198: Computer Vision: Homework #3
Camera Calibration & Augmented Reality

Due on Saturday, May 2, 2015

Prof. Elgammal

Eric Wengrowski

Contents

Part 1 - Camera calibration with 3D object

```
worldCorners = [
    2   2   2;
   -2   2   2;
   -2   2  -2;
    2   2  -2;
    2  -2   2;
   -2  -2   2;
   -2  -2  -2;
    2  -2  -2;];

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

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

```
figure(1), plot(cameraCorners(:,1),cameraCorners(:,2),'o'), title('World corners of 3D
```

1.2

```
% 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 P (slide 30 of the Camera Calibration pdf
% document). This matrix P will be used to compute the 12 elements of the
% projection matrix M such that lambda*p_i = M*P_i
    P = matrixP(worldCorners,cameraCorners);
```

1.3 - Print matrix P

```
disp('Matrix P')
disp(P);
```

1.4

```
% 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.
```

```
    [~,~,Vprime] = svd(P);

    % Find M, where p=M*P
    M(1,1:4) = Vprime(1:4,end)';
    M(2,1:4) = Vprime(5:8,end)';
    M(3,1:4) = Vprime(9:12,end)';
    % Print matrix M
    disp('Matrix M')
    disp(M);
```

1.5

```
% Now we need to recover the translation vector which is a null vector of
% M. Find the singular value decomposition of matrix M = U*S*V'. 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
    [~,~,Vprime] = svd(M);

    % V(:,end) are homogeneous coordinates of Camera center
    camCenter = Vprime(:,end)'/Vprime(end,end)';
    disp('Camera Center')
    disp(camCenter)

\subsection{1.6}
% 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.
% Therefore M can be written as M' = K * R_x' * R_y' * R_z'
    M3 = M(1:3,1:3);
    M3 = M3/M3(end,end);
    % Print matrix M' (M3)
    disp('M''')
    disp(M3)
```

1.7 - RQ factorization of M (aka M3)

```
% First, find a rotation matrix Rx that sets the term at position (3,2)
% to zero when Rx is multiplied to M. The cosine and sine used in this
% matrix are of the form:
% cos(theta_x) = m(3,3)/sqrt(m(3,3)^2 + m(3,2)^2)
% sin(theta_x) = -m(3,2)/sqrt(m(3,3)^2 + m(2,3)^2)
% Note that the term at position (3,2) would also be set to zero if the
% signs of cos(theta_x) and sin(theta_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.
    crx = M3(3,3)/sqrt(M3(3,3)^2 + M3(3,2)^2);
    srx = -M3(3,2)/sqrt(M3(3,3)^2 + M3(3,2)^2);
%     theta_x = asin(srx);
    theta_x = acos(crx);
%     Rx = [1     0               0
%           0     cos(theta_x)    -sin(theta_x)
%           0     sin(theta_x)    cos(theta_x)];

    Rx = [1      0      0
          0      crx   -srx
          0      srx    crx];
    N = M3*Rx;
    disp('R_x = ')
    disp(Rx)
    disp('theta_x = ')
    disp(theta_x)
    disp('Matrix N')
    disp(N)
```

1.8 - Rotation in Z direction

```
    crz = N(2,2)/sqrt(N(2,1)^2 + N(2,2)^2);
    srz = -N(2,1)/sqrt(N(2,1)^2 + N(2,2)^2);
%     theta_z = asin(srz);
    theta_z = acos(crz);
    Rz = [crz   -srz    0
          srz    crz    0
          0      0      1];
%     N = N*Rz;
    disp('theta_z = ')
    disp(theta_z)
```

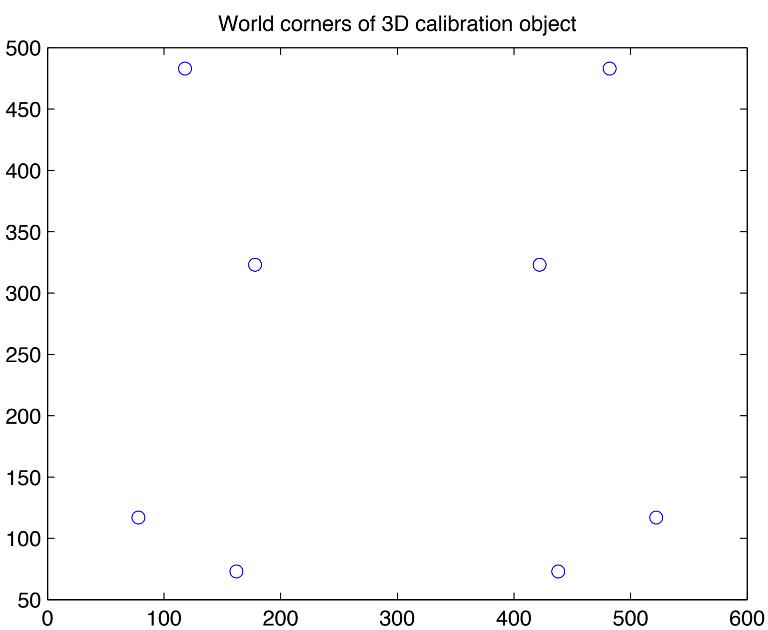
1.9 - Compute K

```
    % M3 = K*R
    R = Rx*eye(3)*Rz;
    % R^-1 = transp(R) = R'
    % M3 * R' = K
    K = M3 * R;
    K = K/K(3,3);
    disp('Matrix K')
    disp(K)
    %{
    K = [f*mx   skew   u0
         0      f*my   v0
         0      0      1];
    %}
```

```
% Compute image center
u0 = K(1,3);
v0 = K(2,3);
disp('Focal length = ')
disp((K(1,1)+K(2,2))/2)
disp('Image center = ')
disp([u0,v0])
```

```
Matrix 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
Matrix 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
Camera Center
   -0.0000   -2.9912   -8.2695    1.0000
```

```
M'
  734.6289  107.8955  299.9999
    0.0009  780.1442    0.2641
    0.0000    0.3597    1.0000
R_x =
    1.0000         0         0
         0    0.9410    0.3384
         0   -0.3384    0.9410
theta_x =
    0.3452
Matrix N
  734.6289   -0.0000  318.8125
    0.0009  734.0199  264.2723
    0.0000         0    1.0627
theta_z =
   1.2602e-06
Matrix K
  691.2797    0.0009  299.9999
    0.0000  690.7067  248.6780
    0.0000    0.0000    1.0000
Focal length =
  690.9932
Image center =
  299.9999  248.6780
```



Part 2 - Camera calibration with 2D calibration object

Checkerboard is 9 squares in width, 7 squares in height, 30x30mm squares Bottom left corner of grid is origin of world cooridnate system. Checkerboard grid is plane at Z=0

```
grid1 = imread('images2.png');
grid2 = imread('images9.png');
grid3 = imread('images12.png');
grid4 = imread('images20.png');

gridWidth = 9;
gridHeight = 7;

% Corners in 3x4 homogeneous world coordinates
wc = [0,            gridHeight*30,  1;
      gridWidth*30, gridHeight*30,  1;
      gridWidth*30, 0,              1;
      0,            0,              1];
wc=wc';
```

2.1 - Corner Extraction and Homography computation

```
% Manually detect checkerboard corners: clockwise from top right
% Order: TopRight, TopLeft, BottomLeft, BottomRight
if ~exist('x1','var')
    figure(2), imshow(grid1), [x1,y1] = ginput(4);
    figure(2), imshow(grid2), [x2,y2] = ginput(4);
    figure(2), imshow(grid3), [x3,y3] = ginput(4);
    figure(2), imshow(grid4), [x4,y4] = ginput(4);
    close(2);
end

% Convert to 3xN homogenous cooridnates, N=4
ic1 = [x1'; y1'; [1,1,1,1]];
ic2 = [x2'; y2'; [1,1,1,1]];
ic3 = [x3'; y3'; [1,1,1,1]];
ic4 = [x4'; y4'; [1,1,1,1]];

% Compute homography H that relates the grid 3d coordinates to the corners
H1 = homography2d(wc,ic1);
H2 = homography2d(wc,ic2);
H3 = homography2d(wc,ic3);
H4 = homography2d(wc,ic4);
% NOTE: It was not clear in the documentation, that the world
% coordinates WC should be the first input variable, rather than the
% pixel coordinates PC in homography2d(WC,PC).
% But that that seems to make a huge difference.

