

# ECE661: Computer Vision (Fall 2014)

Shaobo Fang: s-fang@purdue

September 11, 2014

## Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Estimation of the Homography Matrix <math>H</math></b>                               | <b>2</b>  |
| <b>2</b> | <b>Project Audrey Image <math>x</math> To the Frame PQRS <math>x'</math></b>            | <b>3</b>  |
| <b>3</b> | <b>Project frame ABCD in Frame <math>x'</math> image to image Audrey <math>x</math></b> | <b>6</b>  |
| <b>4</b> | <b>Project Images Obtained in Previous Parts To World Plane</b>                         | <b>10</b> |
| <b>5</b> | <b>Projective Transformation Performed On My Own Pair of Images</b>                     | <b>15</b> |

# 1 Estimation of the Homography Matrix $H$

Section 1 is the estimation of the homography matrix  $H$  that is used implementing point correspondences.

From lecture notes we know that given a point in a planer scene and its corresponding pixel  $x'$  in the image plane, for most cameras we can write  $x' = Hx$ . Assuming that  $x$  and  $x'$  are expressed using homogeneous coordinates:  $x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$  and  $x' = \begin{pmatrix} x'_1 \\ x'_2 \\ x'_3 \end{pmatrix}$ , with

$H = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix}$ . Easily, we can obtain:

$$\begin{pmatrix} x'_1 \\ x'_2 \\ x'_3 \end{pmatrix} = \begin{pmatrix} h_{11}x_1 + h_{12}x_2 + h_{13}x_3 \\ h_{21}x_1 + h_{22}x_2 + h_{23}x_3 \\ h_{31}x_1 + h_{32}x_2 + h_{33}x_3 \end{pmatrix}$$

Denoting the physical plane scene coordinates by  $(x,y)$  and the physical pixel coordinates by  $(x',y')$ , we have  $x = \frac{x_1}{x_3}$ ,  $y = \frac{x_2}{x_3}$  and  $x' = \frac{x'_1}{x'_3}$ ,  $y' = \frac{x'_2}{x'_3}$ .

Now, we can write for the physical coordinates of the image pixel:

$$x' = \frac{h_{11}x_1 + h_{12}x_2 + h_{13}x_3}{h_{31}x_1 + h_{32}x_2 + h_{33}x_3}$$
$$y' = \frac{h_{21}x_1 + h_{22}x_2 + h_{23}x_3}{h_{31}x_1 + h_{32}x_2 + h_{33}x_3}$$

Now, substitute  $h_{33} = 1$  and  $x_3 = 1$ :

$$H = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{pmatrix}$$

.

Accordingly:

$$x' = \frac{h_{11}x_1 + h_{12}x_2 + h_{13}}{h_{31}x_1 + h_{32}x_2 + 1}$$
$$y' = \frac{h_{21}x_1 + h_{22}x_2 + h_{23}}{h_{31}x_1 + h_{32}x_2 + 1}$$

In order to solve for projective transformation matrix  $H$ , form the linear equations below

based on the two sets of coordinates  $(x_i, y_i)$  and  $(x'_i, y'_i)$ :

$$\begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x'_1 & -y_1y'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y'_1 & -y_1x'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x'_2 & -y_2y'_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y'_2 & -y_2x'_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x'_3 & -y_3y'_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y'_3 & -y_3x'_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x'_4 & -y_4y'_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y'_4 & -y_4x'_4 \end{pmatrix} \begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{pmatrix} = \begin{pmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \\ x'_4 \\ y'_4 \end{pmatrix}$$

**Solve the linear equation  $A\bar{h} = [\bar{x}, \bar{y}]$  above to obtain the matrix  $H$ .**

After the  $H$  is obtained, point correspondences between two planes could be easily calculated by :

**1. Transform the points  $x$  in planer scene to the pixel locations  $x'$  in the digital image plane**

$$x' = Hx$$

**2. Transform the points pixel locations  $x'$  in the digital image plane to planer scene  $x$**

$$x = H^{-1}x'$$

## 2 Project Audrey Image $x$ To the Frame PQRS $x'$

**In this part, we will map image Audrey to the digital image Frame.**

The pixel locations in both digital image plane ( $x'$ ) and 'planer scene' ( $x$ ) were hard coded into the matlab script:

