Problem1

part(a)

As given:
f = 8mm
ux = 800/4 = 200 pixels/mm
uy = 600/3 = 200 pixels/mm
u = 400;
v = 300
Camera Matrix

$$C = \begin{bmatrix} fu_x & s & u_0 \\ 0 & fu_y & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

scaling is zero so s=0;

$$C = \begin{bmatrix} 8*200 & 0 & 400 \\ 0 & 8*200 & 300 \\ 0 & 0 & 1 \end{bmatrix}$$

$$C = \begin{bmatrix} 1600 & 0 & 400 \\ 0 & 1600 & 300 \\ 0 & 0 & 1 \end{bmatrix}$$

part (b).

Quaternion

```
rotate = makehgtform('axisrotate',[3/sqrt(26),4/sqrt(26),-1/sqrt(26)],pi/3);
%% for translation
rotate(:,4) = [0 0 10 1];
Q = rotate;
```

```
Q =

    0.6731    0.4006    0.6217         0
    0.0609    0.8077   -0.5864         0
   -0.7371    0.4326    0.5192   10.0000
    0.000     0.000      0.000    1.000
```

Part C

```
% Cube coordinates (cc) are along the cube center (0,0,0)

cc = [-1 -1 -1 1; 1 -1 -1 1; -1 1 -1 1; 1 1 1 1; -1 1 1 1; 1 -1 1 1; 1 1 -1 1; -1 1 -1 1];

pixel_coordinates = C * WM * cc';
```

converting to homogeneous form

```
for i = 1:8
    pixel_coordinates(1,i) = pixel_coordinates(1,i)/pixel_coordinates(3,i);
    pixel_coordinates(2,i) = pixel_coordinates(2,i)/pixel_coordinates(3,i);
    pixel_coordinates(3,i) = pixel_coordinates(3,i)/pixel_coordinates(3,i);

end

pixel_coordinates
```

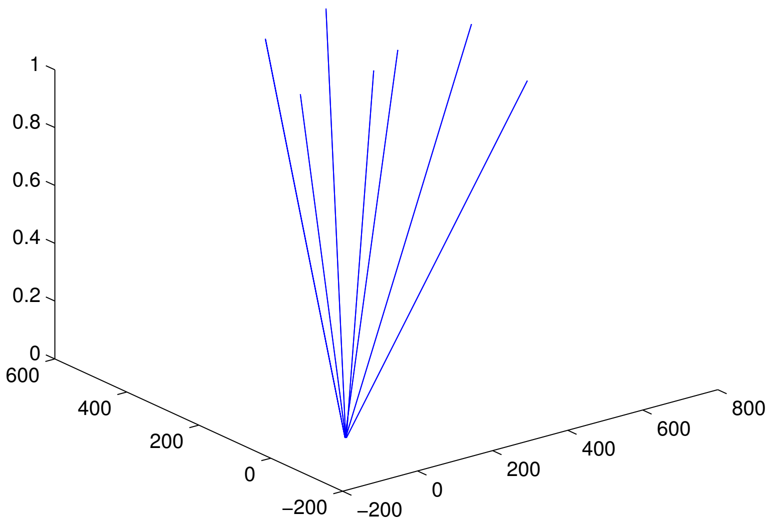
```
pixel_coordinates =

   122.7884   332.7728   265.6745   665.5544   447.8003   553.0149   478.8140   265.6745
  253.8615   269.1368   500.2867   344.1983   321.9445    71.8462   553.7085    500.2867
    1.0000     1.0000     1.0000     1.0000     1.0000     1.0000     1.0000    1.0000
```

