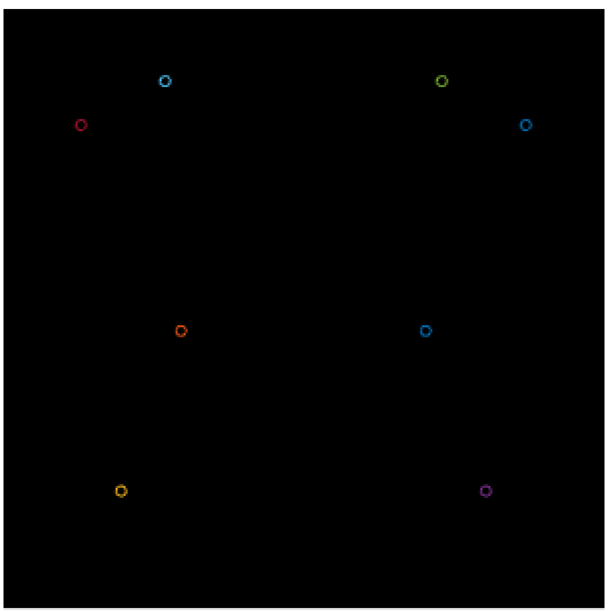
Part I: Camera Calibration using 3D calibration object

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

```
World_coord=[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_coord=[422 323;178 323;118 483;482 483;438 73;162 73;78 117;522 117];

I=zeros(600,600);
imshow(I);
hold on;
for i=1:8
    draw_circle(Image_coord(i,1),Image_coord(i,2));
end
hold off;
snapnow;

italicP=[];
```



3. Use this Matlab 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. italicP=[];

```
% 2. Write a Matlab 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 P (slide 30
of the Camera Calibration pdf document).
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
```

```
for i=1:8
    capP=transpose([World_coord(i,:) 1]);
    smallp=transpose(Image_coord(i,:));
    rows=Prows(smallp,capP);
    italicP=[italicP;rows];
end
disp("P");
disp(italicP);
```

```
P
  Columns 1 through 6

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

  Columns 7 through 12

         0            0         -844         -844         -844         -422
         2            1         -646         -646         -646         -323
         0            0          356         -356         -356         -178
         2            1          646         -646         -646         -323
         0            0          236         -236          236         -118
        -2            1          966         -966          966         -483
         0            0         -964         -964          964         -482
        -2            1         -966         -966          966         -483
         0            0         -876          876         -876         -438
         2            1         -146          146         -146          -73
         0            0          324          324         -324         -162
         2            1          146          146         -146          -73
         0            0          156          156          156          -78
        -2            1          234          234          234         -117
         0            0        -1044         1044         1044         -522
```

```
    -2          1       -234        234        234       -117
```

4. 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,S,V]=svd(italicP);
Melements=transpose(V(:,end));
M=[Melements(1:4);Melements(5:8);Melements(9:12)];
disp("M");
disp(M);
```

```
M
   -0.1925   -0.0283   -0.0786   -0.7346
   -0.0000   -0.2044   -0.0001   -0.6120
   -0.0000   -0.0001   -0.0003   -0.0024
```

5. Now we need to recover the translation vector which is a null vector of M. Find the singular value decomposition of matrix M = U?VT. 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);
center_cam=V(:,end);
center_cam=center_cam/center_cam(end);
disp("Translation vector/camera center");
disp(center_cam(1:3));
```

```
Translation vector/camera center
   -0.0000
   -2.9912
   -8.2695
```

6. Consider the 3x3 matrix M' composed of the rst 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 de ned in slide 38 where the axes e1, e2, e3 are the x, y, z axes respectively. Therefore M' can be written as M0 = KRTz RTy RTx

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

```
M'
  734.6289  107.8955  299.9999
    0.0009  780.1442    0.2641
    0.0000    0.3597    1.0000
```

7. We will perform the RQ factorization of M' in several steps. First, nd a rotation matrix Rx that sets the term at position (3,2) to zero when Rx is multiplied to M'. Compute matrix N = M' ? Rx. Print Rx, ?x and N.

