CSE252A Computer Vision Homework 3

Yingyan Hua & Yanli Wang

Question 1. Epipolar Geometry

a) Solution:

Given that the world coordinate of (x,y) = (0, 0) is (-15, 0, 1) and that of (u,v) = (0, 0) is (15, 0, 1), we find the world coordinates of (x,y) = (6, 6) and (u,v) = (1, 6) to be (-9, 6, 1) and (16, 6, 1), respectively. The two lines defined by camera focal points (-15, 0, 0), (15, 0, 0) and the two image points (-9, 6, 1), (16, 6, 1) are:

$l1$ = (-15, 0, 0) + s((-9, 6, 1) - (-15, 0, 0))
$l2$ = (15, 0, 0) + t((16, 6, 1) - (15, 0, 0)).

The point to be located is at the intersection of these two lines, when $l1 = l2$. By solving this equation we get s = t = 6. Inserting s=6 back to $l1$, we get $l1$ = (-15, 0, 0) + 6(6, 6, 1) = (21, 36, 6). Therefore, the 3D location of the point is at (21, 36, 6).

b) Solution:

Based the given information, we set up geometry relationships as shown in Figure 1. We first draw a line BO_2 that is parallel to AO_1. The distance between their crossing points D and E is disparity. From triangles AO_2B and BO_2F, we derive the following equation: $DE/AB = O_2G/O_2F = DG/AF$. Replacing the line segments by their length we get d/30 = f/z = -u/(15-x). Combining this equation with f=1 and x + z = 0, we can solve the equation and derive the following expression for disparity: d = -2(u+1).

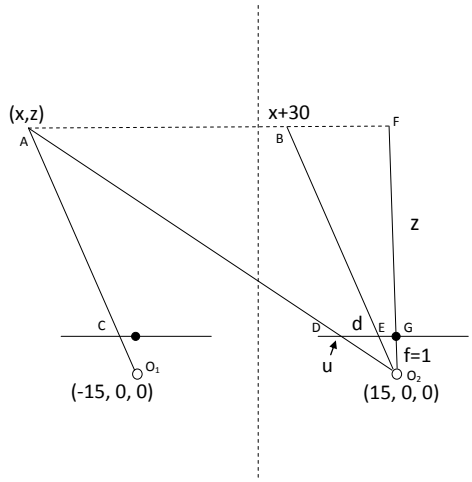

Figure 1: Problem 2 Set-up

Question 2. NCC

Let $\tilde{w}_1 = \tilde{W}_1(:)$ and $\tilde{w}_2 = \tilde{W}_2(:)$, and then we can rewrite the expression for NCC and NSSD as:
$c_{NCC} = \tilde{w}_1^T \tilde{w}_2$
$c_{NSSD} = (\tilde{w}_1 - \tilde{w}_2)^T (\tilde{w}_1 - \tilde{w}_2)$
$= (\tilde{w}_1^T - \tilde{w}_2^T)(\tilde{w}_1 - \tilde{w}_2)$
$= \tilde{w}_1^T \tilde{w}_1 - \tilde{w}_1^T \tilde{w}_2 - \tilde{w}_2^T \tilde{w}_1 + \tilde{w}_2^T \tilde{w}_2$
$= \tilde{w}_1^T \tilde{w}_1 + \tilde{w}_2^T \tilde{w}_2 - 2\tilde{w}_1^T \tilde{w}_2$ (since $\tilde{w}_1^T \tilde{w}_2$ and $\tilde{w}_2^T \tilde{w}_1$ are equivalent)
$= \tilde{w}_1^T \tilde{w}_1 + \tilde{w}_2^T \tilde{w}_2 - 2c_{NCC}$.

Therefore when we minimize c_{NSSD}, we are maximizing the negative term in the equation, which is the same as maximizing c_{NCC}.

