Hao-en Sung (A53204772)
Tsung-han Lee (A53219632)
University of California, San Diego

November 8, 2016

1 Epipolar Geometry Theory [6pt]

1. Given calibrations of two cameras (a stereo pair) to a common external coordinate system, represented by R_1, T_1, R_2, T_2, provide an expression that will map points expressed in the coordinate system of camera 1 to that of camera 2.

Sol. It is known that

$$\begin{aligned} {}^{c_1}P &= R_1 \cdot {}^{w}P + T_1 \\ {}^{w}P &= R_1^{-1} \cdot ({}^{c_1}P - T_1)\,. \end{aligned}$$

Then I can convert the world coordinate to that of camera 2 as follows.

$$\begin{aligned} {}^{c_2}P &= R_2 \cdot {}^{w}P + T_2 \\ &= R_2 \cdot R_1^{-1} \cdot ({}^{c_1}P - T_1) + T_2 \end{aligned}$$

□

2. What is the length of the baseline of the stereo pair.

Sol. Consider the point O in camera 1. I know

$$\begin{aligned} {}^{c_2}O &= R_2 \cdot R_1^{-1} \cdot ({}^{c_1}O - T_1) + T_2 \\ &= -R_2 \cdot R_1^{-1} \cdot T_1 + T_2. \end{aligned}$$

□

3. Give an expression for the Essential Matrix in terms of R_1, T_1, R_2, T_2.

Sol. By transforming all points on camera 2 coordinate, I have

$$\begin{aligned} {}^{c_2}O &= R_2 \cdot R_1^{-1} \cdot (-T_1) + T_2 \\ {}^{c_2}\overrightarrow{OO'} &= R_2 \cdot R_1^{-1} \cdot (-T_1) + T_2 \\ {}^{c_2}P &= R_2 \cdot R_1^{-1} \cdot (P - T_1) + T_2 \\ {}^{c_2}\overrightarrow{OP} &= R_2 \cdot R_1^{-1} \cdot P \end{aligned}$$

Under the coordinate of camera 2, I also have the coplanar constraint that

$$\left(\overrightarrow{OP} \times \overrightarrow{OO'}\right) \cdot \overrightarrow{O'P'} = P^T \cdot \left[(R_2 \cdot R_1^{-1}) \times (R_2 \cdot R_1^{-1} \cdot (-T_1) + T_2)\right] \cdot P'.$$

Thus, the essential matrix equals to $(R_2 \cdot R_1^{-1}) \times (R_2 \cdot R_1^{-1} \cdot (-T_1) + T_2)$. □

2 Epipolar Geometry [4 pt]

1. Suppose the points $(x, y) = (8, 7)$ is matched with disparity of 6 to the point $(u, v) = (2, 7)$. What is the 3D location of this point?

Sol. From the given information, I know that the baseline $b = 12 - (-12) = 24$, focal length $f = 1$, and disparity $X_L - X_R = 8 - 2 = 6$. Then, I have

$$\begin{aligned} X &= \frac{b \cdot X_L}{X_L - X_R} = \frac{24 \cdot 8}{6} = 32, \\ Y &= \frac{b \cdot Y_L}{X_L - X_R} = \frac{24 \cdot 7}{6} = 28, \\ Z &= \frac{b \cdot f}{X_L - X_R} = \frac{24 \cdot 1}{6} = 4. \end{aligned}$$

□

2. Consider points that lie on the 3-D line $X + Z = 2, Y = 0$. Use the same stereo set up as before. Write an analytic expression giving the disparity of a point on this line after it projects onto the two images, as a function of its position in the right image. So your expression should only involve the variables u and d (for disparity). Your expression only needs to be valid for points on the line that are in front of the cameras, i.e. with $Z > 1$.

Sol. It is known that $X_R = u$. Thus, I can express X_L as $u + d$. After that I can rewrite X and Z as follows.

$$\begin{aligned} X &= \frac{b \cdot X_L}{X_L - X_R} \\ &= \frac{24 \cdot (u + d)}{d} \\ Z &= \frac{b \cdot f}{d} \\ &= \frac{24}{d} \end{aligned}$$

Since $X + Z = 2$, I have

$$\begin{aligned} X + Z = 2 &= \frac{24u + 24d + 24}{d} \\ 0 &= 22d + 24u + 24 \\ d &= \frac{-12(u + 1)}{11}. \end{aligned}$$

□

3 Reconstruction Accuracy [3pt]

Sol. From lecture slides, I know that $Z' = \frac{bf}{(X_L - X_R)}$. To find out the dependence between $\Delta Z'$ and $\Delta(X_L - X_R)$, I have

$$\begin{aligned} \frac{\partial Z'}{\partial (X_L - X_R)} &= \frac{-bf}{(X_L - X_R)^2} \\ &= \frac{-(Z')^2}{bf}. \end{aligned}$$

□

4 Filters as Templates [16pt]

4.1 Warmup [3pt]

Sol. I first use the filter to convolute with the target image and plot the intensity figure, which is shown in the left subfigure of Fig 1. Then I use the local maximum as the center of the bounding box, as shown in right subfigure of Fig. 1.

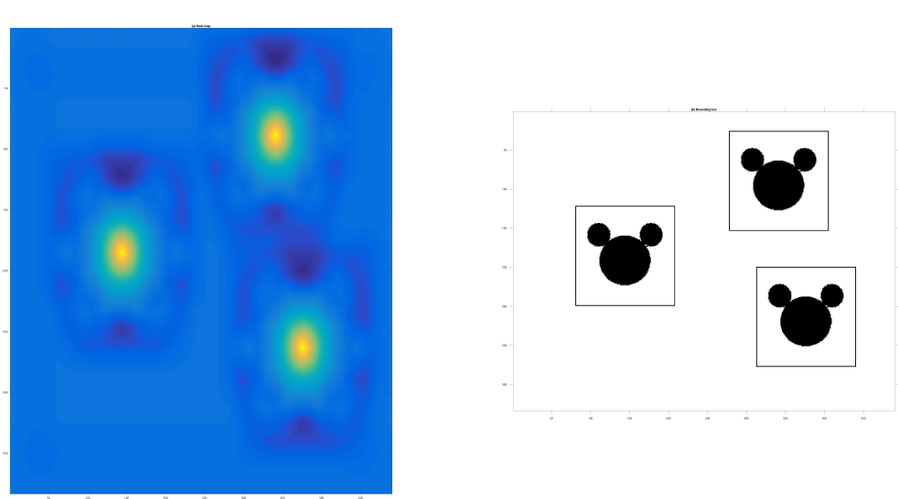

Figure 1: Warmup

□

4.2 Detection Error [3pt]

Sol. In this subsection, I draw three arbitrary bounding boxes and calculate the corresponding detection errors as shown in Fig. 2.

