ECE661: Computer Vision (Fall 2014)

Shaobo Fang: s-fang@purdue

September 11, 2014

Contents

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:

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

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):

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

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.

```
close all
clear all
clc;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Load the data images into matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Img_Frame = imread('Frame.jpg');
Img_Audrey = imread('Audrey.jpg');
SizeImgFrame = size(Img_Frame);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Hard-coding the corrdinates location that were used to obtain
% the homography matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x1w = 1;
y1w = 1;
x2w = 1;
y2w = 500;
x3w = 508;
y3w = 1;
x4w = 508;
y4w = 500;
```

```
x1 = 150;
y1 = 188;
x2 = 176;
y2 = 346;
x3 = 462;
y3 = 184;
x4 = 433;
y4 = 345;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Solving for the homography matrix H
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

A = [x1w,y1w,1,0,0,0,-x1w*x1,-y1w*x1;...
     0,0,0,x1w,y1w,1,-x1w*y1,-y1w*y1;...
     x2w,y2w,1,0,0,0,-x2w*x2,-y2w*x2;...
     0,0,0,x2w,y2w,1,-x2w*y2,-y2w*y2;...
     x3w,y3w,1,0,0,0,-x3w*x3,-y3w*x3;...
     0,0,0,x3w,y3w,1,-x3w*y3,-y3w*y3;...
     x4w,y4w,1,0,0,0,-x4w*x4,-y4w*x4;...
     0,0,0,x4w,y4w,1,-x4w*y4,-y4w*y4]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Alternative approach to obatin H, however the H would be in symbolic
% and the calculation in the for loops below would be EXTREMELY SLOW!
% Using inverse matrix has been proved a lot faster than using 'solve'
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% syms h11 h12 h13 h21 h22 h23 h31 h32
% S = solve(A*[h11;h12;h13;h21;h22;h23;h31;h32] == [x1;y1;x2;y2;x3;y3;x4;y4])
% H = [S.h11,S.h12,S.h13;
%       S.h21,S.h22,S.h23;
%       S.h31,S.h32,1];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

h_vector = inv(A)*[x1;y1;x2;y2;x3;y3;x4;y4];
H = reshape([h_vector;1],[3,3])'



tic

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Perform projective transformation and save the projected image generated
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

New_Img_Frame = Img_Frame;
SizeImgAudrey = size(Img_Audrey);
for i = 1:1:SizeImgAudrey(1)
    i
    for j = 1:1:SizeImgAudrey(2)
        LocFrame = H*[i;j;1];
        x = round(LocFrame(1)/LocFrame(3));
        y = round(LocFrame(2)/LocFrame(3));
```

```
        New_Img_Frame(x,y,:) = Img_Audrey(i,j,:);
    end
end
image(New_Img_Frame)
imwrite(New_Img_Frame,'projected_img_1.jpg','jpeg');
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

```
close all
clear all
clc;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Load the data images into matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Img_Frame = imread('Frame.jpg');
Img_Audrey = imread('Audrey.jpg');

