CS 231A Computer Vision (Winter 2016)
Problem Set 2

Solution Set

April 25th 2016 11:59pm

1 Fundamental Matrix (20 points)

In this question, you will explore some properties of fundamental matrix and derive a minimal parameterization for it.

a Show that two 3×4 camera matrices M and M' can always be reduced to the following canonical forms by an appropriate projective transformation in 3D space, which is represented by a 4×4 matrix H. Here, we assume $e_3^T(-A'A^{-1}b+b') \neq 0$, where $e_3 = (0,0,1)^T$, $M = [A, b]$ and $M' = [A', b']$.
Note: You don't have to show the explicit form of H for the proof. **[10 points]**
Hint: The camera matrix has rank 3. Block matrix multiplication may be helpful. If you construct a projective transformation matrix H_0 that reduces M to $\hat{M}$, (i.e., $\hat{M} = MH_0$) can a H_1 be constructed such that not only does it not affect the reduction of M to $\hat{M}$ (i.e., $\hat{M} = MH_0H_1$), but it also reduces M' to $\hat{M}'$? (i.e., $\hat{M}' = M'H_0H_1$)

$$\hat{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \text{ and } \hat{M}' = \begin{bmatrix} a_{11} & a_{12} & a_{13} & b_1 \\ a_{21} & a_{22} & a_{23} & b_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

b Given a 4×4 matrix H representing a projective transformation in 3D space, prove that the fundamental matrices corresponding to the two pairs of camera matrices (M, M') and $(MH, M'H)$ are the same. **[5 points]**
(Hint: Think about point correspondences)

c Using the conclusions from (a) and (b), derive the fundamental matrix F of the camera pair (M, M') using a_{11}, a_{12}, a_{13}, a_{21}, a_{22}, a_{23}, b_1, b_2. Then use the fact that F is only defined up to a scale factor to construct a seven-parameter expression for F. *(Hint: The fundamental matrix corresponding to a pair of camera matrices $M = [I \mid 0]$ and $M' = [A \mid b]$ is equal to $[b]_\times A$.)* **[5 points]**

Solution:

(a) Since the camera matrix M has rank 3, we can always first find 4×4 matrix H_0

$$H_0 = \begin{bmatrix} A^{-1} & -A^{-1}b \\ 0 & 1 \end{bmatrix}.$$

such that

$$MH_0 = [I, 0] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \hat{M}$$

where I is the 3x3 Identity matrix.

Applying the same transformation to M' we will get a new matrix,

$$M'H_0 = [A'A^{-1}, -A'A^{-1}b + b'] = \begin{bmatrix} m'_{11} & m'_{12} & m'_{13} & m'_{14} \\ m'_{21} & m'_{22} & m'_{23} & m'_{24} \\ m'_{31} & m'_{32} & m'_{33} & m'_{34} \end{bmatrix}.$$

As $e_3^T(-A'A^{-1}b + b') \neq 0$, $m'_{34} \neq 0$.

We also know that we can multiply MH_0 by another matrix H_1 of the form

$$H_1 = \begin{bmatrix} I & 0 \\ c & d \end{bmatrix}$$

without changing the result, i.e.

$$(MH_0)H_1 = [I, 0] \begin{bmatrix} I & 0 \\ c & d \end{bmatrix} = [I, 0] = \hat{M}$$

We therefore have the flexibility to choose c and d such that $M'H_0H_1 = \hat{M}'$. We can thus choose H_1 to be

$$H_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\frac{m'_{31}}{m'_{34}} & -\frac{m'_{32}}{m'_{34}} & -\frac{m'_{33}}{m'_{34}} & \frac{1}{m'_{34}} \end{bmatrix}$$

so that we have

$$MH_0H_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \text{ and } M'H_0H_1 = \begin{bmatrix} m'_{11} - \frac{m'_{14}m'_{31}}{m'_{34}} & m'_{12} - \frac{m'_{14}m'_{32}}{m'_{34}} & m'_{13} - \frac{m'_{14}m'_{33}}{m'_{34}} & \frac{m'_{14}}{m'_{34}} \\ m'_{21} - \frac{m'_{24}m'_{31}}{m'_{34}} & m'_{22} - \frac{m'_{24}m'_{32}}{m'_{34}} & m'_{23} - \frac{m'_{24}m'_{33}}{m'_{34}} & \frac{m'_{24}}{m'_{34}} \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

So a projection transformation matrix $H = H_0H_1$ will reduce M and M' to the canonical forms.

(b) Observe that $MX = (MH)(H^{-1}\mathbf{X})$, and similarly for M'. Thus if $\mathbf{x}$ and $\mathbf{x}'$ are matched points with respect to the pair of cameras (M, M'), corresponding to a 3D point $\mathbf{X}$, then they are also matched points with respect to the pair of cameras $(MH, M'H)$, corresponding to the point $H^{-1}\mathbf{X}$.