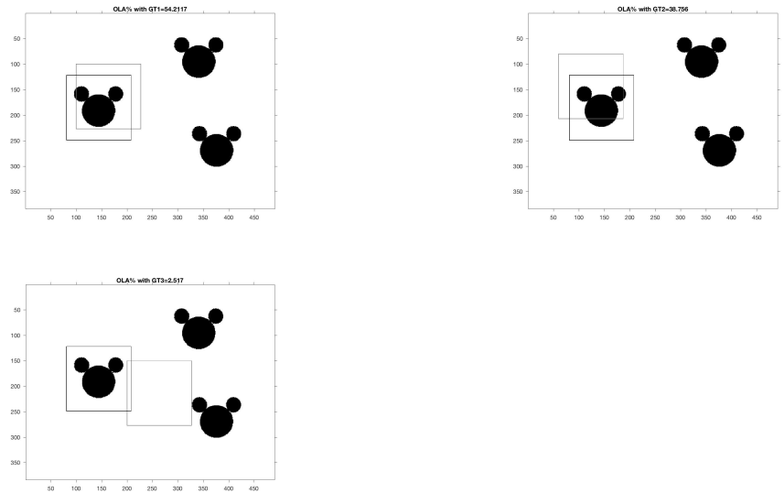

Figure 2: Detection Error

□

4.3 More Realistic Images [8pt]

Sol. I do a lot of parameter tunings, including shading, rotating, rescaling, and blurring, for five car figures as shown in Fig. 3 to 7.

I use the Code 2 to generate the Fig. 3. It can be found that I achieve a very good result and obtain around 90.283% accuracy score.

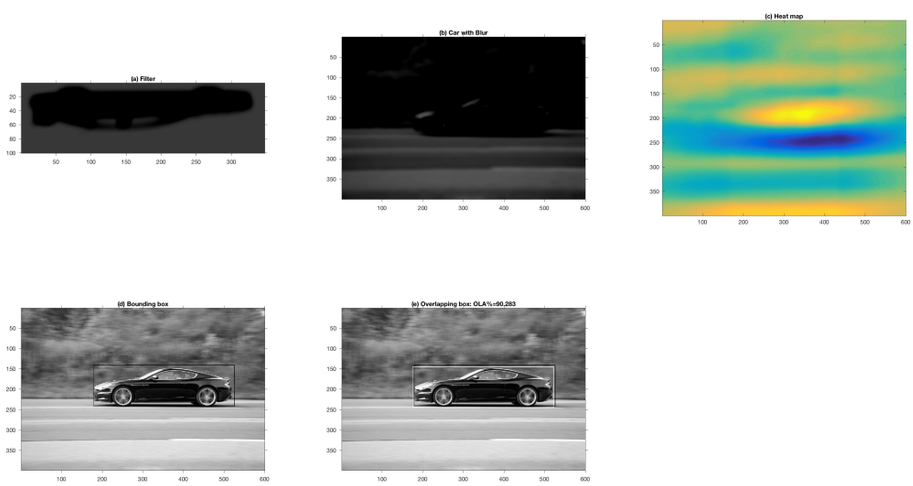

Figure 3: More Realistic Images: Car 1

I use the Code 3 to generate the Fig. 4. It can be found that I also achieve a very good result and obtain around 84.761% accuracy score.

I use the Code 4 to generate the Fig. 5. In this task, I achieve only 11.237 accuracy score. From the intensity figure, it can be observed that there are multiple possible local maximum value. That is the reason why I fail to bound the image correctly.

I use the Code 5 to generate the Fig. 6. This task is not that easy, because there are multiple people stand in front of the car. Thus, I achieve merely around 60.802 accuracy score.

I use the Code 6 to generate the Fig. 7. Since the template car shape is different from the target car shape, the result is not so good — around 64.902 accuracy score.

□

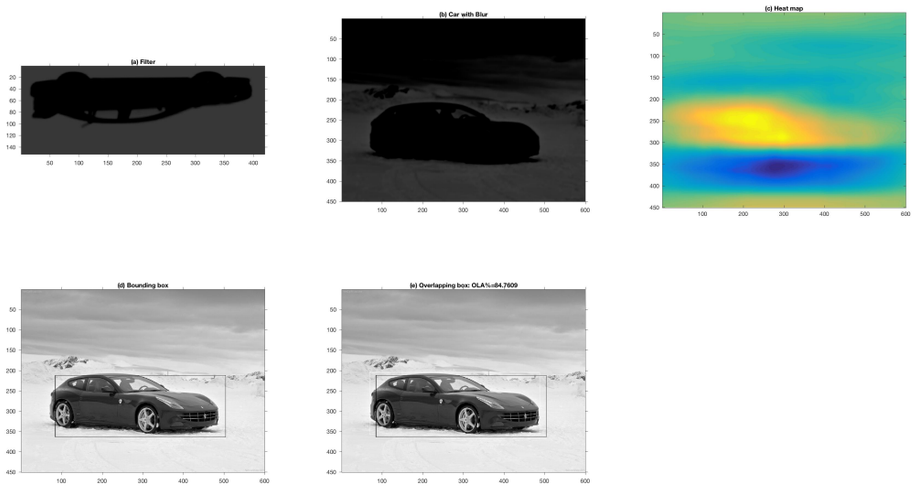

Figure 4: More Realistic Images: Car 2

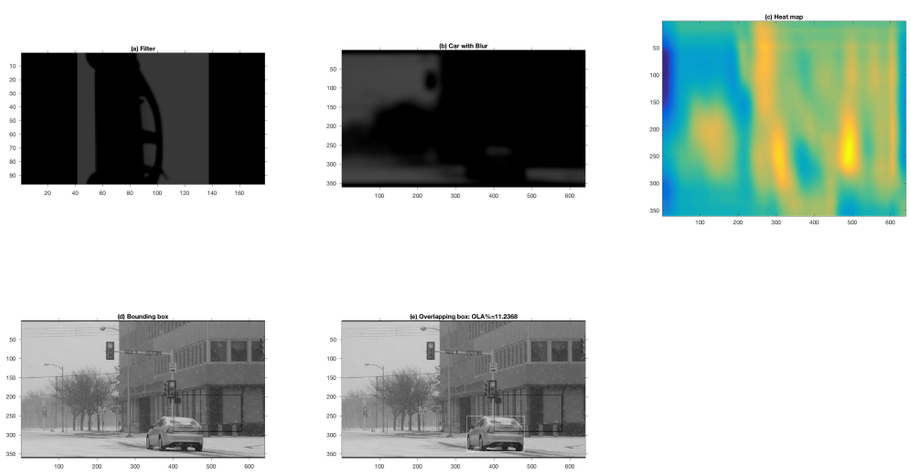

Figure 5: More Realistic Images: Car 3

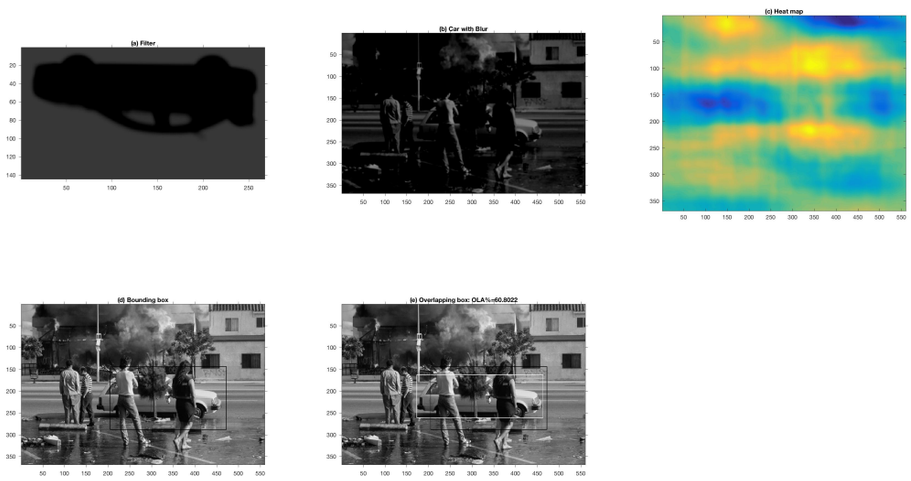

Figure 6: More Realistic Images: Car 4

4.4 Invariance [2pt]

Sol. It is not scale-invariant, blurring-invariant, flipping-invariant, and rotating-invariant. For example, different size of rescaling filters, different blurring parameters, with or without flipping, and rotating

