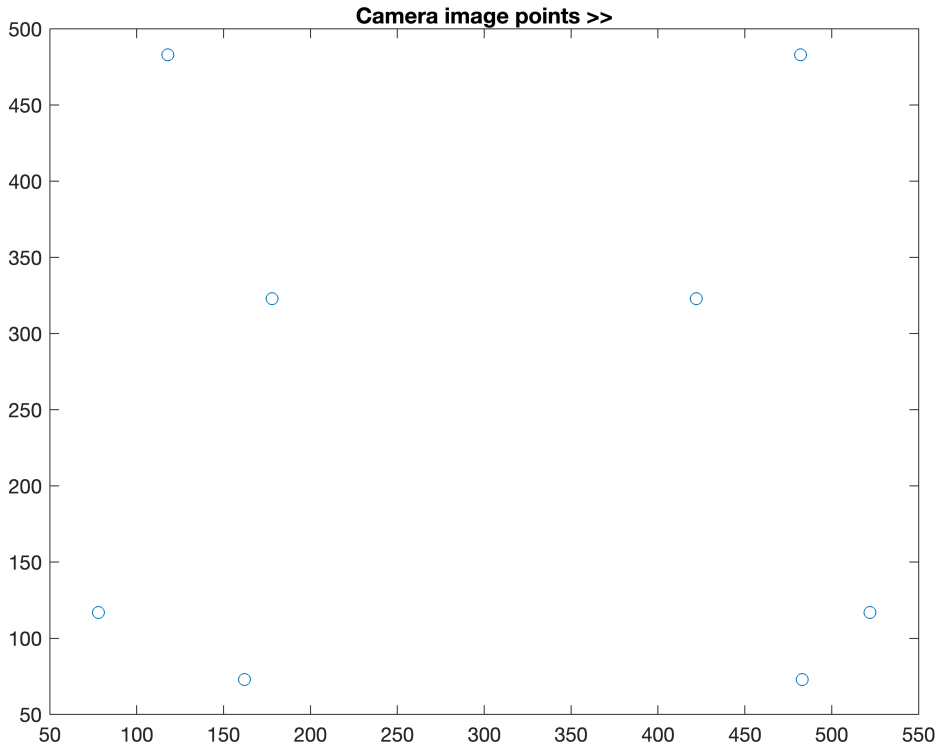
PART I: Camera Calibration using 3D calibration object

Q1. Draw the image points, using small circles for each image point

```
world_coordinates = [2 2 2; -2 2 2;-2 2 -2;2 2 -2;2 -2 2; -2 -2 2; -2 -2 -2; 2 -2 -2];
image_coordinates = [422 323; 178 323; 118 483; 482 483;483 73;162 73; 78 117;522 117]

plot(image_coordinates(:,1),image_coordinates(:,2),'o');
title("Camera image points >>")
```




```
p_matrix = [];
```

Q2 is at the bottom

Q3. Use this function to generate 2 rows of the matrix P for each cube corner and its image and obtain a matrix with 16 rows and 12 columns. Print matrix P.

```
for i = 1:8
    p_cap = transpose([world_coordinates(i,:) 1]);
    p_small = transpose(image_coordinates(i,:));
    rows = Prows(p_small, p_cap);
    p_matrix = [p_matrix; rows];
end
disp("P")
```

```
P
```

```
disp(p_matrix)
```

```
Columns 1 through 8

     2     2     2     1     0     0     0     0
     0     0     0     0     2     2     2     1
    -2     2     2     1     0     0     0     0
     0     0     0     0    -2     2     2     1
    -2     2    -2     1     0     0     0     0
     0     0     0     0    -2     2    -2     1
     2     2    -2     1     0     0     0     0
     0     0     0     0     2     2    -2     1
     2    -2     2     1     0     0     0     0
     0     0     0     0     2    -2     2     1
    -2    -2     2     1     0     0     0     0
     0     0     0     0    -2    -2     2     1
    -2    -2    -2     1     0     0     0     0
     0     0     0     0    -2    -2    -2     1
     2    -2    -2     1     0     0     0     0
     0     0     0     0     2    -2    -2     1

Columns 9 through 12

  -844  -844  -844  -422
  -646  -646  -646  -323
   356  -356  -356  -178
   646  -646  -646  -323
   236  -236   236  -118
   966  -966   966  -483
  -964  -964   964  -482
  -966  -966   966  -483
  -966   966  -966  -483
  -146   146  -146   -73
   324   324  -324  -162
   146   146  -146   -73
   156   156   156   -78
   234   234   234  -117
 -1044  1044  1044  -522
  -234   234   234  -117
```

Q4. Now we need to solve the system Pm =0. Find the singular value decomposition of matrix P using matlab svd function. The last column vector of V obtained by svd(P) should be the 12 elements in row order of the projection matrix that transformed the cube corner coordinates into their images. Print the matrix M

```
[U,sigma,V_transpose] = svd(p_matrix);

M(1,1:4) = V_transpose(1:4,end)';
M(2,1:4) = V_transpose(5:8,end)';
M(3,1:4) = V_transpose(9:12,end)';

disp("Matrix M");
```

```
Matrix M
```

```
disp(M);
```

```
    0.1959    0.0293    0.0771    0.7337
   -0.0002    0.2082   -0.0068    0.6109
   -0.0000    0.0001    0.0002    0.0024
```

Q5. Now we need to recover the translation vector which is a null vector of M. Find the singular value decomposition of matrix M=U ^V T . The 4 elements of the last column of V are the homogeneous coordinates of the position of the camera center of projection in the frame of reference of the cube (as in slide 36). Print the corresponding 3 Euclidean coordinates of the camera center in the frame of reference of the cube.

```
[u, s, v] = svd(M);
camera_center = v(:,end);
camera_center = camera_center/camera_center(end);
disp("Translation vector/camera center");
```

```
Translation vector/camera center
```

```
disp(camera_center(1:3));
```

```
    0.1293
   -3.2170
   -8.6275
```

Q6. Consider the 3x3 matrix M' composed of the first 3 columns of matrix M. Rescale the elements of this matrix so that its element m33 becomes equal to 1. Print matrix M' . Now let the rotation matrices be as defined in slide 38 where the axes e1, e2, e3 are the x, y, z axes respectively

```
m_dash = M(:,1:3);
m_dash = m_dash/m_dash(3,3);
disp("M");
```

```
M
```

```
disp(m_dash);
```

```
  843.7086  126.0766  331.8648
   -0.9935  896.5559  -29.3752
   -0.0040    0.5256    1.0000
```