SizeImgFrame = size(Img_Frame);

```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Hard-coding the corrdinates location that were used to obtain
% the homography matrix. PLEASE NOTE THAT IN ORDER TO REDUCE SOME
% UNNECESSARY COMPUTATION, we will map frame ABCD to a 100*100 pixels
% squared frame. Of course we can easily change x2w, x4w = 508 and
% y2w, y4w = 500. The computation scale would be significantly larger
% while the result did not improve significatntly. (So I did not see
% the advantage)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

x1w = 1;
y1w = 1;
x2w = 1;
y2w = 100;
x3w = 100;
y3w = 1;
x4w = 100;
y4w = 100;

x1 = 201;
y1 = 486;
x2 = 212;
y2 = 564;
x3 = 422;
y3 = 484;
x4 = 405;
y4 = 566;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Solving for the homography matrix H
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
A = [x1w,y1w,1,0,0,0,-x1w*x1,-y1w*x1;
     0,0,0,x1w,y1w,1,-x1w*y1,-y1w*y1;
     x2w,y2w,1,0,0,0,-x2w*x2,-y2w*x2;
     0,0,0,x2w,y2w,1,-x2w*y2,-y2w*y2;
     x3w,y3w,1,0,0,0,-x3w*x3,-y3w*x3;
     0,0,0,x3w,y3w,1,-x3w*y3,-y3w*y3;
     x4w,y4w,1,0,0,0,-x4w*x4,-y4w*x4;
     0,0,0,x4w,y4w,1,-x4w*y4,-y4w*y4;];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Alternative approach to obatin H, however the H would be in symbolic
% and the calculation in the for loops below would be EXTREMELY SLOW!
% Using inverse matrix has been proved a lot faster than using 'solve'
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% syms h11 h12 h13 h21 h22 h23 h31 h32
% S = solve(A*[h11;h12;h13;h21;h22;h23;h31;h32] == [x1;y1;x2;y2;x3;y3;x4;y4])
% H = [S.h11,S.h12,S.h13;
%      S.h21,S.h22,S.h23;
%      S.h31,S.h32,1];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
h_vector = inv(A)*[x1;y1;x2;y2;x3;y3;x4;y4];
H = reshape([h_vector;1],[3,3])'
```

```matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The below part generated the minimum size image frame so that all
% pixles in image 'frame' could be mapped to our generated image.
% Note that there will be plenty of blank area because after projection
% the image is not retangle anymore and hence will only utilize
% a portion of pixles in a retangle frame.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

H*[1;1;1]
H*[1;100;1]
H*[100;1;1]
H*[100;100;1]


newp1 = H^(-1)*[1;1;1]
newp2 = H^(-1)*[1;SizeImgFrame(2);1]
newp3 = H^(-1)*[SizeImgFrame(1);1;1]
newp4 = H^(-1)*[SizeImgFrame(1);SizeImgFrame(2);1]

xp1 = round(newp1(1)/newp1(3))
xp2 = round(newp2(1)/newp2(3))
xp3 = round(newp3(1)/newp3(3))
xp4 = round(newp4(1)/newp4(3))

yp1 = round(newp1(2)/newp1(3))
yp2 = round(newp2(2)/newp2(3))
yp3 = round(newp3(2)/newp3(3))
yp4 = round(newp4(2)/newp4(3))

pointxp1 = double(xp1);
pointxp2 = double(xp2);
pointxp3 = double(xp3);
pointxp4 = double(xp4);

pointyp1 = double(yp1);
pointyp2 = double(yp2);
pointyp3 = double(yp3);
pointyp4 = double(yp4);

A = [pointxp1,pointxp2,pointxp3,pointxp4];
B = [pointyp1,pointyp2,pointyp3,pointyp4];

tx = abs(min(A)) + 1;
ty = abs(min(B)) + 1;

frame_x = max(A)  - min(A) + 1;
frame_y = max(B)  - min(B) + 1;
Corrected_Img(1:frame_x,1:frame_y,1:3) = 255;
Projected_Img = imread('Frame.jpg');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Perform projective transformation and save the projected image generated
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tic
```

```
 for i = 1:1:frame_x
     i
     for j = 1:1:frame_y
         LocFrame = H*[(i-tx);(j-ty);1];
         x = double(round(LocFrame(1)/LocFrame(3)));
         y = double(round(LocFrame(2)/LocFrame(3)));
         if (x<SizeImgFrame(1))&&(x>0)&&(y<SizeImgFrame(2))&&(y>0)
         Corrected_Img(i,j,:) = Projected_Img(x,y,:);
         x;
         y;
         else
         end
     end
 end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Mapping the audrey image to the squared box A'B'C'D' in our projected frame
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for i = 1:1:508
    for j = 1:1:500
        x = round(i/5) + tx;
        y = round(j/5) + ty;
        Corrected_Img(x,y,:) = Img_Audrey(i,j,:);
    end
end

Corrected_Img = uint8(Corrected_Img);
image(Corrected_Img)
truesize
imwrite(Corrected_Img,'projected_img_2.jpg','jpeg');
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 pale 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 pale of the image obtained in part 2. Audrey is in frame ABCD.

Matlab Code used in this section