$$x' = Hx$$

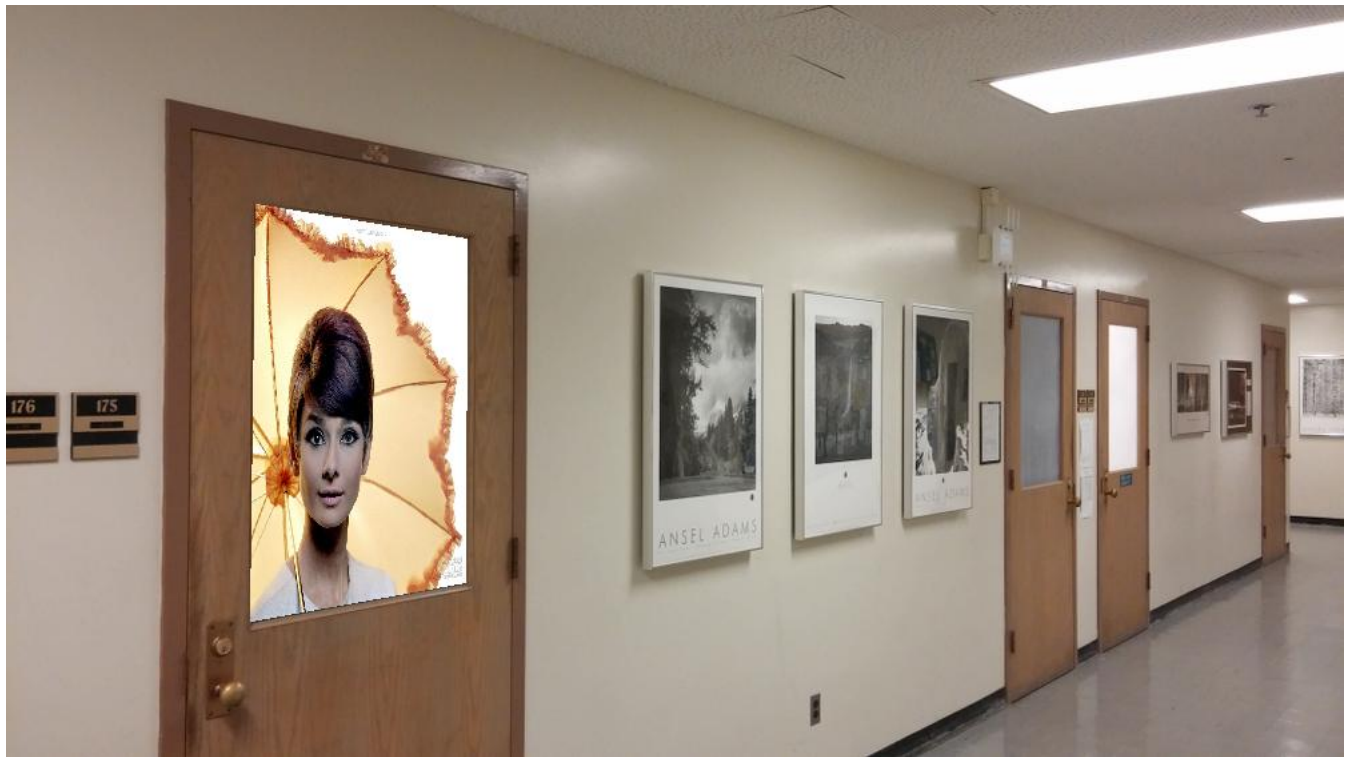


Fig 1. Audrey's Image Projected To the Wall.

Matlab code used in this section.

```

1 close all
2 clear all
3 clc;
4
5 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
6 % Load the data images into matlab
7 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
8 Img_Frame = imread('Frame.jpg');
9 Img_Audrey = imread('Audrey.jpg');
10 SizeImgFrame = size(Img_Frame);
11
12 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
13 % Hard-coding the coordinates location that were used to obtain
14 % the homography matrix
15 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
16 x1w = 1;
17 y1w = 1;
18 x2w = 1;
19 y2w = 500;
20 x3w = 508;
21 y3w = 1;
22 x4w = 508;
23 y4w = 500;

```

```

24
25 x1 = 150;
26 y1 = 188;
27 x2 = 176;
28 y2 = 346;
29 x3 = 462;
30 y3 = 184;
31 x4 = 433;
32 y4 = 345;
33
34 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
35 % Solving for the homography matrix H
36 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
37
38 A = [x1w,y1w,1,0,0,0,-x1w*x1,-y1w*x1;...
39      0,0,0,x1w,y1w,1,-x1w*y1,-y1w*y1;...
40      x2w,y2w,1,0,0,0,-x2w*x2,-y2w*x2;...
41      0,0,0,x2w,y2w,1,-x2w*y2,-y2w*y2;...
42      x3w,y3w,1,0,0,0,-x3w*x3,-y3w*x3;...
43      0,0,0,x3w,y3w,1,-x3w*y3,-y3w*y3;...
44      x4w,y4w,1,0,0,0,-x4w*x4,-y4w*x4;...
45      0,0,0,x4w,y4w,1,-x4w*y4,-y4w*y4]
46
47 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
48 % Alternative approach to obtain H, however the H would be in symbolic
49 % and the calculation in the for loops below would be EXTREMELY SLOW!
50 % Using inverse matrix has been proved a lot faster than using 'solve'
51 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
52 % syms h11 h12 h13 h21 h22 h23 h31 h32
53 % S = solve(A*[h11;h12;h13;h21;h22;h23;h31;h32] == [x1;y1;x2;y2;x3;y3;x4;y4])
54 % H = [S.h11,S.h12,S.h13;
55 %      S.h21,S.h22,S.h23;
56 %      S.h31,S.h32,1];
57 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
58
59 h_vector = inv(A)*[x1;y1;x2;y2;x3;y3;x4;y4];
60 H = reshape([h_vector;1],[3,3])'
61
62
63
64 tic
65
66 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
67 % Perform projective transformation and save the projected image generated
68 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
69
70 New_Img_Frame = Img_Frame;
71 SizeImgAudrey = size(Img_Audrey);
72 for i = 1:1:SizeImgAudrey(1)
73     i
74     for j = 1:1:SizeImgAudrey(2)
75         LocFrame = H*[i;j;1];
76         x = round(LocFrame(1)/LocFrame(3));
77         y = round(LocFrame(2)/LocFrame(3));

```

```

78         New_Img_Frame(x,y,:) = Img_Audrey(i,j,:);
79     end
80 end
81 image(New_Img_Frame)
82 imwrite(New_Img_Frame, 'projected_img_1.jpg', 'jpeg');
83 toc

```

### 3 Project frame ABCD in Frame $x'$ image to image Audrey $x$

In this part, we will map image Frame to the digital image Audrey so that ABCD will fit around Audrey

The pixel locations in both digital image plane ( $x'$ ) and 'planer scene' ( $x$ ) were hard coded into the matlab script:

$$x = H^{-1}x'$$



Fig 2. Project image Frame so that frame ABCD will fit around Audrey. Note that Audrey is still squared in this case.