Question 3. Sparse Stereo Matching

Part 1 Corner Detection

We first calculated the first-order derivatives of the warrior and matrix images and then smoothed them using gaussian filter with sigma set to 2. Using window size of 4 we computed the eigenvalues $\lambda 1$ and $\lambda 2$. We then performed non-maximum suppression to find local minima and the desired number of corners. The resulting figures are shown in Figures 2 and 3.

Figure 2: The twenty best corners detected for warrior images.

Figure 3: The twenty best corners detected for matrix images.

Part 2 SSD matching

The detailed implementation of the SSD matching algorithm is attached at the end of this report.

Part 3 Naive matching

The results of naive matching procedure are shown in Figure 4. As expected, naive matching does not produce good matching results because the corners detected from one image may not be the same set of corners in the other image.

Figure 4: The matching results using the SSD naive matching method.

Part 4 Epipolar geometry

The fundamental matrix was calculated using the provided fund.m function. The epipolar lines in the warrior and matrix image pairs are shown in Figures 5 and 6, respectively.

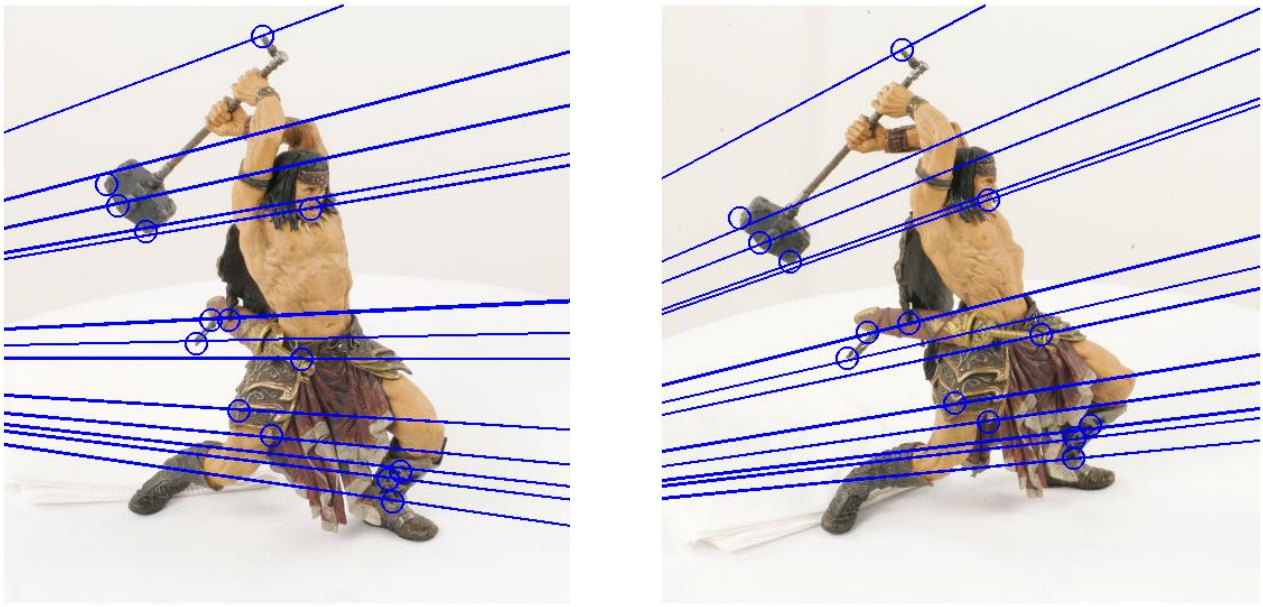

Figure 5: The epipolar lines detected for warrior images.