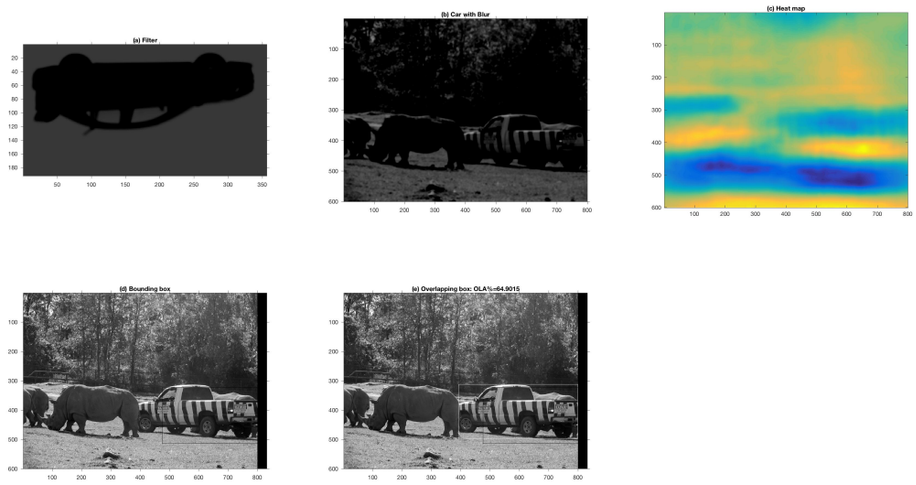

Figure 7: More Realistic Images: Car 5

angles can lead to different results. □

5 Canny Edge Detection [9pt]

Sol. Note that all the codes for problem 5 are recorded as Code 7 in the Appendix, and the figure is shown as Fig. 8.

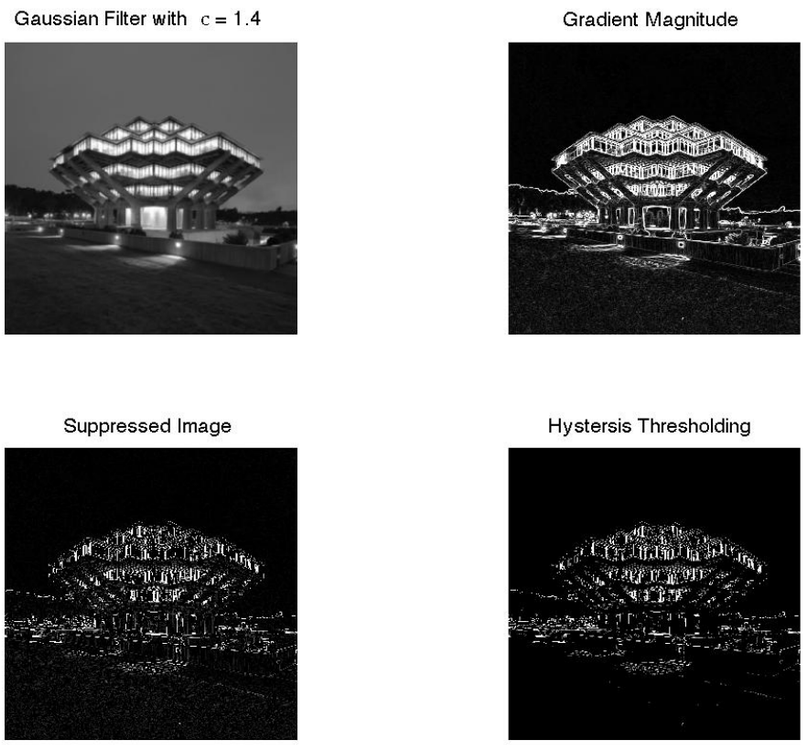

Figure 8: Problem 5

□

5.1 Smoothing [1pt]

Sol. I just use the given Gaussian kernel filter with $\sigma = 1.4$ to smooth the image, as shown in top-left subfigure in Fig. 8. Function *conv2* is used here. □

5.2 Gradient Computation [2pt]

Sol. With two gradient matrices s_x and s_y, I can find out G_x and G_y with function *conv2*. After that, gradient magnitude image can be drawn as the top-right subfigure in Fig. 8. □

5.3 Non Maximum suppression [3pt]

Sol. With G_θ, I can check whether current pixel can be considered as edge candidate or not in two of the eight directions. Corresponding figure are drawn in the bottom-left subfigure in Fig. 8. □

5.4 Hysteresis Thresholding [3pt]

Sol. To fulfill *Hysteresis Thresholding*, I use java linkedlist to implement breadth-first-search (BFS). The result is shown in the bottom-right subfigure in Fig. 8. □

6 Sparse Stereo Matching [27pt]

6.1 Corner Detection [8pt]

Sol. First I convert the RGB figures into gray-level ones, as shown in Fig. 9 and 11. Later, I smooth the images by smoothSTD parameter and use the windowsize parameter to calculate the C matrix. After that, I choose top 20 corners and label them with red circles, as shown in Fig. 10 and 12.