```
close all
clear all
clc;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Load the data images into matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Img_Frame = imread('Frame.jpg');
Img_Audrey = imread('Audrey.jpg');
SizeImgFrame = size(Img_Frame);
Projected_Img = imread('projected_img_1.jpg');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Hard-coding the corrdinates location that were used to obtain
% the homography matrix.
% Please note what matters in the world plane is the correct ratio.
% Thus we set the desired ratio for frame PQRS(and ABCD) to be 63*91
% (equivalently 126 * 182. We can approximately assume the frame PQRS
% and frame ABCD have the same ratio.)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

x1w = 1;
y1w = 1;
x2w = 1;
```

```
y2w = 126;
x3w = 182;
y3w = 1;
x4w = 182;
y4w = 126;

x1 = 150;
y1 = 188;
x2 = 176;
y2 = 346;
x3 = 462;
y3 = 184;
x4 = 433;
y4 = 345;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Solving for the homography matrix H
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
A = [x1w,y1w,1,0,0,0,-x1w*x1,-y1w*x1;
     0,0,0,x1w,y1w,1,-x1w*y1,-y1w*y1;
     x2w,y2w,1,0,0,0,-x2w*x2,-y2w*x2;
     0,0,0,x2w,y2w,1,-x2w*y2,-y2w*y2;
     x3w,y3w,1,0,0,0,-x3w*x3,-y3w*x3;
     0,0,0,x3w,y3w,1,-x3w*y3,-y3w*y3;
     x4w,y4w,1,0,0,0,-x4w*x4,-y4w*x4;
     0,0,0,x4w,y4w,1,-x4w*y4,-y4w*y4;];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Alternative approach to obatin H, however the H would be in symbolic
% and the calculation in the for loops below would be EXTREMELY SLOW!
% Using inverse matrix has been proved a lot faster than using 'solve'
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% syms h11 h12 h13 h21 h22 h23 h31 h32
% S = solve(A*[h11;h12;h13;h21;h22;h23;h31;h32] == [x1;y1;x2;y2;x3;y3;x4;y4])
% H = [S.h11,S.h12,S.h13;
%      S.h21,S.h22,S.h23;
%      S.h31,S.h32,1];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
h_vector = inv(A)*[x1;y1;x2;y2;x3;y3;x4;y4];
H = reshape([h_vector;1],[3,3])'



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The below part generated the minimum size image frame so that all
% pixles in image 'frame' could be mapped to our generated image.
% Note that there will be plenty of blank area because after projection
% the image is not retangle anymore and hence will only utilize
% a portion of pixles in a retangle frame.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

newp1 = H^(-1)*[1;1;1];
newp2 = H^(-1)*[1;SizeImgFrame(2);1];
newp3 = H^(-1)*[SizeImgFrame(1);1;1];
newp4 = H^(-1)*[SizeImgFrame(1);SizeImgFrame(2);1];

```

```
xp1 = round(newp1(1)/newp1(3))
xp2 = round(newp2(1)/newp2(3))
xp3 = round(newp3(1)/newp3(3))
xp4 = round(newp4(1)/newp4(3))

yp1 = round(newp1(2)/newp1(3))
yp2 = round(newp2(2)/newp2(3))
yp3 = round(newp3(2)/newp3(3))
yp4 = round(newp4(2)/newp4(3))

pointxp1 = double(xp1);
pointxp2 = double(xp2);
pointxp3 = double(xp3);
pointxp4 = double(xp4);

pointyp1 = double(yp1);
pointyp2 = double(yp2);
pointyp3 = double(yp3);
pointyp4 = double(yp4);

A = [pointxp1,pointxp2,pointxp3,pointxp4];
B = [pointyp1,pointyp2,pointyp3,pointyp4];

tx = abs(min(A)) + 1;
ty = abs(min(B)) + 1;

frame_x = max(A) + tx
frame_y = max(B) + ty
Corrected_Img(1:frame_x,1:frame_y,1:3) = 255;


tic
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Perform projective transformation and save the projected image generated
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 for i = 1:1:frame_x
     i
     for j = 1:1:frame_y
         LocFrame = H*[(i-tx);(j-ty);1];
         x = double(round(LocFrame(1)/LocFrame(3)));
         y = double(round(LocFrame(2)/LocFrame(3)));
         if (x<SizeImgFrame(1))&&(x>0)&&(y<SizeImgFrame(2))&&(y>0)
         Corrected_Img(i,j,:) = Projected_Img(x,y,:);
         x;
         y;
         else
         end
     end
end

Corrected_Img = uint8(Corrected_Img);
image(Corrected_Img)
imwrite(Corrected_Img,'corrected_img_1.jpg','jpeg');
```