H1 = H1/H1(3,3);
```

```
H2 = H2/H2(3,3);
H3 = H3/H3(3,3);
H4 = H4/H4(3,3);

% Display results
disp('Homography for images2.png')
disp(H1)
disp('Homography for images9.png')
disp(H2)
disp('Homography for images12.png')
disp(H3)
disp('Homography for images20.png')
disp(H4)
```

```
Homography for images2.png
    1.7435    0.1589   67.0000
    0.0339   -1.5770  413.0000
    0.0000    0.0004    1.0000
Homography for images9.png
    2.1928    0.0615  132.0000
    0.2893   -1.8937  422.0000
    0.0010    0.0003    1.0000
Homography for images12.png
    1.1147    0.0814  104.0000
   -0.2860   -1.4058  394.0000
   -0.0009    0.0003    1.0000
Homography for images20.png
    1.6477    0.5275  128.0000
   -0.0289   -0.7825  277.0000
   -0.0001    0.0016    1.0000
```

2.2 - Computing Intrinsic and Extrinsic parameters

```
H = [H1, H2, H3, H4];
h = zeros(3,3);
V = [];

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

    k=1;j=2;
    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),

    k=1;j=1;
    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),
```

```
    k=2;j=2;
    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

% Solve V*b=0,
% Sove for b by finding the eigenvector of V'*V associated with the
% smallest eigenvalue
%(equivalently, the right singular vector of V associated with the
% smallest singular value)
[U,S,Vprime] = svd(V);
b = Vprime(:,end)';

% b = [B11, B12, B22, B13, B23, B33]';
% B = inv(A')*inv(A), symetric
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];

%Intrinsic parameters
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;

% B=lambda*inv(A)'*A
A = [alpha, gamma,   u0;
         0,  beta,   v0;
         0,     0,    1];

disp('Intrinsic Parameters matrix K = ')
disp(A)

H = [H1, H2, H3, H4];
photoSet = [2,9,12,20];

for i = 1:4
    Ho = eval(['H' num2str(i)]);
    h1 = Ho(:,1);
    h2 = Ho(:,2);
```

```
        h3 = Ho(:,3);
        lambda = 1/norm(A\h1); %%%% NEW LAMBDA - VERY CONFUSING NOTATION
        r1 = lambda*inv(A)*h1;
        r2 = lambda*inv(A)*h2;
        r3 = cross(r1,r2);
        t  = lambda*inv(A)*h3;

%         R(:,:,i) = [r1, r2, r3];
%         t(:,i) = t;
        R = [r1, r3, r2]; %%% Not sure why this has to be the case

        disp(['Rotation matrix for images' num2str(photoSet(i)) '.png'])
        disp(R)
        disp(['Translation vector for images' num2str(photoSet(i)) '.png'])
        disp(t)
        disp(['R''*R for images' num2str(photoSet(i)) '.png'])
        disp(R'*R)

    % R'*R is not eye(3)
    % Better method of estimating R: use SVD, set singular values to 1s
    % New Rotation matrix = Rnew = U*V', where R = U*S*V'

        [U,S,Vprime] = svd(R);
        Rnew = U*Vprime;

        disp(['New R via SVD for images' num2str(photoSet(i)) '.png'])
        disp(Rnew)
        disp(['New R''*R via SVD for images' num2str(photoSet(i)) '.png'])
        disp(Rnew'*Rnew)
    end
```

```
Intrinsic Parameters matrix K =
  718.0150    2.9850  316.9536
         0  703.3625  232.5124
         0         0    1.0000
Rotation matrix for images2.png
    0.9998    0.0108    0.0165
    0.0173   -0.1790   -0.9857
    0.0079   -0.9858    0.1792
Translation vector for images2.png
 -144.2853
  106.0317
  413.2068
R'*R for images2.png
    1.0000         0    0.0008
         0    1.0040         0
    0.0008         0    1.0040
New R via SVD for images2.png
    0.9998    0.0108    0.0160
```

```
    0.0177   -0.1787   -0.9837
    0.0078   -0.9838    0.1788
New R'*R via SVD for images2.png
    1.0000    0.0000   -0.0000
    0.0000    1.0000   -0.0000
   -0.0000   -0.0000    1.0000
Rotation matrix for images9.png
    0.9322    0.3616   -0.0087
    0.0277   -0.0948   -0.9948
    0.3608   -0.9272    0.0983
Translation vector for images9.png
  -92.4713
   96.2931
  357.4320
R'*R for images9.png
    1.0000   -0.0000   -0.0002
   -0.0000    0.9994   -0.0000
   -0.0002   -0.0000    0.9994
New R via SVD for images9.png
    0.9322    0.3617   -0.0086
    0.0276   -0.0948   -0.9951
    0.3608   -0.9274    0.0984
New R'*R via SVD for images9.png
    1.0000         0    0.0000
         0    1.0000         0
    0.0000         0    1.0000
Rotation matrix for images12.png
    0.9117   -0.4101   -0.0041
   -0.0572   -0.1257   -0.9884
   -0.4068   -0.9014    0.1397
Translation vector for images12.png
 -140.2684
  108.2363
  471.4255
R'*R for images12.png
    1.0000         0   -0.0040
         0    0.9964    0.0000
   -0.0040    0.0000    0.9964
New R via SVD for images12.png
    0.9117   -0.4108   -0.0023
   -0.0592   -0.1259   -0.9903
   -0.4065   -0.9030    0.1391
New R'*R via SVD for images12.png
    1.0000    0.0000   -0.0000
    0.0000    1.0000         0
   -0.0000         0    1.0000
Rotation matrix for images20.png
    0.9997   -0.0231    0.0103
   -0.0104   -0.7014   -0.7117
```

```
   -0.0223   -0.7113    0.7014
Translation vector for images20.png
 -113.6192
   27.2807
  431.3167
R'*R for images20.png
    1.0000         0    0.0020
         0    0.9985         0
    0.0020         0    0.9985
New R via SVD for images20.png
    0.9997   -0.0232    0.0093
   -0.0097   -0.7019   -0.7122
   -0.0230   -0.7119    0.7019
New R'*R via SVD for images20.png
    1.0000   -0.0000    0.0000
   -0.0000    1.0000   -0.0000
    0.0000   -0.0000    1.0000
```

2.3 - Improving Accuracy

compute the approximate location of each grid corner in the image use previously computed homographies and known 3d locations of the grid corners