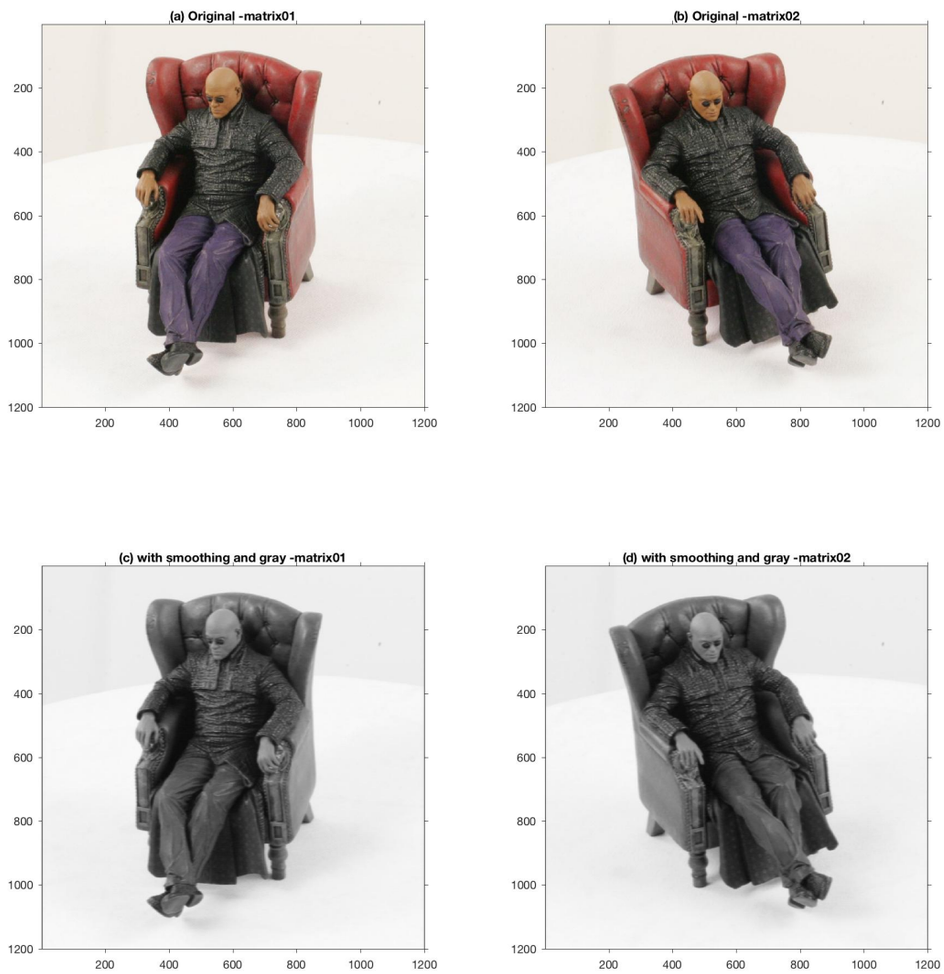

Figure 9: Corner Detection Matrix Original Figure

□

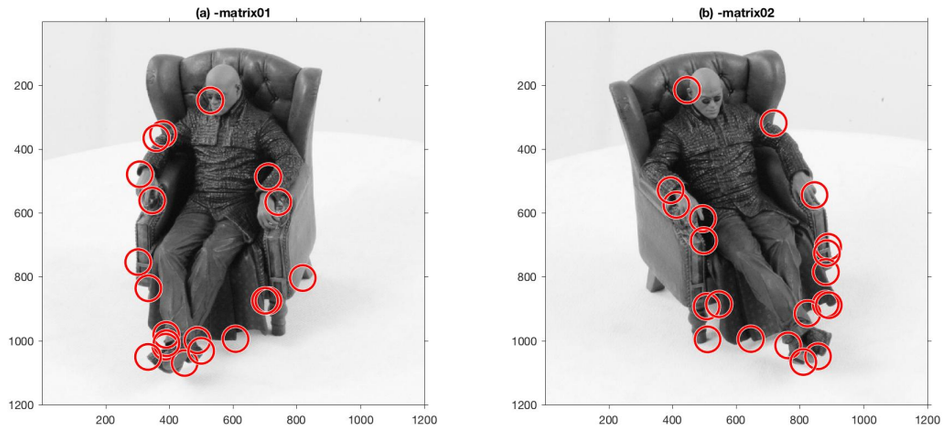

Figure 10: Corner Detection Matrix Result Figure

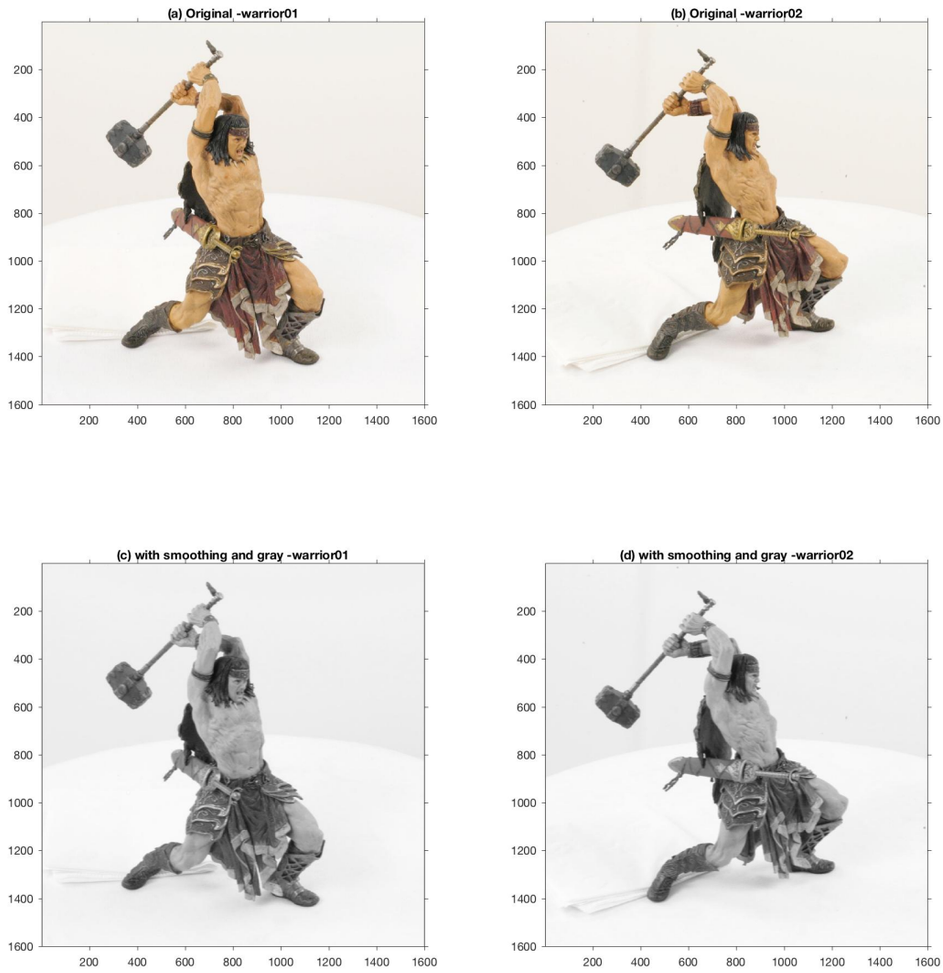

Figure 11: Corner Detection Warrior Original Figure

6.2 SSD Matching [2pt]

Sol. I just calculate the value $\sum_{i,j}[f(i,j)-g(i,j)]^2$, where f and g are some specific points in two different figures, as written in Code 8 in Appendix. □

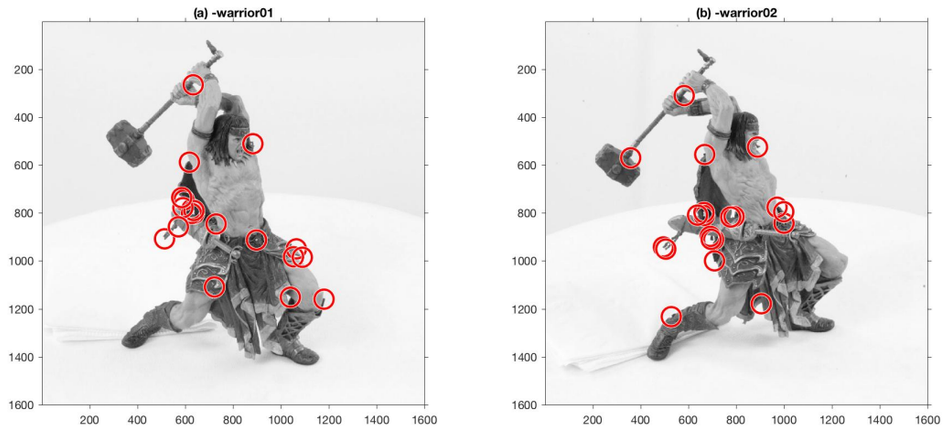

Figure 12: Corner Detection Warrior Result Figure

6.3 Naive Matching [4pt]

Sol. After using corner detection to find out the top-ten corner points in two figures, I can use the SSD matching algorithm to find out the best pairing results with specific window size. Figures for Matrix and Warrior are shown in Fig. 13 and 14, respectively.