plotting the points with line and plot3

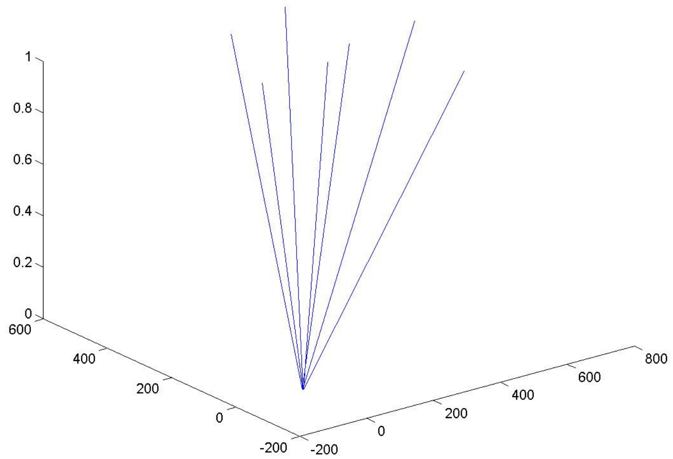
The points in this plot looks like merging at one point this is because all the cube points are very close in world coordinate and they are at different position in the image (the points in the image are not very close).

```
figure;
for i = 1:8
    x = [cc(i,1) pixel_coordinates(1,i)];
    y = [cc(i,2) pixel_coordinates(2,i)];
    z = [0 pixel_coordinates(3,i)];
    plot3(x,y,z)
    line(x,y,z);
    hold on
end
```



plot with plot3

The points are cube are two close as compared to there pixel coordinates that's they look converg- ing. We look our result by zooming the image to.



part d

taking point X at infinity by taking high values of X ~ 10^20 and y=1 z =1

```
pixel_coordinates_infinity_x = C * WM * [10^10 1 1 1]';
pixel_coordinates_infinity_x(1,1) = pixel_coordinates_infinity_x(1,1)/pixel_coordinates_infinity_x
pixel_coordinates_infinity_x(2,1) = pixel_coordinates_infinity_x(2,1)/pixel_coordinates_infinity_x
pixel_coordinates_infinity_x(3,1) = pixel_coordinates_infinity_x(3,1)/pixel_coordinates_infinity_x
pixel_coordinates_infinity_x
```

```
pixel_coordinates_infinity_x =

   1.0e+03 *

   -1.0611
    0.1677
    0.0010
```

taking point Y at infinity by taking high values of Y ~ 10^10 and x=1 z =1

```
pixel_coordinates_infinity_y = C * WM * [1 10^10 1 1]';
pixel_coordinates_infinity_y(1,1) = pixel_coordinates_infinity_y(1,1)/pixel_coordinates_infinity_y
pixel_coordinates_infinity_y(2,1) = pixel_coordinates_infinity_y(2,1)/pixel_coordinates_infinity_y
pixel_coordinates_infinity_y(3,1) = pixel_coordinates_infinity_y(3,1)/pixel_coordinates_infinity_y
pixel_coordinates_infinity_y
```

```
pixel_coordinates_infinity_y =

   1.0e+03 *

    1.8817
    3.2873
    0.0010
```

taking point Z at infinity by taking high values of z ~ 10^10 and x=1 z =1

```
pixel_coordinates_infinity_z = C * WM * [1 1 10^10 1]';
pixel_coordinates_infinity_z(1,1) = pixel_coordinates_infinity_z(1,1)/pixel_coordinates_infinity_z
pixel_coordinates_infinity_z(2,1) = pixel_coordinates_infinity_z(2,1)/pixel_coordinates_infinity_z
pixel_coordinates_infinity_z(3,1) = pixel_coordinates_infinity_z(3,1)/pixel_coordinates_infinity_z
pixel_coordinates_infinity_z
```

```
pixel_coordinates_infinity_z =

   1.0e+03 *

    2.3157
   -1.5071
    0.0010
```

taking point at X, Y , Z at infinity by taking high values of x, y, z

```
pixel_coordinates_infinity_xyz = C * WM * [10^10 10^10 10^10 1]';
pixel_coordinates_infinity_xyz(1,1) = pixel_coordinates_infinity_xyz(1,1)/pixel_coordinates_infini
pixel_coordinates_infinity_xyz(2,1) = pixel_coordinates_infinity_xyz(2,1)/pixel_coordinates_infini
pixel_coordinates_infinity_xyz(3,1) = pixel_coordinates_infinity_xyz(3,1)/pixel_coordinates_infini
pixel_coordinates_infinity_xyz
```

```
pixel_coordinates_infinity_xyz =

   1.0e+04 *

    1.3030
    0.2402
    0.0001
```