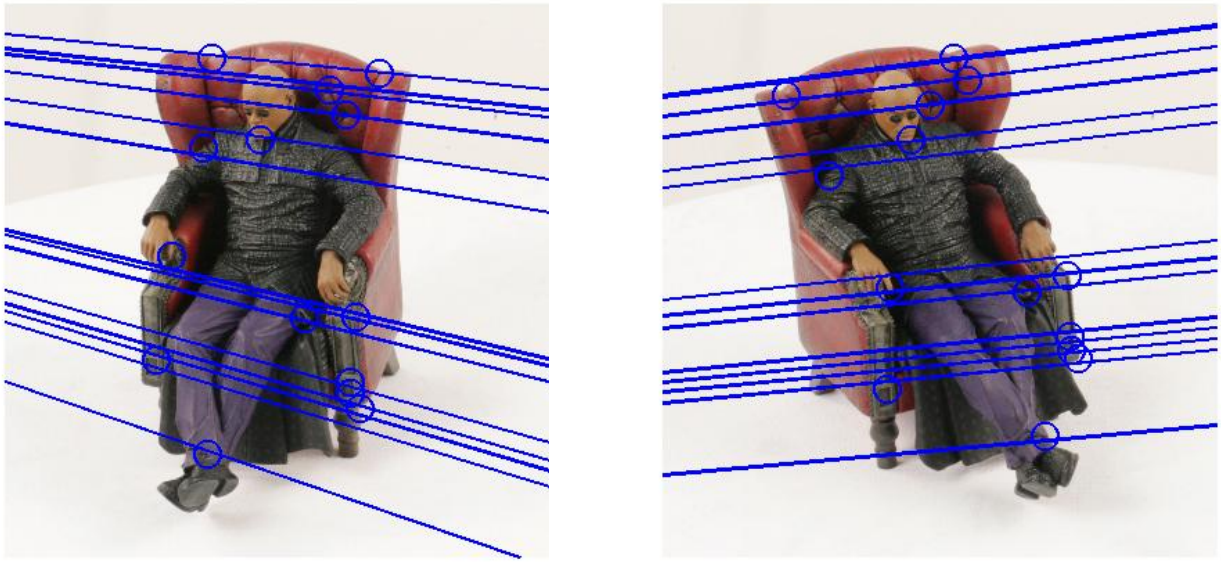

Figure 6: The epipolar lines detected for matrix images.

Part 5 Epipolar geometry based matching

We first computed the epipolar lines and then performed SSD matching using the epipolar lines. The matching results are shown in Figure 7. Compared to the results from Part 3, the current results are much more accurate. This is because searching for similarities along epipolar lines is much more efficient and accurate.

Figure 7: Matching results based on the epipolar lines detected for the two images.

Part 6 Triangulation

The results from triangulation are shown in Figure 8.