```
% known 3d locations of the grid corners
wc_allpoints = [];

for i = 0:gridWidth
    for j = 0:gridHeight
        wc_allpoints = [wc_allpoints; i*30, j*30, 1];
    end
end

% Compute all grid locations with homography- NOW WORKING PROPERLY :D
p_approx = zeros((gridWidth+1)*(gridHeight+1), 3);
H = [H1, H2, H3, H4];
Hnew = zeros(3,3,4);

for i = 1:4     % For all 4 calibration images
    H = eval(['H' num2str(i)]);
    p_approx = H*wc_allpoints';
    for j=1:length(p_approx)    % Normalize
        p_approx(:,j) = p_approx(:,j) / p_approx(3,j);
    end

    grid = eval(['grid' num2str(i)]);
    figure(), imshow(grid), title(['Figure1: Projected grid corners for images' num2st
    hold on
    plot(p_approx(1,:),p_approx(2,:),'ro');
    hold off
```

```
    % Harris corner detection
        sigma = 2;
        thresh = 500;
        radius = 2;
        [cim,r,c,rsubp,csubp]=harris(rgb2gray(grid),sigma,thresh,radius,1);
        title(['Figure 2 : Harris corners for images' num2str(photoSet(i)) '.png'])

    % Compute the closest Harris corner to each approx grid corner
        D = dist2(p_approx(1:2,:)',[csubp, rsubp]);
        [D_sorted, D_index] = sort(D, 2);
        p_correct(:,:,i) = [csubp(D_index(:,1)),rsubp(D_index(:,1)),ones((gridWidth+1)*(gri
        figure(), imshow(grid), title(['Figure3: Grid Points for images' num2str(photoSet(i
        hold on
        plot(p_correct(:,1,i),p_correct(:,2,i),'g+')
        hold off

    % Compute new homography from p_correct
        Hnew(:,:,i) = homography2d(wc_allpoints',p_correct(:,:,i)');
        Hnew(:,:,i) = Hnew(:,:,i)/Hnew(3,3,i);
        disp(['Recomputed Homography H of ' num2str(photoSet(i)) '.png'])
        disp(Hnew(:,:,i))

    end % for all 4 images

    Hn1 = Hnew(:,:,1);
    Hn2 = Hnew(:,:,2);
    Hn3 = Hnew(:,:,3);
    Hn4 = Hnew(:,:,4);

% Use Hnew to estimate K,R,t for each image
    Hn = [Hn1, Hn2, Hn3, Hn4];
    H = [H1, H2, H3, H4];
    %h = zeros(3,3);
    V = [];

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

        k=1;j=2;
        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),

        k=1;j=1;
        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),

        k=2;j=2;
```

```
    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

% Solve V*b=0,
% Sove for b by finding the eigenvector of V'*V associated with the
% smallest eigenvalue
%(equivalently, the right singular vector of V associated with the
% smallest singular value)
[U,S,Vprime] = svd(V);
b = Vprime(:,end)';

% b = [B11, B12, B22, B13, B23, B33]';
% B = inv(A')*inv(A), symetric
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];

%Intrinsic parameters
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;

% B=lambda*inv(A)'*A
A = [alpha, gamma,   u0;
         0,  beta,   v0;
         0,     0,    1];
disp('Intrinsic Parameters matrix K = ')
disp(A)

photoSet = [2,9,12,20];
Hn = [Hn1, Hn2, Hn3, Hn4];
H = [H1, H2, H3, H4];

for i = 1:4 % for all 4 images
    Hnn = eval(['Hn' num2str(i)]);
    h1 = Hnn(:,1);
    h2 = Hnn(:,2);
    h3 = Hnn(:,3);
```

```
        lambda = 1/norm(A\h1); %%%% NEW LAMBDA - VERY CONFUSING NOTATION
        r1 = lambda*inv(A)*h1;
        lambda = 1/norm(A\h2); %%%% NEW LAMBDA - VERY CONFUSING NOTATION
        r2 = lambda*inv(A)*h2;
        r3 = cross(r1,r2);
        t(:,i)  = lambda*inv(A)*h3;
        R = [r1, r3, r2]; %%% Not sure why this has to be the case

    % Better method of estimating R: use SVD, set singular values to 1s
    % New Rotation matrix = Rnew = U*V', where R = U*S*V'

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

        disp(['Rotation matrix R for images' num2str(photoSet(i)) '.png'])
        disp(Rnew(:,:,i))
        disp(['Translation vector for images' num2str(photoSet(i)) '.png'])
        disp(t(:,i))

    % Compute reprojection error
        % compute the errors between points in p_correct and points you get
        % by projecting grid corners to the image
        Hnn = eval(['Hn' num2str(i)]);
        Hh = eval(['H' num2str(i)]);
        p_projection = Hnn*wc_allpoints';

        for j=1:length(p_projection)    % Normalize
            p_projection(:,j) = p_projection(:,j) /p_projection(3,j);
        end
        p_projection = p_projection';

        err_xline = [p_correct(:,1,i) p_projection(:,1)];
        err_yline = [p_correct(:,2,i) p_projection(:,2)];

        grid = eval(['grid' num2str(i)]);
        figure(), imshow(grid), title(['Figure4: Grid Point Reprojection Error for images'
        hold on
        plot(p_correct(:,1,i),  p_correct(:,2,i),   'g+')
        plot(p_projection(:,1), p_projection(:,2),  'bo')
        plot(err_xline', err_yline', 'r-')
        hold off

        err_reprojection = [err_xline,err_yline];
        tot_err_reprojection = sum(sqrt((err_xline(:,1)-err_xline(:,2)).^2 + (err_yline(:,
        avg_err_reprojection = tot_err_reprojection/80;

    % Display reprojection error
        disp(['Total Grid Point Reprojection Error (in pixels) for images' num2str(photoSe
        disp(tot_err_reprojection)
```

```
        disp(['Average Grid Point Reprojection Error (in pixels) for images' num2str(photoS
        disp(avg_err_reprojection)

    % Compare reprojection error to part 2 - manual only 4 corner input
        p_projection_old = Hh*wc_allpoints';
        for j=1:length(p_projection_old)
            p_projection_old(:,j) = p_projection_old(:,j) /p_projection_old(3,j);
        end
        p_projection_old = p_projection_old';
        err_xline_old = [p_correct(:,1,i) p_projection_old(:,1)];
        err_yline_old = [p_correct(:,2,i) p_projection_old(:,2)];
        err_reprojection_old = [err_xline_old,err_yline_old];
        tot_err_reprojection_old = sum(sqrt((err_xline_old(:,1)-err_xline_old(:,2)).^2 + (e
        avg_err_reprojection_old = tot_err_reprojection_old/80;

        disp(['Manual Only Grid Point Input - Total Reprojection Error (in pixels) for imag
        disp(tot_err_reprojection_old)
        disp(['Manual Only Grid Point Input - Average Reprojection Error (in pixels) for in
        disp(avg_err_reprojection_old)

    end % for all 4 images
```

Fully automatic checkerboard grid detection

To calibrate a camera with no user manual input (not even the 4 corners), the user can simply specify the number of blocks, as well as the block dimensions (measurements in mm). This establishes the world coordinates of the checkerboard grid. To determine the image coordinates of the checkerboard grid, first run corner detection. Then, search you image for sets of colinear corners. In particular, you are looking for colinear points that correspond to the already known number of checkerboard grid points in the row and column direction. The sets of colinear corners should approximate a rectangular grid, with parallel and perpendicular line segmemnts with a projective transformation applied. In practice, Matlab's function detectCheckerboard() can be used.

