CS283 Assignment 3 Solution

Note: Matlab requires that all local functions in a live script appear at the end of the script. So, your code defining functions must appear at the very bottom of this file, after all other code that makes references to those functions. Also, each function definition must terminate with "end". More information is in the Matlab documentation.

Question 1

For each of the two cases, we answer both parts (a) and (b) together.

(i) The camera matrix P can be decomposed as $P = M[I_{3\times3}| - C]$ where M is a 3×3 matrix, C is a 3×1 vector corresponding to the camera center in 3-D inhomogeneous coordinates, and therefore $[I_{3\times3}| - C]$ is a 3×4 matrix.
Using the above, $x_i = PX_i$ can be written as

$$x_i = PX_i = M[I_{3\times3}| - C]X_i = M\tilde{X}_i, \tag{1}$$

where $\tilde{X}_i = [I_{3\times3}| - C]X_i$ is a 3×1 vector.
As the camera location (or equivalently center C) is known, we only need to estimate the 3×3 matrix M. To do this, note that for each 2D-3D match, we have up to scale that $x_i = M\tilde{X}_i$. This can be alternatively expressed as

$$x_i \times M\tilde{X}_i = 0. \tag{2}$$

Then, writing $M^T = [m_1\ m_2\ m_3]$ and $x_i = [x_{i1}\ x_{i2}\ x_{i3}]$, we have

$$\underbrace{\begin{bmatrix} 0\,0\,0 & x_{i3}\tilde{X}_i^T & -x_{i2}\tilde{X}_i^T \\ x_{i2}\tilde{X}_i^T & -x_{i1}\tilde{X}_i^T & 0\,0\,0 \end{bmatrix}}_{A_i} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \end{bmatrix} = 0, \tag{3}$$

where we do not include the third equation from (2), as it is linearly dependent on the other two. Stacking up equations from all available matches, we get

$$\underbrace{\begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_N \end{bmatrix}}_{\Lambda} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \end{bmatrix} = 0, \tag{4}$$

where Λ is a $2N\times9$ matrix. By adding a constraint on the l_2-norm, to avoid the trivial zero solution, the linear system (4) can be solved, in the least-squares sense, using SVD.
As M has 8 degrees of freedom and each match gives 2 linearly independent equations, we need at least 4 2D-3D correspondences.

(ii) The matrix M can be further decomposed as $M = KR$, where R is the known 3×3 orthogonal rotation matrix (corresponding to the camera's orientation), and K is a 3×3 upper triangular matrix defined as

$$K = \begin{bmatrix} \alpha_x & s & c_x \\ 0 & \alpha_y & c_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{5}$$

As in the previous case, we rewrite $x_i = PX_i$ as $x_i = K\tilde{X}_i$, where

$$\tilde{X}_i = R[I| - C]X_i. \tag{6}$$

For each correspondence, we write $x \times (K\tilde{X}) = 0$ as before. In order to solve for K, we write the two independent constraints provided by each correspondence in matrix form, as

$$\underbrace{\begin{bmatrix} 0 & 0 & 0 & x_{i3}\tilde{X}_{i2} & x_{i3}\tilde{X}_{i3} \\ x_{i2}\tilde{X}_{i1} & x_{i2}\tilde{X}_{i2} & x_{i2}\tilde{X}_{i3} & -x_{i1}\tilde{X}_{i2} & -x_{i1}\tilde{X}_{i3} \end{bmatrix}}_{A_i} \begin{bmatrix} \alpha_x \\ s \\ c_x \\ \alpha_y \\ c_y \end{bmatrix} = \underbrace{\begin{bmatrix} x_{i2}\tilde{X}_{i3} \\ 0 \end{bmatrix}}_{B_i}. \tag{7}$$

Stacking these up we get,

$$\underbrace{\begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_N \end{bmatrix}}_{\Lambda} \begin{bmatrix} \alpha_x \\ s \\ c_x \\ \alpha_y \\ c_y \end{bmatrix} = \underbrace{\begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_N \end{bmatrix}}_{\Gamma}. \tag{8}$$

Then, as we have seen in Assignment One, the least-squares solution for the vector of elements of K is given by

$$K = (\Lambda^T\Lambda)^{-1}\Lambda^T\Gamma. \tag{9}$$

Note, of course, that the inverse here is used only for notational purposes. The solution is usually calculated fast, stably and accurately by using an appropriate matrix factorization.

