Assignment1 of CMSC 733

1. (Camera models) For a camera, the image x of a point X in space is given by: $x = PX$, with x and X homogeneous 3 and 4-vectors respectively.

(a) If $PC = 0$, where C is a homogeneous 4-vector, show that C is the camera center.

==========

Consider the point C and another point A in 3D space, any point on the line passing through A and C can be expressed as,

$$X(\lambda) = \lambda A + (1 - \lambda)C$$

Give a mapping that maps X to x by matrix P, x can be represented as,

$$x = PX(\lambda) = P(\lambda A + (1 - \lambda)C) = \lambda PA + (1 - \lambda)PC$$

If $PC = 0$, then we have

$$x = \lambda PA$$

which means all points on the line AC are mapped to the same image point λPA. Thus this ray must be a ray through the camera center, and C is the homogeneous representation of the camera center.

==========

(b) The camera projection matrix P can be written as : $P = KR[I| - \tilde{C}]$, where K is the calibration matrix, R is the rotation between the camera and world coordinate frames, and the camera center C is expressed as $C = (\tilde{C}, 1)$. Show that the calibration matrix can be obtained from a RQ decomposition of the first 3×3 sub matrix of the camera matrix P.

==========

On one hand, we know that the camera projection matrix $P = KR[I| - \tilde{C}]$. K is the calibration matrix, which is also an upper triangular:

$$K = \begin{bmatrix} \alpha_x & s & x_0 \\ & \beta_x & y_0 \\ & & 1 \end{bmatrix}$$

And R is the orthogonal rotation matrix between the camera and world coordinate frames.

On the other hand, given a 3×4 camera projection matrix P, then,

$$P = [M| - M\tilde{C}] = M[I| - \tilde{C}]$$

where M is the first 3×3 sub matrix of P, and $(\tilde{C}, 1)$ is the camera center.
By RQ decomposition of M, we obtain, $M = AB$, where A is an upper triangular with the last element of the last row is 1, and B is orthogonal matrix. Thus,

$$P = M[I| - \tilde{C}] = KR[I| - \tilde{C}]$$

As a result, we can see that the two matrix A and B of RQ decomposition of M match with the calibration matrix K and rotation matrix R respectively. Thus the calibration matrix K can be obtained from a RQ decomposition of the first 3×3 sub matrix of the camera matrix P.

==========

(c) In some application we obtained the following camera matrix P. Find the camera center and the calibration parameters.

$$P = \begin{bmatrix} 707 & 679.2 & 555.4 & -2898920 \\ 207 & 46.6 & 919.2 & 6325250 \\ 1.4 & -0.7 & 1.22 & -1837 \end{bmatrix}$$

==========

Given the camera matrix P and the equation $P = [M| - M\tilde{C}]$, we have,

$$M = \begin{bmatrix} 707 & 679.2 & 555.4 \\ 207 & 46.6 & 919.2 \\ 1.4 & -0.7 & 1.22 \end{bmatrix} \qquad -M\tilde{C} = \begin{bmatrix} -2898920 \\ 6325250 \\ -1837 \end{bmatrix}$$

Then we can compute $\tilde{C}$ by,

$$-\begin{bmatrix} 707 & 679.2 & 555.4 \\ 207 & 46.6 & 919.2 \\ 1.4 & -0.7 & 1.22 \end{bmatrix} \tilde{C} = \begin{bmatrix} -2898920 \\ 6325250 \\ -1837 \end{bmatrix}$$

By solving the linear equations, we get,

$$\tilde{C} = [-4228.776 \;\; 2362.218 \;\; 7713.802]^T$$

Thus the camera center C is,

$$C = [-4228.776 \;\; 2362.218 \;\; 7713.802 \;\; 1]^T$$

To compute the calibration matrix, we apply RQ decomposition to M, the first 3×3 sub matrix of P,

$$M = KR$$

$$\rightarrow \begin{bmatrix} 707 & 679.2 & 555.4 \\ 207 & 46.6 & 919.2 \\ 1.4 & -0.7 & 1.22 \end{bmatrix} = \begin{bmatrix} -853.3613 & -425.0469 & 302.6478 \\ 0 & -638.2636 & 350.0416 \\ 0 & 0 & 1.0000 \end{bmatrix} \begin{bmatrix} -0.5529 & -0.8166 & 0.1659 \\ 0.4435 & -0.4569 & -0.7711 \\ 1.4000 & -0.7000 & 1.2200 \end{bmatrix}$$

Thus, the calibration matrix K is,

$$K = \begin{bmatrix} -853.3613 & -425.0469 & 302.6478 \\ 0 & -638.2636 & 350.0416 \\ 0 & 0 & 1.0000 \end{bmatrix}$$

For the details of computation, please check the Appendix: Prob1c.