Problem2

part a

Camera Matrix

$$C = \begin{bmatrix} fu_x & s & u_0 \\ 0 & fu_y & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

Externel matrix

$$P = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix}$$

prespective Projection Matrix:

$$P2 = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix}$$

A equation for converting any 3D homogeneous point (X,Y,Z,1) to image coordinates (wx,wy,w);

$$\begin{bmatrix} wx \\ wy \\ w \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

In the given situation the beam of light is a 1D line in the world coordinates. As our point move on the 1D line we can represented it in a form of line and we can say that in the world it will be having only one coordinate (X). As Y = Z = 0 We can rewrite our matrix as:

$$\begin{bmatrix} wx \\ wy \\ w \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} X \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Writing the equations with reduced parameters:

$$\begin{bmatrix} wx \\ wy \\ w \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \\ p_{31} & p_{32} \end{bmatrix} \begin{bmatrix} X \\ 1 \end{bmatrix}$$

So any new point (X,1) can be mapped to image coordinates using this new projective camera model as:

$$\begin{bmatrix} wx \\ wy \\ w \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \\ p_{31} & p_{32} \end{bmatrix} \begin{bmatrix} X \\ 1 \end{bmatrix}$$

part b

The degree of freedom of the given system is 5. (As we have 6 parameters in the projective camera model and we can scale one parameter by dividing all the remaining parameters with it) We need at least 3 points to solve this equation.

part c

```
% Forming the equation in n*n form in this case n = 6; and equation it
% forms is AX=0 so we have to find the null space of A which we can do in two ways
% 1). By SVD
% 2). By zero Eigen Value and with corresponding eigen vector
A1 = [500 1 0  0 -50000 -100;
    0  0 500  1 -125000 -250;
    100  1 0  0 -14000 -140;
    0  0 100  1 -34000 -340;
    200  1 0  0 -40000 -200;
    0  0 200  1 -90000 -450];
```

SVD of matrix

```
[U S V] = svd(A1)


U =

   -0.2927    0.8432   -0.2219   -0.1346    0.3248    0.1744
   -0.7318   -0.3496   -0.5081   -0.2331   -0.1473   -0.0900
   -0.0820    0.1706    0.2472   -0.2712   -0.7431    0.5267
   -0.1991   -0.0651    0.6071   -0.6689    0.2677   -0.2616
   -0.2342    0.3391    0.2077    0.3223   -0.4405   -0.6993
   -0.5269   -0.1356    0.4688    0.5501    0.2343    0.3558


S =

   1.0e+05 *
```

```
    1.7081         0         0         0         0         0
         0    0.0055         0         0         0         0
         0         0    0.0036         0         0         0
         0         0         0    0.0008         0         0
         0         0         0         0    0.0000         0
         0         0         0         0         0    0.0000


V =

   -0.0012    0.9247   -0.1237   -0.3600    0.0022   -0.0013
   -0.0000    0.0025    0.0006   -0.0010   -0.9242    0.3818
   -0.0029   -0.3805   -0.2756   -0.8827   -0.0016   -0.0033
   -0.0000   -0.0010    0.0016   -0.0042    0.3818    0.9242
    1.0000    0.0000    0.0023   -0.0040   -0.0000   -0.0000
    0.0034   -0.0100   -0.9533    0.3020    0.0002    0.0029
```