(c) From the conclusion of (b), the fundamental matrix of the camera pair (M, M') is the same as the fundamental matrix of the camera pair $(\hat{M}, \hat{M}')$, which is $[\hat{b}]_\times \hat{A}$ where

$$\hat{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 0 \end{bmatrix}, \hat{b} = \begin{bmatrix} b_1 \\ b_2 \\ 1 \end{bmatrix}$$

Thus F can be found.

$$F = [\hat{b}]_\times \hat{A} = \begin{bmatrix} 0 & -1 & b_2 \\ 1 & 0 & -b_1 \\ -b_2 & b_1 & 0 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} -a_{21} & -a_{22} & -a_{23} \\ a_{11} & a_{12} & a_{13} \\ -a_{11}b_2 + a_{21}b_1 & -a_{12}b_2 + a_{22}b_1 & -a_{13}b_2 + a_{23}b_1 \end{bmatrix}$$

We can divide F by a_{21} to get a seven-parameter expression,

$$\begin{bmatrix} -1 & -\frac{a_{22}}{a_{21}} & -\frac{a_{23}}{a_{21}} \\ \frac{a_{11}}{a_{21}} & \frac{a_{12}}{a_{21}} & \frac{a_{13}}{a_{21}} \\ -\frac{a_{11}}{a_{21}}b_2 + b_1 & -\frac{a_{12}}{a_{21}}b_2 + \frac{a_{22}}{a_{21}}b_1 & -\frac{a_{13}}{a_{21}}b_2 + \frac{a_{23}}{a_{21}}b_1 \end{bmatrix}.$$

The new seven parameters are $\frac{a_{11}}{a_{21}}$, $\frac{a_{12}}{a_{21}}$, $\frac{a_{13}}{a_{21}}$, $\frac{a_{22}}{a_{21}}$, $\frac{a_{23}}{a_{21}}$, b_1, b_2.

2 Fundamental Matrix Estimation From Point Correspondences (20 points)

This problem is concerned with the estimation of the fundamental matrix from point correspondences. Implement both the linear least-squares version of the eight-point algorithm and its normalized version. In both cases, enforce the rank-two constraint for the fundamental matrix via singular value decomposition.

The data for this problem is here: `http://cs231a.stanford.edu/hw/ps2/data.zip`. The zipped file contains two datasets. For each dataset, you will find in the corresponding directory the files

```
pt_2d_1.txt, pt_2d_2.txt
image1.jpg, image2.jpg
```

The first two of these files contain matching image points (in the following format: number n of points followed by n pairs of floating-point coordinates). The two remaining files are the images where the points were found.
You must turn in:

a. a copy of your code, [**15 points**]

b. For both methods applied to both datasets, the average distance between the points and the corresponding epipolar lines for each image, as well as a drawing similar to Figure 1 (with the selected corresponding points and their epipolar lines). [**5 points**]

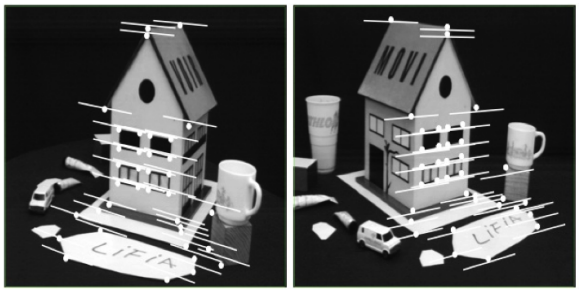

Figure 1: Example illustration, with epipolar lines shown in both images (Image courtesy Forsyth & Ponce)

Solution:

1. Code Implementation

```
    for useDataset = 1:2

clearvars -EXCEPT useDataset

[p1,p2] = readTextFiles(useDataset);

W(:,1) = p1(1,:)'.*p2(1,:)';
```

```
W(:,2) = p1(1,:)'.*p2(2,:)';
W(:,3) = p1(1,:)';
W(:,4) = p1(2,:)'.*p2(1,:)';
W(:,5) = p1(2,:)'.*p2(2,:)';
W(:,6) = p1(2,:)';
W(:,7) = p2(1,:)';
W(:,8) = p2(2,:)';
W(:,9) = ones(length(p1),1);

[U,S,V] = svd(W);
oldF = reshape(V(:,end),3,3)';
[U2,S2,V2] = svd(oldF);
F = zeros(3,3);
for i = 1:2
    F = F + S2(i,i)*U2(:,i)*V2(:,i)';
end

% Normalizing coordinates
xmean1 = mean(p1(1,:));
ymean1 = mean(p1(2,:));
d1 = (1/length(p1))*sum(sqrt( ((p1(1,:)-xmean1).^2)+...
    ((p1(2,:)-ymean1).^2) ) );
T1 = [sqrt(2)/d1 0 -xmean1*sqrt(2)/d1;...
      0 sqrt(2)/d1 -ymean1*sqrt(2)/d1;...
      0 0 1];
xmean2 = mean(p2(1,:));
ymean2 = mean(p2(2,:));
d2 = (1/length(p2))*sum(sqrt(((p2(1,:)-repmat(xmean2,1,length(p2))).^2)+...
    ((p2(2,:)-repmat(ymean2,1,length(p2))).^2)));
T2 = [sqrt(2)/d2 0 -xmean2*sqrt(2)/d2;...
      0 sqrt(2)/d2 -ymean2*sqrt(2)/d2;...
      0 0 1];
q1 = zeros(size(p1));
q2 = zeros(size(p1));
for i = 1:length(p1)
    q1(:,i) = T1*p1(:,i);
    q2(:,i) = T2*p2(:,i);
end

Wnorm(:,1) = q1(1,:)'.*q2(1,:)';
Wnorm(:,2) = q1(1,:)'.*q2(2,:)';
Wnorm(:,3) = q1(1,:)';
Wnorm(:,4) = q1(2,:)'.*q2(1,:)';
Wnorm(:,5) = q1(2,:)'.*q2(2,:)';
Wnorm(:,6) = q1(2,:)';
Wnorm(:,7) = q2(1,:)';
Wnorm(:,8) = q2(2,:)';
Wnorm(:,9) = ones(length(q1),1);
```

```
[U,S,V] = svd(Wnorm);
oldF = reshape(V(:,end),3,3)';
[U2,S2,V2] = svd(oldF);
Fq = zeros(3,3);
for i = 1:2
    Fq = Fq + S2(i,i)*U2(:,i)*V2(:,i)';
end
Fnorm = T1'*Fq*T2;

% Calculating Epipolar Lines
l1norm = Fnorm*p2;
l2norm = Fnorm'*p1;
l1 = F*p2;
l2 = F'*p1;

pdist1 = mean(abs(sum(p1.*l1,1))./sqrt((l1(1,:).^2)+(l1(2,:).^2)))
pdist2 = mean(abs(sum(p2.*l2,1))./sqrt((l2(1,:).^2)+(l2(2,:).^2)))

pdist1n = mean(abs(sum(p1.*l1norm,1))./sqrt((l1norm(1,:).^2)+(l1norm(2,:).^2)))
pdist2n = mean(abs(sum(p2.*l2norm,1))./sqrt((l2norm(1,:).^2)+(l2norm(2,:).^2)))

% Plotting Epipolar Lines
dataSets={'set1','set2'};
dataset=dataSets{useDataset};

x1l = p1(1,:)-15;
x2l = p2(1,:)-15;
x1r = p1(1,:)+15;
x2r = p2(1,:)+15;
y1l = (-l1(3,:)  -l1(1,:).*x1l)./l1(2,:);
y1r = (-l1(3,:)  -l1(1,:).*x1r)./l1(2,:);
y2l = (-l2(3,:)  -l2(1,:).*x2l)./l2(2,:);
y2r = (-l2(3,:)  -l2(1,:).*x2r)./l2(2,:);


figure
subplot(1,2,1)
imshow([dataset '/image1.jpg'])
hold on
for i = 1:length(p1)
    plot([x1l(i) x1r(i)],[y1l(i) y1r(i)],'r')
    plot(p1(1,i),p1(2,i),'b*')
end
datstr = ['Data',num2str(dataset),', Image 1, Non-normalized Results'];
title(datstr)

subplot(1,2,2)
```

```
imshow([dataset '/image2.jpg'])
hold on
for i = 1:length(p1)
    plot([x2l(i) x2r(i)],[y2l(i) y2r(i)],'r')
    plot(p2(1,i),p2(2,i),'b*')
end
datstr = ['Data',num2str(dataset),', Image 2, Non-normalized Results'];
title(datstr)
print('-dpng','-r300',num2str(dataset));


% Now for normalized
x1l = p1(1,:)-15;
x2l = p2(1,:)-15;
x1r = p1(1,:)+15;
x2r = p2(1,:)+15;
y1l = (-l1norm(3,:)  -l1norm(1,:).*x1l)./l1norm(2,:);
y1r = (-l1norm(3,:)  -l1norm(1,:).*x1r)./l1norm(2,:);
y2l = (-l2norm(3,:)  -l2norm(1,:).*x2l)./l2norm(2,:);
y2r = (-l2norm(3,:)  -l2norm(1,:).*x2r)./l2norm(2,:);
figure
subplot(1,2,1)
imshow([dataset '/image1.jpg'])
hold on
for i = 1:length(p1)
    plot([x1l(i) x1r(i)],[y1l(i) y1r(i)],'r')
    plot(p1(1,i),p1(2,i),'b*')
end
datstr = ['Data',num2str(dataset),', Image 1, Normalized Results'];
title(datstr)

subplot(1,2,2)
imshow([dataset '/image2.jpg'])
hold on
for i = 1:length(p1)
    plot([x2l(i) x2r(i)],[y2l(i) y2r(i)],'r')
    plot(p2(1,i),p2(2,i),'b*')
end
datstr = ['Data',num2str(dataset),', Image 2, Normalized Results'];
title(datstr)
print('-dpng','-r300',[num2str(dataset) ' norm']);

end
```