==========

2. (Affine and metric rectification) Consider the image posted in the class web page. Read the image into matlab and use matlab to perform the calculations necessary for the following questions:

(a) Calculate two vanishing points.

==========

The vanishing point is the image of a point at infinity. Since parallel lines intersect at infinity, the

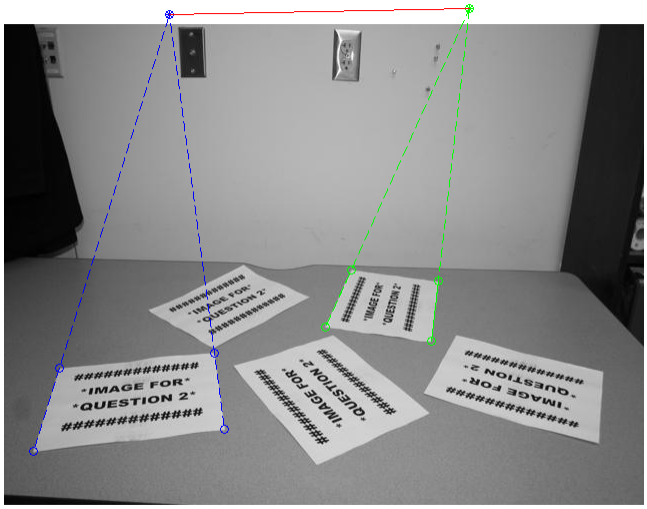

Figure 1: Vanishing points and vanishing line.

intersection of parallel lines in the image is the vanishing point. As a result, we can find two parallel lines, and find the intersection in the image, and the intersection is one vanishing points. Repeat this process, we can find a second vanishing point.
In the given image, four points are enough to define two groups of parallel lines. However, to make it is easier to display, two set of parallel lines (eight points) are selected, so we do not need to extend the image too far. Please see the code in Appendix Prob2ab for the detailed computation. Here is just the two vanishing points in the image (Figure 1).

==========

(b) Find the image of the line at infinity.

==========

Following 2(a), the image of the line at infinity should pass through the two vanishing points, obtained in 2(a). As a result, the line that connects the two vanishing points are the vanishing line. See Figure 1. The details of computation are in Appendix Prob2ab.

==========

(c) Calculate a homography that maps the line at infinity to its canonical position. This homography produces an affine rectification. Apply this homography to the image.

==========

In prob 2(c) (e), we will use a small part of the whole image, shown in Figure 2.
Using the same way introduced in 2(a) and 2(b), we compute the vanishing line of this small image as $l = (l_1, l_2, l_3)^T = (-0.0005, 0.0035, 18)$, while the line at infinity is $l_\infty = (0, 0, 1)^T$. So the

Figure 2: A small part of the whole image

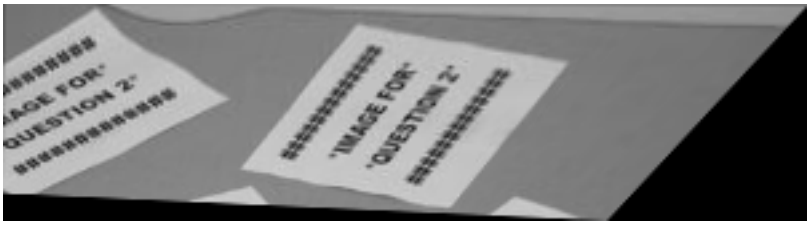

Figure 3: Affine rectified image

homography to map l_∞ to l satisfies:

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

where H_A is any affine transformation. Here, we suppose H_A is the identity matrix. Thus, the transformation matrix is,

$$H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -0.0005 & 0.0035 & 1 \end{bmatrix}$$

Apply this homography to this small image, we get Figure 3.
The corresponding code is in Appendix prob2cde.
==========‘

(d) Calculate the dual conic (the image of the dual conic).