```
toc

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Perform the similar approach for the image we obatianed in part 2.
% Note that if we think about our result in part 2, we will notice
% we only need to stretch/compress x,y axis to get the corrected projection
% in world plane.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


close all
clear all
clc;
Img_Frame = imread('Frame.jpg');
Img_Audrey = imread('Audrey.jpg');

SizeImgFrame = size(Img_Frame);
Projected_Img = imread('projected_img_2.jpg');
Projected_Img_Size = size(Projected_Img);
Corrected_Img(1:Projected_Img_Size(1),1:round(Projected_Img_Size(2)*(2/3))
    ,1:3) = 255;
for i = 1:1:Projected_Img_Size(1)
    for j = 1:1:Projected_Img_Size(2)
        x = i;
        y = round(j*(2/3));
        Corrected_Img(x,y,:) = Projected_Img(i,j,:);
    end
end

Corrected_Img = uint8(Corrected_Img);
image(Corrected_Img)
truesize
imwrite(Corrected_Img,'corrected_img_2.jpg','jpeg');
toc
```

5 Projective Transformation Performed On My Own Pair of Images

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

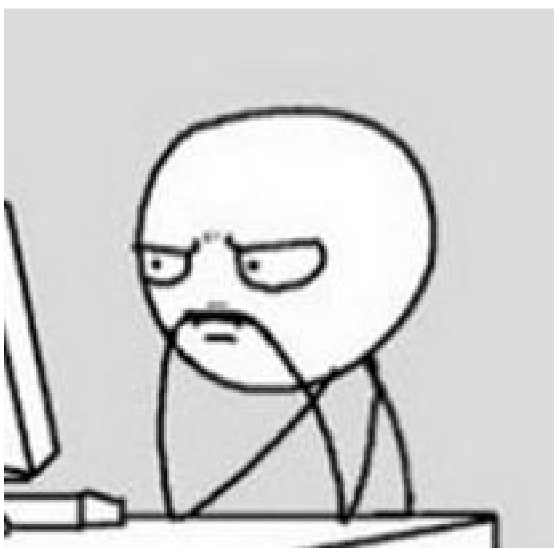

Fig 6. Face of a Typical ECE Graduate Student at Purdue University that we want to project to cover `Mona Lisa`.

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 $77cm \times 53cm$.

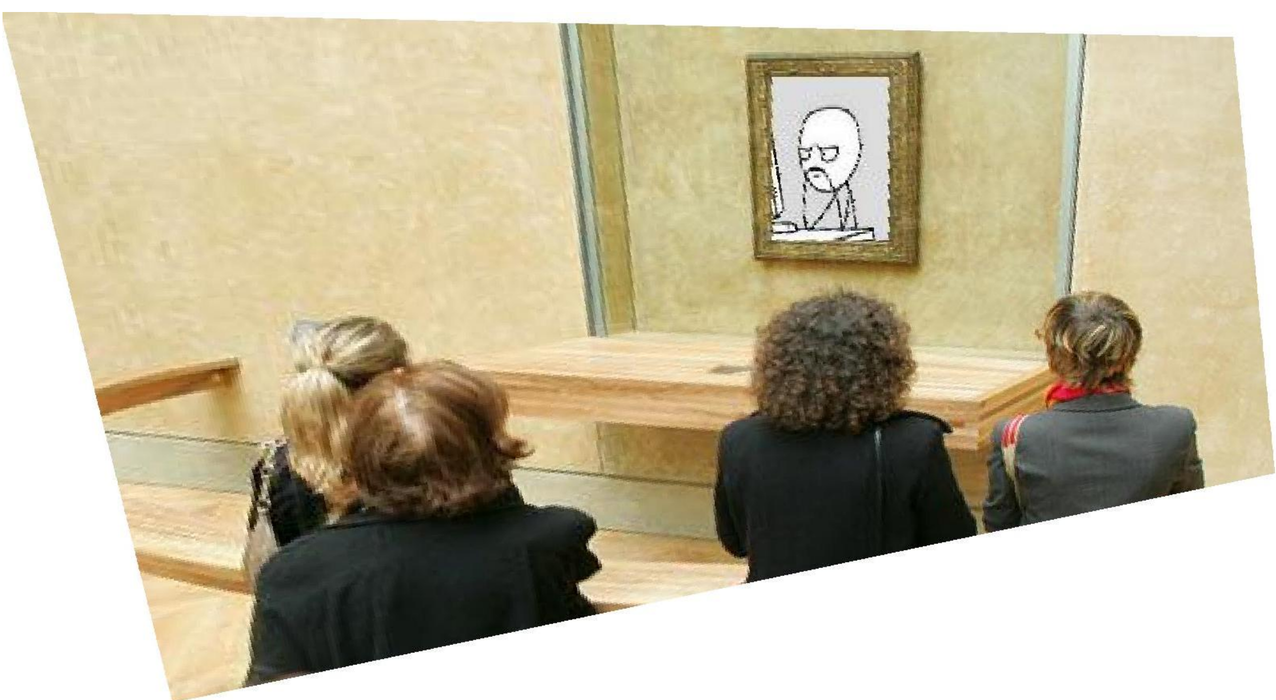

Fig 10. Project Fig 8. to real world plane. It is known that the size of `Mona Lisa` is

$$77cm \times 53cm.$$

Matlab Code Used in this section