Eigen values and vector

```
[V1 D1] = eigs(A1)
```

```
V1 =

  Columns 1 through 4

  -0.2889 + 0.0000i  -0.0151 + 0.1897i  -0.0151 - 0.1897i  -0.1955 + 0.0000i
  -0.7303 + 0.0000i   0.8697 + 0.0000i   0.8697 + 0.0000i   0.5537 + 0.0000i
  -0.0826 + 0.0000i  -0.0967 - 0.0552i  -0.0967 + 0.0552i  -0.3499 + 0.0000i
  -0.2023 + 0.0000i  -0.0601 - 0.2340i  -0.0601 + 0.2340i  -0.6880 + 0.0000i
  -0.2342 + 0.0000i  -0.0017 + 0.0018i  -0.0017 - 0.0018i  -0.0022 + 0.0000i
  -0.5297 + 0.0000i   0.3231 - 0.1796i   0.3231 + 0.1796i   0.2437 + 0.0000i

  Columns 5 through 6

   0.0034 + 0.0000i   0.0014 + 0.0000i
  -0.9506 + 0.0000i  -0.3850 + 0.0000i
  -0.0048 + 0.0000i   0.0034 + 0.0000i
   0.3103 + 0.0000i  -0.9229 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i
  -0.0071 + 0.0000i  -0.0030 + 0.0000i


D1 =

   1.0e+04 *

  Columns 1 through 4

  -4.0203 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0092 - 0.0244i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0092 + 0.0244i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0065 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i

  Columns 5 through 6

   0.0000 + 0.0000i   0.0000 + 0.0000i
```

```
   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0004 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  -0.0000 + 0.0000i
```

```
m = V(:,end);
M = reshape(m,2,3)';
abs_lambda=sqrt(M(3,2)^2 + M(3,1)^2);
M = M / abs_lambda;
```

calibration parameter

```
M
```

```
M =

   -0.4625  130.9066
   -1.1371  316.8807
   -0.0040    1.0000
```

eigen vector corresponding to minimum eigen value

```
eigenV = V1(:,6)
eigenV  = reshape(eigenV ,2,3)';
abs_lambda=sqrt(eigenV(3,1)^2 + eigenV(3,2)^2);
eigenV  = eigenV  / abs_lambda;
```

```
eigenV =

    0.0014
   -0.3850
    0.0034
   -0.9229
    0.0000
   -0.0030
```

new heights

```
pM = pinv(M);
peigen = pinv(eigenV);
```

first Point

```
  p1_m = pM * [130; 310; 1];
  p1_m(1,1) = p1_m(1,1)/p1_m(2,1);
  p1_m(2,1) = p1_m(2,1)/p1_m(2,1);

  p1_m(1,1)
```

```
ans =  137.4421
```

Second Point

```
p1_m = pM * [170; 380; 1];
p1_m(1,1) = p1_m(1,1)/p1_m(2,1);
p1_m(2,1) = p1_m(2,1)/p1_m(2,1);

p1_m(1,1)
```

```
ans =  234.7793
```

3rd Point

```
p1_m = pM * [190; 300; 1];
p1_m(1,1) = p1_m(1,1)/p1_m(2,1);
p1_m(2,1) = p1_m(2,1)/p1_m(2,1);

p1_m(1,1)
```

```
ans =  270.7898
```

Problem3

part (a) I have tested my results on my two set of images (sofa) and (HSC) the results are in the particular folder Besides that I have also created a full mosaic on humanity image.

part b

the points in the left image

```
ptsa1 = [249,336,1];
ptsa2 = [312,140,1];
ptsa3 = [292,55,1];
ptsa4 = [332,188,1];
ptsa5 = [405,317,1];
ptsa6 = [423,111,1];
ptsa7 = [332,238,1];
ptsa8 = [528,139,1];
ptsa9 = [425,294,1];
ptsa10 = [322,281,1];

X = [ptsa1; ptsa2;ptsa3;ptsa4;ptsa5;ptsa6];
```

points in right image

```
ptsb1 = [101,340,1];
ptsb2 = [162,134,1];
ptsb3 = [121,38,1];
ptsb4 = [181,171,1];
ptsb5 = [264,292,1];
ptsb6 = [265,89,1];
ptsb7 = [173,226,1];
ptsb8 = [360,111,1];
ptsb9 = [285,264,1];
ptsb10 = [181,275,1];


Y = [ptsb1; ptsb2;ptsb3;ptsb4;ptsb5;ptsb6];
```