Q7. We will perform the RQ factorization of M' in several steps. Note that the term at position (3, 2) would also be set to zero if the signs of cos(✓x)and sin(✓x)were reversed, but this would lead to finding a negative focal length for the camera. So we should choose the signs that leads to a positive focal length. Compute the angle ✓x of this rotation in degrees. Compute matrix N=M' ⇤Rx. Print Rx, ✓x and N.

```
cos=m_dash(3,3)/sqrt((m_dash(3,3)^2)+m_dash(3,2)^2);
sin=-m_dash(3,2)/sqrt((m_dash(3,3)^2)+m_dash(3,2)^2);
Rx=[1 0 0;0 cos -sin; 0 sin cos];
disp("Rx")
```

```
Rx
```

```
disp(Rx)
```

```
    1.0000         0         0
         0    0.8852    0.4652
         0   -0.4652    0.8852
```

```
theta=rad2deg(atan(sin/cos));
fprintf("ThetaX: %f \n", theta);
```

```
ThetaX: -27.724678
```

```
N=m_dash*Rx;
disp("N");
```

```
N
```

```
disp(N);
```

```
  843.7086  -42.7891  352.4183
   -0.9935  807.2913  391.0960
   -0.0040         0    1.1297
```

Q8. The element n31 of Nis small enough so that there is no need for a rotation Ry. However, element n21 is large and a rotation matrix Rz is needed to set it to zero. Compute the rotation matrix Rz using cosine and sine. Compute the rotation angle ✓z in degrees.

```
cosz=m_dash(2,2)/sqrt((m_dash(2,2)^2)+m_dash(2,1)^2);
sinz=-m_dash(2,1)/sqrt((m_dash(2,2)^2)+m_dash(2,1)^2);
theta_z=rad2deg(atan(sinz/cosz));
fprintf("theta_z: %f \n", theta_z);
```

```
theta_z: 0.063488
```

```
Rz=[cosz -sinz 0;sinz cosz 0; 0 0 1];
```

Q9. Since we factorized out Rz we can directly compute the calibration matrix K, how? Compute K and rescale so that its element K33 is set to 1. Print K. What are the focal lengths of the camera in pixels? What are the pixel coordinates of the image center of the camera?

```
RxRz=transpose(Rx)*transpose(Rz);
K=m_dash*inv(RxRz);
K=K/K(3,3);
disp("K");
```

```
K
```

```
disp(K);
```

```
  746.9672  -38.6092  311.5728
         0  714.6081  346.1952
   -0.0030    0.0000    1.0000
```

```
disp("----------------------------Intrinsic Parameters------------------------- \n")
```

```
----------------------------Intrinsic Parameters------------------------- \n
```

```
fprintf("Focal lengths--------------- \n");
```

```
Focal lengths---------------
```

```
fprintf("alpha: %f, beta: %f, gamma: %f \n", K(1,1),K(2,1),K(1,2));
```

```
alpha: 746.967211, beta: 0.000000, gamma: -38.609164
```

```
fprintf("Image centers--------------- \n");
```

```
Image centers---------------
```

```
fprintf("u0: %f, v0: %f \n", K(1,3), K(2,3))
```

```
u0: 311.572797, v0: 346.195211
```

Q2. Write a function that takes as argument the homogeneous coordinates of one cube corner and the homogeneous coordinates of its image, and returns 2 rows of the matrix

```
function y = Prows(uv,xyz1)
xyz1Transpose = transpose(xyz1);
uXxyz1Transpose = -uv(1)*xyz1Transpose;
vXxyz1Transpose = -uv(2)*xyz1Transpose;
Zeros = [0 0 0 0];
y = [xyz1Transpose Zeros uXxyz1Transpose;Zeros xyz1Transpose vXxyz1Transpose];
end
```

PART II: Camera Calibration using 2D calibration object

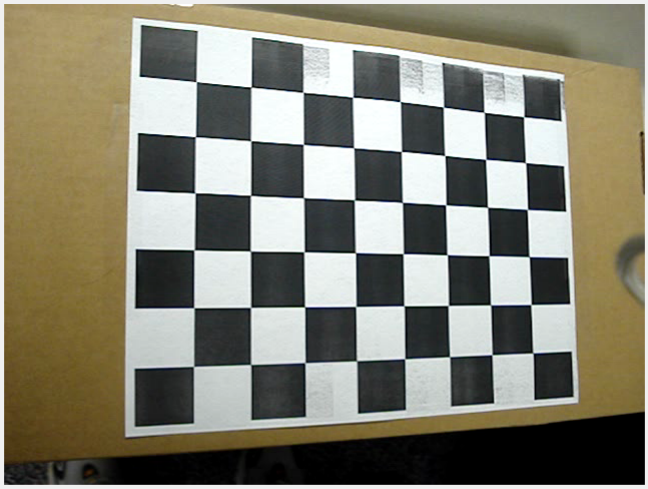
(a). Corner Extraction and Homography computation

```
width = 9;
height = 7;
squareSize = 30;

world_coordinates = [0,0,1; 0,(height*squareSize),1; (width*squareSize),(height*squareS

world_coordinates=world_coordinates';
disp(world_coordinates);
```

```
     0     0   270   270
     0   210   210     0
     1     1     1     1
```

```
img1 = imread("images2.png");
img2 = imread("images9.png");
img3 = imread("images12.png");
img4 = imread("images20.png");

imshow(img1), [x1,y1] = ginput(4);
```

```
imshow(img2), [x2,y2] = ginput(4);
```



```
imshow(img3), [x3,y3] = ginput(4);
```

```
imshow(img4), [x4,y4] = ginput(4);
```

```
disp([x1,y1]);
```

```
   60.0000  418.0000
   86.0000   68.0000
  528.0000   74.0000
  544.0000  422.0000
```

```
disp([x2,y2]);
```

```
  122.0000  424.0000
  128.0000   16.0000
  558.0000   72.0000
  574.0000  390.0000
```

```
disp([x3,y3]);
```

```
   98.0000  396.0000
  110.0000   88.0000
  516.0000   18.0000
  536.0000  414.0000
```

```
disp([x4,y4]);
```

```
  120.0000  278.0000
  176.0000   84.0000
  522.0000   76.0000
  590.0000  276.0000
```

```
ones = [1,1,1,1];

image_coordinates_1 = [x1(:), y1(:), ones(:)]';
image_coordinates_2 = [x2(:), y2(:), ones(:)]';
image_coordinates_3 = [x3(:), y3(:), ones(:)]';
image_coordinates_4 = [x4(:), y4(:), ones(:)]';

H1 = homography2d(world_coordinates, image_coordinates_1);
```

```
H2 = homography2d(world_coordinates, image_coordinates_2);
H3 = homography2d(world_coordinates, image_coordinates_3);
H4 = homography2d(world_coordinates, image_coordinates_4);

disp("Homography of --> images2.png")
```

```
Homography of --> images2.png
```

```
disp(H1)
```

```
    0.9953    0.0897   33.0388
    0.0145   -0.9008  230.1702
    0.0000    0.0002    0.5506
```

```
disp("Homography of --> images9.png")
```

```
Homography of --> images9.png
```

```
disp(H2)
```

```
   -1.1231   -0.0324  -59.5977
   -0.1459    0.9468 -207.1266
   -0.0005   -0.0001   -0.4885
```

```
disp("Homography of --> images12.png")
```

```
Homography of --> images12.png
```

```
disp(H3)
```

```
    0.7260    0.0580   61.4728
   -0.1834   -0.9023  248.4005
   -0.0005    0.0002    0.6273
```

```
disp("Homography of --> images20.png")
```

```
Homography of --> images20.png
```

```
disp(H4)
```

```
   -0.8560   -0.2914  -62.4569
    0.0273    0.4080 -144.6918
    0.0001   -0.0009   -0.5205
```