```
Recomputed Homography H of 2.png
    1.7456    0.1571   63.4936
    0.0316   -1.6043  414.4186
    0.0000    0.0004    1.0000
Recomputed Homography H of 9.png
    2.2440    0.0727  128.7522
    0.3056   -1.9304  424.4314
    0.0011    0.0003    1.0000
Recomputed Homography H of 12.png
    1.1305    0.0819  101.1891
   -0.2824   -1.4298  394.5842
   -0.0009    0.0003    1.0000
Recomputed Homography H of 20.png
    1.6909    0.5302  125.9504
   -0.0143   -0.7965  277.1073
    0.0000    0.0016    1.0000
Intrinsic Parameters matrix K =
```

```
  723.2192   -0.9648  329.6334
         0  709.8053  234.3268
         0         0    1.0000
Rotation matrix R for images2.png
    0.9999    0.0043    0.0154
    0.0159   -0.1644   -0.9863
    0.0017   -0.9864    0.1645
Translation vector for images2.png
 -151.6065
  104.6242
  412.3610
Total Grid Point Reprojection Error (in pixels) for images2.png
  143.9755
Average Grid Point Reprojection Error (in pixels) for images2.png
    1.7997
Manual Only Grid Point Input - Total Reprojection Error (in pixels) for images2.png
  256.3953
Manual Only Grid Point Input - Average Reprojection Error (in pixels) for images2.png
    3.2049
Rotation matrix R for images9.png
    0.9238    0.3827   -0.0109
    0.0271   -0.0937   -0.9952
    0.3819   -0.9191    0.0970
Translation vector for images9.png
  -98.2373
   94.8461
  354.1330
Total Grid Point Reprojection Error (in pixels) for images9.png
  149.1836
Average Grid Point Reprojection Error (in pixels) for images9.png
    1.8648
Manual Only Grid Point Input - Total Reprojection Error (in pixels) for images9.png
  222.6190
Manual Only Grid Point Input - Average Reprojection Error (in pixels) for images9.png
    2.7827
Rotation matrix R for images12.png
    0.9151   -0.4032   -0.0052
   -0.0567   -0.1157   -0.9917
   -0.3993   -0.9078    0.1288
Translation vector for images12.png
 -148.6400
  106.3454
  471.0204
Total Grid Point Reprojection Error (in pixels) for images12.png
  166.4552
Average Grid Point Reprojection Error (in pixels) for images12.png
    2.0807
Manual Only Grid Point Input - Total Reprojection Error (in pixels) for images12.png
  255.2531
```