#### Matlab code used in this section

```

1  close all
2  clear all
3  clc;
4  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
5  % Load the data images into matlab
6  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
7  Img_Frame = imread('Frame.jpg');
8  Img_Audrey = imread('Audrey.jpg');
9
10 SizeImgFrame = size(Img_Frame);
11

```

```

12 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
13 % Hard-coding the coordinates location that were used to obtain
14 % the homography matrix. PLEASE NOTE THAT IN ORDER TO REDUCE SOME
15 % UNNECESSARY COMPUTATION, we will map frame ABCD to a 100*100 pixels
16 % squared frame. Of course we can easily change x2w, x4w = 508 and
17 % y2w, y4w = 500. The computation scale would be significantly larger
18 % while the result did not improve significatntly. (So I did not see
19 % the advantage)
20 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
21
22 x1w = 1;
23 y1w = 1;
24 x2w = 1;
25 y2w = 100;
26 x3w = 100;
27 y3w = 1;
28 x4w = 100;
29 y4w = 100;
30
31 x1 = 201;
32 y1 = 486;
33 x2 = 212;
34 y2 = 564;
35 x3 = 422;
36 y3 = 484;
37 x4 = 405;
38 y4 = 566;
39
40 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
41 % Solving for the homography matrix H
42 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
43 A = [x1w,y1w,1,0,0,0,-x1w*x1,-y1w*x1;
44      0,0,0,x1w,y1w,1,-x1w*y1,-y1w*y1;
45      x2w,y2w,1,0,0,0,-x2w*x2,-y2w*x2;
46      0,0,0,x2w,y2w,1,-x2w*y2,-y2w*y2;
47      x3w,y3w,1,0,0,0,-x3w*x3,-y3w*x3;
48      0,0,0,x3w,y3w,1,-x3w*y3,-y3w*y3;
49      x4w,y4w,1,0,0,0,-x4w*x4,-y4w*x4;
50      0,0,0,x4w,y4w,1,-x4w*y4,-y4w*y4];
51 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
52 % Alternative approach to obtain H, however the H would be in symbolic
53 % and the calculation in the for loops below would be EXTREMELY SLOW!
54 % Using inverse matrix has been proved a lot faster than using 'solve'
55 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
56 % syms h11 h12 h13 h21 h22 h23 h31 h32
57 % S = solve(A*[h11;h12;h13;h21;h22;h23;h31;h32] == [x1;y1;x2;y2;x3;y3;x4;y4])
58 % H = [S.h11,S.h12,S.h13;
59 %      S.h21,S.h22,S.h23;
60 %      S.h31,S.h32,1];
61 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
62 h_vector = inv(A)*[x1;y1;x2;y2;x3;y3;x4;y4];
63 H = reshape([h_vector;1],[3,3])'
64
65

```

```

66 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
67 % The below part generated the minimum size image frame so that all
68 % pixels in image 'frame' could be mapped to our generated image.
69 % Note that there will be plenty of blank area because after projection
70 % the image is not rectangle anymore and hence will only utilize
71 % a portion of pixels in a rectangle frame.
72 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
73
74 H*[1;1;1]
75 H*[1;100;1]
76 H*[100;1;1]
77 H*[100;100;1]
78
79
80 newp1 = H^(-1)*[1;1;1]
81 newp2 = H^(-1)*[1;SizeImgFrame(2);1]
82 newp3 = H^(-1)*[SizeImgFrame(1);1;1]
83 newp4 = H^(-1)*[SizeImgFrame(1);SizeImgFrame(2);1]
84
85 xp1 = round(newp1(1)/newp1(3))
86 xp2 = round(newp2(1)/newp2(3))
87 xp3 = round(newp3(1)/newp3(3))
88 xp4 = round(newp4(1)/newp4(3))
89
90 yp1 = round(newp1(2)/newp1(3))
91 yp2 = round(newp2(2)/newp2(3))
92 yp3 = round(newp3(2)/newp3(3))
93 yp4 = round(newp4(2)/newp4(3))
94
95 pointxp1 = double(xp1);
96 pointxp2 = double(xp2);
97 pointxp3 = double(xp3);
98 pointxp4 = double(xp4);
99
100 pointyp1 = double(yp1);
101 pointyp2 = double(yp2);
102 pointyp3 = double(yp3);
103 pointyp4 = double(yp4);
104
105 A = [pointxp1,pointxp2,pointxp3,pointxp4];
106 B = [pointyp1,pointyp2,pointyp3,pointyp4];
107
108 tx = abs(min(A)) + 1;
109 ty = abs(min(B)) + 1;
110
111 frame_x = max(A) - min(A) + 1;
112 frame_y = max(B) - min(B) + 1;
113 Corrected_Img(1:frame_x,1:frame_y,1:3) = 255;
114 Projected_Img = imread('Frame.jpg');
115
116 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
117 % Perform projective transformation and save the projected image generated
118 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
119 tic

```



```
120 for i = 1:1:frame_x
121     i
122     for j = 1:1:frame_y
123         LocFrame = H*[(i-tx);(j-ty);1];
124         x = double(round(LocFrame(1)/LocFrame(3)));
125         y = double(round(LocFrame(2)/LocFrame(3)));
126         if (x<SizeImgFrame(1)) && (x>0) && (y<SizeImgFrame(2)) && (y>0)
127             Corrected_Img(i,j,:) = Projected_Img(x,y,:);
128             x;
129             y;
130         else
131             end
132     end
133 end
134
135 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
136 % Mapping the audrey image to the squared box A'B'C'D' in our projected frame
137 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
138
139 for i = 1:1:508
140     for j = 1:1:500
141         x = round(i/5) + tx;
142         y = round(j/5) + ty;
143         Corrected_Img(x,y,:) = Img_Audrey(i,j,:);
144     end
145 end
146
147 Corrected_Img = uint8(Corrected_Img);
148 image(Corrected_Img)
149 truesize
150 imwrite(Corrected_Img,'projected_img_2.jpg','jpeg');
151 toc
```

## 4 Project Images Obtained in Previous Parts To World Plane

Project Images From Task 1 to 'World Plane'



Fig 3. The projection in world plane of the image obtained in part 1. Audrey is in frame PQRS.