(b). Computing the Intrinsic and Extrinsic parameters

```
V = [];
for i = 1:4
    H = eval(['H' num2str(i)]);

    h1 = H(:,1);
    h2 = H(:,2);
    h3 = H(:,3);

    v11 = [h1(1)*h1(1), h1(1)*h1(2)+h1(2)*h1(1), h1(2)*h1(2), h1(3)*h1(1)+h1(1)*h1(3),
    v12 = [h1(1)*h2(1), h1(1)*h2(2)+h1(2)*h2(1), h1(2)*h2(2), h1(3)*h2(1)+h1(1)*h2(3),
    v22 = [h2(1)*h2(1), h2(1)*h2(2)+h2(2)*h2(1), h2(2)*h2(2), h2(3)*h2(1)+h2(1)*h2(3),

    V = [V; v12'; (v11-v22)'];
end

[U, Sigma, V_transpose] = svd(V);

b = V_transpose(:,end);

B11 = b(1);
B12 = b(2);
B22 = b(3);
B13 = b(4);
B23 = b(5);
B33 = b(6);

B = [B11, B12, B13; B12, B22, B23; B13, B23, B33];

disp("Matrix B >>");
```

```
Matrix B >>
```

```
disp(B);
```

```
   -0.0000   -0.0000    0.0005
   -0.0000   -0.0000    0.0004
    0.0005    0.0004   -1.0000
```

```
v0 = (B12*B13 - B11*B23)/(B11*B22 - B12^2);
lambda = B33 - (B13^2 + v0*(B12*B13-B11*B23))/B11;
alpha = sqrt(lambda/B11);
beta = sqrt(lambda*B11/(B11*B22-B12^2));
gamma = -B12*alpha^2*beta/lambda;
u0 = gamma*v0/alpha - B13*alpha^2/lambda;

A = [alpha, gamma, u0; 0, beta, v0; 0, 0, 1];

disp("Intrinsic Parameters matrix >>");
```

Intrinsic Parameters matrix >>

```
disp(A);
```

```
  686.5929   -2.2802  341.2583
         0  671.0162  238.1869
         0         0    1.0000
```

```
files = ["images2", "images9", "images12", "images20"];

for i = 1:4
    H = eval(['H' num2str(i)]);

    h1 = H(:,1);
    h2 = H(:,2);
    h3 = H(:,3);

    lambda_r = 1/ norm(A\h1);
    r1 = lambda_r*(A\h1);
    r2 = lambda_r*(A\h2);
    r3 = cross(r1,r2);
    t  = lambda_r*(A\h3);

    R = [r1, r2, r3];

    disp(["Rotation Matrix for >> " files(i)]);
    disp(R);
    disp(["Translation vector for >> " files(i)]);
    disp(t);

    disp(["Transpose(R)*R for >> " files(i)]);
    disp(R'*R);

    [U, Sigma, V_transpose] = svd(R);
    R_new = U*V_transpose;

    disp(["New rotation matrix for >> " files(i)]);
    disp(R_new);
    disp(["Transpose(R_new)*R_new for >> " files(i)]);
    disp(R_new'*R_new);

end
```

```
    "Rotation Matrix for >> "    "images2"
   -0.9998   -0.0047   -0.0124
   -0.0128    0.9926   -0.1630
    0.0146   -0.1630   -0.9925
    "Translation vector for >> "    "images2"
  157.6667
 -102.6320
 -382.9962
    "Transpose(R)*R for >> "    "images2"
```

```
   1.0000   -0.0104         0
  -0.0104    1.0119         0
        0         0    1.0118
   "New rotation matrix for >> "    "images2"
  -0.2586   -0.8696    0.4207
  -0.5706    0.4889    0.6598
  -0.7794   -0.0694   -0.6226
   "Transpose(R_new)*R_new for >> "    "images2"
   1.0000    0.0000   -0.0000
   0.0000    1.0000   -0.0000
  -0.0000   -0.0000    1.0000
   "Rotation Matrix for >> "    "images9"
   0.9313   -0.0143    0.3636
   0.0179   -0.9948   -0.0937
   0.3638    0.0951   -0.9262
   "Translation vector for >> "    "images9"
-104.9055
  93.1996
 329.4738
   "Transpose(R)*R for >> "    "images9"
   1.0000    0.0035   -0.0000
   0.0035    0.9988   -0.0000
  -0.0000   -0.0000    0.9988
   "New rotation matrix for >> "    "images9"
   0.7700    0.6136    0.1749
  -0.5399    0.7727   -0.3339
  -0.3400    0.1627    0.9262
   "Transpose(R_new)*R_new for >> "    "images9"
   1.0000    0.0000   -0.0000
   0.0000    1.0000   -0.0000
  -0.0000   -0.0000    1.0000
   "Rotation Matrix for >> "    "images12"
  -0.9311    0.0198   -0.3625
   0.0546    0.9898   -0.0875
   0.3607   -0.1016   -0.9227
   "Translation vector for >> "    "images12"
 154.0761
-103.3543
-439.4594
   "Transpose(R)*R for >> "    "images12"
   1.0000   -0.0011         0
  -0.0011    0.9905         0
        0         0    0.9905
   "New rotation matrix for >> "    "images12"
  -0.9620   -0.0832   -0.2600
  -0.0640    0.9946   -0.0814
   0.2653   -0.0616   -0.9622
   "Transpose(R_new)*R_new for >> "    "images12"
   1.0000    0.0000   -0.0000
   0.0000    1.0000   -0.0000
  -0.0000   -0.0000    1.0000
   "Rotation Matrix for >> "    "images20"
   0.9991   -0.0245    0.0174
  -0.0152   -0.7213   -0.6885
   0.0386    0.6882   -0.7211
   "Translation vector for >> "    "images20"
-125.1451
  24.2143
 388.5909
   "Transpose(R)*R for >> "    "images20"
   1.0000    0.0131    0.0000
   0.0131    0.9945    0.0000
   0.0000    0.0000    0.9943
   "New rotation matrix for >> "    "images20"
```

```
 0.5988    0.6535    0.4630
-0.7949    0.5553    0.2444
-0.0973   -0.5144    0.8520
 "Transpose(R_new)*R_new for >> "    "images20"
 1.0000    0.0000    0.0000
 0.0000    1.0000    0.0000
 0.0000    0.0000    1.0000
```

(c). Improving accuracy

```
grid_coordinates = [];
p_correct = [];
arr = [];

for i = 0:width
    for j = 0:height
        grid_coordinates = [grid_coordinates; i*30, j*30, 1];
    end
end

for i = 1:80
    arr(i) = 1
end
```