```
Manual Only Grid Point Input - Average Reprojection Error (in pixels) for images12.png
    3.1907
Rotation matrix R for images20.png
    1.0000   -0.0044   -0.0076
   -0.0085   -0.7011   -0.7130
    0.0022   -0.7130    0.7011
Translation vector for images20.png
 -120.8294
   25.8653
  429.1528
Total Grid Point Reprojection Error (in pixels) for images20.png
  135.8244
Average Grid Point Reprojection Error (in pixels) for images20.png
    1.6978
Manual Only Grid Point Input - Total Reprojection Error (in pixels) for images20.png
  172.9386
Manual Only Grid Point Input - Average Reprojection Error (in pixels) for images20.png
    2.1617
```

Figure1: Projected grid corners for images2.png

Figure 2 : Harris corners for images2.png

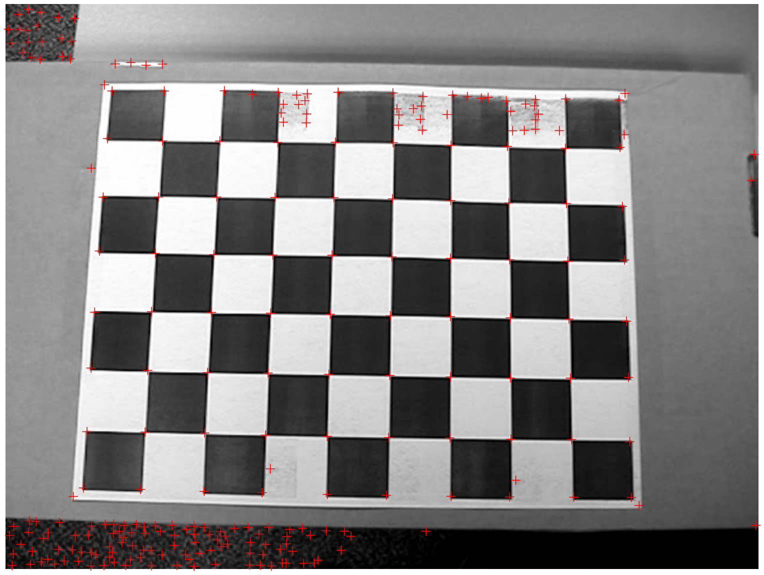


Figure3: Grid Points for images2.png

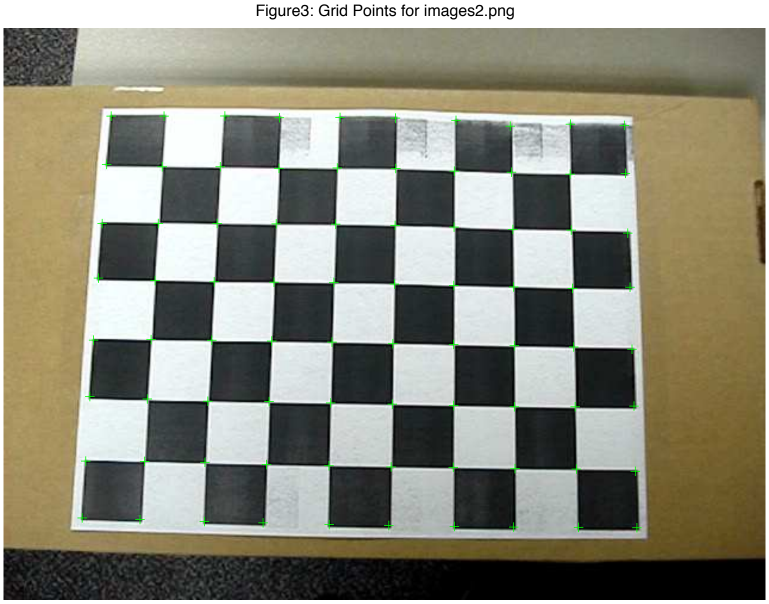


Figure1: Projected grid corners for images9.png

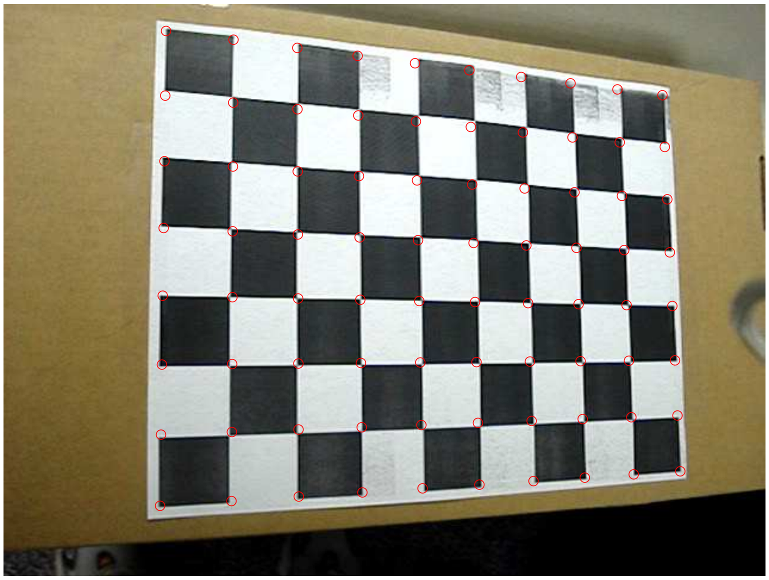


Figure 2 : Harris corners for images9.png

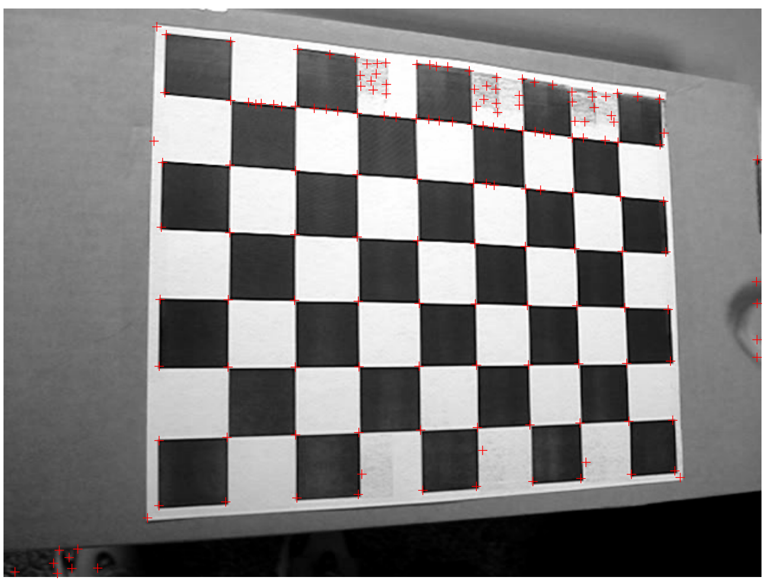


Figure3: Grid Points for images9.png

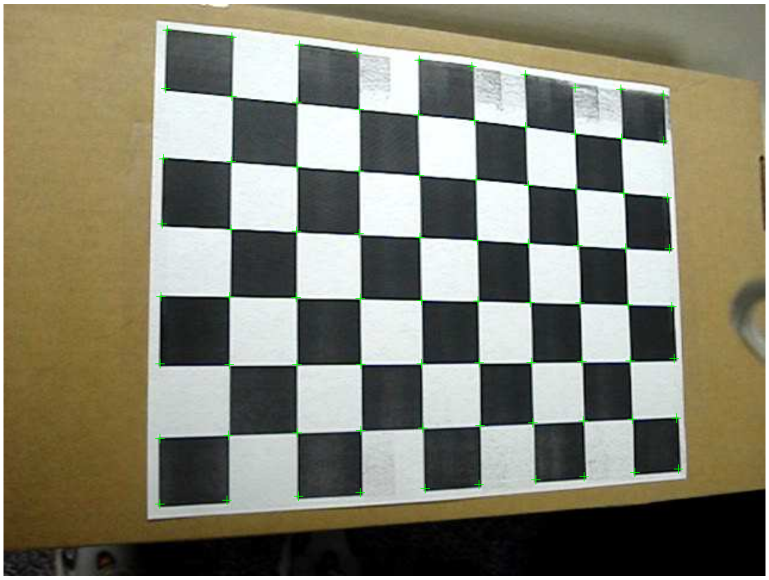


Figure1: Projected grid corners for images12.png

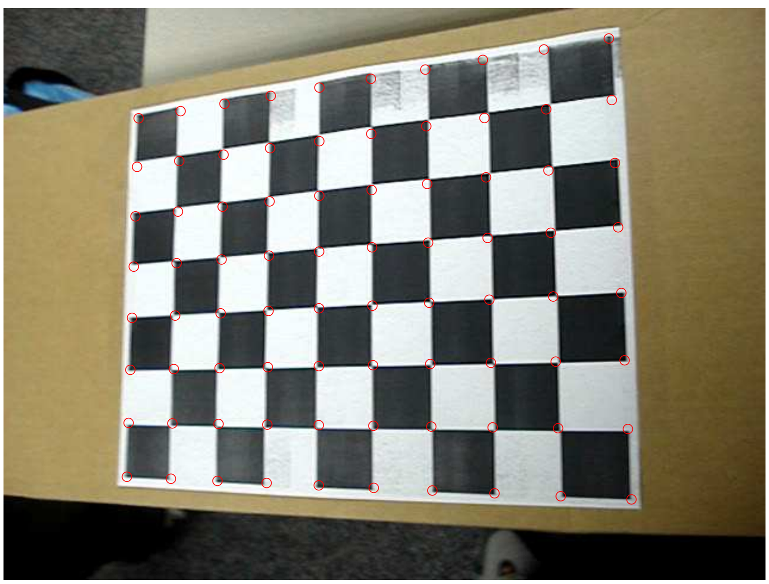


Figure 2 : Harris corners for images12.png

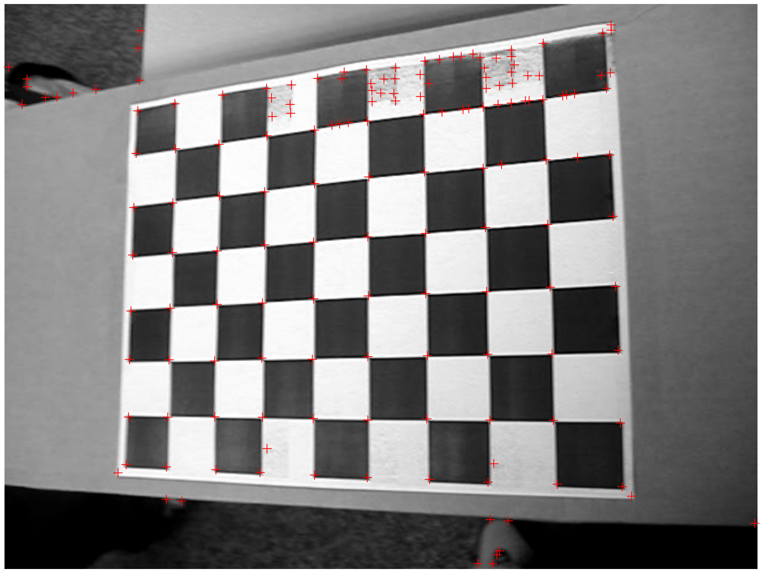


Figure3: Grid Points for images12.png

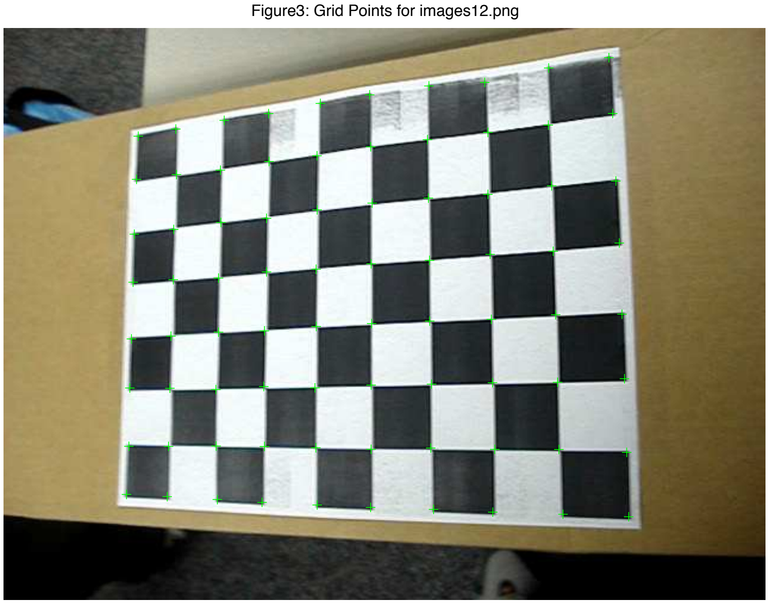


Figure1: Projected grid corners for images20.png

Figure 2 : Harris corners for images20.png

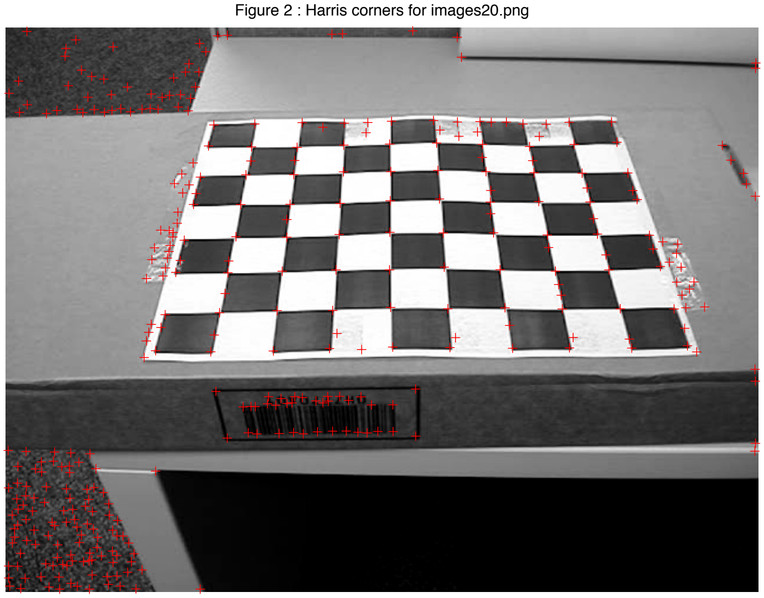


Figure3: Grid Points for images20.png

Figure4: Grid Point Reprojection Error for images2.png

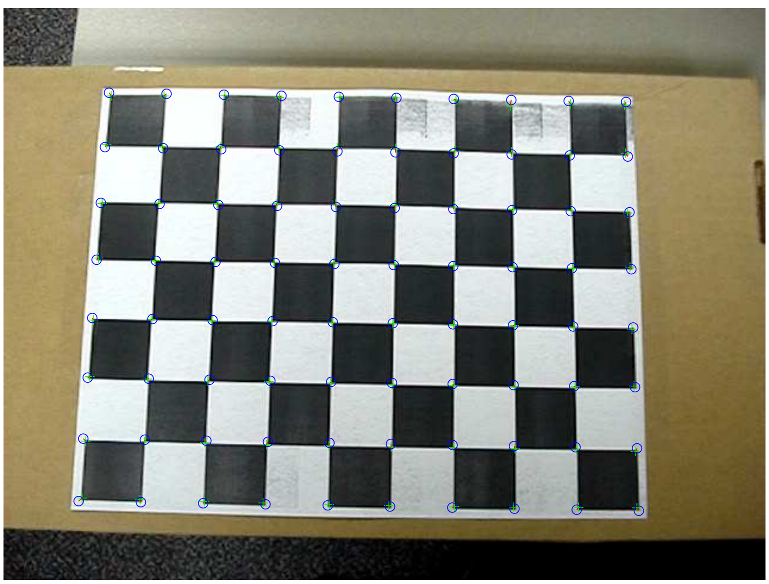


Figure4: Grid Point Reprojection Error for images9.png

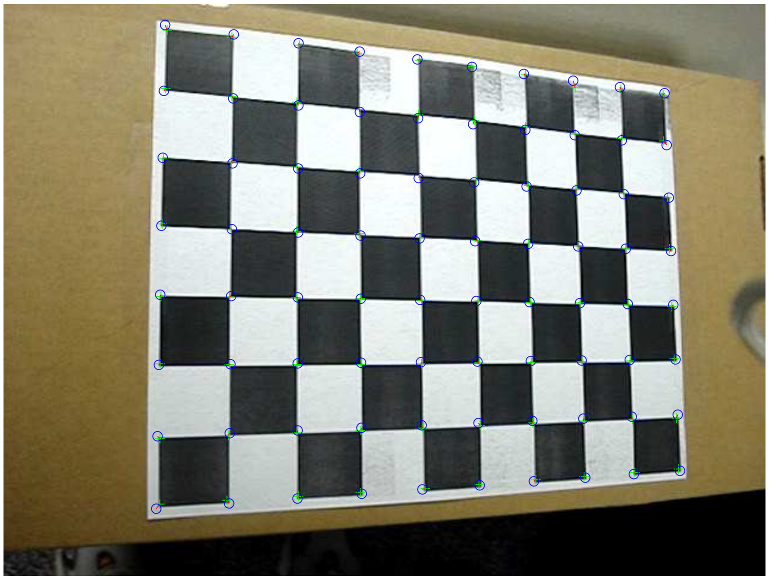


Figure4: Grid Point Reprojection Error for images12.png

Figure4: Grid Point Reprojection Error for images20.png

Part 3 - Augmented Reality 101

3.1 - Augmenting an Image