Project Images From Task 2 to 'World Plane', note that this part could be performed easily by scaling  $(Ax, By) = (x', y')$



Fig 4. The projection in world plane of the image obtained in part 2. Audrey is in frame ABCD.

### Matlab Code used in this section

```
1 close all
2 clear all
3 clc;
4 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
5 % Load the data images into matlab
6 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
7 Img_Frame = imread('Frame.jpg');
8 Img_Audrey = imread('Audrey.jpg');
9 SizeImgFrame = size(Img_Frame);
10 Projected_Img = imread('projected-img-1.jpg');
11
12 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
13 % Hard-coding the coordinates location that were used to obtain
14 % the homography matrix.
15 % Please note what matters in the world plane is the correct ratio.
16 % Thus we set the desired ratio for frame PQRS (and ABCD) to be 63*91
17 % (equivalently 126 * 182. We can approximately assume the frame PQRS
18 % and frame ABCD have the same ratio.)
19 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
20
21 x1w = 1;
22 y1w = 1;
23 x2w = 1;
```

```

24 y2w = 126;
25 x3w = 182;
26 y3w = 1;
27 x4w = 182;
28 y4w = 126;
29
30 x1 = 150;
31 y1 = 188;
32 x2 = 176;
33 y2 = 346;
34 x3 = 462;
35 y3 = 184;
36 x4 = 433;
37 y4 = 345;
38 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
39 % Solving for the homography matrix H
40 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
41 A = [x1w,y1w,1,0,0,0,-x1w*x1,-y1w*x1;
42      0,0,0,x1w,y1w,1,-x1w*y1,-y1w*y1;
43      x2w,y2w,1,0,0,0,-x2w*x2,-y2w*x2;
44      0,0,0,x2w,y2w,1,-x2w*y2,-y2w*y2;
45      x3w,y3w,1,0,0,0,-x3w*x3,-y3w*x3;
46      0,0,0,x3w,y3w,1,-x3w*y3,-y3w*y3;
47      x4w,y4w,1,0,0,0,-x4w*x4,-y4w*x4;
48      0,0,0,x4w,y4w,1,-x4w*y4,-y4w*y4;];
49 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
50 % Alternative approach to obtain H, however the H would be in symbolic
51 % and the calculation in the for loops below would be EXTREMELY SLOW!
52 % Using inverse matrix has been proved a lot faster than using 'solve'
53 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
54 % syms h11 h12 h13 h21 h22 h23 h31 h32
55 % S = solve(A*[h11;h12;h13;h21;h22;h23;h31;h32] == [x1;y1;x2;y2;x3;y3;x4;y4])
56 % H = [S.h11,S.h12,S.h13;
57 %      S.h21,S.h22,S.h23;
58 %      S.h31,S.h32,1];
59 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
60 h_vector = inv(A)*[x1;y1;x2;y2;x3;y3;x4;y4];
61 H = reshape([h_vector;1],[3,3])'
62
63
64
65 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
66 % The below part generated the minimum size image frame so that all
67 % pixels in image 'frame' could be mapped to our generated image.
68 % Note that there will be plenty of blank area because after projection
69 % the image is not rectangle anymore and hence will only utilize
70 % a portion of pixels in a rectangle frame.
71 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
72
73 newp1 = H^(-1)*[1;1;1];
74 newp2 = H^(-1)*[1;SizeImgFrame(2);1];
75 newp3 = H^(-1)*[SizeImgFrame(1);1;1];
76 newp4 = H^(-1)*[SizeImgFrame(1);SizeImgFrame(2);1];
77