==========
Suppose l' and m' are the two lines in the affinely rectified image correspond to the two orthogonal lines in the world plane. Then, the image of the dual conic $C_\infty^{*'}$, l' and m' should satisfy,

$$l'^T C_\infty^{*'} m' = 0$$

where, $C_\infty^{*'} = \begin{bmatrix} KK^T & 0 \\ 0 & 0 \end{bmatrix}$. And KK^T is actually symmetric with three independent elements.
Since we have already have the affinely rectified image from 2(c), we can choose two pairs of lines corresponding to the orthogonal lines in the world plane, and use the above equation to compute $C_\infty^{*'}$. The details of computation is in Appendix Prob2cde, and the obtained image of the dual conic is,

$$C_\infty^{*'} = \begin{bmatrix} 2.568 & -0.359 & 0 & -0.359 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

==========

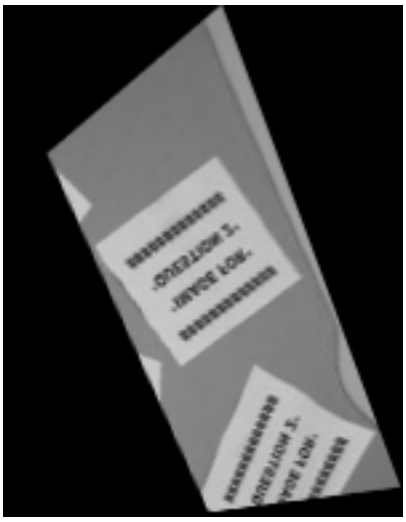

Figure 4: Metric rectified image

(e) Use it to produce a metric rectification of the image.

==========

Once we have $C_\infty^{*'}$, we can recover K from $C_\infty^{*'}$ by eigenvector decomposition. The obtained matrix K is,

$$K = \begin{bmatrix} -0.2046 & -1.5893 \\ -0.9380 & 0.3467 \end{bmatrix}$$

As a result, the affine transformation matrix Ha is,

$$Ha = \begin{bmatrix} -0.2046 & -1.5893 & 0 \\ -0.9380 & 0.3467 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Apply Ha to this image, we get Figure 4. Please check Appendix Prob2cde for details of computation.

==========

3. (Planar mosaics) Consider the images posted in the class web site. They are images of a scene taken by a camera only rotating around an axis passing through its center. Thus any two of the images are separated by a rotation. Consider a point X in space and its images x and x' in two of the images. Then, assuming that the first camera coordinate system coincides with the world coordinate system, we have: $x = K[I|O]X, x' = K[R|O]X = KRK^{-1}x$. Thus, $x' = Hx$, with $H = KRK^{-1}$. H is called a conjugate rotation.

(a) Show that H and R have the same eigenvalues up to scale (one real and two complex).

==========

Given $H = KRK^{-1}$, the eigenvalues λ and eigenvectors v of H satisfy:

$$\begin{aligned} & Hv = \lambda v \\ \rightarrow \quad & KRK^{-1}v = \lambda v \\ \rightarrow \quad & RK^{-1}v = K^{-1}\lambda v = \lambda K^{-1}v \\ \rightarrow \quad & Rv' = \lambda v', where \;\; v' = K^{-1}v \end{aligned}$$

As a result, we can see that λ is also the eigenvalues of R, while v' is the eigenvectors of R. Thus H and R have the same eigenvalues up to scale.

==========

(b) Show that the eigenvector corresponding to the real eigenvalue is the vanishing point of the rotation axis.

==========

The rotation of a vector v can be represented by

$$v' = Rv$$

where R is the rotation matrix. If v is the direction of the rotation axis, v' should be the same as v up to scale. Thus we have,

$$v' = \lambda v = Rv$$

Where λ is scaling parameter, thus it is a real number. Also, from the above equation, we see that actually v is eigenvector corresponding to the real eigenvalue λ of R.