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

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

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

```
Rx
    1.0000         0         0
         0    0.9410    0.3384
         0   -0.3384    0.9410

ThetaX: -19.781219
N
  734.6289   -0.0000  318.8125
    0.0009  734.0199  264.2723
    0.0000         0    1.0627
```

8. The element n31 of N is 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. Compute the rotation angle ?z in degrees. This angle is actually very small.

```
cosz=mdash(2,2)/sqrt((mdash(2,2)^2)+mdash(2,1)^2);
sinz=-mdash(2,1)/sqrt((mdash(2,2)^2)+mdash(2,1)^2);
thetaz=rad2deg(atan(sinz/cosz));
fprintf("ThetaZ: %f \n", thetaz);
Rz=[cosz -sinz 0;sinz cosz 0; 0 0 1];
```

```
ThetaZ: -0.000068
```

9. 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=mdash*inv(RxRz);
K=K/K(3,3);
disp("K");
disp(K);
disp("-----------------------------Intrinsic Parameters-------------------------- \n");
fprintf("Focal legnths--------------- \n");
fprintf("alpha: %f, beta: %f, gamma: %f \n", K(1,1),K(2,1),K(1,2));
fprintf("Image centers--------------- \n");
fprintf("u0: %f, v0: %f \n", K(1,3), K(2,3))
```

```
K
  691.2796    0.0008  300.0002
   -0.0000  690.7067  248.6780
    0.0000    0.0000    1.0000

----------------------------Intrinsic Parameters------------------------- \n
Focal legnths---------------
alpha: 691.279580, beta: -0.000000, gamma: 0.000768
Image centers---------------
u0: 300.000169, v0: 248.678031
```

Published with MATLAB® R2019a

Part II: Camera Calibration using 2D calibration object

Corner Extraction and Homography computation (10 points)

Take corner input and compute homographies

```
images=["images2.png","images9.png","images12.png","images20.png"];
max_x=270;
max_y=210;
world_coordinates=[0 max_x max_x 0; 0 0 max_y max_y;1 1 1 1];
V=[];

Hs=zeros(3,3,4);
for i=1:4
    I=imread(images(i));
    imshow(I);
    [x,y]=ginput(4);
    image_coordinates=[transpose(x);transpose(y);1 1 1 1];
    H=homography2d(world_coordinates, image_coordinates);
    fprintf("--------------------------------%s------------------------------- \n",images(i))
    disp("H:");
    disp(H);
    Hs(:,:,i)=H;
    v12=transpose(vij(1,2,H));
    v11=vij(1,1,H);
    v22=vij(2,2,H);
    v11v22=transpose(v11-v22);
    V=[V;v12;v11v22];
end
```

```
--------------------------------images2.png-------------------------------
H:
   -0.9679    0.0836  -53.4511
   -0.0076   -0.9024  -44.4425
    0.0000    0.0002   -0.6006
--------------------------------images9.png-------------------------------
H:
    1.0782   -0.0256   68.9334
    0.1236    0.9371   15.3774
    0.0004   -0.0001    0.5303

--------------------------------images12.png-------------------------------
H:
    0.7157   -0.0471   73.7297
   -0.1837    0.9189   60.4451
   -0.0005   -0.0002    0.6642
```

```
--------------------------------images20.png-------------------------------
H:
   -0.8414    0.2773 -121.2006
    0.0271   -0.4126  -59.2072
    0.0001    0.0008   -0.6966
```

Compute B