```

```
78 xp1 = round(newp1(1)/newp1(3))
79 xp2 = round(newp2(1)/newp2(3))
80 xp3 = round(newp3(1)/newp3(3))
81 xp4 = round(newp4(1)/newp4(3))
82
83 yp1 = round(newp1(2)/newp1(3))
84 yp2 = round(newp2(2)/newp2(3))
85 yp3 = round(newp3(2)/newp3(3))
86 yp4 = round(newp4(2)/newp4(3))
87
88 pointxp1 = double(xp1);
89 pointxp2 = double(xp2);
90 pointxp3 = double(xp3);
91 pointxp4 = double(xp4);
92
93 pointyp1 = double(yp1);
94 pointyp2 = double(yp2);
95 pointyp3 = double(yp3);
96 pointyp4 = double(yp4);
97
98 A = [pointxp1,pointxp2,pointxp3,pointxp4];
99 B = [pointyp1,pointyp2,pointyp3,pointyp4];
100
101 tx = abs(min(A)) + 1;
102 ty = abs(min(B)) + 1;
103
104 frame_x = max(A) + tx
105 frame_y = max(B) + ty
106 Corrected_Img(1:frame_x,1:frame_y,1:3) = 255;
107
108
109 tic
110 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
111 % Perform projective transformation and save the projected image generated
112 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
113
114 for i = 1:1:frame_x
115     i
116     for j = 1:1:frame_y
117         LocFrame = H*[(i-tx);(j-ty);1];
118         x = double(round(LocFrame(1)/LocFrame(3)));
119         y = double(round(LocFrame(2)/LocFrame(3)));
120         if (x<SizeImgFrame(1)) && (x>0) && (y<SizeImgFrame(2)) && (y>0)
121             Corrected_Img(i, j, :) = Projected_Img(x, y, :);
122             x;
123             y;
124         else
125             end
126     end
127 end
128
129 Corrected_Img = uint8(Corrected_Img);
130 image(Corrected_Img)
131 imwrite(Corrected_Img, 'corrected_img_1.jpg', 'jpeg');
```

```
132 toc
133
134 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
135 % Perform the similar approach for the image we obtained in part 2.
136 % Note that if we think about our result in part 2, we will notice
137 % we only need to stretch/compress x,y axis to get the corrected projection
138 % in world plane.
139 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
140
141
142 close all
143 clear all
144 clc;
145 Img_Frame = imread('Frame.jpg');
146 Img_Audrey = imread('Audrey.jpg');
147
148 SizeImgFrame = size(Img_Frame);
149 Projected_Img = imread('projected_img_2.jpg');
150 Projected_Img_Size = size(Projected_Img);
151 Corrected_Img(1:Projected_Img_Size(1),1:round(Projected_Img_Size(2)*(2/3))
    ,1:3) = 255;
152 for i = 1:1:Projected_Img_Size(1)
153     for j = 1:1:Projected_Img_Size(2)
154         x = i;
155         y = round(j*(2/3));
156         Corrected_Img(x,y,:) = Projected_Img(i,j,:);
157     end
158 end
159
160 Corrected_Img = uint8(Corrected_Img);
161 image(Corrected_Img)
162 truesize
163 imwrite(Corrected_Img, 'corrected_img_2.jpg', 'jpeg');
164 toc
```

## 5 Projective Transformation Performed On My Own Pair of Images



Fig 5. The Image of 'Mona Lisa' Hanging on Louvre Museum's Wall

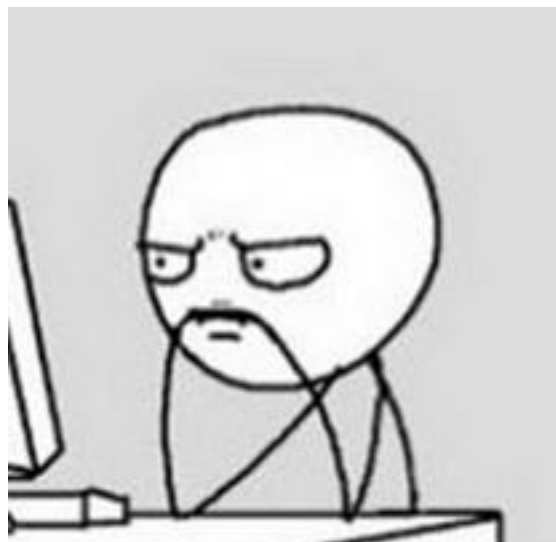


Fig 6. Face of a Typical ECE Graduate Student at Purdue University that we want to project to cover Mona Lisa.



**Part 4.1: Perform Projective Transformation Similar As Was Done In Task 1**



Fig 7. Project the face of a typical ECE student to the frame of Mona Lisa on the wall.  
Similar in part 1.

**Part 4.2: Perform Projective Transformation Similar As Was Done In Task 2,  
Note That the Projected Comic Is Squared Shape**

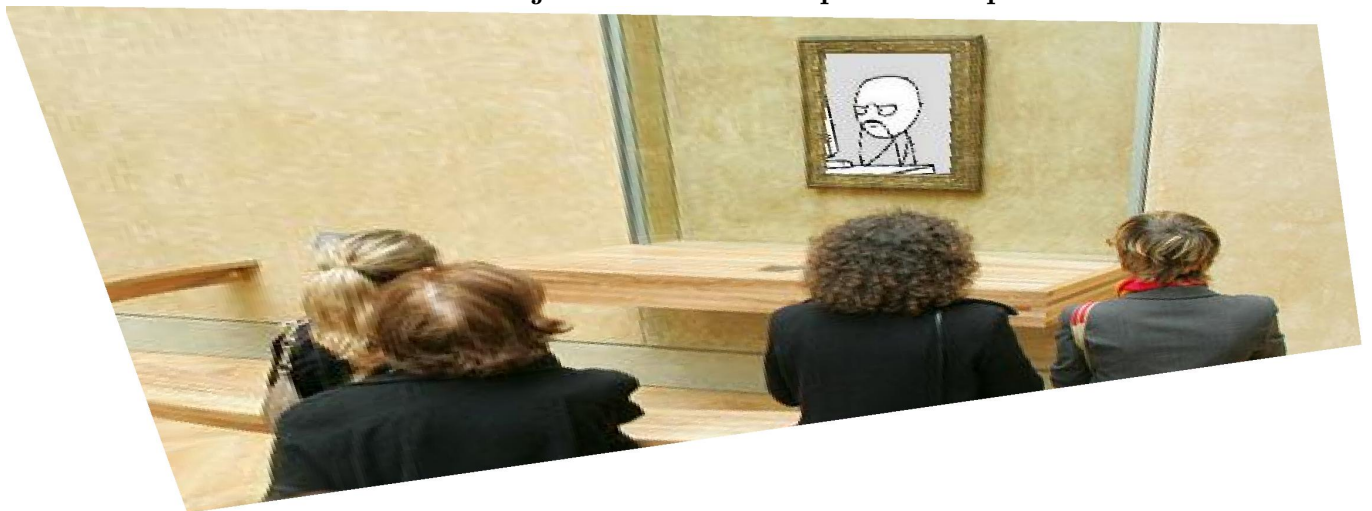


Fig 8. Project the Louvre image so that the face of ECE student could fit in the frame of  
Mona Lisa.