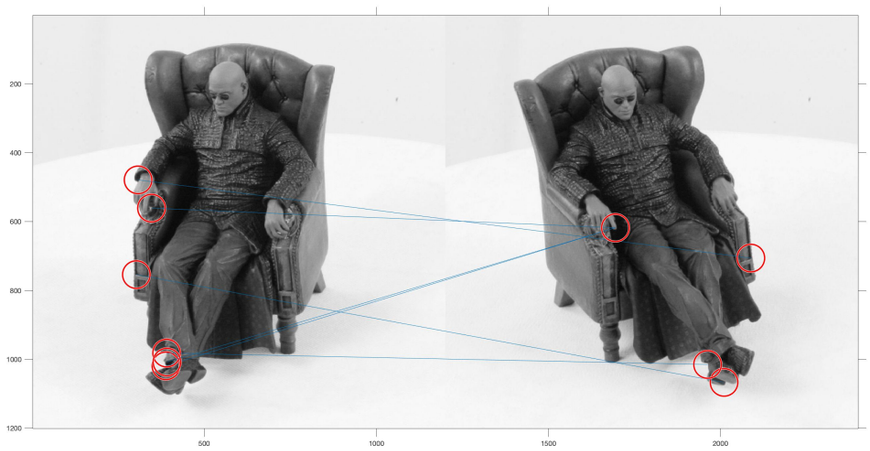

Figure 13: Naive Matching Matrix Figure

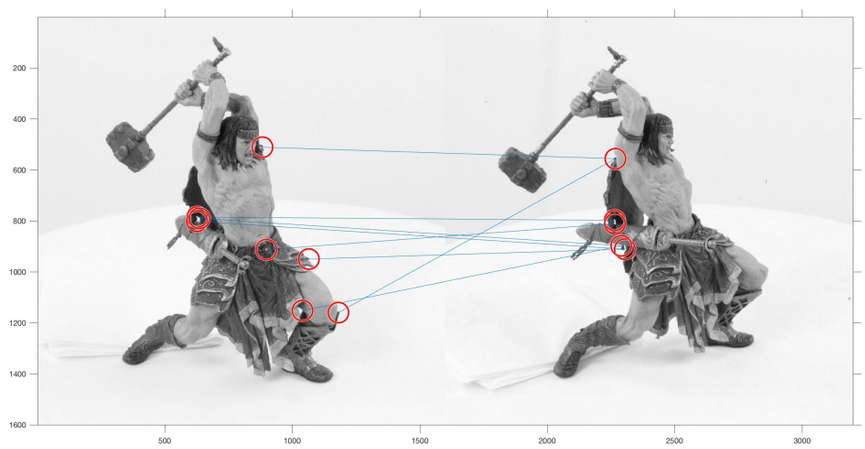

Figure 14: Naive Matching Warrior Figure

□

6.4 Epipolar Geometry [3pt]

Sol. With the *fund.m*, I can find out the corresponding epipolar lines in second figure from each point in first figure, and vice-versa. After that, I can use the *linePts.m* to find out the line boundary and draw it, as shown in Fig. 15, 16, 17, 18.

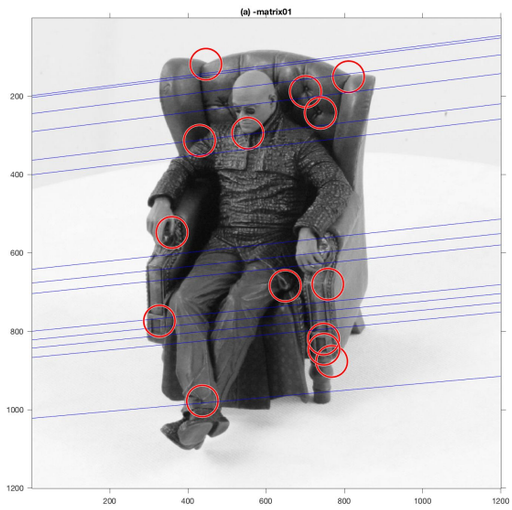

Figure 15: Epipolar Geometry Matrix Figure 1

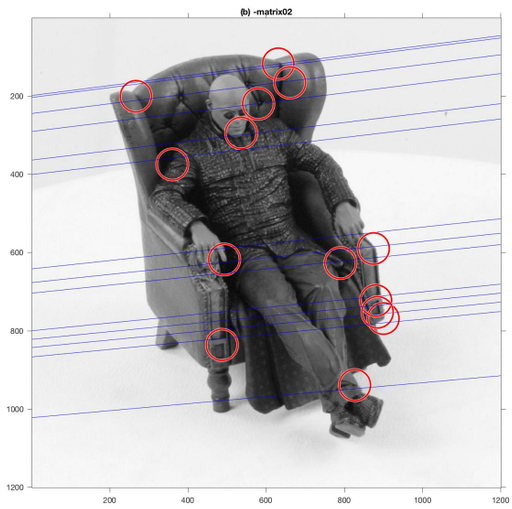

Figure 16: Epipolar Geometry Matrix Figure 2

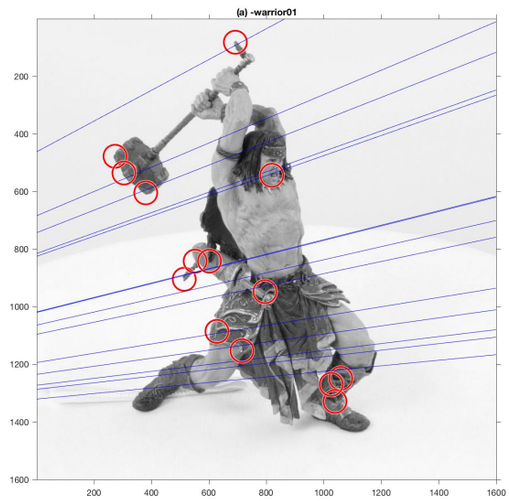

Figure 17: Epipolar Geometry Warrior Figure 1

□

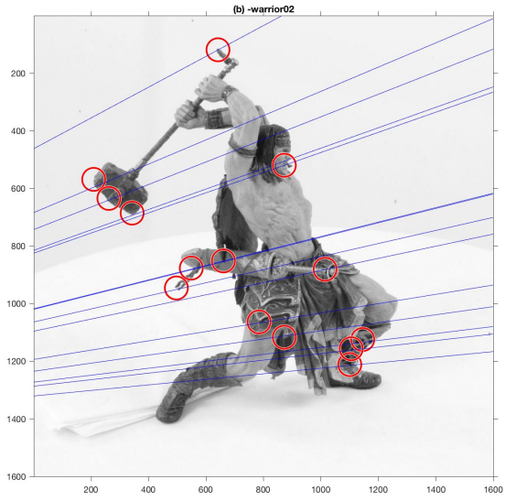

Figure 18: Epipolar Geometry Warrior Figure 2

6.5 Epipolar Geometry Based Matching [5pt]

Sol. From last subsection, I successively find out the corresponding epipolar line for each given corner point. Later, I will search the best-fit point on that epipolar line with a specific window size. At the end, I can link the pair of points and show it in Fig. 19 and 20.