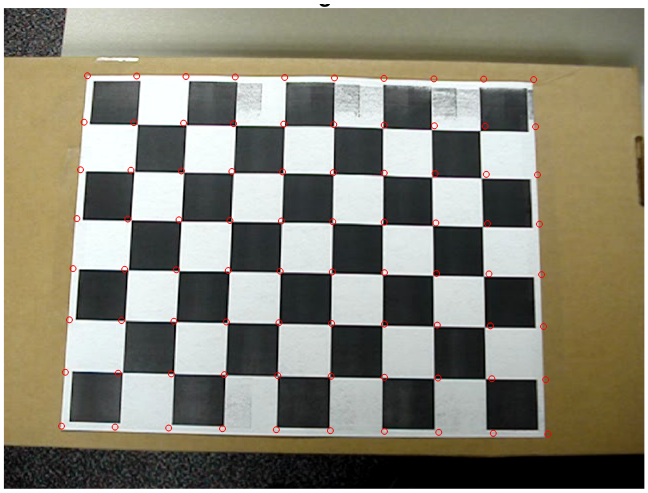
```
arr = 1
arr = 1×2
     1     1
arr = 1×3
     1     1     1
arr = 1×4
     1     1     1     1
arr = 1×5
     1     1     1     1     1
arr = 1×6
     1     1     1     1     1     1
arr = 1×7
     1     1     1     1     1     1     1
arr = 1×8
     1     1     1     1     1     1     1     1
arr = 1×9
     1     1     1     1     1     1     1     1     1
arr = 1×10
     1     1     1     1     1     1     1     1     1     1
arr = 1×11
     1     1     1     1     1     1     1     1     1     1     1
arr = 1×12
     1     1     1     1     1     1     1     1     1     1     1     1
arr = 1×13
     1     1     1     1     1     1     1     1     1     1     1     1     1
arr = 1×14
     1     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×15
     1     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×16
     1     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×17
     1     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×18
     1     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×19
     1     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×20
     1     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×21
     1     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×22
     1     1     1     1     1     1     1     1     1     1     1     1     1 ···
```

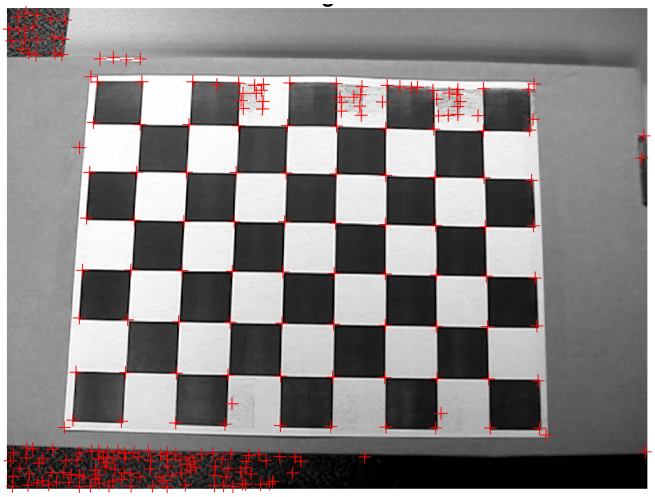
```
arr = 1×23
     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×24
     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×25
     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×26
     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×27
     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×28
     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×29
     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×30
     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×31
     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×32
     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×33
     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×34
     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×35
     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×36
     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×37
     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×38
     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×39
     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×40
     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×41
     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×42
     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×43
     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×44
     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×45
     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×46
     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×47
     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×48
     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×49
     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×50
     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×51
     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×52
     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×53
     1     1     1     1     1     1     1     1     1     1     1     1 ···
arr = 1×54
```

```
     1     1     1     1     1     1     1     1     1     1     1     1     1···
arr = 1×55
     1     1     1     1     1     1     1     1     1     1     1     1     1···
arr = 1×56
     1     1     1     1     1     1     1     1     1     1     1     1     1···
arr = 1×57
     1     1     1     1     1     1     1     1     1     1     1     1     1···
arr = 1×58
     1     1     1     1     1     1     1     1     1     1     1     1     1···
arr = 1×59
     1     1     1     1     1     1     1     1     1     1     1     1     1···
arr = 1×60
     1     1     1     1     1     1     1     1     1     1     1     1     1···
arr = 1×61
     1     1     1     1     1     1     1     1     1     1     1     1     1···
arr = 1×62
     1     1     1     1     1     1     1     1     1     1     1     1     1···
arr = 1×63
     1     1     1     1     1     1     1     1     1     1     1     1     1···
arr = 1×64
     1     1     1     1     1     1     1     1     1     1     1     1     1···
arr = 1×65
     1     1     1     1     1     1     1     1     1     1     1     1     1···
arr = 1×66
     1     1     1     1     1     1     1     1     1     1     1     1     1···
arr = 1×67
     1     1     1     1     1     1     1     1     1     1     1     1     1···
arr = 1×68
     1     1     1     1     1     1     1     1     1     1     1     1     1···
arr = 1×69
     1     1     1     1     1     1     1     1     1     1     1     1     1···
arr = 1×70
     1     1     1     1     1     1     1     1     1     1     1     1     1···
arr = 1×71
     1     1     1     1     1     1     1     1     1     1     1     1     1···
arr = 1×72
     1     1     1     1     1     1     1     1     1     1     1     1     1···
arr = 1×73
     1     1     1     1     1     1     1     1     1     1     1     1     1···
arr = 1×74
     1     1     1     1     1     1     1     1     1     1     1     1     1···
arr = 1×75
     1     1     1     1     1     1     1     1     1     1     1     1     1···
arr = 1×76
     1     1     1     1     1     1     1     1     1     1     1     1     1···
arr = 1×77
     1     1     1     1     1     1     1     1     1     1     1     1     1···
arr = 1×78
     1     1     1     1     1     1     1     1     1     1     1     1     1···
arr = 1×79
     1     1     1     1     1     1     1     1     1     1     1     1     1···
arr = 1×80
     1     1     1     1     1     1     1     1     1     1     1     1     1···
```

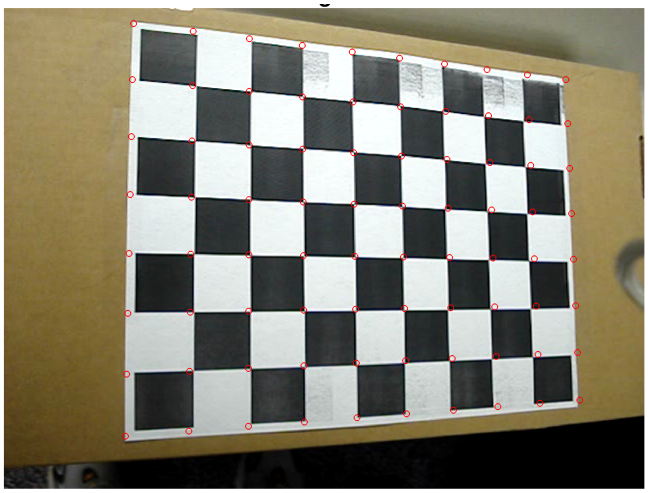
```
for i = 1:4

    H = eval(['H' num2str(i)]);
    p_approx = H*grid_coordinates';
    for j = 1:length(p_approx)
        p_approx(:,j) = p_approx(:,j) / p_approx(3,j);
    end
```