Since the vanishing point of the rotation axis is defined as the direction of the rotation axis, we can say that the eigenvector corresponding to the real eigenvalue is the vanishing point of the rotation axis.

==========

(c) Show that H is the infinite homography (i.e. the homography induced by the plane at infinity).

==========

In this problem, two camera matrix are provided. Let's denote it as,

$$P_1 = K[I|0] \qquad P_2 = K[R|0]$$

Suppose the world plane is $\pi_E = (n^T, d)^T$, where d is the orthogonal distance from the first position of the camera to the world plane. Thus the points on this plane should satisfy: $n^T\hat{X} + d = 0$. As a result, the homography induced by this plane can be represented as,

$$H = K(R - 0 \cdot n^T/d)K^{-1} = KRK^{-1} = \lim_{d\to\infty} K(R - t \cdot n^T/d)K^{-1}$$

As a result, we can think H as a homography induced by a world plane at infinity. Thus, we call H is the infinite homography.

==========

Figure 5: A mosaic image of the 7 given images.

(d) Use the images provided to produce a planar panoramic mosaic.

==========

Given two images, the basic idea for mapping them is: define one of them as the base image, the other one as new image. Then, find two corresponding points on each image and calculate the transformation matrix. Finally, warp the new image to the base image according to the transformation matrix. One thing to notice is that the coordinates of image points should change while warping, thus we need to compute that offset.

Similarly, given a sequence of images, we can always define the first one as the base image, a second one as the new image. After the first warping, use the result as the updated base image, and map with the next one. Repeat this process till all the images are mapped to the base image, which is finally the required planar panoramic mosaic.

The details can be found in Appendix Prob3d. The mosaic image is shown in 5.

==========

4. (Projective reconstruction using coplanar points). Assume two sets of four known coplanar points in the scene A,B,C,D and E,F,G,H giving images a,b,c,d and e,f,g,h for an non calibrated camera with camera center O. Consider an unknown point M in the scene producing an image m.

(a) Show how to determine the viewing ray Om.

==========

Given five coplanar points O, A, B, C, D, we can think it as four lines OA, OB, OC and OD cross at point O, thus the cross ratio is defined as,

$$CR_O(A,B,C,D) = \frac{sin(OA,OC)sin(OB,OD)}{sin(OA,OD)sin(OB,OC)}$$

In projective transformation, the cross ratio is invariant. We will use the notation $CR_O(A, B, C, D)$ to denote the cross ratio of four points A, B, C, D at point O.
In this problem, suppose the point M is projected to M_1 in the plane of A, B, C, D, then we can compute M_1 by the invariant cross ratio.

$$CR_a(b, c, d, m) = CR_A(B, C, D, M_1)$$

$$CR_b(a, c, d, m) = CR_B(A, C, D, M_1)$$

$$CR_c(a, b, d, m) = CR_C(A, B, D, M_1)$$

Since A, B, C, D and a, b, c, d and m are given, we can get the specific coordinates of M_1.

In the same way, we can compute M_2, the projection of point M to the plane of E, F, G, H, by

$$CR_e(f, g, h, m) = CR_E(F, G, H, M_2)$$

$$CR_f(e, g, h, m) = CR_F(E, G, H, M_2)$$

$$CR_g(e, f, h, m) = CR_G(E, F, H, M_2)$$

As a result, the viewing ray Om can be determined by M_1 and M_2.
==========

(b) Use (a) to develop a method for finding the camera center.

==========
Similarly, consider the projection A' of A to the plane of E, F, G, H, we can obtain A' by,

$$CR_e(f, g, h, a) = CR_E(F, G, H, A')$$

$$CR_f(e, g, h, a) = CR_F(E, G, H, A')$$

$$CR_g(e, f, h, a) = CR_G(E, F, H, A')$$

By now, we have A, A', M_1 and M_2, and we have two rays AA' and M_1M_2 from the camera center. Thus the camera center O can be determined by the intersection of the ray AA' and the ray M_1M_2.
==========

(c) Repeat (a) and (b) for the case where we are given 6 points in space and their images (instead of two sets of four coplanar points).