2. Results

File Name	Mean Distance for Normalized Coordinates (px)	Mean Distance for non-normalized Coordinates (px)
Dataset1/Image1.png	28.0257	0.8906
Dataset1/Image2.png	25.1629	0.8287
Dataset2/Image1.png	9.7014	0.8895
Dataset2/Image2.png	14.5682	0.8917

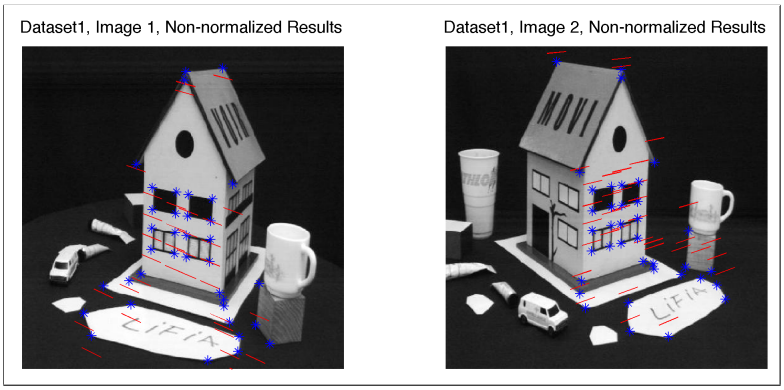

Figure 2: Dataset 1 Epipolar Lines

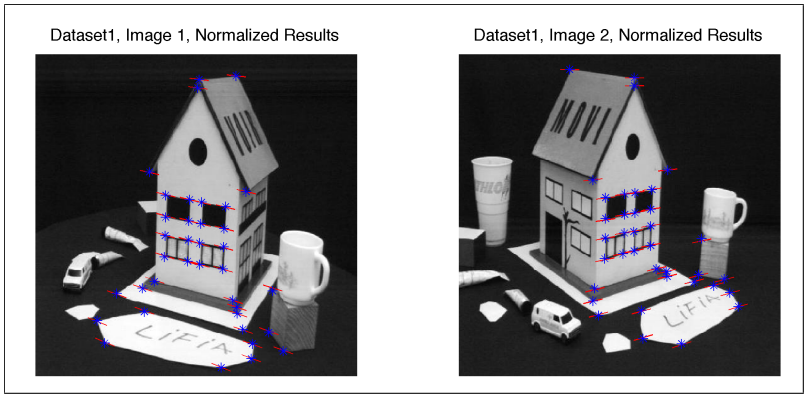

Figure 3: Dataset 1 Epipolar Lines with Normalized Coordinates

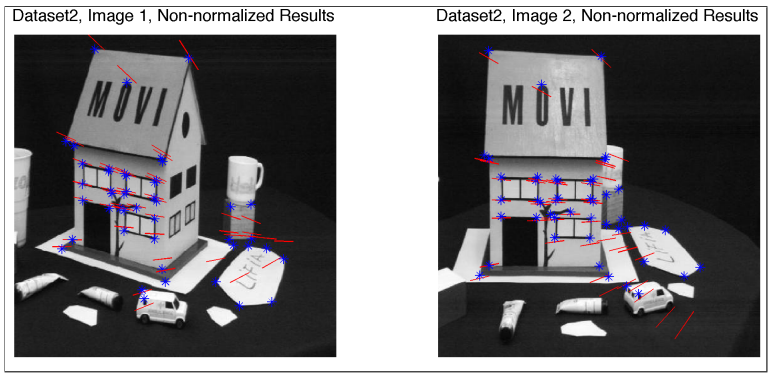

Figure 4: Dataset 2 Epipolar Lines

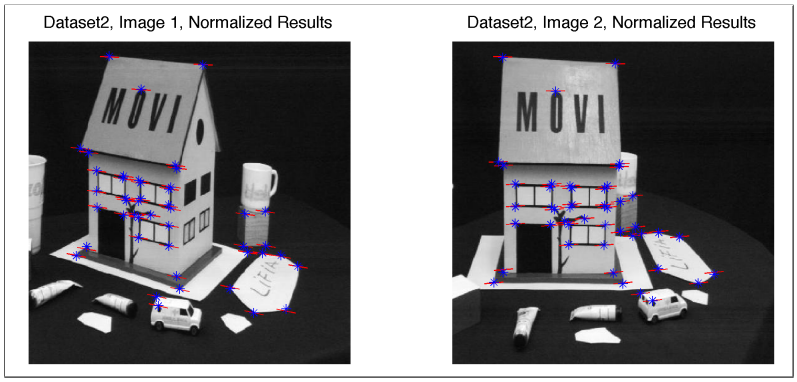

Figure 5: Dataset 2 Epipolar Lines with Normalized Coordinates

3 Affine Triangulation using the Factorization Method (30 points)

In this question, you will explore the factorization method, initially presented by Tomasi and Kanade, for solving the affine structure from motion problem.

Begin by downloading `http://cs231a.stanford.edu/hw/ps2/ps2_factor.zip` and running the function ps2q2.m. Observe the point correspondences that have been selected in the two images.

(a) Now implement the factorization method presented in the lecture slides. You only need to modify the file ps2q2.m, and you only need to enter code in the location that says "YOUR CODE HERE." [**7 points**]

Solution:
Code:

```
%Center data
x1center=1/size(x1,2)*sum(x1,2);
x2center=1/size(x2,2)*sum(x2,2);

x1c=x1-repmat(x1center, 1,size(x1,2));
x2c=x2-repmat(x2center, 1,size(x2,2));

%Generate measurement matrix
D=[x1c(1,:);x2c(1,:);x1c(2,:);x2c(2,:)];

%Factorize
[U,Sigma,V] = svd(D);

%Extract M(motion) and S(structure) Matrices
motion = U(:,1:3) * Sigma(1:3,1:3)^0.5;
structure = Sigma(1:3,1:3)^0.5 * (V(:,1:3)');
```

(b) Run your code and visualize the result by plotting the resulting 3d points; the `plot3` Matlab function may be helpful. We also recommend calling `axis equal` after making your plots so that the axes will be scaled appropriately.

Compare your result to the ground truth provided. The results should look identical, except for a scaling and rotation. Explain why this occurs. [**5 points**]

Solution:

An example of the factorization output can be seen in Figure 6.

The decomposition of our measurement matrix into our motion and structure matrices was not unique. The factorization processed allowed us to generate a matrix M and S such that:

$$D = MS$$

However we can get the same measurement matrix D by applying an invertible 3×3 linear transformation Q, as follows:

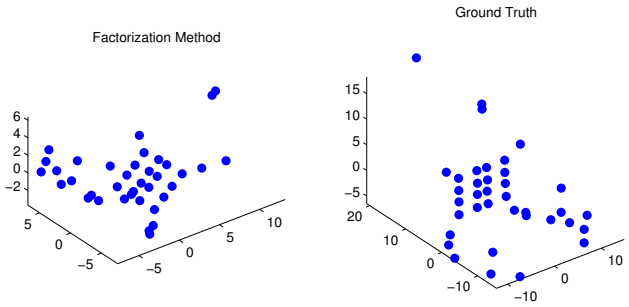

Figure 6: Example output of factorization code

$$D = (MQ)\left(Q^{-1}S\right)$$

So an arbitrary motion matrix MQ and structure matrix $Q^{-1}S$ can be generated. This affine ambiguity means that our estimate motion and structure matrices are linear transformation of the true motion and structure matrices, hence the apparent rotation and scaling of the code output compared to the ground truth.

(c) Report the 4 singular values from the SVD decomposition. Why are there 4 non-zero singular values? How many non-zero singular values would you expect to get in the idealized version of the method, and why? **[6 points]**

Solution:
Singular values: 959.5852, 540.4761, 184.4317, 27.9152

If exact correspondences were found, we would expect only 3 non-zero singular values. This is because D = MS, where M is a (2m x 3) motion matrix and S is a (3 x n) structure matrix, where m is the number of cameras and n is the number of correspondences. Thus D should be rank 3, and there should be only 3 non-zero singular values.

In reality, the correspondences are not exact, due to human error in selecting the correspondence points. Thus the 4th singular value is small but non-zero.

(d) Change the file readTextFiles.m by setting usesubset to true, and re-run the code. The file will now only load a subset of the original correspondences. Compare your new results to the ground truth, and explain why they no longer appear similar. **[6 points]**

Solution:
The selected subset of points all lie in a single plane, so the resulting matrix is only rank 2. The solution is thus degenerate.

An example of the factorization output with a subset of the original data can be seen in Figure 7.