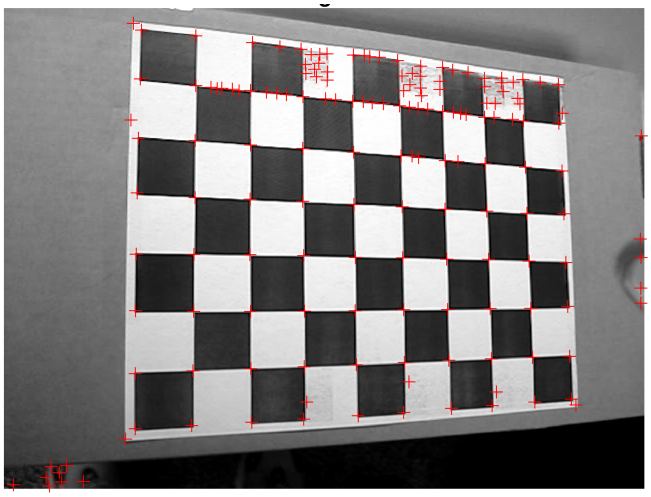
```
    img = eval(['img' num2str(i)]);
    figure(), imshow(img)
    hold on
    title(['Figure 1 : Projected grid corners for >> ' files(i)])
    plot(p_approx(1,:),p_approx(2,:),'ro', 'MarkerSize',3);
    hold off

    sigma = 2;
    thresh = 500;
    radius = 2;
    [cim,r,c,rsubp,csubp]=harris(rgb2gray(img),sigma,thresh,radius,1);
    title(["Figure 2 : Harris corners for >> " files(1)])

    D = dist2(p_approx(1:2,:)',[csubp, rsubp]);
    [D_sorted, D_index] = sort(D, 2);
    p_correct(:,:,i) = cat(2, csubp(D_index(:,1)), rsubp(D_index(:,1)), transpose(arr)
    %p_correct(:,:,i) = [csubp(D_index(:,1)), rsubp(D_index(:,1)), transpose(arr)];
    figure(), imshow(img), title(["Figure3 : Grid Points for >> " files(i)])
    hold on
    plot(p_correct(:,1,i),p_correct(:,2,i),'g+')
    hold off

    H_new(:,:,i) = homography2d(grid_coordinates',p_correct(:,:,i)');
    H_new(:,:,i) = H_new(:,:,i)/H_new(3,3,i);
    disp(["New Homography H for >> " files(i)])
    disp(H_new(:,:,i))
end
```

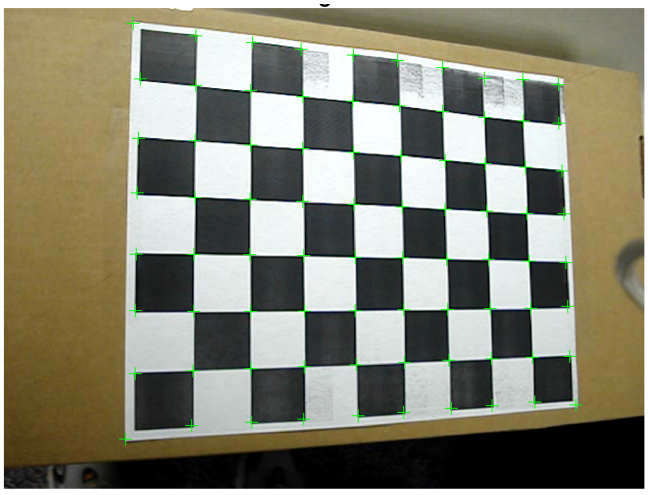




```
 "New Homography H for >> "    "images2"
 1.7433    0.1589   62.8090
 0.0244   -1.6122  415.5829
-0.0000    0.0004    1.0000
```

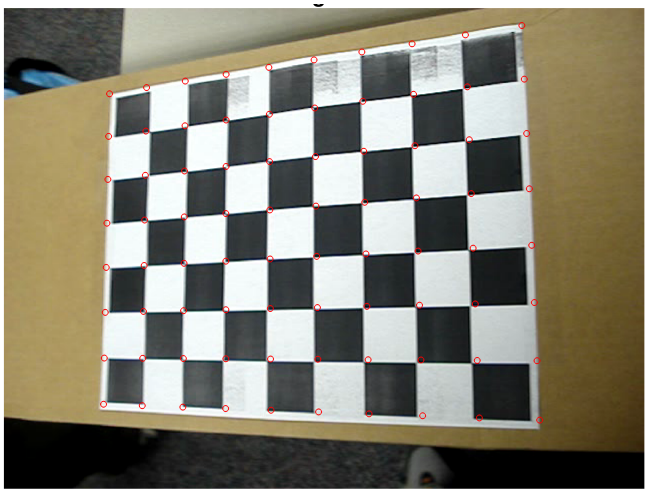


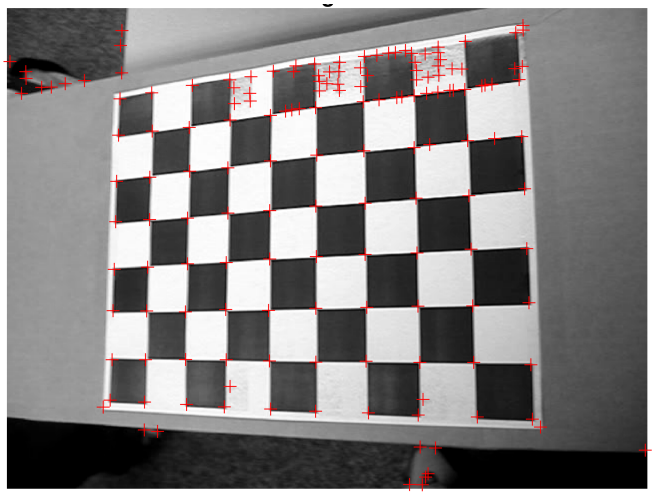


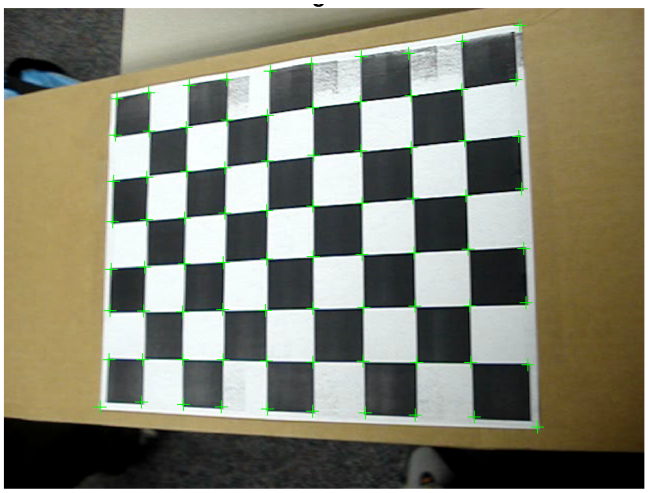


"New Homography H for >> "   "images9"