```
mouseIn = imread('0.png');
scale = [gridHeight*30 NaN];
%scale = [NaN gridWidth*30];
mouseIn = imresize(mouseIn,scale);
[rowsM, colsM, ~] = size(mouseIn);
Hn = [Hn1, Hn2, Hn3, Hn4];

mouseIn = mouseIn + 1;
mouseIn = imrotate(mouseIn,-90);

for k = 1:4 % all checkerboard images

    grid = eval(['grid' num2str(k)]);
    [rowsG, colsG, ~] = size(grid);
    Hnn = eval(['Hn' num2str(k)]);

    mouse_homo = [];
    for i = 1:colsM
    for j = 1:rowsM
        mouse_homo = [mouse_homo; i, j, 1, mouseIn(i,j,1), mouseIn(i,j,2), mouseIn(i,j,3)];
    end
    end

    mouse_homo_tformed = double(mouse_homo(:,1:3))*Hnn';
    mouse_homo_tformed(:,1) = mouse_homo_tformed(:,1)./mouse_homo_tformed(:,3);
    mouse_homo_tformed(:,2) = mouse_homo_tformed(:,2)./mouse_homo_tformed(:,3);
    mouse_homo_tformed = abs(round(mouse_homo_tformed+1));

    mouseNew = zeros(max(mouse_homo_tformed(:,1)), max(mouse_homo_tformed(:,2)), 3);
    for i = 1:length(mouse_homo_tformed)
        r = mouse_homo_tformed(i,2);    %%%
        c = mouse_homo_tformed(i,1);    %%%
        mouseNew(r,c,1) = mouse_homo(i,4);
        mouseNew(r,c,2) = mouse_homo(i,5);
        mouseNew(r,c,3) = mouse_homo(i,6);
    end
    mouseNew = uint8(mouseNew);
%     figure, imshow(mouseNew)