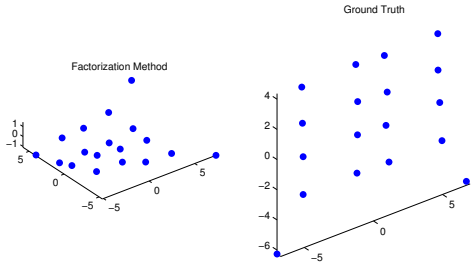

Figure 7: Example output of factorization code with degenerate output.

(e) Report the new singular values, and compare them to the singular values that you found previously. Explain any major changes. **[6 points]**

Solution:
Singular values: 264.5440, 210.0607, 7.2192, 5.1286

The selected correspondences all lie in a single plane, so the D matrix is now rank 2. Thus there are now 2 small singular values.

4 Projective Triangulation in Structure From Motion (30 points)

Figure 8: The set of images used in this structure from motion reconstruction.

Structure from motion is inspired by our ability to learn about the 3D structure in the surrounding environment by moving through it. Given a sequence of images, we are able to simultaneously estimate both the 3D structure and the path the camera took. In this problem, you will implement significant parts of a structure from motion framework, estimating both R and T of the cameras, as well as generating the locations of points in 3D space. Recall that in the previous problem we triangulated points assuming affine transformations. However, in the actual structure from motion problem, we assume projective transformations. By doing this problem, you will learn how to solve this type of triangulation.

The starter code and data can be found at `http://cs231a.stanford.edu/hw/ps2/SFM.zip`

(a) Given correspondences between pairs of images, we compute the respective Fundamental and Essential matrices. Given the Essential matrix, we must now compute the R and T

between the two cameras. On pages 257-259 of HZ, the authors explain that this is possible as follows:

1. To compute R: Given the singular value decomposition $E = UDV^T$, we can rewrite $E = MQ$ where $M = UZU^T$ and $Q = UWV^T$ *or* UW^TV^T, where $Z = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ and $W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$. Note that this factorization of E only guarantees that Q is orthogonal. To find a rotation, we simply compute $R = (\det Q)Q$.
2. To compute T: Given that $E = UDV^T$, T is simply either u_3 or $-u_3$, where u_3 is the third column vector of U.

Implement this in the function `computeRTFromE.m`. For now, verify that you get the correct four pairs of R and T. Give the four pairs of R and T for the sample matrix in `sfm.m`. You will submit your code for this problem in part e.[**5 points**]

Solution:

$$\begin{bmatrix} 0.9831 & -0.1179 & -0.1404 & 0.9994 \\ -0.1193 & -0.9929 & -0.0015 & -0.0089 \\ -0.1392 & 0.0182 & -0.9901 & 0.0331 \end{bmatrix}$$

$$\begin{bmatrix} 0.9831 & -0.1179 & -0.1404 & -0.9994 \\ -0.1193 & -0.9929 & -0.0015 & 0.0089 \\ -0.1392 & 0.0182 & -0.9901 & -0.0331 \end{bmatrix}$$

$$\begin{bmatrix} 0.9736 & -0.0988 & -0.2056 & 0.9994 \\ 0.1019 & 0.9948 & 0.0045 & -0.0089 \\ 0.2041 & -0.0254 & 0.9786 & 0.0331 \end{bmatrix}$$

$$\begin{bmatrix} 0.9736 & -0.0988 & -0.2056 & -0.9994 \\ 0.1019 & 0.9948 & 0.0045 & 0.0089 \\ 0.2041 & -0.0254 & 0.9786 & -0.0331 \end{bmatrix}$$

(b) In order to distinguish the correct R,T pair, we must first know how to find the 3D point given matching correspondences in different images. On page 312 of HZ, the authors explain a linear estimate (DLT) of this 3D point:

1. For each image i, we have $x_i = P_iX$, where X is the 3D point, x_i is the homogenous image coordinate of that point, and P_i is the projective camera matrix.
2. Formulate matrix
$$A = \begin{bmatrix} x_{1,1}p^{3\top} - p^{1\top} \\ x_{1,2}p^{3\top} - p^{2\top} \\ \vdots \\ x_{n,1}p^{3\top} - p^{1\top} \\ x_{n,2}p^{3\top} - p^{2\top} \end{bmatrix}$$
where $x_{i,1}$ and $x_{i,2}$ are the xy coordinates in image i and $p^{k\top}$ is the k-th row of P.
3. The 3D point is simply the vector in V corresponding to the smallest singular value of A.