```
[b,D]=eigs(transpose(V)*V,1,'SM');
B=[b(1) b(2) b(4);b(2) b(3) b(5); b(4) b(5) b(6)];

v0=(B(1,2)*B(1,3)-B(1,1)*B(2,3))/(B(1,1)*B(2,2)-B(1,2)^2);
lambda=B(3,3)-(B(1,3)^2+v0*(B(1,2)*B(1,3)-B(1,1)*B(2,3)))/B(1,1);
alpha=sqrt(lambda/B(1,1));
beta=sqrt(lambda*B(1,1)/(B(1,1)*B(2,2)-B(1,2)^2));
gamma=-B(1,2)*(alpha^2)*beta/lambda;
u0=(gamma*v0/alpha)-(B(1,3)*(alpha^2)/lambda);

disp("B");
disp(B);

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

```
B
    0.0000   -0.0000   -0.0003
   -0.0000    0.0000    0.0000
   -0.0003    0.0000    1.0000
```

Computing the Intrinsic and Extrinsic parameters (30 points)

Compute R,t and R'X R for each image

```
Rs=zeros(3,3,4);
for i=1:4
    H=Hs(:,:,i);
    Ah=inv(A)*H(:,1);
    lambda=1/sqrt(transpose(Ah)*Ah);
    r1=lambda*inv(A)*H(:,1);
    r2=lambda*inv(A)*H(:,2);
    r3=cross(r1,r2);
    t=lambda*inv(A)*H(:,3);
    R=[r1 r2 r3];
    R_T=transpose(R);
    Rs(:,:,i)=R;
    fprintf("--------------------------------%s------------------------------- \n",images(i))
    disp("R:");
    disp(R);
    disp("t:");
    disp(t);
```

```
    disp("R'R:");
    disp(R_T*R);
end
```

```
--------------------------------images2.png-------------------------------
R:
   -0.9998    0.1584    0.0137
   -0.0089   -0.9977    0.1786
    0.0153    0.1762    0.9989

t:
  103.8280
  -30.3413
 -561.8889

R'R:
    1.0000   -0.1468    0.0000
   -0.1468    1.0515         0
    0.0000         0    1.0299

--------------------------------images9.png-------------------------------
R:
    0.9151   -0.1074   -0.3873
    0.1140    0.9649    0.0720
    0.3867   -0.1241    0.8953

t:
  -61.6641
    0.5240
  462.8006

R'R:
    1.0000   -0.0363    0.0000
   -0.0363    0.9581    0.0000
    0.0000    0.0000    0.9567

--------------------------------images12.png-------------------------------
R:
    0.8558   -0.1254    0.4930
   -0.1758    0.9638    0.1778
   -0.4866   -0.1365    0.8028

t:
  -96.2080
   43.6620
  590.2626

R'R:
    1.0000   -0.2103         0
   -0.2103    0.9633   -0.0000
         0   -0.0000    0.9191

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

```
R:
   -0.9957    0.1476    0.0730
    0.0304   -0.5347    0.8645
    0.0880    0.8552    0.5279

t:
   67.5390
  -48.7085
 -727.8444

R'R:
    1.0000   -0.0880         0
   -0.0880    1.0391         0
         0         0    1.0313
```

Compute R and R' X R under constraint

```
for i=1:4
    R=Rs(:,:,i);
    [U,S,V]=svd(R);
    newR=U*transpose(V);
    fprintf("--------------------------------%s------------------------------- \n",images(i))
    disp("R:");
    disp(newR)
    disp("R'R:");
    disp(transpose(newR)*newR);
end
```

```
--------------------------------images2.png-------------------------------
R:
   -0.9964    0.0841    0.0135
   -0.0804   -0.9811    0.1760
    0.0280    0.1743    0.9843

R'R:
    1.0000    0.0000    0.0000
    0.0000    1.0000    0.0000
    0.0000    0.0000    1.0000

--------------------------------images9.png-------------------------------
R:
    0.9136   -0.0926   -0.3960
    0.1321    0.9885    0.0736
    0.3846   -0.1196    0.9153