==========

Given the camera center O, six points A, B, C, D, E, F and their images, we have six planes namely OAB, OAC, OAD, OAE, OAF, which intersect an image plane at I_AI_B, I_AI_C, I_AI_D, I_AI_E, I_AI_F. Take another plane into scene, OA intersects the plane at $I_{A'}$, and AB, AC, AD, AE, AF intersect with this plane at $I_{B'}$, $I_{C'}$, $I_{D'}$, $I_{E'}$, $I_{F'}$ respectively.

Then we can measure the cross ratio using the intersections I_AI_B, I_AI_C, ..., I_AI_F with the image plane, thus we know cross ratio of $I_{A'}I_{B'}$, $I_{A'}I_{C'}$, $I_{A'}I_{D'}$, $I_{A'}I_{E'}$. Similarly, we can get the cross ratio of $I_{A'}I_{B'}$, $I_{A'}I_{C'}$, $I_{A'}I_{D'}$, $I_{A'}I_{F'}$. As a result, $I_{A'}$ lies on the intersection of the two conics. As long as we find $I_{A'}$, we have two points A and $I_{A'}$ on the ray passing from the camera center O.

In the similar way, we can find two points on another ray passing from the camera center O. As a result, the intersection of the two rays gives up the camera center O.
==========

Appendix: Prob1c

```
% Original data
P = [707 679.2 555.4 -2898920;
     207 46.6 919.2 -6325250;
     1.4 -0.7 1.22 -1837];
M = P(:, 1:3);

% Compute the camera center
C = sym('[c1; c2; c3]');
f = M*C + P(:,4);
res = solve(f);

C_hat = ones(3, 1);
names = fieldnames(res);
for j = 1 : numel(names)
    C_hat(j) = res.(names{j});
end

% Compute the calibration matrix K by RQ decomposition
ReverseRows = [0 0 1; 0 1 0 ; 1 0 0];
[Q R] = qr((ReverseRows * M)');
K = ReverseRows * R' * ReverseRows;
R = ReverseRows * Q';
scale = K(3,3);
K(:,3) = K(:, 3) ./ scale;
R(3,:) = R(3, :) .* scale;
```

Appendix: Prob2ab

```
clear;
close all;

%% Get the image.
image = imread('hw1_q2_vpoint-data.jpg');
image = rgb2gray(image);
img_sz = size(image);

figure, imshow(image);

%% One vanishing point
[x, y] = ginput2(2);
pA1 = [x y];
pA1_ext = [x y [1;1]];

[x, y] = ginput2(2);
pA2 = [x y];
pA2_ext = [x y [1;1]];

lA1 = cross(pA1_ext(1,:), pA1_ext(2,:));
lA2 = cross(pA2_ext(1,:), pA2_ext(2,:));
vA = cross(lA1, lA2);
pA = vA ./ vA(3);
pA = pA(1:2);

%% Another vanishing point
[x, y] = ginput2(2);
pB1 = [x, y];
pB1_ext = [x y [1;1]];

[x, y] = ginput2(2);
pB2 = [x y];
pB2_ext = [x y [1;1]];

lB1 = cross(pB1_ext(1,:), pB1_ext(2,:));
lB2 = cross(pB2_ext(1,:), pB2_ext(2,:));
vB = cross(lB1, lB2);
pB = vB ./ vB(3);
pB = pB(1:2);

%%  Vanishing line
vAB = cross(vA, vB);
vAB = vAB ./ vAB(3);

%% Extend the image

pA = floor(pA);
pB = floor(pB);

LEFT = 0; RIGHT = 0; UP = 0; DOWN = 0;
MARGIN = 0;
img_sz_new = img_sz;
```

```
if pA(1) < 1
    LEFT = LEFT + (1 - pA(1));
elseif pA(1) > img_sz_new(2)
    RIGHT = RIGHT + (pA(1) - img_sz_new(2));
end

if pA(2) < 1
    UP = UP + (1 - pA(2));
elseif pA(2) > img_sz_new(1)
    DOWN = DOWN + (pA(2) - img_sz_new(1));
end

img_sz_new(2) = img_sz(2) + LEFT + RIGHT ;
img_sz_new(1) = img_sz(1) + UP + DOWN;


if pB(1) < 1
    LEFT = LEFT + (1 - pB(1));
elseif pB(1) > img_sz_new(2)
    RIGHT = RIGHT + (pB(1) - img_sz_new(2));
end

if pB(2) < 1
    UP = UP + (1 - pB(2)) + 10;
elseif pB(2) > img_sz_new(1)
    DOWN = DOWN + (pB(2) - img_sz_new(1));
end

% extend
UP = UP + MARGIN;
if LEFT > 0
    LEFT = LEFT + MARGIN;
end
if RIGHT > 0
    RIGHT = RIGHT + MARGIN;
end
img_sz_new(2) = img_sz(2) + LEFT + RIGHT;
img_sz_new(1) = img_sz(1) + UP + DOWN;

img_ext = ones(img_sz_new) + 255;
img_ext(1+UP : img_sz(1)+UP , 1 + LEFT : img_sz(2) + LEFT) = image;
figure, imshow(uint8(img_ext));
hold on;

%% plot the vanishing line

Aline1 = [pA1; pA];
Aline2 = [pA2; pA];
plot(Aline1(:, 1) + LEFT, Aline1(:, 2) + UP, 'g--o');
plot(Aline2(:, 1) + LEFT, Aline2(:, 2) + UP, 'g--o');
hold on;

plot(pA(1) + LEFT, pA(2) + UP, 'g*');
hold on;
```

```
Bline1 = [pB1; pB];
Bline2 = [pB2; pB];
plot(Bline1(:, 1) + LEFT, Bline1(:, 2) + UP, 'b--o');
plot(Bline2(:, 1) + LEFT, Bline2(:, 2) + UP, 'b--o');
hold on;

plot(pB(1) + LEFT, pB(2) + UP, 'b*');
hold on;

vline = [pA; pB];
plot(vline(:,1)+LEFT, vline(:,2) + UP, 'r-');
```