As we have 5 degrees of freedom, we need 5 linearly independent equations for finding K. As each correspondence gives us 2 such equations, we need 2 and a "half" points, where by "half" we mean a third correspondence from which we use just one of the two equations it provides.

Question 2

The homogeneous coordinates of a point in the world and the corresponding point in the image I_0 are related by $x_0 = P_0X$. Let x_1 be the homogenous coordinate of the corresponding point in the image I_1. We want to find the matrix P' such that $x_1 = P'X$.

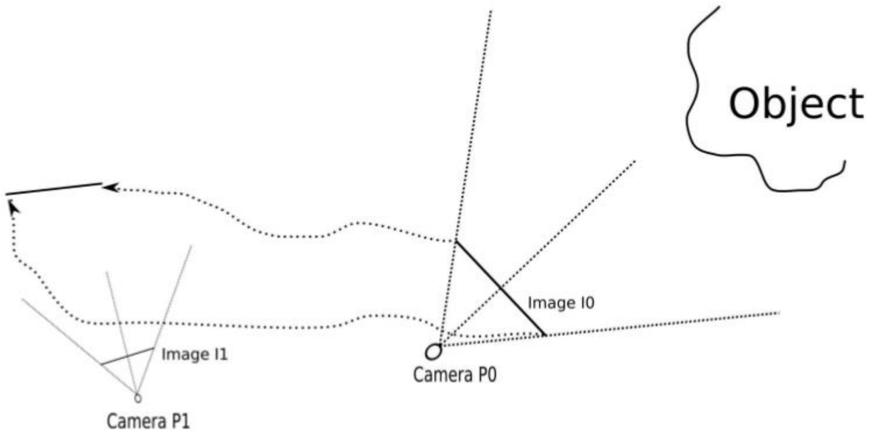


Figure 1: Three views of the 3D reconstructed model.

To do this, we first need to realize that I_1 is not an image of the P_0 camera's sensor plane but rather an image of the printed/developed picture generated from the camera. Therefore, even though you may tempted to backproject the co-ordinate x_0 to find the 3D world co-ordinate to a ray and find its intersection with the sensor plane and then apply P_1 to that, this would be incorrect.

Consider though that the image I_0 will be flat, that is, a plane, and therefore the image I_1 is a projection of that plane. Therefore, x_1 and x_0 can be related by a 3×3 *non-singular* matrix H corresponding to a homography, which will be defined by the camera P_1 and the location of the printed image I_0. Since the latter is unknown, H can be any general 3×3 invertible matrix, and therefore $x_1 = Hx_0 = HP_0X$. We therefore have that $P' = HP_0$.

Let C be the camera center corresponding to P_0. Therefore, we have $P_0C = 0$. But, this also implies that $P'C = HP_0C = H\ 0 = 0$. Also, since H is an invertible 3×3 matrix, $x = C$ is the only vector for which $P'x = 0$. Therefore, C is also the apparent camera center of P'.

A portrait is essentially a captured image of a person, where the subject is usually looking at the photographer/painter, that is, the camera center. When we look at the picture, we are essentially capturing an image of that image, and as shown above, the apparent camera center does not change.

Therefore, no matter where we are in the room, the image will appear to be looking at the camera center, that is, us.

Let $P_0 = M_0[I| - C]$, and therefore $P' = HM_0[I| - C] = M'[I| - C]$. The matrix $M' = HM_0$ encodes all the other parameters of the "apparent camera" P'.

For orientation, the third row of the M matrix is parallel to the principal axis, and since H is a general 3×3 matrix, the third row of M_0 and M', and therefore the orientations of P_0 and P' can be different.

For intrinsic parameters, let $M_0 = K_0R_0$, where the upper-triangular matrix K_0 defines the intrinsic parameters of the camera. Now, if H were an upper triangular matrix as well, HK_0 would be upper-triangular too and therefore $K' = HK_0$ would describe the intrinsic parameters of P'. By choosing arbitrary values for this H, we can affect every non-zero entry of K.