R'R:
    1.0000    0.0000    0.0000
    0.0000    1.0000   -0.0000
    0.0000   -0.0000    1.0000

--------------------------------images12.png-------------------------------
R:
```

```
    0.8570   -0.0347    0.5142
   -0.0716    0.9800    0.1855
   -0.5104   -0.1957    0.8374

R'R:
    1.0000   -0.0000   -0.0000
   -0.0000    1.0000   -0.0000
   -0.0000   -0.0000    1.0000

--------------------------------images20.png-------------------------------
R:
   -0.9921    0.1025    0.0719
    0.0075   -0.5247    0.8512
    0.1250    0.8451    0.5198

R'R:
    1.0000   -0.0000    0.0000
   -0.0000    1.0000    0.0000
    0.0000    0.0000    1.0000
```

Improving accuracy

• First given the computed homographies from Section 2, compute the approximate location of each grid corner in the image. (Hint : This can be done since we know the 3d locations of the grid corners and the approximate homography. Call these points p_approx. Create a gure with the image and approximate grid locations. Call this Figure 1 : Projected grid corners [deliverable]

```
fprintf("--------------------------------%s------------------------------- \n",images(i))
old_Hs=Hs;
V=[];
new_Hs=zeros(3,3,4);
mean_errors=[];
old_mean_errors=[];
for i=1:4
```

```
    p_approx=[];
    H=Hs(:,:,i);
    Xs=linspace(0,max_x,10);
    Ys=linspace(0,max_y,8);
    homo_world_coordinates=[];
    I=imread(images(i));
    imshow(I);
    title("Figure 1: Projected grid corners")
    hold on;
    for x_index=1:10
        for y_index=1:8
            X2=[Xs(x_index);Ys(y_index);1];
```

```
            X1=H*X2;
            X1=X1/X1(3);
            homo_world_coordinates=[homo_world_coordinates X2];
            p_approx=[p_approx;X1(1) X1(2)];
            draw_circle(X1(1),X1(2));
        end
    end
    hold off;
    snapnow;
```

Figure 1: Projected grid corners

Figure 1: Projected grid corners

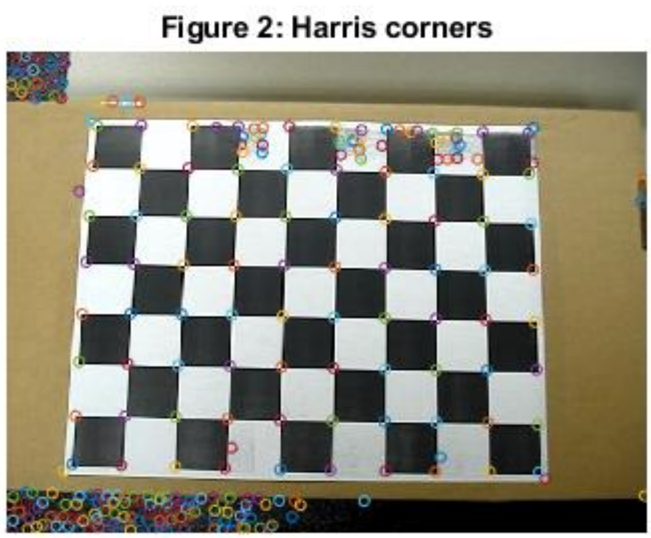
Figure 1: Projected grid corners

Figure 1: Projected grid corners

• Second, using the provided Harris function detect Harris corners in the image and display them. Use the following parameter values for the Harris detection : sigma = 2, thresh = 500, radius = 2. [cim, r, c, rsubp, csubp] = harris(rgb2gray(im), sigma, thresh, radius, disp); Here r is the y-coordinate of the Harris corner, c is the x-coordinate of the Harris corner, rsubp is the y coordinate with subpixel accuracy, csubp is the x coordinate with subpixel accuracy. Use rsubp, csubp. Create a gure with image and overlayed Harris corners. Call this Figure 2 : Harris corners . [deliverable]