parta

the linear equation is in the form of AX = Y

$$A = \begin{bmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ 0 & 0 & 1 \end{bmatrix}$$

```
 So the equation for 1st point is:
 249 * a1 + 336 * a2 + a3 = 162
 249 * a4 + 336 * a5 + a6 = 134

We can have these equations for other points to for other equation too.
```

function ComputeWarpMapping

```
solution with least square regression is:
   X = A\B
```

function WarpImage

I have implemented both forward and backward mapping. The results are in codes folder and forward mapping is commented.
In backward mapping, I have also implemented **first order interpolation**.

```
 %%Backward Mapping with first order interpolation
for row = ceil(min_y):ceil(max_y)
   for col = ceil(min_x):ceil(max_x)
pts = [col row 1];
transform_point = pts * A;
%% First Order Interpolation
transform_point(1) = transform_point(1)/transform_point(3);
transform_point(2) = transform_point(2)/transform_point(3);
delta_x = transform_point(1) - floor(transform_point(1));
delta_y = transform_point(1) - floor(transform_point(1));

if(ceil(transform_point(1)) > 1 && ceil(transform_point(1)) < n && ceil(transform_point(2)) > 1 &&
  for channel = 1:3
    f00 = img1(floor(transform_point(2)), floor(transform_point(1)), channel);
    f10 = img1(ceil(transform_point(2)), floor(transform_point(1)), channel);
    f01 = img1(floor(transform_point(2)), ceil(transform_point(1)), channel);
    f11 = img1(ceil(transform_point(2)), ceil(transform_point(1)), channel);
    val = f00 + (f10-f00) * delta_x + (f01-f00) * delta_y + [f11-f10-f01+f00]*delta_x *delta_y;
    img1warp(ceil(row-min_y), ceil(col-min_x), channel) = val;
  end
end

   end
end
```

Result of warping Humanity01.JPG

Merge and Results

After backward mapping I have also merged the other image in the same image. The results are shown below.

Results of merging Humanity01.JPG and Humanity02.JPG

Results of mosaicing Humanity01.JPG and Humanity02.JPG and Humanity03.JPG

Results of full mosaic

HSC mosaic

Results of sofa mosaic

In the sofa mosaic results are not upto mark because camera was rotated.
Other results are in results section.

Problem4

I have completed this problem like this:

1). Got the matched features by hough transformation (as compared to kd-tree in the paper).

2). I randomly selected 10 matched features and calculated homography with it and measured the error on non selected points. I repeat this process for 100 iterations.

```
%%RANSAC randomly selecting 6 points upto 100 iteration
min_points = 0;
min_err = 0;
selected = [];
condition = false;
for i = 1:100
 left = [];
 right = [];
 points = [];
 count = 0;
 while(1)

  j = floor(10 * rand + 1);
  if(length(points) == 0)
    points = [points;j];
    count = count+1;
  else
   for k = 1:length(points)
      if(points(k) == j)
 condition = true;
       end
    end
     if(condition == false)
      points = [points;j];
      count = count + 1;
   end
```

```
  end

   if (count == 6)
    break;
   end
   condition = false;

end


 for k = 1 : length(points)
  left = [left; img1pts(points(k),:)];
  right = [right; img2pts(points(k),:)];
 end

 H = ComputeWarpMapping(left, right);
total_e = 0;
 for k = 1 : length(img1pts)
   trans = img1pts(k,:) * H;
   orig = img2pts(k,:);
   total_e = total_e + sum((trans - orig).^2);
 end
 total_e
if (min_err == 0)
  min_err = total_e;
  min_points = points;
elseif total_e < min_err
   min_err = total_e;
  min_points = points;
end

end
```

3). Finally took the features which gives the minimum error and make the mosaic with that homography.

HSC mosaic