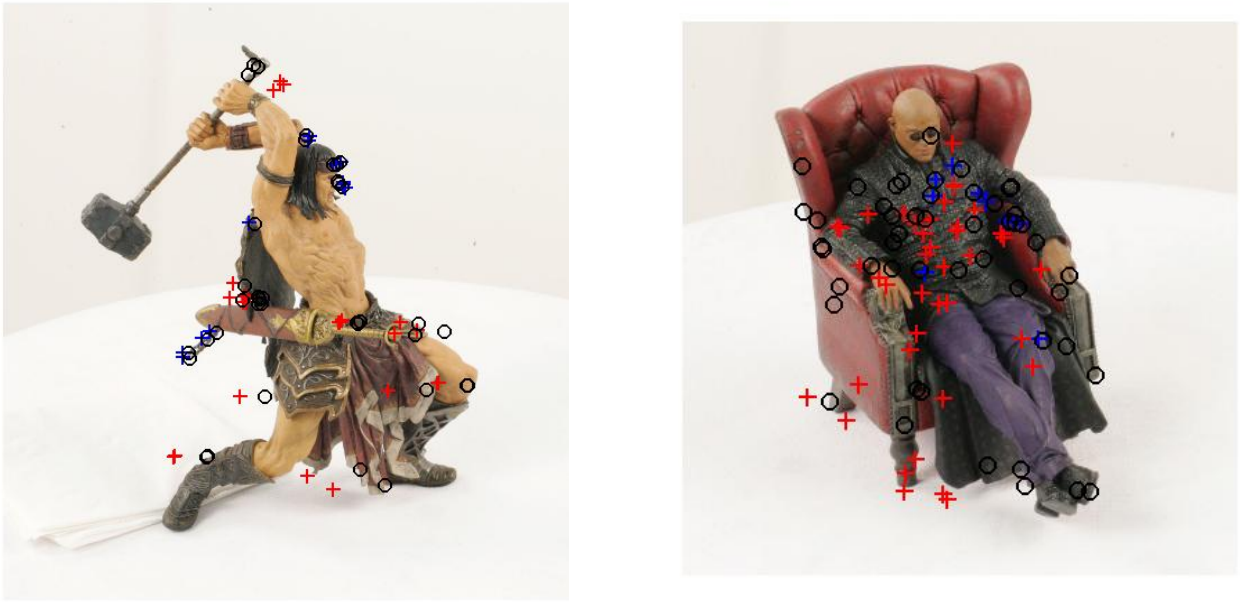

Figure 8: The inliers, outliers, and original points for the warrior and matrix images.

Appendix: MATLAB codes
Question 3 Part 1

```
function corners = CornerDetect(Image, nCorners, SmoothSTD, windowSize)
dx = [-1 0 1; -1 0 1; -1 0 1];
dy = dx';
im1 = rgb2gray(Image);
Ix = conv2(im1, dx, 'valid');
Iy = conv2(im1, dy, 'valid');
[dx, dy] = meshgrid(-(windowSize-1)/2: (windowSize-1)/2);
K = inline('exp(-(x.^2 + y.^2)/2/SmoothSTD^2)');
weight = K(SmoothSTD, dx, dy)/sum(sum(K(SmoothSTD, dx, dy)));
Ix2 = conv2(Ix.^2, weight, 'valid');
Iy2 = conv2(Iy.^2, weight, 'valid');
Ixy = conv2(Ix.*Iy, weight, 'valid');
sz = size(Image);
q1 = zeros(sz(1,1)*sz(1,2),1);
q2 = zeros(sz(1,1)*sz(1,2),1);
q3 = zeros(sz(1,1)*sz(1,2),1);
sumFilter = ones(windowSize);
q1 = conv2(Ix2, sumFilter, 'same');
q2 = conv2(Iy2, sumFilter, 'same');
q3 = conv2(Ixy, sumFilter, 'same');
S = sqrt(-4.*(q1.*q2-q3.^2) + (q1 + q2).^2);
l1 = (q1+q2+S)/2;
l2 = (q1+q2-S)/2;
lambda(:,:,1) = l1;
lambda(:,:,2) = l2;
sz1 = size(l1);
lambda = reshape(lambda, [size(lambda,1)*size(lambda,2)
size(lambda,3)]);
localmin = min(lambda, [], 2);
localmin = reshape(localmin, sz1(1, 1), sz1(1,2));
R = localmin(2:size(localmin,1)-1, 2:size(localmin,2)-1) >
localmin(2:size(localmin,1)-1, 3:size(localmin,2));
L = localmin(2:size(localmin,1)-1, 2:size(localmin,2)-1) >
localmin(2:size(localmin,1)-1, 1:size(localmin,2)-2);
U = localmin(2:size(localmin,1)-1, 2:size(localmin,2)-1) >
localmin(1:size(localmin,1)-2, 2:size(localmin,2)-1);
D = localmin(2:size(localmin,1)-1, 2:size(localmin,2)-1) >
localmin(3:size(localmin,1), 2:size(localmin,2)-1);
```

```
UR = localmin(2:size(localmin,1)-1, 2:size(localmin,2)-1) >
localmin(1:size(localmin,1)-2, 3:size(localmin,2));
DR = localmin(2:size(localmin,1)-1, 2:size(localmin,2)-1) >
localmin(3:size(localmin,1), 3:size(localmin,2));
UL = localmin(2:size(localmin,1)-1, 2:size(localmin,2)-1) >
localmin(1:size(localmin,1)-2, 1:size(localmin,2)-2);
DL = localmin(2:size(localmin,1)-1, 2:size(localmin,2)-1) >
localmin(3:size(localmin,1), 1:size(localmin,2)-2);
m = zeros(sz1);
m(2:sz1(1,1)-1, 2:sz1(1,2)-1) = R .* L .* U .* D.* UR .* DR .* UL .* DL;
localmin1 = m .* localmin;
[sortedValues, sortIndex] = sort(localmin1(:), 'descend');
ind = sortIndex(1:nCorners);
[r1 c1] = ind2sub([size(localmin,1), size(localmin,2)], ind);
corners = [r1 c1];
figure(1), imshow(Image), hold on
plot(c1, r1, 'o', 'MarkerSize', 10, 'linewidth',2),
title('corners detected');
end
```