Question 3

(a)

The code for this part is very simlar to that for function `getH` from Assignment 2. Also, we could use directly use function `getT` from that assignment to do the normalization, however here we choose to re-implement it. See the function `P = getCamera(X3, X2)` below.

(b)

Using the function `P = getCamera(X3, X2)` from part a, we can use the following code to estimate the camera matrices based on the given data.

```
load ./data/corners.mat
Pl = getCamera(corners_3D, leftpts);
Pr = getCamera(corners_3D, rightpts);
save ./data/StereoCameras.mat Pl Pr
```

(c)

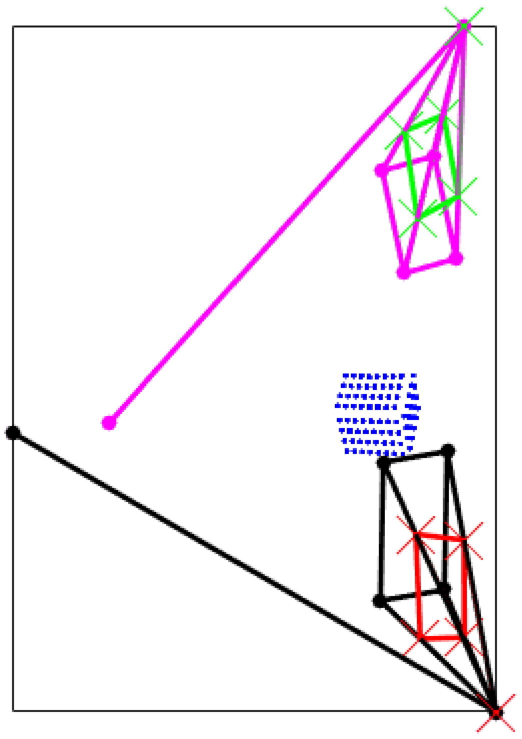
As suggested in the hints section, one way to visualize the camera is by backprojecting the four corners of the image to four rays, taking one point on each ray, and then drawing the pentahedron connecting those four points and the camera center. The question then is, which points on the backprojected rays should we use?

One strategy is to select points so that the distance between the camera center and any of the four points is the same. Though this is an acceptable strategy for this homework assignment, it is worth noting that the plane formed by the four backprojected points in this case is not parallel to the image plane. Put another way, the vector perpendicular to this plane and passing through the camera center is not the principal axis. Put yet another way, we cannot use the plotted plane to determine which way the camera is “looking” at. This strategy is used by the function `viscam(P, sz, D, clr)` below.

An alternative strategy is to select points on the four backprojected rays which all lie on the same plane parallel to the image plane. This strategy is used by the function `viscam_principal(P, sz, D, clr)` (which in addition to the tetrahedron plots the principal axis itself). In the function, line pV `= - det(P(1:3, 1:3)) *` pV uses the equation from page 161, middle of the page (right before the beginning of 6.2.2) of [2], with one difference: we multiply by the negative of the determinant of the matrix M. The reason for this is that, in MATLAB, the image plane uses a left-handed coordinate system (x axis going horizontally from left to right, y axis going vertically from top to bottom). In [2], the coordinate system of the image plane is right-handed (x axis going horizontally from left to right, y axis going vertically from bottom to top). By following the reasoning in page 161 of [2], it is easy to see that this change results in the negative sign.

We use the following script to plot the results produced by both strategies.

```
load ./data/corners.mat
load ./data/StereoCameras.mat
figure;
hold on
plot3(corners_3D(:,1), corners_3D(:,2), corners_3D(:,3), '.b');
viscam(Pl, size(imread('./data/calib_left.bmp')), 3, 'r');
viscam(Pr, size(imread('./data/calib_right.bmp')), 3, 'g');
viscam_principal(Pl, size(imread('./data/calib_left.bmp')), 3, 'k');
viscam_principal(Pr, size(imread('./data/calib_right.bmp')), 3, 'm');
hold off
axis tight; axis equal;
set(gca, 'xtick', [], 'ytick', [], 'ztick', []);
box on;
```



```
Pl
```

```
Pl =

   -7.8078  316.4370   36.4066  206.8591
 -528.1388   -8.3251   24.6380  114.1650
   -0.0830    0.0480   -0.0602    0.4848
```

-

```
Pr
```

```
Pr =

  151.3380 -252.1552   66.3111 -129.7924
  451.3765   76.0416  -24.6274 -114.8921
    0.0443    0.0497    0.0489   -0.4860
```