Implement the linear estimate of this 3D point in `linearEstimate3D.m`. Submit your code and run the check in `sfm.m`. [**5 points**]

Solution:
Code:

```
function [ X ] = linearEstimate3D( x, P, image_sizes )
%LINEARESTIMATE3D Given a match in K different images, find the best 3D point
%
%   INPUTS:
%       x - measured points in each of the K images (2xK matrix)
%       P - camera matrices (K-cell of 3x4 matrices)
%       image_sizes - 2xK matrix of image sizes, 1st row are image widths,
%           2nd row are image heights
%   OUTPUTS:
%       X - 3D point that is MLE given the K projections in different
%       images
%
% We do some preprocessing at the start of the code to ensure stability at
% runtime. You don't have to modify it.
%
% This problem is described in Multiple View Geometry, Hartley & Zisserman pg. 312

if iscell(P)
  P = cat(3,P{:});
end
K = size(P,3);

% We need to normalize x to range from -1 to 1 (recall that
% this also changes the camera projection matrices). Otherwise your
% matrices may end up being near singular.
for k = 1:K
    H = [2/image_sizes(1,k) 0 0
         0 2/image_sizes(2,k) 0
         0 0              1];
    P(:,:,k) = H*P(:,:,k);
    x(:,k) = H(1:2,1:2)*x(:,k) + H(1:2,3);
end

% IMPLEMENT THE FUNCTION BELOW
A = zeros(2*K, 4);
for i = 1:K
    A(i*2-1, :) = x(1, i) * P(3, :, i) - P(1, :, i);
    A(i*2, :) = x(2, i) * P(3, :, i) - P(2, :, i);
end

[U, S, V] = svd(A);
X = V(:, end);
```

```
end
```

(c) However, we can do better than linear estimates, but usually this falls under some iterative nonlinear optimization. To do this kind of optimization, we need some residual. A simple one is the reprojection error of the correspondences, which is computed as follows:
For each image i, given camera matrix P_i, the 3D point X, we compute $y = P_i X$, and find that the image coordinates

$$x_i' = \frac{1}{y_3} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

Given the ground truth image coordinates x_i, the reprojection error e_i for image i is

$$e_i = x_i' - x_i$$

The Jacobian is written as follows:

$$J = \begin{bmatrix} \frac{\partial e_1}{\partial X_1} & \frac{\partial e_1}{\partial X_2} & \frac{\partial e_1}{\partial X_3} \\ \vdots & \vdots & \vdots \\ \frac{\partial e_m}{\partial X_1} & \frac{\partial e_m}{\partial X_2} & \frac{\partial e_m}{\partial X_3} \end{bmatrix}$$

Recall that each e_i is a vector of length two, so the whole Jacobian is a $2K \times 3$ matrix, where K is the number of cameras. Fill in `reprojectionError.m`, which computes the reprojection error and Jacobian for a 3D point and its list of images. Submit your code and run the check in `sfm.m`. [**5 points**]

Solution:
Code:

```
function [ error, J ] = reprojectionError( point3D, image_points, projection_matrices )
%REPROJECTIONERROR Computes reprojection error, given 3d point and its
% matching points
%   INPUTS:
%       point3D - The 3x1 point in 3D space that provides the
%           correspondences in each of the cameras
%       image_points - The 2xK  coordinates in each of the K images
%       projection_matrices - a list of 3x4xK P matrices between the K cameras
%   OUTPUTS:
%       error - The 2Kx1 reprojection error
%       J - The 2Kx3 jacobian matrix

    % reformatting data in case projection_matrices is a cell array of matrices
    if iscell(projection_matrices)
      projection_matrices = cat(3,projection_matrices{:});
    end

    K = size(projection_matrices,3);
    error = zeros(2*K,1);
```

```
    J = zeros(2*K,3);
    for k = 1:K
      R = projection_matrices(:,1:3,k);
      t = projection_matrices(:,4,k);
      projected = R*point3D + t;
      error(2*(k-1)+1:2*k,:) = projected(1:2)/projected(3)-image_points(:,k);
      J(2*(k-1)+1:2*k,:) = [projected(3)*R(1,:)-projected(1)*R(3,:)
               projected(3)*R(2,:)-projected(2)*R(3,:)]/projected(3)^2;
    end
end
```

(d) Implement the Newton Step algorithm which finds an approximation to the 3D point that minimizes this reprojection error. Recall that the Newton Step needs a good initialization, which we have from our linear estimate in part b. Then we continually update X for ten iterations:

$$X = X - J^T(JJ^T)^{-1}e$$

where J and e are the Jacobian and error computed from the previous part.
Implement this Newton step algorithm in the `nonlinEstimate3D.m` function. Submit your code and run the check in `sfm.m`. [**5 points**]