Question 3 Part 2

```
function [match score] = SSDmatch(template, image, threshold)
lr = template;
lrm = mean(lr(:));
lr = lr - lrm;
rr = image;
rrm = mean(rr(:));
t = (lr-(rr-rrm)).^2;
s = sum(t(:));
score=s;
if(s<threshold)
match=1;
else
match=0;
end
```

Question 3 Part 3

```
function [I, corsSSD] = naiveCorrespondanceMatching(I1, I2, corners1, corners2, R, SSDth)
sz=size(corners1);
corsSSD=zeros(sz(1,1),2);
szI=size(I1);
for i=1:sz(1,1)
if (corners1(i,1)+R)<szI(1,1)
template=I1(corners1(i,1)-R:corners1(i,1)+R,
corners1(i,2)-R:corners1(i,2)+R);
min=4000;
for j=1:sz(1,1)
image=I2(corners2(j,1)-R:corners2(j,1)+R,
corners2(j,2)-R:corners2(j,2)+R);
[match score]=SSDmatch(template, image, SSDth);
if match==1&&score<min
min=score;
corsSSD(i,1)=corners2(j,1);
corsSSD(i,2)=corners2(j,2);
end
end
end
end
corsSSD(:,3)=corners1(:,1);
corsSSD(:,4)=corners1(:,2);
I=[I1 I2];
figure;
imshow(I);
hold on;
for i=1:sz(1,1)
plot(corsSSD(i,4),corsSSD(i,3),'ro', 'MarkerSize', 10, 'linewidth',2);
end
for i=1:sz(1,1)
plot(corsSSD(i,2)+szI(1,1),corsSSD(i,1),'ro', 'MarkerSize', 10, 'linewidth',2);
end
for i=1:sz(1,1)
if corsSSD(i,2)>1
plot([corsSSD(i,4),corsSSD(i,2)+szI(1,1)],[corsSSD(i,3),corsSSD(i,1)], 'MarkerSize', 10, 'linewidth',2);
end
end
```

Question 3 Part 4

```
function [c1 c2]=epiLine(I1, I2, cor1, cor2)
sz=size(I1);
F=fund(cor2, cor1);
cor11=cor1';
for i=1:14
cor11(3,i)=1;
end;
cor21=cor2';
for i=1:14
cor21(3,i)=1;
end;
x=[1,sz(1,1)];
y=[1,sz(1,1)];
figure
imshow(I1);
hold on;
for i=1:14
plot(cor1(i,1),cor1(i,2),'o');
end
for i=1:14
c2(:,i)=F*cor21(:,i);
pts2(:,:,i)=linePts(c2(:,i)',x,y);
end
for i=1:14
plot([pts2(1,1,i),pts2(2,1,i)],[pts2(1,2,i),pts2(2,2,i)]);
end
figure
imshow(I2);
hold on;
for i=1:14
plot(cor2(i,1),cor2(i,2),'o');
end
for i=1:14
c1(:,i)=F'*cor11(:,i);
pts1(:,:,i)=linePts(c1(:,i)',x,y);
end
for i=1:14
plot([pts1(1,1,i),pts1(2,1,i)],[pts1(1,2,i),pts1(2,2,i)]);end
```

