_h(3)
        ); CQ_h(2) = round(CQ_h(2)/CQ_h(3));
    CS_h = H_p*[CS_i';1]; CS_h(1) = round(CS_h(1)/CS_h(3)
        ); CS_h(2) = round(CS_h(2)/CS_h(3));
    CR_h = H_p*[CR_i';1]; CR_h(1) = round(CR_h(1)/CR_h(3)
        ); CR_h(2) = round(CR_h(2)/CR_h(3));

    xmin_h = min([CP_h(1) CQ_h(1) CS_h(1) CR_h(1)]);
    ymin_h = min([CP_h(2) CQ_h(2) CS_h(2) CR_h(2)]);

    % ==== point coordinates after removing affine, x_h =
         H_p*x - xmin
    temp = H_p*[P_i2 1]'; P_o = double(int64([temp(1)
        temp(2)]./temp(3) - [xmin_h ymin_h]));
    temp = H_p*[Q_i2 1]'; Q_o = double(int64([temp(1)
        temp(2)]./temp(3) - [xmin_h ymin_h]));
    temp = H_p*[S_i2 1]'; S_o = double(int64([temp(1)
        temp(2)]./temp(3) - [xmin_h ymin_h]));
    temp = H_p*[R_i2 1]'; R_o = double(int64([temp(1)
        temp(2)]./temp(3) - [xmin_h ymin_h]));

    % ==== calculate homography to remove affine
        distortion, using two orthogonal lines in world
        plane
    H_a = calculate_homography_affine(P_o,Q_o,S_o,R_o); %
         X_i = H_a*X_w
    H_a_inv = H_a^(-1); % H_a^(-1)*X_i = X_w

end

% ====== determine the homography that we would like to
    use
switch Case_2
    case '1' % remove projective only
        H = H_p;
        H_inv = H^(-1);
    case '2' % remove both projective and affine
        H = H_a_inv*H_p;
        H_inv = H^(-1);
end

% ====== find the corner points in image, and map these
    points to world plane
```

```
CP_w = H*[CP_i';1]; CP_w(1) = round(CP_w(1)/CP_w(3));
    CP_w(2) = round(CP_w(2)/CP_w(3));
CQ_w = H*[CQ_i';1]; CQ_w(1) = round(CQ_w(1)/CQ_w(3));
    CQ_w(2) = round(CQ_w(2)/CQ_w(3));
CS_w = H*[CS_i';1]; CS_w(1) = round(CS_w(1)/CS_w(3));
    CS_w(2) = round(CS_w(2)/CS_w(3));
CR_w = H*[CR_i';1]; CR_w(1) = round(CR_w(1)/CR_w(3));
    CR_w(2) = round(CR_w(2)/CR_w(3));

% ====== find the boundary in the output where we are
    going to find the pixel values in our image
xmin = min([CP_w(1) CQ_w(1) CS_w(1) CR_w(1)]);
xmax = max([CP_w(1) CQ_w(1) CS_w(1) CR_w(1)]);
ymin = min([CP_w(2) CQ_w(2) CS_w(2) CR_w(2)]);
ymax = max([CP_w(2) CQ_w(2) CS_w(2) CR_w(2)]);
width = xmax-xmin;
height = ymax-ymin;

% ====== initial output image
output = zeros(height,width,3);

% ====== map the points in the output frame to our image
    using H_p, then use bilinear
% interpolation to determine the corresponding pixel
    value
image_pixel = double(image);    % pixel values we want

for h = 1:height
    for w = 1:width
        homo_temp = H_inv*[w+xmin-1;h+ymin-1;1];
        x_i_homo = homo_temp(1)/homo_temp(3); %
            homogeneous representation
        y_i_homo = homo_temp(2)/homo_temp(3);
        pixel_value = Bilinear(x_i_homo,y_i_homo,
            image_pixel);
        output(h,w,:) = pixel_value;
    end
end
output = uint8(output);
figure; imshow(output);
imwrite(output, ['Own_TwoStep_' Case '_' Case_2 '.tif']);
```