-

The resulting plot is shown in above figure. The tetrahedra marked with X pointers correspond to the first strategy (function `viscam(P, sz, D, clr)`); those marked with dots correspond to the second strategy (function `viscam_principal(P, sz, D, clr)`). The lone vectors correspond to the principal axes for each camera. As expected, the plane produced by the second strategy is perpendicular to the principal axis, whereas the plane by the first strategy is not. We can now also use the principal axis to determine where the camera is looking. We see that both cameras appear to be looking at a point very far from the 3D calibration object. In other words, the calibration object does not appear to be well “centered” on the image plane. This indicates that the images calib_left.bmp and calib_right.bmp must have be cropped, and that the calibration object was near the boundary of the original image in both cases.

Question 4

(a)

Assuming that all coordinates are homogeneous, for two 2-D observations x and x' of a 3-D point X by cameras P and P', we have

$$x \times PX = 0, \tag{10}$$
$$x' \times P'X = 0. \tag{11}$$

Denoting $x = [x_1\ x_2\ x_3]^T$, $x' = [x'_1\ x'_2\ x'_3]^T$, $P = [p_1\ p_2\ p_3]^T$ and $P' = [p'_1\ p'_2\ p'_3]^T$, we can rewrite the above equations as

$$\begin{bmatrix} x_2 p_3^T - x_3 p_2^T \\ x_3 p_1^T - x_1 p_3^T \\ x'_2 p_3'^T - x'_3 p_2'^T \\ x'_3 p_1'^T - x'_1 p_3'^T \end{bmatrix} X = 0. \tag{12}$$

This linear system can be solved, under an l_2-norm constraint on X, using SVD, as we have seen in class and previous assignments (see also [1]). Finally, note that equations (10) and (11) provide us with two more relations analogous to those included in the rows of the system matrix. However, these additional equations can be derived as linear combinations of the other four rows of the system matrix—in other words, of a total of six constraints, only four are linearly independent. For this reason, we omit them from the system matrix.

(b)

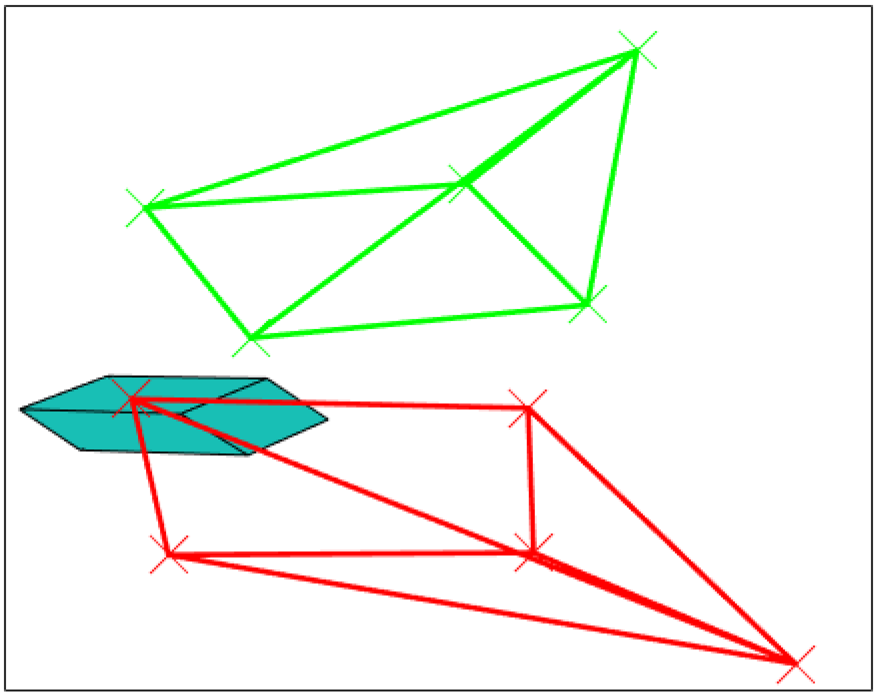
Function `X = triangulate(x1, x2, P1, P2)` is one possible implementation for triangulating two points given the corresponding camera matrices, using the linear system of (12).