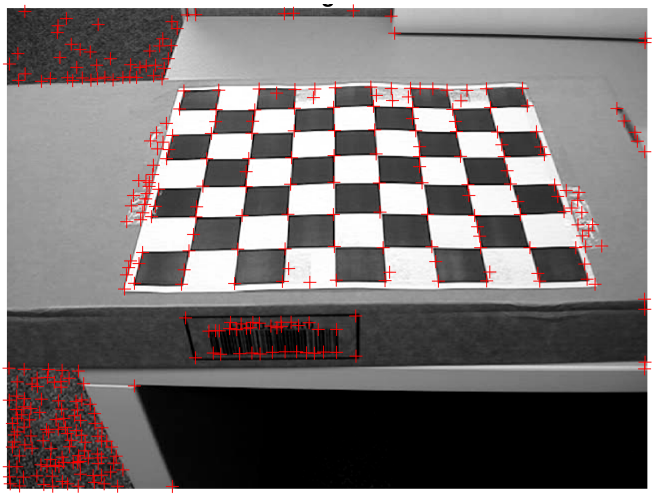
```
2.2713    0.0789  126.9778
0.3139   -1.9443  426.3384
0.0011    0.0003    1.0000
```







```
 "New Homography H for >> "    "images12"
  1.1300    0.0884  100.4824
 -0.2866   -1.4312  395.3177
 -0.0009    0.0003    1.0000
```



```
 "New Homography H for >> "    "images20"
  1.7279    0.5775  121.5967
 -0.0127   -0.7906  278.3214
  0.0000    0.0018    1.0000
```

```
Hnew1 = H_new(:,:,1);
Hnew2 = H_new(:,:,2);
Hnew3 = H_new(:,:,3);
Hnew4 = H_new(:,:,4);
```

```
V = [];
for i = 1:4
    H_temp = eval(['Hnew' num2str(i)]);
    h1 = H_temp(:,1);
    h2 = H_temp(:,2);
    h3 = H_temp(:,3);

    v11 = [h1(1)*h1(1), h1(1)*h1(2)+h1(2)*h1(1), h1(2)*h1(2), h1(3)*h1(1)+h1(1)*h1(3),
    v12 = [h1(1)*h2(1), h1(1)*h2(2)+h1(2)*h2(1), h1(2)*h2(2), h1(3)*h2(1)+h1(1)*h2(3),
    v22 = [h2(1)*h2(1), h2(1)*h2(2)+h2(2)*h2(1), h2(2)*h2(2), h2(3)*h2(1)+h2(1)*h2(3),

    V = [V; v12'; (v11-v22)'];
end

[U, Sigma, V_transpose] = svd(V);

b = V_transpose(:,end);

B11 = b(1);
B12 = b(2);
B22 = b(3);
B13 = b(4);
B23 = b(5);
B33 = b(6);

B = [B11, B12, B13; B12, B22, B23; B13, B23, B33];

v0 = (B12*B13 - B11*B23)/(B11*B22 - B12^2);
lambda = B33 - (B13^2 + v0*(B12*B13-B11*B23))/B11;
alpha = sqrt(lambda/B11);
beta = sqrt(lambda*B11/(B11*B22-B12^2));
gamma = -B12*alpha^2*beta/lambda;
u0 = gamma*v0/alpha - B13*alpha^2/lambda;

A = [alpha, gamma, u0; 0, beta, v0; 0, 0, 1];

for i = 1:4
    H_temp = eval(['Hnew' num2str(i)]);
    h1 = H_temp(:,1);
    h2 = H_temp(:,2);
    h3 = H_temp(:,3);

    lambda_r = 1/ norm(A\h1);
    r1 = lambda_r*(A\h1);
    r2 = lambda_r*(A\h2);
    r3 = cross(r1,r2);
    t(:,i)  = lambda_r*(A\h3);

    R = [r1, r2, r3];
```

```
    [U,S,Vprime] = svd(R);
    Rotation(:,:,i) = U*Vprime;

    disp(["Rotation matrix R for images" files(i)])
    disp(Rotation(:,:,i))
    disp(["Translation vector for images" files(i)])
    disp(t(:,i))

    x1 = p_correct(:,1,i);
    y1 = p_correct(:,2,i);

    H = eval(['Hnew' num2str(i)]);
    points_projection = H*grid_coordinates';
    for j=1:length(points_projection)
        points_projection(:,j) = points_projection(:,j) /points_projection(3,j);
    end
    points_projection = points_projection';

    x2 = points_projection(:,1);
    y2 = points_projection(:,2);

    disp(["New Homography Reprojection error for >> " files(i)])
    total_err_reprojection = sum(sqrt((x1(:)-x2(:)).^2 + (x1(:)-x2(:)).^2));
    disp(["Total Reprojection Error (as Euclidean Distance) >> " total_err_reprojectio
    disp(["Average Reprojection Error per point >> " total_err_reprojection/80]);

    H = eval(['H' num2str(i)]);
    points_projection_2 = H*grid_coordinates';
    for j=1:length(points_projection_2)
        points_projection_2(:,j) = points_projection_2(:,j) /points_projection_2(3,j);
    end
    points_projection_2 = points_projection_2';

    x2 = points_projection_2(:,1);
    y2 = points_projection_2(:,2);

    disp(["Part 2 Homography Reprojection error for >> " files(i)]);
    total_err_reprojection = sum(sqrt((x1(:)-x2(:)).^2 + (x1(:)-x2(:)).^2));
    disp(["Total Reprojection Error (as Euclidean Distance) >> " total_err_reprojectio
    disp(["Average Reprojection Error per point >> " total_err_reprojection/80]);
end
```

```
    "Rotation matrix R for images"    "images2"
    0.2633   -0.8597   -0.4377
    0.5796    0.5037   -0.6406
    0.7712   -0.0850    0.6309
    "Translation vector for images"    "images2"
 -154.5660
  104.6985
  388.9131
    "New Homography Reprojection error for >> "    "images2"
    "Total Reprojection Error (as Euclidean Di…"    "110.7128"
```

```
   "Average Reprojection Error per point >> "    "1.3839"
   "Part 2 Homography Reprojection error for …"    "images2"
   "Total Reprojection Error (as Euclidean Di…"    "346.7359"
   "Average Reprojection Error per point >> "    "4.3342"
   "Rotation matrix R for images"    "images9"
   0.7589    0.6267    0.1769
  -0.5408    0.7579   -0.3647
  -0.3626    0.1811    0.9142
   "Translation vector for images"    "images9"
-100.5481
  94.8001
 332.2943
   "New Homography Reprojection error for >> "    "images9"
   "Total Reprojection Error (as Euclidean Di…"    "109.402"
   "Average Reprojection Error per point >> "    "1.3675"
   "Part 2 Homography Reprojection error for …"    "images9"
   "Total Reprojection Error (as Euclidean Di…"    "238.7597"
   "Average Reprojection Error per point >> "    "2.9845"
   "Rotation matrix R for images"    "images12"
   0.9630   -0.2158    0.1612
   0.2069    0.9759    0.0700
  -0.1724   -0.0341    0.9844
   "Translation vector for images"    "images12"
-151.2509
 105.8987
 442.8284
   "New Homography Reprojection error for >> "    "images12"
   "Total Reprojection Error (as Euclidean Di…"    "141.07"
   "Average Reprojection Error per point >> "    "1.7634"
   "Part 2 Homography Reprojection error for …"    "images12"
   "Total Reprojection Error (as Euclidean Di…"    "409.651"
   "Average Reprojection Error per point >> "    "5.1206"
   "Rotation matrix R for images"    "images20"
   0.5847    0.6470    0.4893
  -0.8042    0.5416    0.2448
  -0.1066   -0.5367    0.8370
   "Translation vector for images"    "images20"
-122.8887
  25.6874
 395.6558
   "New Homography Reprojection error for >> "    "images20"
   "Total Reprojection Error (as Euclidean Di…"    "145.4892"
   "Average Reprojection Error per point >> "    "1.8186"
   "Part 2 Homography Reprojection error for …"    "images20"
   "Total Reprojection Error (as Euclidean Di…"    "250.5126"
   "Average Reprojection Error per point >> "    "3.1314"
```