```
clc
clear all;
close all;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Load the data images into matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
comic_img = imread('ragecomic.jpg');
louvre = imread('louvre.jpg');
size_comic = size(comic_img);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Hard-coding the corrdinates location that were used to obtain
% the homography matrix.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

x1w = 1;
y1w = 1;
x2w = 1;
y2w = size_comic(2)
x3w = size_comic(1);
y3w = 1;
x4w = size_comic(1);
y4w = size_comic(2);

x1 = 25;
y1 = 230;
x2 = 25;
y2 = 278;
x3 = 104;
y3 = 220;
x4 = 108;
y4 = 268;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Solving for the homography matrix H
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
A = [x1w,y1w,1,0,0,0,-x1w*x1,-y1w*x1;
     0,0,0,x1w,y1w,1,-x1w*y1,-y1w*y1;
     x2w,y2w,1,0,0,0,-x2w*x2,-y2w*x2;
     0,0,0,x2w,y2w,1,-x2w*y2,-y2w*y2;
     x3w,y3w,1,0,0,0,-x3w*x3,-y3w*x3;
     0,0,0,x3w,y3w,1,-x3w*y3,-y3w*y3;
     x4w,y4w,1,0,0,0,-x4w*x4,-y4w*x4;
     0,0,0,x4w,y4w,1,-x4w*y4,-y4w*y4;];
h_vector = inv(A)*[x1;y1;x2;y2;x3;y3;x4;y4];
H = reshape([h_vector;1],[3,3])'

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Perform projective transformation and save the projected image generated
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
tic

New_Img_Frame = louvre;
for i = 1:1:size_comic(1)
    i
    for j = 1:1:size_comic(2)
        LocFrame = H*[i;j;1];
        x = round(LocFrame(1)/LocFrame(3));
        y = round(LocFrame(2)/LocFrame(3));
        New_Img_Frame(x,y,:) = comic_img(i,j,:);
    end
end
image(New_Img_Frame)
imwrite(New_Img_Frame,'projected_comic.jpg','jpeg');
toc

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Task 2 in Part 4%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all
clear all
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Load the data images into matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
comic_img = imread('ragecomic.jpg');
louvre = imread('louvre.jpg');
size_comic = size(comic_img);


Img_Frame = imread('projected_comic.jpg');
SizeImgFrame = size(Img_Frame);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Hard-coding the corrdinates location that were used to obtain
% the homography matrix.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x1w = 1;
y1w = 1;
x2w = 1;
y2w = size_comic(2)
x3w = size_comic(1);
y3w = 1;
x4w = size_comic(1);
y4w = size_comic(2);