```
fprintf("--------------------------------%s------------------------------- \n",images(i))
[cim, r, c, rsubp, csubp] = harris(rgb2gray(I), 2, 500,2,0);
harris_corners=[csubp rsubp];
imshow(I);
title("Figure 2: Harris corners");
hold on;
for index=1:length(rsubp)
    draw_circle(csubp(index),rsubp(index));
end
hold off;
snapnow;
```

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




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

Figure 2: Harris corners

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

Figure 2: Harris corners

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

Figure 2: Harris corners

- Third, compute the closest Harris corner to each approximate grid corner. (You may nd it useful to the image and p_correct overlayed. Call this Figure 3 : grid points . [deliverable]

```
fprintf("------------------------------%s------------------------------ \n",images(i))
distance_matrix=dist2(harris_corners,p_approx);
[M,A]=min(distance_matrix);
p_correct=harris_corners(A,:);
imshow(I);
title("Figure 3: Grid corners");
hold on;
for index=1:length(p_correct)
    draw_circle(p_correct(index,1),p_correct(index,2));
end
hold off;

snapnow;
```

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

Figure 3: Grid corners

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

Figure 3: Grid corners

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

Figure 3: Grid corners

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

Figure 3: Grid corners

• Finally, compute a new homography from p_correct, print H [deliverable]

```
    fprintf("--------------------------------%s------------------------------- \n",images(i))

homo_image_coordinates=[transpose(p_correct(:,1));transpose(p_correct(:,2));ones(1,length(p_corre
ct))];
    H=homography2d(homo_world_coordinates,homo_image_coordinates);
```

```
    errors=[];
    for j=1:length(p_correct)
        X2=homo_world_coordinates(:,j);
        X1=H*X2;
        X1=X1/X1(3);
        error=sqrt((X1(1)-p_correct(j,1))^2+(X1(2)-p_correct(j,2))^2);
        errors=[errors error];
    end
    mean_errors=[mean_errors mean(errors)];

    old_errors=[];
    old_H=old_Hs(:,:,i);
    for j=1:length(p_correct)
        X2=homo_world_coordinates(:,j);
        X1=old_H*X2;
        X1=X1/X1(3);
        old_error=sqrt((X1(1)-p_correct(j,1))^2+(X1(2)-p_correct(j,2))^2);
        old_errors=[old_errors old_error];
    end
    old_mean_errors=[old_mean_errors mean(old_errors)];

    disp("H:");
    disp(H);

    new_Hs(:,:,i)=H;
    v12=transpose(vij(1,2,H));
    v11=vij(1,1,H);
    v22=vij(2,2,H);
    v11v22=transpose(v11-v22);
    V=[V;v12;v11v22];
```

```
--------------------------------images2.png-------------------------------
H:
    0.9681   -0.0853   53.5654
    0.0153    0.8942   43.0470
   -0.0000   -0.0002    0.6022


--------------------------------images9.png-------------------------------
H:
    1.0995   -0.0346   70.6543
    0.1486    0.9483    9.4997
    0.0005   -0.0001    0.5203


--------------------------------images12.png------------------------------
H:
    0.7144   -0.0565   74.9893
   -0.1791    0.9014   59.7154
   -0.0005   -0.0002    0.6689


--------------------------------images20.png------------------------------
H:
```

```
    0.8621   -0.2761  119.1720
   -0.0039    0.3989   55.1261
    0.0000   -0.0008    0.6757
```

```
end
```

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

- Repeat this for the other three images. Then use the homographies to estimate K and R, t for each image. Report your K, R's, and t's [deliverable]. Save your results, you will need to use them in Part III

```
Hs=new_Hs;