**Part 4.3: Perform Projective Transformation Similar As Was Done In Task 3**



Fig 9. Project Fig 7. to real world plane. It is known that the size of Mona Lisa is  $77cm \times 53cm$ .

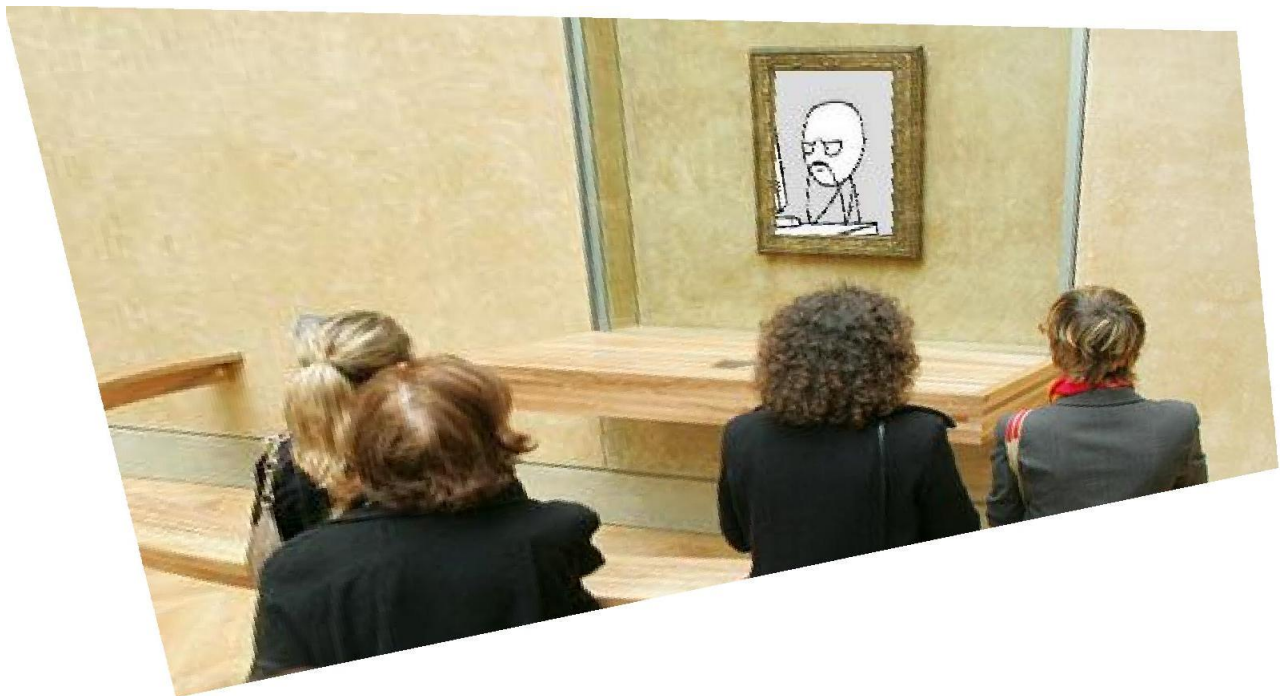


Fig 10. Project Fig 8. to real world plane. It is known that the size of Mona Lisa is

$77cm \times 53cm$ .

### Matlab Code Used in this section

```

1  clc
2  clear all;
3  close all;
4
5  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
6  % Load the data images into matlab
7  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
8  comic_img = imread('ragecomic.jpg');
9  louvre = imread('louvre.jpg');
10 size_comic = size(comic_img);
11 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
12 % Hard-coding the corrdinates location that were used to obtain
13 % the homography matrix.
14 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
15
16 x1w = 1;
17 y1w = 1;
18 x2w = 1;
19 y2w = size_comic(2)
20 x3w = size_comic(1);
21 y3w = 1;
22 x4w = size_comic(1);
23 y4w = size_comic(2);
24
25 x1 = 25;
26 y1 = 230;
27 x2 = 25;
28 y2 = 278;
29 x3 = 104;
30 y3 = 220;
31 x4 = 108;
32 y4 = 268;
33 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
34 % Solving for the homography matrix H
35 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
36 A = [x1w,y1w,1,0,0,0,-x1w*x1,-y1w*x1;
37      0,0,0,x1w,y1w,1,-x1w*y1,-y1w*y1;
38      x2w,y2w,1,0,0,0,-x2w*x2,-y2w*x2;
39      0,0,0,x2w,y2w,1,-x2w*y2,-y2w*y2;
40      x3w,y3w,1,0,0,0,-x3w*x3,-y3w*x3;
41      0,0,0,x3w,y3w,1,-x3w*y3,-y3w*y3;
42      x4w,y4w,1,0,0,0,-x4w*x4,-y4w*x4;
43      0,0,0,x4w,y4w,1,-x4w*y4,-y4w*y4];
44 h_vector = inv(A)*[x1;y1;x2;y2;x3;y3;x4;y4];
45 H = reshape([h_vector;1],[3,3]) '
46
47 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
48 % Perform projective transformation and save the projected image generated
49 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

50 tic
51
52 New_Img_Frame = louvre;
53 for i = 1:1:size_comic(1)
54     i
55     for j = 1:1:size_comic(2)
56         LocFrame = H*[i;j;1];
57         x = round(LocFrame(1)/LocFrame(3));
58         y = round(LocFrame(2)/LocFrame(3));
59         New_Img_Frame(x,y,:) = comic_img(i,j,:);
60     end
61 end
62 image(New_Img_Frame)
63 imwrite(New_Img_Frame, 'projected_comic.jpg', 'jpeg');
64 toc
65
66 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Task 2 in Part
67     4%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
68 close all
69 clear all
70 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
71 % Load the data images into matlab
72 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
73 comic_img = imread('ragecomic.jpg');
74 louvre = imread('louvre.jpg');
75 size_comic = size(comic_img);
76
77
78 Img_Frame = imread('projected_comic.jpg');
79 SizeImgFrame = size(Img_Frame);
80
81 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
82 % Hard-coding the coordinates location that were used to obtain
83 % the homography matrix.
84 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
85 x1w = 1;
86 y1w = 1;
87 x2w = 1;
88 y2w = size_comic(2)
89 x3w = size_comic(1);
90 y3w = 1;
91 x4w = size_comic(1);
92 y4w = size_comic(2);
93
94 x1 = 25;
95 y1 = 230;
96 x2 = 25;
97 y2 = 278;
98 x3 = 104;
99 y3 = 220;
100 x4 = 108;
101 y4 = 268;
102 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
103 % Solving for the homography matrix H