x1 = 25;
y1 = 230;
x2 = 25;
y2 = 278;
x3 = 104;
y3 = 220;
x4 = 108;
y4 = 268;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Solving for the homography matrix H
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
A = [x1w,y1w,1,0,0,0,-x1w*x1,-y1w*x1;
     0,0,0,x1w,y1w,1,-x1w*y1,-y1w*y1;
     x2w,y2w,1,0,0,0,-x2w*x2,-y2w*x2;
     0,0,0,x2w,y2w,1,-x2w*y2,-y2w*y2;
     x3w,y3w,1,0,0,0,-x3w*x3,-y3w*x3;
     0,0,0,x3w,y3w,1,-x3w*y3,-y3w*y3;
     x4w,y4w,1,0,0,0,-x4w*x4,-y4w*x4;
     0,0,0,x4w,y4w,1,-x4w*y4,-y4w*y4;];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Alternative approach to obatin H, however the H would be in symbolic
% and the calculation in the for loops below would be EXTREMELY SLOW!
% Using inverse matrix has been proved a lot faster than using 'solve'
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% syms h11 h12 h13 h21 h22 h23 h31 h32
% S = solve(A*[h11;h12;h13;h21;h22;h23;h31;h32] == [x1;y1;x2;y2;x3;y3;x4;y4])
% H = [S.h11,S.h12,S.h13;
%       S.h21,S.h22,S.h23;
%       S.h31,S.h32,1];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
h_vector = inv(A)*[x1;y1;x2;y2;x3;y3;x4;y4];
H = reshape([h_vector;1],[3,3])'

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The below part generated the minimum size image frame so that all
% pixles in image 'frame' could be mapped to our generated image.
% Note that there will be plenty of blank area because after projection
% the image is not retangle anymore and hence will only utilize
% a portion of pixles in a retangle frame.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

newp1 = H^(-1)*[1;1;1];
newp2 = H^(-1)*[1;SizeImgFrame(2);1];
newp3 = H^(-1)*[SizeImgFrame(1);1;1];
newp4 = H^(-1)*[SizeImgFrame(1);SizeImgFrame(2);1];

xp1 = round(newp1(1)/newp1(3))
xp2 = round(newp2(1)/newp2(3))
xp3 = round(newp3(1)/newp3(3))
xp4 = round(newp4(1)/newp4(3))

yp1 = round(newp1(2)/newp1(3))
yp2 = round(newp2(2)/newp2(3))
yp3 = round(newp3(2)/newp3(3))
yp4 = round(newp4(2)/newp4(3))

pointxp1 = double(xp1);
pointxp2 = double(xp2);
pointxp3 = double(xp3);
pointxp4 = double(xp4);

pointyp1 = double(yp1);
pointyp2 = double(yp2);
pointyp3 = double(yp3);
```

```
pointyp4 = double(yp4);

A = [pointxp1,pointxp2,pointxp3,pointxp4];
B = [pointyp1,pointyp2,pointyp3,pointyp4];

tx = abs(min(A)) + 1;
ty = abs(min(B)) + 1;

frame_x = max(A) + tx
frame_y = max(B) + ty
Corrected_Img(1:frame_x,1:frame_y,1:3) = 255;
Projected_Img = imread('projected_comic.jpg');

tic
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Perform projective transformation and save the projected image generated
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 for i = 1:1:frame_x
     i
     for j = 1:1:frame_y
         LocFrame = H*[(i-tx);(j-ty);1];
         x = double(round(LocFrame(1)/LocFrame(3)));
         y = double(round(LocFrame(2)/LocFrame(3)));
         if (x<SizeImgFrame(1))&&(x>0)&&(y<SizeImgFrame(2))&&(y>0)
         Corrected_Img(i,j,:) = Projected_Img(x,y,:);
         x;
         y;
         else
         end
     end
end


Corrected_Img = uint8(Corrected_Img);
image(Corrected_Img)
imwrite(Corrected_Img,'corrected_img_comic_1.jpg','jpeg');
toc

%%%%%%%%%%%%%%%%%%%%%%%%%Task 3 in part 4%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all
clear all

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Load the data images into matlab
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Img_Frame = imread('projected_comic.jpg');
SizeImgFrame = size(Img_Frame);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Hard-coding the corrdinates location that were used to obtain
% the homography matrix.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
x1w = 1;
```