(c)

We use the following script to plot the results.

```
%
% Visualize results
%
load ./data/StereoCameras.mat
load ./data/KleenexPoints.mat

img_l = rgb2gray(imread('./data/kleenex_left.bmp'));
img_r = rgb2gray(imread('./data/kleenex_right.bmp'));

% preallocate matrix for 3D coordinates
X = zeros(8,3);

% triangulate provided 2D points to find their 3D matches
for iter = 1:7,
 xpt = triangulate(lpts(iter, :), rpts(iter, :), Pl, Pr);
 X(iter, :) = transpose(xpt(1:3) / xpt(end));
end;

% Compute 8th point by symmetry
```

```
X(8, :) = 2 * mean(X(2:7, :), 1) - X(1, :);

% Plot box with cameras
figure;
hold on
viscam(Pl, size(img_l), 5, 'r');
viscam(Pr, size(img_r), 5, 'g');
surf([X(1, 1) X(2, 1); X(3, 1) X(5, 1)], [X(1, 2) X(2, 2); X(3, 2) X(5, 2)], ...
[X(1, 3) X(2, 3); X(3, 3) X(5, 3)], 1);

surf([X(1, 1) X(3, 1); X(4, 1) X(6, 1)], [X(1, 2) X(3, 2); X(4, 2) X(6, 2)], ...
[X(1, 3) X(3, 3); X(4, 3) X(6, 3)],1);

surf([X(1, 1) X(4, 1); X(2, 1) X(7, 1)], [X(1, 2) X(4, 2); X(2, 2) X(7, 2)], ...
[X(1, 3) X(4, 3); X(2, 3) X(7, 3)], 1);

surf([X(4, 1) X(7, 1); X(6, 1) X(8, 1)], [X(4, 2) X(7, 2); X(6, 2) X(8, 2)], ...
[X(4, 3) X(7, 3); X(6, 3) X(8, 3)], 1);

surf([X(2, 1) X(5, 1); X(7, 1) X(8, 1)], [X(2, 2) X(5, 2); X(7, 2) X(8, 2)], ...
[X(2, 3) X(5, 3); X(7, 3) X(8, 3)], 1);

surf([X(3, 1) X(5, 1); X(6, 1) X(8, 1)], [X(3, 2) X(5, 2); X(6, 2) X(8, 2)], ...
[X(3, 3) X(5, 3); X(6, 3) X(8, 3)], 1);

hold off
set(gca, 'xtick', [], 'ytick', [], 'ztick', []);
box on;
```



```
X
```

```
X =
   -0.0374   -0.0834   -0.0810
    0.0248   -0.2947   -0.3537
    0.0421    0.0974   -0.3322
   -0.1859   -0.0598   -0.2835
    0.0988   -0.1128   -0.5840
   -0.1048    0.1096   -0.5284
   -0.1295   -0.2690   -0.5452
   -0.0475   -0.0931   -0.7947
```

-

(d)

We take the following approach for calculating the volume of the box (note that there are many alternatives): we find the lengths of the three sides as the distances of the corresponding corners in the previous part, and their product will be the volume. This gives us the volume to be approximately $0.35 \times 0.32 \times 0.25 = 0.028$ units.
From the calibration array, we find that the squares are roughly 0.047×0.3 in size in terms of

world units. This gives us,

$$\text{Volume} = 0.028 \times \left(\frac{2\text{cm}}{0.047}\right)^3 = 2158\text{cm}^3. \qquad (13)$$

(Due to the large calibration variance, we accept all results from 1700 to 2500.)

The shape of the checkerboard pattern is actually not square, so that we accept all answers that has appropriate assumptions and compute correctly.