```

```

103 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
104 A = [x1w,y1w,1,0,0,0,-x1w*x1,-y1w*x1;
105      0,0,0,x1w,y1w,1,-x1w*y1,-y1w*y1;
106      x2w,y2w,1,0,0,0,-x2w*x2,-y2w*x2;
107      0,0,0,x2w,y2w,1,-x2w*y2,-y2w*y2;
108      x3w,y3w,1,0,0,0,-x3w*x3,-y3w*x3;
109      0,0,0,x3w,y3w,1,-x3w*y3,-y3w*y3;
110      x4w,y4w,1,0,0,0,-x4w*x4,-y4w*x4;
111      0,0,0,x4w,y4w,1,-x4w*y4,-y4w*y4];
112 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
113 % Alternative approach to obtain H, however the H would be in symbolic
114 % and the calculation in the for loops below would be EXTREMELY SLOW!
115 % Using inverse matrix has been proved a lot faster than using 'solve'
116 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
117 % syms h11 h12 h13 h21 h22 h23 h31 h32
118 % S = solve(A*[h11;h12;h13;h21;h22;h23;h31;h32] == [x1;y1;x2;y2;x3;y3;x4;y4])
119 % H = [S.h11,S.h12,S.h13;
120 %      S.h21,S.h22,S.h23;
121 %      S.h31,S.h32,1];
122 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
123 h_vector = inv(A)*[x1;y1;x2;y2;x3;y3;x4;y4];
124 H = reshape([h_vector;1],[3,3])'
125
126 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
127 % The below part generated the minimum size image frame so that all
128 % pixels in image 'frame' could be mapped to our generated image.
129 % Note that there will be plenty of blank area because after projection
130 % the image is not rectangle anymore and hence will only utilize
131 % a portion of pixels in a rectangle frame.
132 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
133
134 newp1 = H^(-1)*[1;1;1];
135 newp2 = H^(-1)*[1;SizeImgFrame(2);1];
136 newp3 = H^(-1)*[SizeImgFrame(1);1;1];
137 newp4 = H^(-1)*[SizeImgFrame(1);SizeImgFrame(2);1];
138
139 xp1 = round(newp1(1)/newp1(3))
140 xp2 = round(newp2(1)/newp2(3))
141 xp3 = round(newp3(1)/newp3(3))
142 xp4 = round(newp4(1)/newp4(3))
143
144 yp1 = round(newp1(2)/newp1(3))
145 yp2 = round(newp2(2)/newp2(3))
146 yp3 = round(newp3(2)/newp3(3))
147 yp4 = round(newp4(2)/newp4(3))
148
149 pointxp1 = double(xp1);
150 pointxp2 = double(xp2);
151 pointxp3 = double(xp3);
152 pointxp4 = double(xp4);
153
154 pointyp1 = double(yp1);
155 pointyp2 = double(yp2);
156 pointyp3 = double(yp3);

```

```

157 pointyp4 = double(yp4);
158
159 A = [pointxp1,pointxp2,pointxp3,pointxp4];
160 B = [pointyp1,pointyp2,pointyp3,pointyp4];
161
162 tx = abs(min(A)) + 1;
163 ty = abs(min(B)) + 1;
164
165 frame_x = max(A) + tx
166 frame_y = max(B) + ty
167 Corrected_Img(1:frame_x,1:frame_y,1:3) = 255;
168 Projected_Img = imread('projected_comic.jpg');
169
170 tic
171 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
172 % Perform projective transformation and save the projected image generated
173 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
174 for i = 1:1:frame_x
175     i
176     for j = 1:1:frame_y
177         LocFrame = H*[(i-tx);(j-ty);1];
178         x = double(round(LocFrame(1)/LocFrame(3)));
179         y = double(round(LocFrame(2)/LocFrame(3)));
180         if (x<SizeImgFrame(1)) && (x>0) && (y<SizeImgFrame(2)) && (y>0)
181             Corrected_Img(i,j,:) = Projected_Img(x,y,:);
182             x;
183             y;
184         else
185             end
186     end
187 end
188
189
190 Corrected_Img = uint8(Corrected_Img);
191 image(Corrected_Img)
192 imwrite(Corrected_Img,'corrected_img_comic-1.jpg','jpeg');
193 toc
194
195 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Task 3 in part
196 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
197 close all
198 clear all
199
200 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
201 % Load the data images into matlab
202 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
203
204 Img_Frame = imread('projected_comic.jpg');
205 SizeImgFrame = size(Img_Frame);
206 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
207 % Hard-coding the corrdinates location that were used to obtain
208 % the homography matrix.
209 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
210 x1w = 1;