```
y1w = 1;
x2w = 1;
y2w = 106;
x3w = 154;
y3w = 1;
x4w = 154;
y4w = 106;

x1 = 25;
y1 = 230;
x2 = 25;
y2 = 278;
x3 = 104;
y3 = 220;
x4 = 108;
y4 = 268;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Solving for the homography matrix H
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
A = [x1w,y1w,1,0,0,0,-x1w*x1,-y1w*x1;
     0,0,0,x1w,y1w,1,-x1w*y1,-y1w*y1;
     x2w,y2w,1,0,0,0,-x2w*x2,-y2w*x2;
     0,0,0,x2w,y2w,1,-x2w*y2,-y2w*y2;
     x3w,y3w,1,0,0,0,-x3w*x3,-y3w*x3;
     0,0,0,x3w,y3w,1,-x3w*y3,-y3w*y3;
     x4w,y4w,1,0,0,0,-x4w*x4,-y4w*x4;
     0,0,0,x4w,y4w,1,-x4w*y4,-y4w*y4;];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Alternative approach to obatin H, however the H would be in symbolic
% and the calculation in the for loops below would be EXTREMELY SLOW!
% Using inverse matrix has been proved a lot faster than using 'solve'
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% syms h11 h12 h13 h21 h22 h23 h31 h32
% S = solve(A*[h11;h12;h13;h21;h22;h23;h31;h32] == [x1;y1;x2;y2;x3;y3;x4;y4])
% H = [S.h11,S.h12,S.h13;
%      S.h21,S.h22,S.h23;
%      S.h31,S.h32,1];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
h_vector = inv(A)*[x1;y1;x2;y2;x3;y3;x4;y4];
H = reshape([h_vector;1],[3,3])'

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The below part generated the minimum size image frame so that all
% pixles in image 'frame' could be mapped to our generated image.
% Note that there will be plenty of blank area because after projection
% the image is not retangle anymore and hence will only utilize
% a portion of pixles in a retangle frame.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

newp1 = H^(-1)*[1;1;1];
newp2 = H^(-1)*[1;SizeImgFrame(2);1];
newp3 = H^(-1)*[SizeImgFrame(1);1;1];
newp4 = H^(-1)*[SizeImgFrame(1);SizeImgFrame(2);1];

```

```
xp1 = round(newp1(1)/newp1(3))
xp2 = round(newp2(1)/newp2(3))
xp3 = round(newp3(1)/newp3(3))
xp4 = round(newp4(1)/newp4(3))

yp1 = round(newp1(2)/newp1(3))
yp2 = round(newp2(2)/newp2(3))
yp3 = round(newp3(2)/newp3(3))
yp4 = round(newp4(2)/newp4(3))

pointxp1 = double(xp1);
pointxp2 = double(xp2);
pointxp3 = double(xp3);
pointxp4 = double(xp4);

pointyp1 = double(yp1);
pointyp2 = double(yp2);
pointyp3 = double(yp3);
pointyp4 = double(yp4);

A = [pointxp1,pointxp2,pointxp3,pointxp4];
B = [pointyp1,pointyp2,pointyp3,pointyp4];

tx = abs(min(A)) + 1;
ty = abs(min(B)) + 1;

frame_x = max(A) + tx
frame_y = max(B) + ty
Corrected_Img(1:frame_x,1:frame_y,1:3) = 255;
Projected_Img = imread('projected_comic.jpg');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Perform projective transformation and save the projected image generated
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tic

 for i = 1:1:frame_x
     i
     for j = 1:1:frame_y
         LocFrame = H*[(i-tx);(j-ty);1];
         x = double(round(LocFrame(1)/LocFrame(3)));
         y = double(round(LocFrame(2)/LocFrame(3)));
         if (x<SizeImgFrame(1))&&(x>0)&&(y<SizeImgFrame(2))&&(y>0)
         Corrected_Img(i,j,:) = Projected_Img(x,y,:);
         x;
         y;
         else
         end
     end
end

Corrected_Img = uint8(Corrected_Img);
image(Corrected_Img)
imwrite(Corrected_Img,'corrected_img_comic.jpg','jpeg');
toc
```

```
%%Part 3 in part 4 cont.: Mapping Task 2 in part 2 to real world plane %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


Projected_Img = imread('corrected_img_comic_1.jpg');
Projected_Img_Size = size(Projected_Img);
Corrected_Img(1:Projected_Img_Size(1),1:round(Projected_Img_Size(2)*(53/77)),1:3) = 255;
for i = 1:1:Projected_Img_Size(1)
    for j = 1:1:Projected_Img_Size(2)
        x = i;
        y = round(j*(53/77));
        Corrected_Img(x,y,:) = Projected_Img(i,j,:);
    end
end

Corrected_Img = uint8(Corrected_Img);
image(Corrected_Img)
truesize
imwrite(Corrected_Img,'corrected_img_comic_2.jpg','jpeg');
toc
```