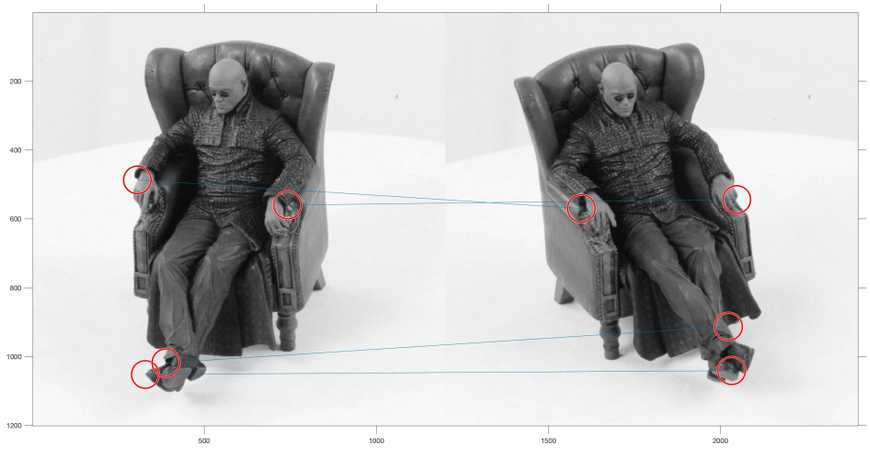

Figure 19: Epipolar Geometry Based Matching Matrix Figure

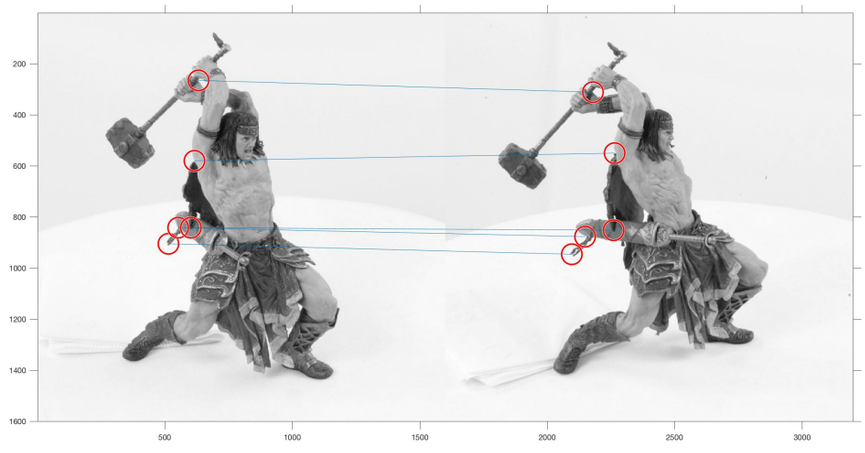

Figure 20: Epipolar Geometry Based Matching Warrior Figure

□

6.6 Triangulation [5pt]

Sol. From previous subsections, I successively find out the corresponding pair of corner points. Later, I cam use the projection matrix to solve linear algebra equation. At the end, I need to calculate the left null space. By doing so, I can find out the precise point in 3D coordinate.

(Due to the time factor, I just put the pseudo algorithm here instead of implementing the algorithm and generate the result picture.) □

Appendix

```
% ***** 4.1 Filter *****

% --- Input image file ---
filter_img = im2double( imread('data/filter.jpg') );
toy_img = im2double( imread('data/toy.png') );

% --- Modification ---
filter_img = filter_img - mean(filter_img(:));
toy_img_mean = toy_img - mean(toy_img(:));

% --- Convolution ---
ans_img(:,:,:) = conv2(toy_img_mean(:,:,:), filter_img(:,:,:), 'same');

% --- Bounding Box Detection and Plotting ---
toy_img_box(:,:,:) = toy_img(:,:,:);
box_length = 127;
box_height = 127;
[x1,y1] = box_Pre(ans_img,100,300,60,200, box_length, box_height);
toy_img_box = boxDrawing(toy_img_box,x1,y1, box_length, box_height, 0);
[x2,y2] = box_Pre(ans_img,20,150,300,400, box_length, box_height);
toy_img_box = boxDrawing(toy_img_box,x2,y2, box_length, box_height, 0);
[x3,y3] = box_Pre(ans_img,200,330,320,430, box_length, box_height);
toy_img_box = boxDrawing(toy_img_box,x3,y3, box_length, box_height, 0);

% --- Plotting ---
graph = figure('visible', 'off');
set(gcf,'units','points','position',[0,0,4000,2000]);

subplot (1,2,1);
imagesc(ans_img);
title('(a) Heat map');

subplot(1,2,2);
imshow(toy_img_box);
axis on;
title('(b) Bounding box');

saveas(graph, 'Output/4.1Warmup.jpg');



% ***** 4.2 Detection Error *****
toy_img_box_err1(:,:,:) = toy_img(:,:,:);
toy_img_box_err2(:,:,:) = toy_img(:,:,:);
toy_img_box_err3(:,:,:) = toy_img(:,:,:);

box_length = 127;
box_height = 127;

x_grou1 = 100;
y_grou1 = 100;
overlappingArea1=errDetecion(x_grou1,y_grou1,x1,y1, box_length, box_height);
toy_img_box_err1 = boxDrawing(toy_img_box_err1,x_grou1,y_grou1, box_length,
    box_height, 0.5);
toy_img_box_err1 = boxDrawing(toy_img_box_err1,x1,y1, box_length, box_height,
    0);

x_grou2 = 80;
y_grou2 = 60;
overlappingArea2=errDetecion(x_grou2,y_grou2,x1,y1, box_length, box_height);
toy_img_box_err2 = boxDrawing(toy_img_box_err2,x_grou2,y_grou2, box_length,
    box_height, 0.5);
toy_img_box_err2 = boxDrawing(toy_img_box_err2,x1,y1, box_length, box_height,
    0);

x_grou3 = 150;
```

```
y_grou3 = 200;
overlappingArea3=errDetecion(x_grou3,y_grou3,x1,y1, box_length, box_height);
toy_img_box_err3 = boxDrawing(toy_img_box_err3,x_grou3,y_grou3, box_length,
    box_height, 0.5);
toy_img_box_err3 = boxDrawing(toy_img_box_err3,x1,y1, box_length, box_height,
    0);

graph = figure('visible', 'off');
set(gcf,'units','points','position',[0,0,2000,1000]);
subplot (2,2,1);
imshow(toy_img_box_err1);
axis on;
title(strcat('OLA% with GT1=', string(overlappingArea1)));

subplot(2,2,2);
imshow(toy_img_box_err2);
axis on;
title(strcat('OLA% with GT2=', string(overlappingArea2)));

subplot(2,2,3);
imshow(toy_img_box_err3);
axis on;
title(strcat('OLA% with GT3=', string(overlappingArea3)));

saveas(graph, 'Output/4.2Error.jpg');


function [Xstart,Ystart] = box_Pre(inten_img,x1,x2,y1,y2, dx, dy)
temp = inten_img(x1:x2,y1:y2);
[M,I] = max(temp(:));
[I_row, I_col] = ind2sub(size(temp),I);
Xstart = I_row - round(dx/2) + x1;
Ystart = I_col - round(dy/2) + y1;
end

function res_img = boxDrawing(toy_img_box, x, y, dx, dy,color)
toy_img_box(x:x+dx,y) = color;
toy_img_box(x:x+dx,y+dy) = color;
toy_img_box(x,y:y+dy) = color;
toy_img_box(x+dx,y:y+dy) = color;
res_img = toy_img_box;
end

function overlappingArea=errDetecion(x_grou,y_grou,x_pre,y_pre, dx, dy)

X_union = dx + max(x_grou, x_pre) - min(x_grou, x_pre);
Y_union = dy + max(y_grou, y_pre) - min(y_grou, y_pre);

X_substract = abs(x_grou - x_pre);
Y_substract = abs(y_grou - y_pre);

X_intersec = dx - X_substract;
Y_intersec = dy - Y_substract;

numerator = X_intersec * Y_intersec;
demoniator = X_union * Y_union - 2 * (X_substract * Y_substract);

overlappingArea = numerator / demoniator *100;
end
```