[b,D]=eigs(transpose(V)*V,1,'SM');
B=[b(1) b(2) b(4);b(2) b(3) b(5); b(4) b(5) b(6)];

v0=(B(1,2)*B(1,3)-B(1,1)*B(2,3))/(B(1,1)*B(2,2)-B(1,2)^2);
lambda=B(3,3)-(B(1,3)^2+v0*(B(1,2)*B(1,3)-B(1,1)*B(2,3)))/B(1,1);
alpha=sqrt(lambda/B(1,1));
beta=sqrt(lambda*B(1,1)/(B(1,1)*B(2,2)-B(1,2)^2));
gamma=-B(1,2)*(alpha^2)*beta/lambda;
u0=(gamma*v0/alpha)-(B(1,3)*(alpha^2)/lambda);

A=[alpha gamma u0;0 beta v0;0 0 1];
disp("K:")
disp(A)

Rs=zeros(3,3,4);
Ts=[];
for i=1:4
    fprintf("--------------------------------%s------------------------------- \n",images(i))
    H=Hs(:,:,i);
    Ah=inv(A)*H(:,1);
    lambda=1/sqrt(transpose(Ah)*Ah);
    r1=lambda*inv(A)*H(:,1);
    r2=lambda*inv(A)*H(:,2);
    r3=cross(r1,r2);
    t=lambda*inv(A)*H(:,3);
    R=[r1 r2 r3];
    R_T=transpose(R);
    Rs(:,:,i)=R;
    disp("R:");
    disp(R);
    disp("t:");
    disp(t);
    Ts=[Ts t];

end
```

```
K:
  838.0415  219.0149  238.9904
         0  812.2935   56.5791
         0         0    1.0000

--------------------------------images2.png-------------------------------
R:
    0.9999   -0.2886    0.0015
    0.0167    0.9686    0.1871
   -0.0047   -0.1858    0.9732

t:
  -96.1110
    9.5909
  522.8100

--------------------------------images9.png-------------------------------
R:
    0.9011   -0.2484   -0.4062
    0.1175    0.9426   -0.0057
    0.4173   -0.1088    0.8786

t:
  -46.1820
  -19.6612
  416.7535

--------------------------------images12.png-------------------------------
R:
    0.8791   -0.2566    0.4462
   -0.1523    0.9357    0.2519
   -0.4516   -0.1547    0.7835

t:
  -90.2868
   22.4381
  557.5697

--------------------------------images20.png-------------------------------
R:
    0.9996   -0.2269   -0.0091
   -0.0066    0.5377    0.8200
    0.0272   -0.8265    0.5360

t:
  -54.6642
   20.3374
  660.5360
```

- Using the new computed H, compute the errors between points in p_correct and points you get by projecting grid corners to the image (Hint there is no need to use R, t for projecting) . Call this err_reprojection. Report your result. [deliverable]

```
for i=1:4
    fprintf("--------------------------------%s------------------------------- \n",images(i))
    fprintf("error_reprojection: %f\n",mean_errors(i));
end
fprintf("\n")
```

```
--------------------------------images2.png-------------------------------
error_reprojection: 1.780278
--------------------------------images9.png-------------------------------
error_reprojection: 2.023190
--------------------------------images12.png-------------------------------
error_reprojection: 2.112336
--------------------------------images20.png-------------------------------
error_reprojection: 1.868357
```

- Now repeat the process using 4 images. Compare your results to your previous results and those of part 2 [deliverable].

```
for i=1:4
    fprintf("--------------------------------%s------------------------------- \n",images(i))
    fprintf("Average error before improvement: %f \n",old_mean_errors(i));
    fprintf("Average error after improvement: %f \n \n",mean_errors(i));
end
```

```
--------------------------------images2.png-------------------------------
Average error before improvement: 2.858509
Average error after improvement: 1.780278

--------------------------------images9.png-------------------------------
Average error before improvement: 5.931050
Average error after improvement: 2.023190

--------------------------------images12.png-------------------------------
Average error before improvement: 5.009999
Average error after improvement: 2.112336