A way in which the process can be done automatically without having the user manually select the corner points is by making use of the detectHarrisFeatures function which is inbuilt in matlab. This basically detects corners using Harris–Stephens algorithm and return cornerPoints object. Simply use

```
corners = detectHarrisFeatures(I);
```

`detectHarrisFeatures(I)` returns a `cornerPoints` object, `points`. The object contains information about the feature points detected in a 2-D input image, `I`. The `detectHarrisFeatures` function uses the Harris–Stephens algorithm to find these feature points.

Input arguments: Input image, specified is an *M*-by-*N* 2-D image. The input image must be real and nonsparse.

Output arguments: Corner points object, returned as a `cornerPoints` object. The object contains information about the feature points detected in the 2-D input image.

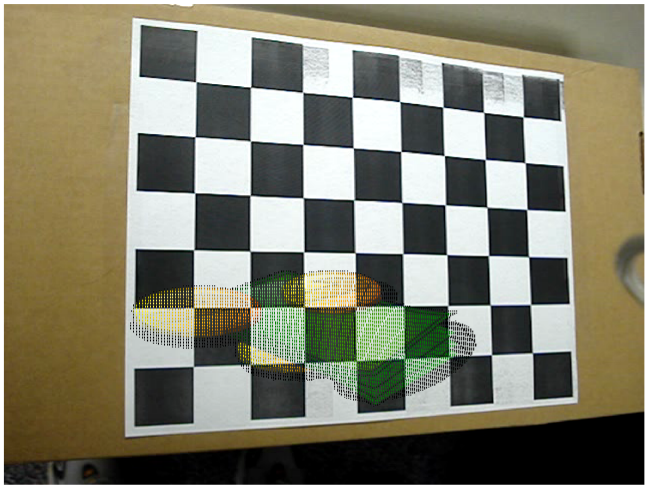
Part III: Augmented Reality 101

(a). Augmenting an Image

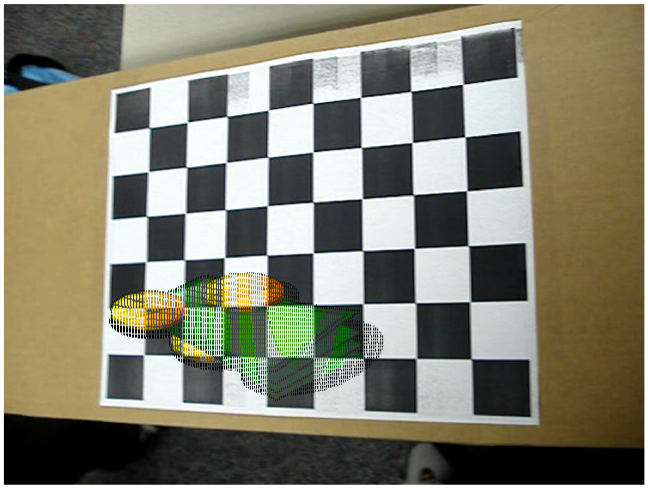
```
[clipart,~,Alpha]=imread("5.png");
height=150;
width=100;
clipart_resized=imresize(clipart,[height,width]);
A_resized=imresize(Alpha,[height,width]);
for i=1:4
 fprintf("--------------------------------%s-------------------------------- \n",images
 H=Hs(:,:,i);
 I=imread(images(i));
 for x=1:height
 for y=1:width
 val=clipart_resized(x,y,:);
 if A_resized(x,y)~=0
 X1=[y;x;1];
 X2=H*X1;
 X2=X2/X2(3);
 I(int64(X2(2)),int64(X2(1)),:)=val;
 end
 end
 end
 figure, imshow(I);
 snapnow;
end
```

--------------------------------images2.png--------------------------------

--------------------------------images9.png--------------------------------



------------------------------images12.png------------------------------



------------------------------images20.png------------------------------

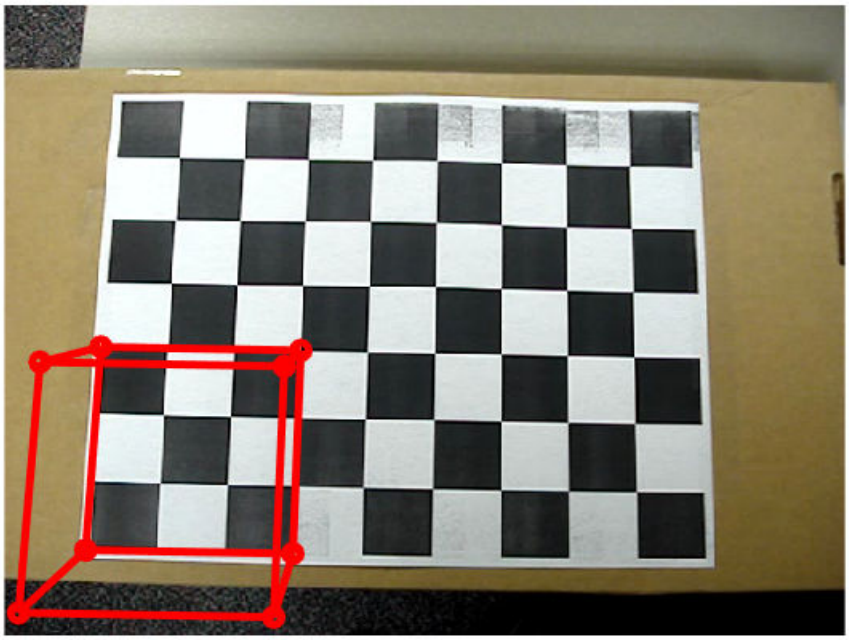
(b). Augmenting an Object

```
Cube=[0 90 90;90 90 90; 90 0 90;0 0 90;0 90 0;90 90 0;90 0 0;0 0 0];
edges=[1 2;1 4;2 3;3 4;5 6;5 8;6 7;7 8;1 5;2 6;3 7;4 8];
J=[]
```