Question 3 Part 5

```
function corsSSD = correspondanceMatchingLine(I1, I2, corners1, F, R, SSDth)
sz=size(corners1);
corsSSD=zeros(sz(1,1),2);
szI=size(I1);
cors1(:,1)=corners1(:,2);
cors1(:,2)=corners1(:,1);
cor11=cors1';
for i=1:sz(1,1)
cor11(3,i)=1; end;
a=[1,szI(1,1)];
b=[1,szI(1,1)];
for i=1:sz(1,1)
c1(:,i)=F'*cor11(:,i);
pts1(:,:,i)=linePts(c1(:,i)',a,b);
template=I1(corners1(i,1)-R:corners1(i,1)+R,
corners1(i,2)-R:corners1(i,2)+R);
min=40000;
for x=pts1(1,1,i)+R+1:pts1(2,1,i)-R-1
y=(-c1(3,i)-c1(1,i)*x)/c1(2,i);
x1=round(x);
y1=round(y);
if((y1-R)>0&&(x1-R)>0)
image=I2(y1-R:y1+R, x1-R:x1+R);
[match score]=SSDmatch(template, image, SSDth);
if match==1&&score<min
min=score;
corsSSD(i,1)=y1;
corsSSD(i,2)=x1;
end end
end end
corsSSD(:,3)=corners1(:,1);
corsSSD(:,4)=corners1(:,2);
I=[I1 I2];
figure; imshow(I); hold on;
for i=1:sz(1,1)
plot([corsSSD(i,4),corsSSD(i,2)+szI(1,1)],[corsSSD(i,3),corsSSD(i,1)], 'MarkerSize', 10, 'linewidth',2);
end
for i=1:sz(1,1)
plot(corsSSD(i,4),corsSSD(i,3),'ro', 'MarkerSize', 10, 'linewidth',2); end
for i=1:sz(1,1)
plot(corsSSD(i,2)+szI(1,1),corsSSD(i,1),'ro', 'MarkerSize', 10, 'linewidth',2); end
```

Question 3 Part 6

```
function[inlier, outlier] = findOutliers(points3D, P2, outlierTH, corsSSD) corsB=corsSSD(:,1:2);
corsB=corsB';
back=P2*points3D;
szback=size(corsB);
bback(1,:)=back(1,:)./back(3,:);
bback(2,:)=back(2,:)./back(3,:);
p=1; q=1;
for i=1:(szback(1,2))
if(((corsB(1,i)-bback(1,i))^2+(corsB(2,i)-bback(2,i))^2)>(outlierTH^2))
outlier(:,p)=bback(:,i);
p=p+1;
else
inlier(:,q)=bback(:,i);
q=q+1;
end
end
szin=q-1;
szout=p-1;

function points3D = triangulate(corsSSD, P1, P2)
x11(:,1)=corsSSD(:,3);
x11(:,2)=corsSSD(:,4);
x11(:,3)=1;
x11=x11';
x22(:,1)=corsSSD(:,1);
x22(:,2)=corsSSD(:,2);
x22(:,3)=1;
x22=x22';
for i=1:size(x11,2)
sx1 = x11(:,i);
sx2 = x22(:,i);
A1 = sx1(1,1).*P1(3,:) - P1(1,:);
A2 = sx1(2,1).*P1(3,:) - P1(2,:);
A3 = sx2(1,1).*P2(3,:) - P2(1,:);
A4 = sx2(2,1).*P2(3,:) - P2(2,:);
A = [A1;A2;A3;A4];
[U,S,V] = svd(A);
Xtemp = V(:,4);
Xtemp = Xtemp ./ repmat(Xtemp(4,1),4,1);
points3D(:,i) = Xtemp;
end
```