--------------------------------images20.png-------------------------------
Average error before improvement: 5.645813
Average error after improvement: 1.868357
```

Part III: Augmented Reality 101

Augmenting an Image

```
[clipart,~,Alpha]=imread("4.png");
height=210;
width=183;
clipart_resized=imresize(clipart,[height,width]);
A_resized=imresize(Alpha,[height,width]);
for i=1:4
    fprintf("--------------------------------%s------------------------------- \n",images(i))
    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
    imshow(I);
    snapnow;
end
```

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

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

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

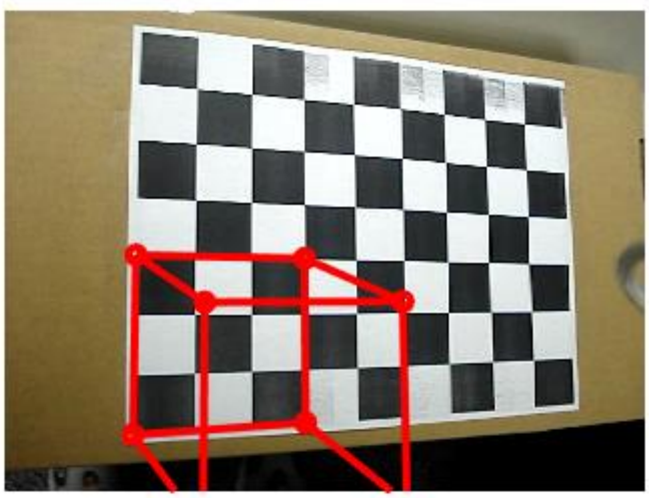
---------------------------------images20.png--------------------------------

Augmenting an Object

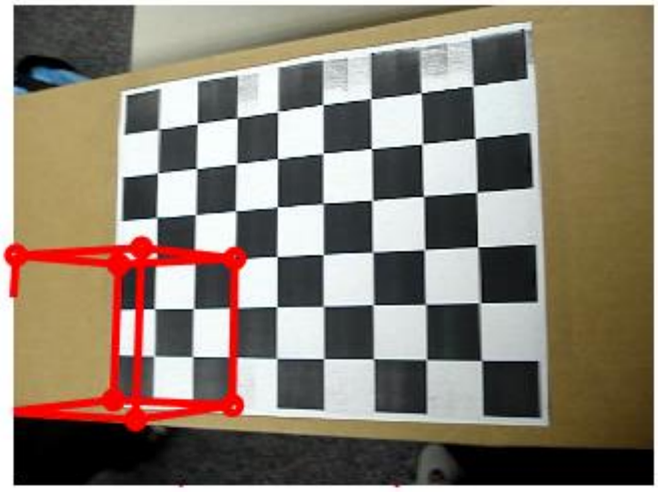
```
Cube=[0 210 0;90 210 0; 90 120 0;0 120 0;0 210 -90;90 210 -90;90 120 -90;0 120 -90];
edges=[1 2;1 4;2 3;3 4;5 6;5 8;6 7;7 8;1 5;2 6;3 7;4 8];
for i=1:4
    fprintf("--------------------------------%s------------------------------- \n",images(i))
    I=imread(images(i));
    R=Rs(:,:,i);
    T=Ts(:,i);
    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------------------------------

Helper function to calculate V vector

```
function v=vij(i,j,H)
    t1=H(1,i)*H(1,j);
    t2=H(1,i)*H(2,j)+H(2,i)*H(1,j);
    t3=H(2,i)*H(2,j);
    t4=H(3,i)*H(1,j)+H(1,i)*H(3,j);
    t5=H(3,i)*H(2,j)+H(2,i)&H(3,j);
    t6=H(3,i)*H(3,j);
```

```
    v=[t1;t2;t3;t4;t5;t6];
end
```

Published with MATLAB® R2019a

Extra credit

2. If only 2 images are available, we can impose the skewless constraint $\gamma = 0$.