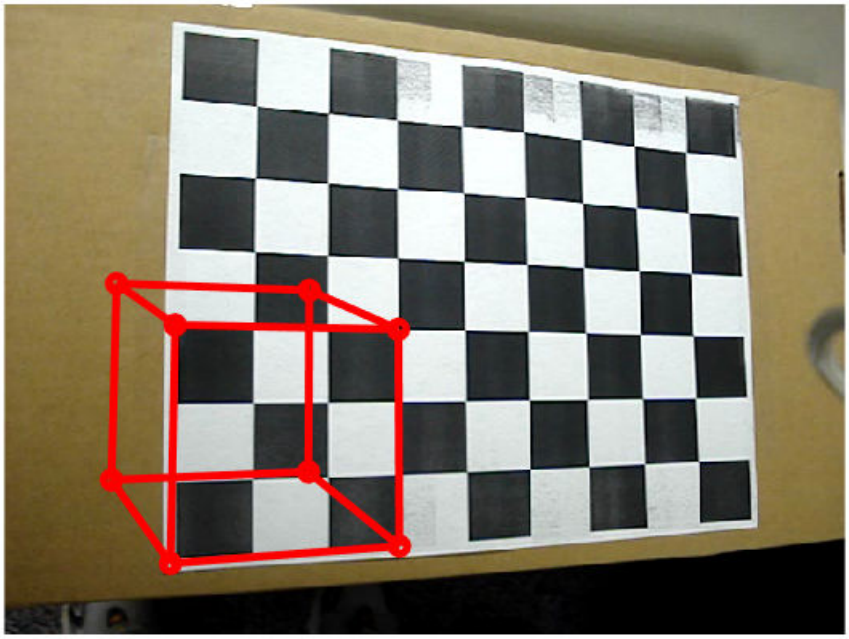
```
J =

     []
```

```
for i=1:4
 fprintf("--------------------------------%s-------------------------------- \n",images
 I=imread(images(i));
 R=Rs(:,:,i);
 T=Ts(:,i);
 figure, imshow(I)
 hold on
 points=[];
 for point_no=1:8
     point=Cube(point_no,:);
     X1=[transpose(point);1];
     X2=A*[R T]*X1;
     X2=X2/X2(3);
     points=[points [X2(1);X2(2)]];
  end
  for edge_no=1:12
     edge=edges(edge_no,:);
     X=[points(1,edge(1)),points(1,edge(2))];
     Y=[points(2,edge(1)),points(2,edge(2))];
     p=plot(X,Y,'-or');
     p.LineWidth =3;
  end
 snapnow;
end
```

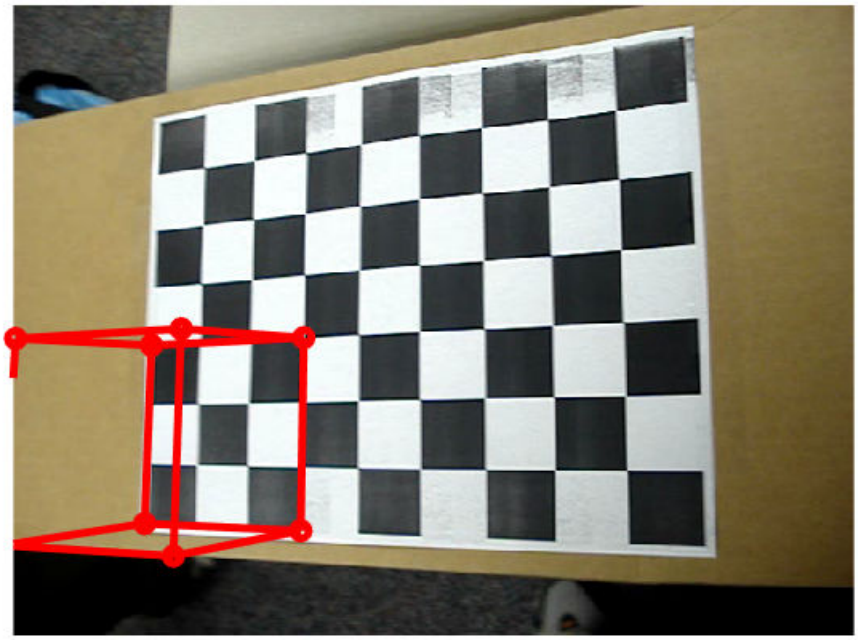
```
--------------------------------images2.png--------------------------------
```



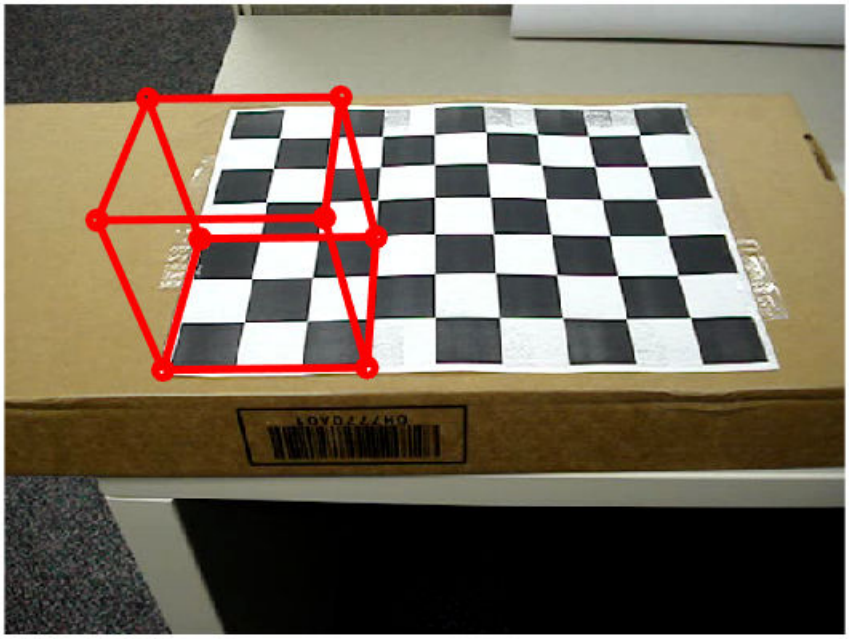
----------images9.png----------



----------images12.png----------



----------images20.png----------



EXTRA CREDIT QUESTION:

2. Can you find a way to estimate the intrinsic and extrinsic parameters from only two images of the grid? What assumptions on the intrinsic parameters are needed to achieve this. (Hint the answer can be found in Sec 2.4)

We know that if we observe n images of the model plane, we have Vb = 0 where V is a 2n X 6 matrix.
When we have n>=3 we have a unique solution b fro Vb = 0 defined up to a scale factor. But when we have only two images, i.e. n=2, we will have to impose a skewness constraint where gamma = 0, that is, [0,1,0,0,0,0]b = 0, which is added as an additional equation, i.e, if only 2 images are available, we can impose the skewness constraint $\gamma = 0$.
But if we have n=1, we can only solve two camera intrinsic parameters alpha and beta, assuming u_0 and v_0 are known and gamma = 0.