%     mouseNew(mouseNew == 0) = [];
%     L = zeros(size(mouseNew));
%     L(:,:,1) = medfilt2(mouseNew(:,:,1),[5 5]);
%     L(:,:,2) = medfilt2(mouseNew(:,:,2),[5 5]);
%     L(:,:,3) = medfilt2(mouseNew(:,:,3),[5 5]);

%     h = fspecial('average', [3 3]);
```

```
%      L = imfilter(mouseNew,h)*3;
%      figure, imshow(L)

    mouse = mouseNew;

    % tform = projective2d(Hnn');
    % mouse = imwarp(mouse,tform,'FillValues', 255);

    % % Overlay onto image
    bw = (mouse > 0) & (mouse < 255);

    % Pad with zeros
    dif = abs(size(grid)-size(mouse));
    mask_pad = padarray(bw, [dif(1),dif(2)], 'post');
    img_masked_pad = mouse .* uint8(bw);

    new_img = zeros(rowsG,colsG,3);
    for i = 1:rowsG
    for j = 1:colsG
        if mask_pad(i,j,:)
            new_img(i,j,:) = img_masked_pad(i,j,:);
        else
            new_img(i,j,:) = grid(i,j,:);
        end
    end
    end
    figure(), imshow(uint8(new_img))

end
```

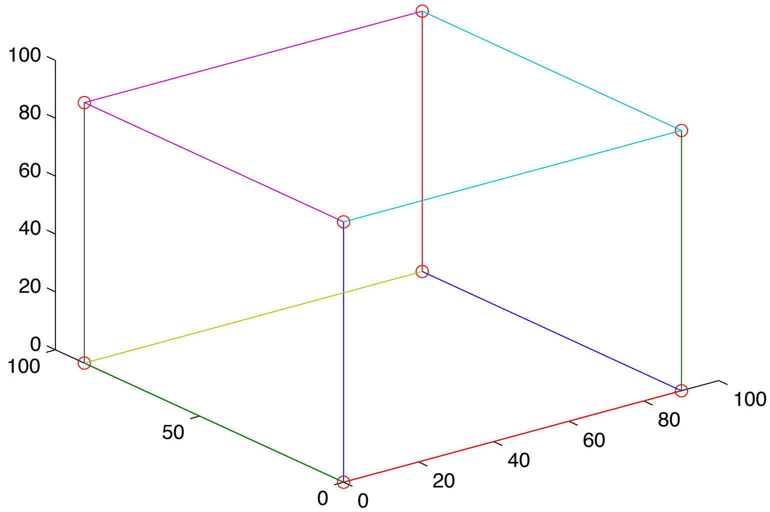
3.2 - Augmenting an Object

```
scale = 30;
cube = [0,          0,          0,          1;
        3*scale,    0,          0,          1;
        0,          3*scale,    0,          1;
        0,          0,          3*scale,    1;
        0,          3*scale,    3*scale,    1;
        3*scale,    0,          3*scale,    1;
        3*scale,    3*scale,    0,          1;
        3*scale,    3*scale,    3*scale,    1;];

newCube(:,1) = cube(:,3);
newCube(:,2) = cube(:,2);
newCube(:,3) = cube(:,1);
newCube(:,4) = cube(:,4);
cube=newCube;

cubeEdges = [cube(1,1:3) cube(2,1:3);
             cube(1,1:3) cube(3,1:3);
             cube(1,1:3) cube(4,1:3);
             cube(2,1:3) cube(6,1:3);
             cube(2,1:3) cube(7,1:3);
             cube(3,1:3) cube(5,1:3);
             cube(3,1:3) cube(7,1:3);
             cube(4,1:3) cube(5,1:3);
             cube(4,1:3) cube(6,1:3);
             cube(8,1:3) cube(5,1:3);
             cube(8,1:3) cube(6,1:3);
             cube(8,1:3) cube(7,1:3);];

cubeEdgesX = [cubeEdges(:,1),cubeEdges(:,4)]';
```

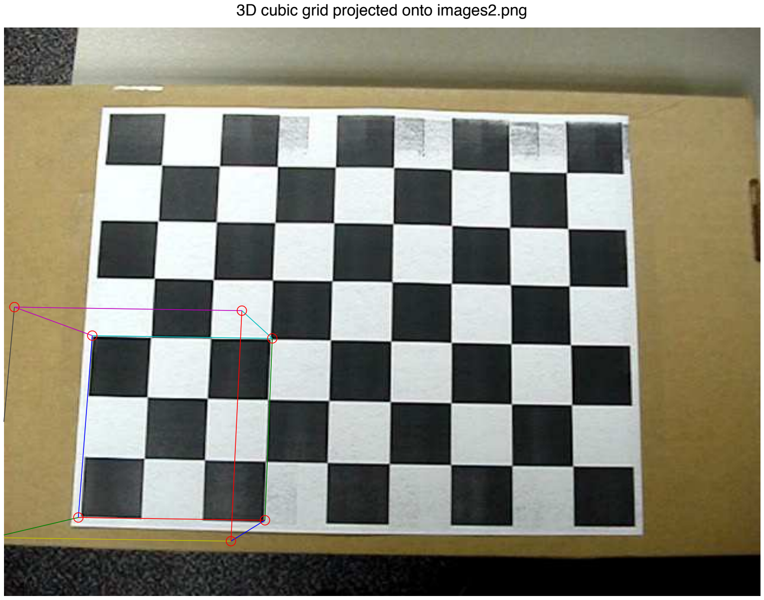
```
cubeEdgesY = [cubeEdges(:,2),cubeEdges(:,5)]';
cubeEdgesZ = [cubeEdges(:,3),cubeEdges(:,6)]';


figure(), plot3(cube(:,1), cube(:,2), cube(:,3), 'ro')
hold on
plot3(cubeEdgesX, cubeEdgesY, cubeEdgesZ)
hold off

extrinsic   = zeros(3,4,4);
Projection  = zeros(3,4,4);

Hn = [Hn1, Hn2, Hn3, Hn4];

for k = 1:4
    grid = eval(['grid' num2str(k)]);
    Hnn = eval(['Hn' num2str(k)]);
    %t(:,k)
    %Rnew(:,:,k)
    %A
    extrinsic(:,:,k) = [Rnew(:,:,k), t(:,k)];
    intrinsic = A;
    Projection(:,:,k) = intrinsic*extrinsic(:,:,k);

    pixels = Projection(:,:,k)*cube';
    pixels = pixels';

%     Hnnew = [Hnn, [0,0,0]'; 0,0,0,1];
%     pixels = Hnn*cube(:,1:3)';
%     pixels = pixels';