Solution:
Code:

```
function [ X ] = nonlinEstimate3D( x, P, image_sizes )
%NONLINESTIMATE3D %Given a match in K different images, find the best 3D point
%   As outlined in Zisserman pg. 95-99, 314 - 315
%   INPUTS:
%       x - measured points in each of the K images (2xK matrix)
%       P - camera matrices (K-cell of 3x4 matrices), is converted to a
%           3x4xK matrix in the code.
%       image_sizes - 2xK matrix of image sizes
%   OUTPUTS:
%       X - 3D point that is MLE given the K projections in different
%           images
%
% We do some preprocessing at the start of the code to ensure stability at
% runtime. You don't have to modify it.
%
% Using, the linear estimate as a starting value, simply keep doing newton
% steps (we do a fixed amount of 10 for speed)

if iscell(P)
  P = cat(3,P{:});
end
K = size(P,3);

% Get the linear estimate as the initial value
X = linearEstimate3D(x,P,image_sizes);
```

```
X = X(1:3)/X(4);

% We need to normalize x to range from -1 to 1 (recall that
% this also changes the camera projection matrices). Otherwise your
% matrices may end up being near singular.
for k = 1:K
    H = [2/image_sizes(1,k) 0 0
         0 2/image_sizes(2,k) 0
         0 0              1];
    P(:,:,k) = H*P(:,:,k);
    x(:,k) = H(1:2,1:2)*x(:,k) + H(1:2,3);
end

% TODO: Implement the Newton Step functionality below
for i = 1:10
    [e, J] = reprojectionError( X, x, P );
    X = X - J'*pinv(J*J')*e;
end

X = [X; 1];


end
```

(e) Now finally, go back and distinguish the correct R, T pair from part a.

1. First, compute the location of the 3D point of each pair of correspondences given each R, T pair
2. Given each R, T you will have to find the 3D point's location in that R, T frame. The correct R, T pair is the one for which the most 3D points have positive depth (z-coordinate) in all camera frames. Page 259 of HZ has an excellent diagram and explanation of this.

Submit your updated code here. After doing so, we now have a robust method to triangulate 3D points given correspondences. Submit the final plot of the reconstruction. Hopefully, you can see a point cloud that looks like the frontal part of the statue in the above sequence of images: Note that to better see the point cloud, try rotating or wiggling the plot. You should be able to see the corner of the base and the cusp that forms the facial part of the statue. [**10 points**]

Solution:
Code:

```
function Rt =computeRTFromE(E, matches, K, im_width, im_height)
%BUNDLEADJUSTMENT runs bundle adjustment on a group of cameras to adjust
% motion and structure
% Arguments:
%          E  - The Essential Matrix between two cameras
%          matches - the matches between two cameras
%          K - camera matrix
```

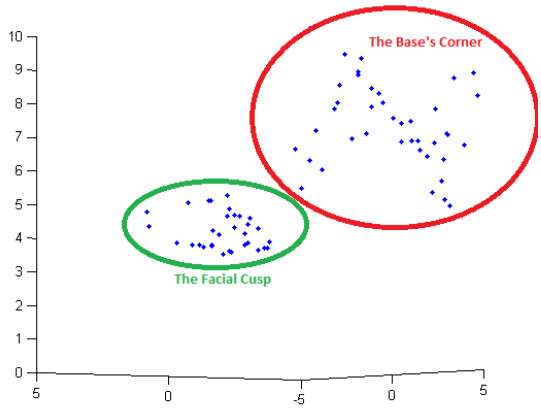

Figure 9: The final reconstruction our solution renders.

```
%          im_width - image width in pixels
%          im_height - image height in pixels
% Returns:
%          Rt - 3x4 Matrix where first 3x3 is R, last 3x1 is T
%
% Initially, for part A, you will return 4 of them, but your final result
% should only be one 3x4 Matrix. Look at pages 257-259 in Multiple View
% Geometry by Hartley and Zisserman for implementation details.

%TODO: FILL IN THE FUNCTION BELOW
[U,D,V] = svd(E);

W = [0 -1 0; 1 0 0; 0 0 1];
Z = [0  1 0;-1 0 0; 0 0 0];
% translaton vector (skew-symmetry matrix)
S = U*Z*U';
% two possible rotation matrices
R1 = U*W*V';
R2 = U*W'*V';
% translation vector
T = U(:,3);
% check determinant
if det(R1) < 0
  R1 = -R1;
end

if det(R2) < 0
  R2 = -R2;
end

P{1} = K*[eye(3) [0 0 0]'];
Rt{1} = [R1 T];
```

```
Rt{2} = [R2 T];
Rt{3} = [R1 -T];
Rt{4} = [R2 -T];
num_pos = zeros(4, 1);

for i = 1:4
    P{2} = K*Rt{i};
    for j = 1:size(matches, 2)
        pt = nonlinEstimate3D(reshape(matches(:,1),2,2),P,repmat([im_height;im_width],1,2)
        pt = pt / pt(4);
        if pt(3) > 0 && (Rt{i}(3,1:3) * ((pt(1:3) - Rt{i}(1:3,4)))) > 0
            num_pos(i) = num_pos(i) + 1;
        end
    end
end

[~, I] = max(num_pos);
Rt = Rt{I};
```