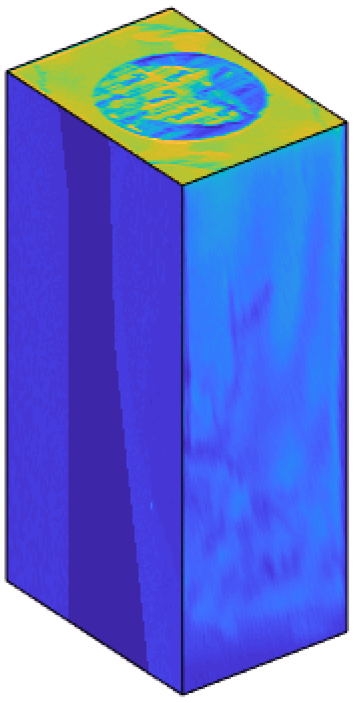
Bonus Question

We use `Iout = getFacet(Iin, H, facet, a, b, c)`, a modified version of function `applyH` from Assignment 2, that only applies the calculated homography to the area required for one facet. This is shown below.

Based on the function `getFacet`, and also using `getH` from assignment 2, we use the following script to create the kleenex.

```
%
% Create and show kleenex box.
%
%%
load ./data/KleenexPoints.mat
Kl = rgb2gray(imread('./data/kleenex_left.bmp'));
Kr = rgb2gray(imread('./data/kleenex_right.bmp'));

%% Dimensions of kleenex box
a = sqrt(sum((X(1,:)-X(2,:)).^2));
b = sqrt(sum((X(1,:)-X(4,:)).^2));
c = sqrt(sum((X(1,:)-X(8,:)).^2));

%% Estimate upper facet from left image
X1 = lpts([1 2 5 3],:);
X2 = [0 0;0 a;b a;b 0];
H1 = getH(X1,X2);
facet1 = getFacet(Kl,H1,1,a,b,c);
```

```matlab
%% Estimate front facet from left image
X1 = lpts([4 6 3 1],:);
X2 = [0 0;b 0;b c;0 c];
H2 = getH(X1,X2);
facet2 = getFacet(Kl,H2,2,a,b,c);

%% Estimate side (right) facet from right image
X1 = rpts([7 4 1 2],:);
X2 = [0 0;a 0;a c;0 c];
H3 = getH(X1,X2);
facet3 = getFacet(Kr,H3,3,a,b,c);

%% Plot various surfaces
figure;
s1 = surf([0 b; 0 b], [0 0; a a], [c c; c c]); hold on;
set(s1, 'facecolor', 'texturemap', 'cdata', im2uint8(facet1));
s2 = surf([0 b; 0 b], [a a; a a], [0 0; c c]);
set(s2, 'facecolor', 'texturemap', 'cdata', im2uint8(facet2));
s3 = surf([b b; b b], [a 0; a 0], [0 0; c c]);
set(s3, 'facecolor', 'texturemap', 'cdata', im2uint8(facet3));
s4 = surf([0 0; 0 0], [a 0; a 0], [0 0; c c]);
set(s4, 'facecolor', 'texturemap', 'cdata', im2uint8(facet3));
s5 = surf([0 b; 0 b], [0 0; 0 0], [0 0; c c]);
set(s5, 'facecolor', 'texturemap', 'cdata', im2uint8(facet2));
s6 = surf([0 b; 0 b], [0 0; a a], [0 0; 0 0]);
set(s6, 'facecolor', 'texturemap', 'cdata', im2uint8(facet1));
axis ij;
axis image;
axis off;
```