    % Normalize by homogenious coordinate
    for i=1:length(pixels)
        pixels(i,:) = pixels(i,:)/pixels(i,3);
    end

    pixelEdges = [pixels(1,1:3) pixels(2,1:3);
                  pixels(1,1:3) pixels(3,1:3);
                  pixels(1,1:3) pixels(4,1:3);
                  pixels(2,1:3) pixels(6,1:3);
                  pixels(2,1:3) pixels(7,1:3);
                  pixels(3,1:3) pixels(5,1:3);
                  pixels(3,1:3) pixels(7,1:3);
                  pixels(4,1:3) pixels(5,1:3);
                  pixels(4,1:3) pixels(6,1:3);
                  pixels(8,1:3) pixels(5,1:3);
                  pixels(8,1:3) pixels(6,1:3);
                  pixels(8,1:3) pixels(7,1:3);];
```

```
    pixelEdgesX = [pixelEdges(:,1),pixelEdges(:,4)]';
    pixelEdgesY = [pixelEdges(:,2),pixelEdges(:,5)]';
    pixelEdgesZ = [pixelEdges(:,3),pixelEdges(:,6)]';

    figure, imshow(grid)
    hold on
    plot(pixelEdgesX, pixelEdgesY)
    plot(pixels(:,1), pixels(:,2), 'ro')
    title(['3D cubic grid projected onto images' num2str(photoSet(k)) '.png'])
    hold off

end
```

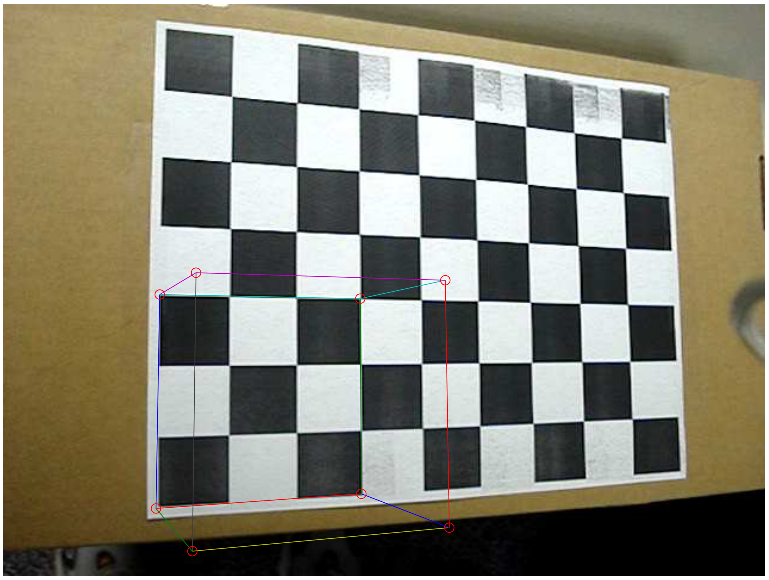
```
Reprojected cube coordinates on images2.png
   63.4936  414.4186    1.0000
   75.2674  260.6822    1.0000
   -8.6060  431.3644    1.0000
  221.3573  416.8171    1.0000
  192.5805  434.4125    1.0000
  227.6583  263.0537    1.0000
    9.4105  236.5551    1.0000
  201.7919  239.5595    1.0000
Reprojected cube coordinates on images9.png
  128.7522  424.4314    1.0000
  131.8714  244.6465    1.0000
  159.3308  460.3117    1.0000
  301.2940  412.0723    1.0000
  375.1122  440.5763    1.0000
  300.3530  248.1139    1.0000
  162.4185  226.3215    1.0000
  371.8755  232.7034    1.0000
Reprojected cube coordinates on images12.png
  101.1891  394.5842    1.0000
  106.1484  259.4707    1.0000
  -14.1361  409.2226    1.0000
  219.2325  399.5005    1.0000
  119.4708  416.7638    1.0000
  221.5283  253.3903    1.0000
   -4.8328  246.1523    1.0000
  125.4480  237.3996    1.0000
Reprojected cube coordinates on images20.png
  125.9504  277.1073    1.0000
  151.1820  179.0929    1.0000
   89.5154  161.9140    1.0000
  277.6402  275.8212    1.0000
  267.8818  160.4634    1.0000
  283.4248  178.0105    1.0000
  123.9001   66.1862    1.0000
  275.9709   64.9930    1.0000
```



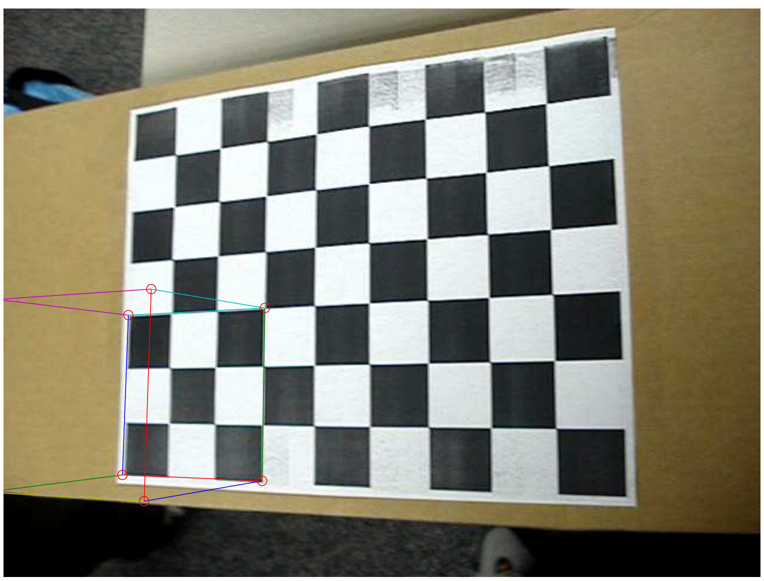
3D cubic grid projected onto images2.png



3D cubic grid projected onto images9.png



3D cubic grid projected onto images12.png



3D cubic grid projected onto images20.png

Extra Credit

Calibrate camera with only 2 images

The intrinsic and extrinsic parameters of a camera can be estimated using only 2 calibration images by treating the system as a stereo calibration system. Zhang describes this in section 2.4.9:

> Let (R_s, t_s) be the rigid transformation between the two cameras such that $(R', t') = (R, t) * (R_s, t_s)$ or more precisely:
> $R' = R * R_s$ and $t' = R * t_s + t$.
> Stereo calibration is then to solve:
> $A, A', k1, k2, k'_1, k'_2, \{(R_i, t_i) | i = 1, ..., n\}$, and (R'_s, t'_s) by minimizing the following function:
> $\sum_{i=1}^{n} \sum_{j=1}^{m} [\delta_{ij} ||m_{ij} - m(A, k_1, k_2, R_i, t_i, M_j||^2 + [\delta'_{ij} ||m'_{ij} - m(A', k'_1, k'_2, R'_i, t'_i, M'_j||^2]$
> subject to $R'_i = R_i R_s$ and $t'_i = R_i t_s + t_i$.

The problem is then solved with nonlinear optimization. To obtain an initial solution, the camera is independently calibrated for each image. SVD is used to compute R_s from $R'_i = R_i R_s$ (for all n point correspondences). t_s is solved with least-squares from $t'_i = R_i t_s + t_i$ (also for all n point correspondences).

The nice part, is that now the number of point correspondences needed to estimate extrinsic parameters is reduced from $12n$ to $6n + 6$. This is the same principle the the fundamental matrix F operates by for stereo camera systems. This is why the 7 point correspondences are needed to estimate the fundamental matrix. The fundamental matrix can be used to derive a projective transformation relating the scene coordinates to world coordinates.