```

```

210 y1w = 1;
211 x2w = 1;
212 y2w = 106;
213 x3w = 154;
214 y3w = 1;
215 x4w = 154;
216 y4w = 106;
217
218 x1 = 25;
219 y1 = 230;
220 x2 = 25;
221 y2 = 278;
222 x3 = 104;
223 y3 = 220;
224 x4 = 108;
225 y4 = 268;
226 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
227 % Solving for the homography matrix H
228 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
229 A = [x1w,y1w,1,0,0,0,-x1w*x1,-y1w*x1;
230      0,0,0,x1w,y1w,1,-x1w*y1,-y1w*y1;
231      x2w,y2w,1,0,0,0,-x2w*x2,-y2w*x2;
232      0,0,0,x2w,y2w,1,-x2w*y2,-y2w*y2;
233      x3w,y3w,1,0,0,0,-x3w*x3,-y3w*x3;
234      0,0,0,x3w,y3w,1,-x3w*y3,-y3w*y3;
235      x4w,y4w,1,0,0,0,-x4w*x4,-y4w*x4;
236      0,0,0,x4w,y4w,1,-x4w*y4,-y4w*y4];
237 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
238 % Alternative approach to obtain H, however the H would be in symbolic
239 % and the calculation in the for loops below would be EXTREMELY SLOW!
240 % Using inverse matrix has been proved a lot faster than using 'solve'
241 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
242 % syms h11 h12 h13 h21 h22 h23 h31 h32
243 % S = solve(A*[h11;h12;h13;h21;h22;h23;h31;h32] == [x1;y1;x2;y2;x3;y3;x4;y4])
244 % H = [S.h11,S.h12,S.h13;
245 %      S.h21,S.h22,S.h23;
246 %      S.h31,S.h32,1];
247 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
248 h_vector = inv(A)*[x1;y1;x2;y2;x3;y3;x4;y4];
249 H = reshape([h_vector;1],[3,3])'
250
251 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
252 % The below part generated the minimum size image frame so that all
253 % pixels in image 'frame' could be mapped to our generated image.
254 % Note that there will be plenty of blank area because after projection
255 % the image is not rectangle anymore and hence will only utilize
256 % a portion of pixels in a rectangle frame.
257 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
258
259 newp1 = H^(-1)*[1;1;1];
260 newp2 = H^(-1)*[1;SizeImgFrame(2);1];
261 newp3 = H^(-1)*[SizeImgFrame(1);1;1];
262 newp4 = H^(-1)*[SizeImgFrame(1);SizeImgFrame(2);1];
263

```

```
264 xp1 = round(newp1(1)/newp1(3))
265 xp2 = round(newp2(1)/newp2(3))
266 xp3 = round(newp3(1)/newp3(3))
267 xp4 = round(newp4(1)/newp4(3))
268
269 yp1 = round(newp1(2)/newp1(3))
270 yp2 = round(newp2(2)/newp2(3))
271 yp3 = round(newp3(2)/newp3(3))
272 yp4 = round(newp4(2)/newp4(3))
273
274 pointxp1 = double(xp1);
275 pointxp2 = double(xp2);
276 pointxp3 = double(xp3);
277 pointxp4 = double(xp4);
278
279 pointyp1 = double(yp1);
280 pointyp2 = double(yp2);
281 pointyp3 = double(yp3);
282 pointyp4 = double(yp4);
283
284 A = [pointxp1,pointxp2,pointxp3,pointxp4];
285 B = [pointyp1,pointyp2,pointyp3,pointyp4];
286
287 tx = abs(min(A)) + 1;
288 ty = abs(min(B)) + 1;
289
290 frame_x = max(A) + tx
291 frame_y = max(B) + ty
292 Corrected_Img(1:frame_x,1:frame_y,1:3) = 255;
293 Projected_Img = imread('projected_comic.jpg');
294 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
295 % Perform projective transformation and save the projected image generated
296 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
297 tic
298
299 for i = 1:1:frame_x
300     i
301     for j = 1:1:frame_y
302         LocFrame = H*[(i-tx);(j-ty);1];
303         x = double(round(LocFrame(1)/LocFrame(3)));
304         y = double(round(LocFrame(2)/LocFrame(3)));
305         if (x<SizeImgFrame(1)) && (x>0) && (y<SizeImgFrame(2)) && (y>0)
306             Corrected_Img(i,j,:) = Projected_Img(x,y,:);
307             x;
308             y;
309         else
310             end
311     end
312 end
313
314 Corrected_Img = uint8(Corrected_Img);
315 image(Corrected_Img)
316 imwrite(Corrected_Img,'corrected_img_comic.jpg','jpeg');
317 toc
```

```
318
319 %%Part 3 in part 4 cont.: Mapping Task 2 in part 2 to real world plane
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
320
321
322 Projected_Img = imread('corrected_img_comic_1.jpg');
323 Projected_Img_Size = size(Projected_Img);
324 Corrected_Img(1:Projected_Img_Size(1),1:round(Projected_Img_Size(2)*(53/77))
    ,1:3) = 255;
325 for i = 1:1:Projected_Img_Size(1)
326     for j = 1:1:Projected_Img_Size(2)
327         x = i;
328         y = round(j*(53/77));
329         Corrected_Img(x,y,:) = Projected_Img(i,j,:);
330     end
331 end
332
333 Corrected_Img = uint8(Corrected_Img);
334 image(Corrected_Img)
335 truesize
336 imwrite(Corrected_Img, 'corrected_img_comic_2.jpg', 'jpeg');
337 toc
```