Local Functions

```
function P = getCamera(X3, X2)
    % Problem 3(a)
    N = size(X3, 1);
    c3 = mean(X3, 1);
    c2 = mean(X2, 1);
    d3 = mean(sqrt(sum((X3 - repmat(c3, [N 1])) .^ 2, 2)));
    d2 = mean(sqrt(sum((X2 - repmat(c2, [N 1])) .^ 2, 2)));
    T2 = [[eye(2) -transpose(c2)] ./ d2; 0 0 1];
    T3 = [[eye(3) -transpose(c3)] ./ d3; 0 0 0 1];

    X3h = [X3 ones(N, 1)];
    X2h = [X2 ones(N, 1)];
    X3h = transpose(T3 * transpose(X3h));
    X2h = transpose(T2 * transpose(X2h));

    % Estimate P
    z = zeros(N, 4);
    x3x = repmat(X2h(:, 3), [1 4]) .* X3h;
    x2x = repmat(X2h(:, 2), [1 4]) .* X3h;
    x1x = repmat(X2h(:, 1), [1 4]) .* X3h;
    A = [z -x3x x2x; x3x z -x1x];

    [~, ~, V] = svd(A,0);
    P = transpose(reshape(V(:, end), [4 3]));
    P = T2 \ P * T3;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Problem 3(c)
% one camera visualization method
function viscam(P, sz, D, clr)
    % Compute center
    [~, ~, v] = svd(P);
    V = transpose(v(1:3, end)) / v(4,end);

    % Back-project corners to rays
    S = [1 1 1; 1 sz(1) 1; sz(2) sz(1) 1;sz(2) 1 1];
    S = transpose(P \ transpose(S));
    S = S(:, 1:3) ./ repmat(S(:, 4), [1 3]);

    S = S - repmat(V, [4 1]);
    S = repmat(V, [4 1]) + D * S ./ repmat(sqrt(sum(S .^ 2, 2)), [1 3]);

    % Draw pentahedron (assumed called with hold on)
    plot3(S([1:4 1], 1), S([1:4 1], 2), S([1:4 1], 3), ['.-' clr], 'linewidth', 2);
    plot3(S([1:4 1], 1), S([1:4 1], 2), S([1:4 1], 3), ['x' clr], 'markersize', 20);

    for iter = 1:4
        plot3([V(1) S(iter, 1)], [V(2) S(iter, 2)], [V(3) S(iter, 3)], ['.-' clr], 'linewidth'
        plot3([V(1) S(iter, 1)], [V(2) S(iter, 2)], [V(3) S(iter, 3)], ['x' clr], 'markersize'
    end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Problem 3(c)
% the other camera visualization method
function viscam_principal(P, sz, D, clr)
    % Compute center
    [~, ~, v] = svd(P);
    C = transpose(v(1:3, end)) / v(4,end);

    % Compute principal axis
    pV = P(3, 1:3)';
    pV = - det(P(1:3, 1:3)) * pV;
    pV = pV' / norm(pV);

    % Find point on principal axis
    pP = C + pV * D;

    % Back-project corners to ray vectors
    S = [1 1 1; 1 sz(1) 1; sz(2) sz(1) 1;sz(2) 1 1];
    S = transpose(P \ transpose(S));
    S = S(:, 1:3) ./ repmat(S(:, 4), [1 3]);
    S = S - repmat(C, [4 1]);
    S = S ./ repmat(sqrt(sum(S .^ 2, 2)), [1 3]);

    % Compute distance in principal axis direction, and find points on rays
    ds =  D ./ (S * pV');
    rP = repmat(C, [4 1]) + repmat(ds, [1 3]) .* S;

    % Draw pentahedron (assumed called with hold on)
    plot3(rP([1:4 1], 1), rP([1:4 1], 2), rP([1:4 1], 3), ['-' clr], 'linewidth', 2);
    plot3(rP([1:4 1], 1), rP([1:4 1], 2), rP([1:4 1], 3), ['.' clr], 'markersize', 20);

    for iter = 1:4
     plot3([C(1) rP(iter, 1)], [C(2) rP(iter, 2)], [C(3) rP(iter, 3)], ['-' clr], 'linewidth',
     plot3([C(1) rP(iter, 1)], [C(2) rP(iter, 2)], [C(3) rP(iter, 3)], ['.' clr], 'markersize'
    end

    % Draw principal axis
    plot3([C(1) pP(1)], [C(2) pP(2)], [C(3) pP(3)], ['-' clr], 'linewidth', 2);
    plot3([C(1) pP(1)], [C(2) pP(2)], [C(3) pP(3)], ['.' clr], 'markersize', 20);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Problem 4(a)
function X = triangulate(x1, x2, P1, P2)
    % create linear system matrix
    A = [x1(1) * P1(3, :) - P1(1, :);
     x1(2) * P1(3, :) - P1(2, :);
     x2(1) * P2(3, :) - P2(1, :);