Listing 1: Code for 4-1 and 4-2

```
% ***** 4.3-1 Realistic Image *****

% --- Input image file ---
filter_img = im2double( imread('data/cartemplate.jpg') );
car_img = im2double( imread('data/car1.jpg') );

fileID = fopen('data/car1.txt','r');
Coord = fscanf(fileID,'%d %d',[2 Inf]);
Coord = Coord';
fclose(fileID);

```

```
filter_img = filter_img - mean(filter_img(:));
car_img_mean = car_img - mean(car_img(:));

%  --- Rescale ---
%filter_img = imresize(filter_img, 0.19);
filter_img = imresize(filter_img, [100,347]);

% --- Blur with std ---
filter_img = imgaussfilt(filter_img, 4);
car_img_mean = imgaussfilt(car_img_mean, 4);

% --- Rotation ---
filter_img = imrotate(filter_img,180,'bilinear','crop');

% --- Convolution ---
ans_img = conv2(car_img_mean, filter_img, 'same');

%  --- Prediction ---
car_img_box(:,:) = car_img(:,:);
[img_x, img_y] = size(car_img);
[box_x, box_y] = size(filter_img);
[x1,y1] = box_Pre(ans_img, box_x, box_y);
car_img_box = boxDrawing(car_img_box,x1,y1, box_x, box_y, 0);

%  --- Overlapping Area ---
car_img_box_err(:,:) = car_img(:,:);
overlappingArea = errDetecion(Coord(1,2),Coord(1,1), Coord(2,2),Coord(2,1), x1
    , y1, x1+box_x, y1+box_y);
car_img_box_err = boxDrawing(car_img_box_err, Coord(1,2), Coord(1,1), Coord
    (2,2)-Coord(1,2), Coord(2,1)-Coord(1,1), 1);
car_img_box_err = boxDrawing(car_img_box_err,x1,y1, box_x, box_y, 0);

disp(overlappingArea);

% --- Plotting ---
graph = figure('visible', 'off');
set(gcf,'units','points','position',[0,0,2000,1000]);

subplot(2,3,1);
imshow(filter_img);
axis on;
title('(a) Filter');

subplot(2,3,2);
imshow(car_img_mean);
axis on;
title('(b) Car with Blur');

subplot(2,3,3);
imagesc(ans_img);
title('(c) Heat map');

subplot(2,3,4);
imshow(car_img_box);
axis on;
title('(d) Bounding box');

subplot(2,3,5);
imshow(car_img_box_err);
axis on;
title(strcat('(e) Overlapping box: OLA%= ', string(overlappingArea)));

saveas(graph, 'Output/4.3_Car1.jpg');



function [Xstart,Ystart] = box_Pre(inten_img, dx, dy)
[M,I] = max(inten_img(:));
[I_row, I_col] = ind2sub(size(inten_img),I);
Xstart = I_row - round(dx/2);
Ystart = I_col - round(dy/2);
end
```

```

function res_img = boxDrawing(toy_img_box, x, y, dx, dy, color)
toy_img_box(x:x+dx,y) = color;
toy_img_box(x:x+dx,y+dy) = color;
toy_img_box(x,y:y+dy) = color;
toy_img_box(x+dx,y:y+dy) = color;
res_img = toy_img_box;
end

function len = overlapLength(x1,x2,y1,y2)
if (x1<y1) && (y2<x2)
    len = abs(y2-y1);
elseif (x1>y1) && (y2>x2)
    len = abs(x2-x1);
elseif max(y2-x1) > max(x2-y1)
    len = abs(x2-y1);
elseif max(y2-x1) < max(x2-y1)
    len = abs(y2-x1);
end
end

function overlappingArea=errDetecion(x0,y0,x1,y1, p0,q0,p1,q1)

X_intersect = overlapLength(x0,x1, p0,p1);
Y_intersect = overlapLength(y0,y1, q0,q1);

numerator = X_intersect * Y_intersect;
demoniator = abs(x0-x1)*abs(y0-y1) + abs(p0-p1)*abs(q0-q1) - numerator;

overlappingArea = numerator / demoniator *100;
end
```

Listing 2: Code for 4-3 Car 1

```

% ***** 4.3-2 Realistic Image *****

% --- Input image file ---
filter_img = im2double( imread('data/cartemplate.jpg') );
car_img = im2double( imread('data/car2.jpg') );

fileID = fopen('data/car2.txt','r');
Coord = fscanf(fileID,'%d %d',[2 Inf]);
Coord = Coord';
fclose(fileID);


filter_img = filter_img - mean(filter_img(:));
car_img_mean = car_img - mean(car_img(:));

%  --- Rescale to 18% ---
%filter_img = imresize(filter_img, 0.18);
filter_img = imresize(filter_img, [152,419]);

% --- Blur with std = 2 ---
filter_img = imgaussfilt(filter_img, 2);
car_img_mean = imgaussfilt(car_img_mean, 2);

% --- Rotation ---
filter_img = imrotate(filter_img,180,'bilinear','crop');

% --- Flip ---
%filter_img = flip(filter_img,2);

% --- Convolution ---
ans_img = conv2(car_img_mean, filter_img, 'same');

%  --- Prediction ---
car_img_box(:,:) = car_img(:,:);
[img_x, img_y] = size(car_img);
[box_x, box_y] = size(filter_img);
[x1,y1] = box_Pre(ans_img, box_x, box_y);
car_img_box = boxDrawing(car_img_box,x1,y1, box_x, box_y, 0);

```

```
% --- Overlapping Area ---
car_img_box_err(:,:) = car_img(:,:);
overlappingArea = errDetecion(Coord(1,2),Coord(1,1), Coord(2,2),Coord(2,1), x1
    , y1, x1+box_x, y1+box_y);
car_img_box_err = boxDrawing(car_img_box_err, Coord(1,2), Coord(1,1), Coord
    (2,2)-Coord(1,2), Coord(2,1)-Coord(1,1), 1);
car_img_box_err = boxDrawing(car_img_box_err,x1,y1, box_x, box_y, 0);

disp(overlappingArea);

% --- Plotting ---
graph = figure('visible', 'off');
set(gcf,'units','points','position',[0,0,2000,1000]);

subplot(2,3,1);
imshow(filter_img);
axis on;
title('(a) Filter');

subplot(2,3,2);
imshow(car_img_mean);
axis on;
title('(b) Car with Blur');

subplot(2,3,3);
imagesc(ans_img);
title('(c) Heat map');

subplot(2,3,4);
imshow(car_img_box);
axis on;
title('(d) Bounding box');

subplot(2,3,5);
imshow(car_img_box_err);
axis on;
title(strcat('(e) Overlapping box: OLA%= ', string(overlappingArea)));

saveas(graph, 'Output/4.3_Car2.jpg');









function [Xstart,Ystart] = box_Pre(inten_img, dx, dy)
[M,I] = max(inten_img(:));
[I_row, I_col] = ind2sub(size(inten_img),I);
Xstart = I_row - round(dx/2);
Ystart = I_col - round(dy/2);
end

function res_img = boxDrawing(toy_img_box, x, y, dx, dy, color)
toy_img_box(x:x+dx,y) = color;
toy_img_box(x:x+dx,y+dy) = color;
toy_img_box(x,y:y+dy) = color;
toy_img_box(x+dx,y:y+dy) = color;
res_img = toy_img_box;
end

function len = overlapLength(x1,x2,y1,y2)
if (x1<y1) && (y2<x2)
    len = abs(y2-y1);
elseif (x1>y1) && (y2>x2)
    len = abs(x2-x1);
elseif max(y2-x1) > max(x2-y1)
    len = abs(x2-y1);
elseif max(y2-x1) < max(x2-y1)
    len = abs(y2-x1);
end
end