Appendix: Prob2cde

```
clc;
close all;
clear;

%% Get a small part of the whole image
image = imread('hw1_q2_vpoint-data.jpg');
image = rgb2gray(image);

image = image(240:320, 200:500);
figure, imshow(image);

imwrite(image, 'prob2c_origin.png')

%% Compute the vanishing line

% One vanishing point
[x, y] = ginput2(2);
pA1 = [x y [1;1]];

[x, y] = ginput2(2);
pA2 = [x y [1;1]];

lA1 = cross(pA1(1,:), pA1(2,:));
lA2 = cross(pA2(1,:), pA2(2,:));
vA = cross(lA1, lA2);

% Another vanishing point
[x, y] = ginput2(2);
pB1 = [x y [1;1]];

[x, y] = ginput2(2);
pB2 = [x y [1;1]];

lB1 = cross(pB1(1,:), pB1(2,:));
lB2 = cross(pB2(1,:), pB2(2,:));
vB = cross(lB1, lB2);

% Vanishing line
vAB = cross(vA, vB);
vAB = vAB ./ vAB(3);

%% Obtain the projective transformation matrix based on the vanishing
line

RE = eye(3);
RE(3,:) = vAB;

HA = [1 0  0; 0 1 0; 0 0 1];
H = HA * RE;

% Projective transformation
```

```
tform = maketform('projective',H');
img2 = imtransform(image, tform, 'Size', size(image));

imwrite(img2, 'prob2c.png')
figure, imshow(img2), hold on;

%% Obtain the image of the dual conic

% Choose one pair of orthogonal lines: line by line, two points on each
line

[x, y] = ginput2(2);
pA1 = [x y];
plot(pA1(:,1), pA1(:,2), 'bx'), hold on;

[x, y] = ginput2(2);
pA2 = [x y];
plot(pA2(:,1), pA2(:,2), 'bo'), hold on;

co = sym('[k; -1; b]');
pA1_ex = [pA1 ones(2,1)];
f = pA1_ex * co;
result = solve(f);
coA1 = [result.k, -1, result.b];

pA2_ex = [pA2 ones(2, 1)];
f = pA2_ex * co;
result = solve(f);
coA2 = [result.k, -1, result.b];

% Choose another pair of orthogonal lines: line by line, two points on
each line

[x, y] = ginput2(2);
pB1 = [x y];
plot(pB1(:,1), pB1(:,2), 'go'), hold on;

[x, y] = ginput2(2);
pB2 = [x y];
plot(pB2(:,1), pB2(:,2), 'gx'), hold on;

co = sym('[k; -1; b]');
pB1_ex = [pB1 ones(2,1)];
f = pB1_ex * co;
result = solve(f);
coB1 = [result.k, -1, result.b];

pB2_ex = [pB2 ones(2, 1)];
f = pB2_ex * co;
result = solve(f);
coB2 = [result.k, -1, result.b];

% Compute the image of dual conic
```

```
dc = sym('[a b 0; b 1 0; 0 0 0]');
f = [coA1 * dc * coA2'; coB1 * dc * coB2'];
result = solve(f);
dualconic = [result.a result.b 0; result.b 1 0; 0 0 0];
dualconic = double(dualconic);

%% Matric Rectification

% Recover K from the image of dual conic
S = dualconic(1:2, 1:2);
[vec, val] = eig(S);
K = vec * (val^0.5);

% Get the affine transformation matrix
Ha = zeros(3,3);
Ha(1:2, 1:2) = K;
% Ha = dualconic;
Ha(3, 3) = 1;

% Affine transformation
tform = maketform('affine',inv(Ha'));
img3 = imtransform(img2, tform);
figure, imshow(img3);

imwrite(img3, 'prob2e.png')
```