     x2(2) * P2(3, :) - P2(2, :)];

    % solve using SVD
    [~, ~, v] = svd(A);
    X = v(:, end);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Bonus
%
% Modified applyH to create facet.
%
function Iout = getFacet(Iin, H, facet, a, b, c)
```

```
    Iin = im2double(Iin);

    %% Resolution
    [M N] = size(Iin);
    num_x_pts = ceil(3 * N / 2);
    num_y_pts = ceil(3 * M / 2);

    % create regularly-space grid of (x,y)-pixel coordinates for selected facet
    if (facet == 1)
     [x,y]=meshgrid(linspace(0,b,num_x_pts), linspace(0,a,num_y_pts));
    elseif (facet == 2)
     [x,y]=meshgrid(linspace(0,b,num_x_pts), linspace(0,c,num_y_pts));
    elseif (facet == 3)
     [x,y]=meshgrid(linspace(0,a,num_x_pts), linspace(0,c,num_y_pts));
    end;

    % reshape them so that a homography can be applied to all points in parallel
    X = transpose([x(:) y(:)]);
    Xh = hom2in(H ^ (-1) * in2hom(X));

    % [Apply a homography to homogeneous coordinates corresponding to X. ] %
    % [Compute inhomogeneous coordinates of mapped points. ] %
    % [Save result in Nx2 matrix named Xh. ] %
    % interpolate I to get intensity values at image points Xh
    Iout = interp2(Iin, Xh(1, :), Xh(2, :));

    % reshape intensity vector into image with correct height and width
    Iout = reshape(Iout, [num_y_pts, num_x_pts]);

    % Points in Xh that are outside the boundaries of the image are assigned
    % value NaN, which means not a number. The final step is to
    % set the intensities at these points to zero.
    Iout(isnan(Iout)) = 0;
end

function Hcoords = in2hom(Icoords)
    Hcoords = [Icoords; ones(1, size(Icoords, 2))];
end

function Icoords = hom2in(Hcoords)
    Icoords = Hcoords(1:2, :) ./ repmat(Hcoords(3, :), [2 1]);
end

%
% Estimate a homography from corresponding pairs of points
%
function H = getH(X1, X2)
    if ((ndims(X1) ~= 2) || (ndims(X2) ~=2) || ~isempty(find(size(X1) ~= size(X2), 1))...
      || (size(X1, 2) ~= 2) || (size(X1, 1) == 0))
     error('Input must be two Nx2 matrices.');
    end

    N = size(X1, 1);

    %% Normalize points using getT
    T1 = getT(X1);
    T2 = getT(X2);

    X1h = T1 * in2hom(transpose(X1));
    X2h = T2 * in2hom(transpose(X2));
    %% Construct matrix A of the homogeneous linear system
    A = zeros(3 * N, 9);
    for iter = 1:N,
     A((iter * 3 - 2): (iter * 3), :) = ...
     [zeros(1, 3) -X2h(3, iter) * transpose(X1h(:, iter)) X2h(2, iter) * transpose(X1h(:, iter
     X2h(3, iter) * transpose(X1h(:, iter)) zeros(1, 3) -X2h(1, iter) * transpose(X1h(:, iter)
     -X2h(2, iter) * transpose(X1h(:, iter))  X2h(1, iter) * transpose(X1h(:, iter)) zeros(1,
    end;

    %% Compute SVD and take last vector of V
    [~, ~, V] = svd(A);
    h = V(:, 9);
    Htild = transpose(reshape(h, [3 3]));

    %% "Remove" normalizations
    H = T2 \ Htild * T1;
end

%
% Center and scale samples
%
function T = getT(X)
    if ((ndims(X) ~= 2) || (size(X, 2) ~= 2) || (size(X, 1) == 0))
     error('Input matrix must be a Nx2 matrix.');
    end

    X = transpose(X);
    %% Calculate tx, ty and s

    Xbar = mean(X(1, :));
    Ybar = mean(X(2, :));

    std = mean(sqrt((X(1, :) - Xbar) .^ 2 + (X(2, :) - Ybar) .^ 2));
    if std > 0
     s = sqrt(2) / std;
    else
     s = 1;
     warning('All points coincide.');
    end

    tx = - s * Xbar;
    ty = - s * Ybar;

    %% Form and return T
    T = [s 0 tx;
      0 s ty;
      0 0 1];
end
```