Appendix: Prob3

```
img_base = double(imread('mosaic-data/mosaic-00000000.jpg')) / 255;

for index = 1 : 1 : 6
   path = strcat('mosaic-data/mosaic-0000000', int2str(index), '.jpg');
   img = double(imread(path)) / 255;

   % obtain size
   [h_base, w_base, d_base] = size(img_base);
   [h, w, d] = size(img);

   close all;

   % get two sets of corresponding points from user:
   % two points per image
   figure; subplot(1,2,1); image(img_base); axis image; hold on;
   title('base image');
   [x_base, y_base] = ginput2(2);
   subplot(1,2,2); image(img); axis image; hold on;
   title('new image');
   [x, y] = ginput2(2);

   % estimate the parameters
   param  = [x' y'; y' -x'; 1 1 0 0; 0 0 1 1]';
   base = [x_base; y_base];
   t  = param \ base; % solve the linear system

   % get the transformation matrix H
   H = [t(1) t(2) t(3); -t(2) t(1) t(4); 0 0 1];

   % warp corners of the new image to determine the size of the output
image
   corner_warp = H*[ 1 1 w w; 1 h 1 h; 1 1 1 1];
   x_wp = min( [ corner_warp(1,:) 0 ] ) : max( [corner_warp(1,:)
w_base] ); % min x : max x
   y_wp = min( [ corner_warp(2,:) 0 ] ) : max( [corner_warp(2,:)
h_base] ); % min y : max y
   [x_w,y_w] = ndgrid(x_wp,y_wp);
   [w_w, h_w] = size(x_w);

   % backwards transform
   bt = H \ [ x_w(:) y_w(:) ones(w_w * h_w,1) ];  % warp

   % re-sample pixel values
   clear img_temp;
   bt1_rs = reshape( bt(1,:), w_w, h_w)';
   bt2_rs = reshape( bt(2,:), w_w, h_w)';
   img_temp(:,:,1) = interp2(img(:,:,1), bt1_rs, bt2_rs, 'nearest');
   img_temp(:,:,2) = interp2(img(:,:,2), bt1_rs, bt2_rs, 'nearest');
   img_temp(:,:,3) = interp2(img(:,:,3), bt1_rs, bt2_rs, 'nearest');

   % get offset
```

```
   offset = -round( [ min( [ corner_warp(1,:) 0 ] )
min( [ corner_warp(2,:) 0 ] ) ] );

   % copy base image into the warped image
   for ii = 1 : h_base
       for jj = 1 : w_base
           if img_base(ii,jj) > 0
               img_temp(ii+offset(2),jj+offset(1), :) =
img_base(ii,jj,:);
           end
       end
   end

   img_base = img_temp;
end

% show the result
figure; image(img_base); axis image;
title('mosaic image');

imwrite(uint8(img_base*255), 'result.png');
```