```

```
function overlappingArea=errDetecion(x0,y0,x1,y1, p0,q0,p1,q1)
X_intersect = overlapLength(x0,x1, p0,p1);
Y_intersect = overlapLength(y0,y1, q0,q1);

numerator = X_intersect * Y_intersect;
demoniator = abs(x0-x1)*abs(y0-y1) + abs(p0-p1)*abs(q0-q1) - numerator;

overlappingArea = numerator / demoniator *100;
end
```

Listing 3: Code for 4-3 Car 2

```
% ***** 4.3-3 Realistic Image *****

% --- Input image file ---
filter_img = im2double( imread('data/cartemplate.jpg') );
car_img = im2double( imread('data/car3.jpg') );

fileID = fopen('data/car3.txt','r');
Coord = fscanf(fileID,'%d %d',[2 Inf]);
Coord = Coord';
fclose(fileID);


filter_img = filter_img - mean(filter_img(:));
car_img_mean = car_img - mean(car_img(:));

%  --- Rescale ---
filter_img = imresize(filter_img, 0.1);

% --- Blur with std ---
filter_img = imgaussfilt(filter_img, 1);
car_img_mean = imgaussfilt(car_img_mean, 10);

% --- Rotation ---
filter_img = imrotate(filter_img,-90,'bilinear','crop');

% --- Convolution ---
ans_img = conv2(car_img_mean, filter_img, 'same');

%  --- Prediction ---
car_img_box(:,:) = car_img(:,:);
[img_x, img_y] = size(car_img);
[box_x, box_y] = size(filter_img);
[x1,y1] = box_Pre(ans_img, box_x, box_y);
car_img_box = boxDrawing(car_img_box,x1,y1, box_x, box_y, 0);

%  --- Overlapping Area ---
car_img_box_err(:,:) = car_img(:,:);
overlappingArea = errDetecion(Coord(1,2),Coord(1,1), Coord(2,2),Coord(2,1), x1
    , y1, x1+box_x, y1+box_y);
car_img_box_err = boxDrawing(car_img_box_err, Coord(1,2), Coord(1,1), Coord
    (2,2)-Coord(1,2), Coord(2,1)-Coord(1,1), 1);
car_img_box_err = boxDrawing(car_img_box_err,x1,y1, box_x, box_y, 0);

disp(overlappingArea);

% --- Plotting ---
graph = figure('visible', 'off');
set(gcf,'units','points','position',[0,0,2000,1000]);

subplot(2,3,1);
imshow(filter_img);
axis on;
title('(a) Filter');

subplot(2,3,2);
imshow(car_img_mean);
axis on;
title('(b) Car with Blur');

subplot(2,3,3);
imagesc(ans_img);
```

```
title('(c) Heat map');

subplot(2,3,4);
imshow(car_img_box);
axis on;
title('(d) Bounding box');

subplot(2,3,5);
imshow(car_img_box_err);
axis on;
title(strcat('(e) Overlapping box: OLA%= ', string(overlappingArea)));

saveas(graph, 'Output/4.3_Car3.jpg');







function [Xstart,Ystart] = box_Pre(inten_img, dx, dy)
[M,I] = max(inten_img(:));
[I_row, I_col] = ind2sub(size(inten_img),I);
Xstart = I_row - round(dx/2);
Ystart = I_col - round(dy/2);
end

function res_img = boxDrawing(toy_img_box, x, y, dx, dy, color)
toy_img_box(x:x+dx,y) = color;
toy_img_box(x:x+dx,y+dy) = color;
toy_img_box(x,y:y+dy) = color;
toy_img_box(x+dx,y:y+dy) = color;
res_img = toy_img_box;
end

function len = overlapLength(x1,x2,y1,y2)
if (x1<y1) && (y2<x2)
    len = abs(y2-y1);
elseif (x1>y1) && (y2>x2)
    len = abs(x2-x1);
elseif max(y2-x1) > max(x2-y1)
    len = abs(x2-y1);
elseif max(y2-x1) < max(x2-y1)
    len = abs(y2-x1);
end
end

function overlappingArea=errDetecion(x0,y0,x1,y1, p0,q0,p1,q1)
X_intersect = overlapLength(x0,x1, p0,p1);
Y_intersect = overlapLength(y0,y1, q0,q1);

numerator = X_intersect * Y_intersect;
demoniator = abs(x0-x1)*abs(y0-y1) + abs(p0-p1)*abs(q0-q1) - numerator;

overlappingArea = numerator / demoniator *100;
end
```

Listing 4: Code for 4-3 Car 3

```
% ***** 4.3-3 Realistic Image *****

% --- Input image file ---
filter_img = im2double( imread('data/cartemplate.jpg') );
car_img = im2double( imread('data/car4.jpg') );

fileID = fopen('data/car4.txt','r');
Coord = fscanf(fileID,'%d %d',[2 Inf]);
Coord = Coord';
fclose(fileID);


filter_img = filter_img - mean(filter_img(:));
car_img_mean = car_img - mean(car_img(:));

```

```
% --- Rescale ---
%filter_img = imresize(filter_img, 0.15);
filter_img = imresize(filter_img, [98,290]);

% --- Blur with std ---
filter_img = imgaussfilt(filter_img, 3);
car_img_mean = imgaussfilt(car_img_mean, 1);

% --- Rotation ---
filter_img = imrotate(filter_img, 180, 'bilinear','crop');

% --- Flip ---
filter_img = flip(filter_img,1);

% --- Convolution ---
ans_img = conv2(car_img_mean, filter_img, 'same');

% --- Prediction ---
car_img_box(:,:) = car_img(:,:);
[img_x, img_y] = size(car_img);
[box_x, box_y] = size(filter_img);

[x1,y1] = box_Pre(ans_img, box_x, box_y);
car_img_box = boxDrawing(car_img_box,x1,y1, box_x, box_y, 0);

% --- Overlapping Area ---
car_img_box_err(:,:) = car_img(:,:);
overlappingArea = errDetecion(Coord(1,2),Coord(1,1), Coord(2,2),Coord(2,1), x1
    , y1, x1+box_x, y1+box_y);
car_img_box_err = boxDrawing(car_img_box_err, Coord(1,2), Coord(1,1), Coord
    (2,2)-Coord(1,2), Coord(2,1)-Coord(1,1), 1);
car_img_box_err = boxDrawing(car_img_box_err,x1,y1, box_x, box_y, 0);

disp(overlappingArea);

% --- Plotting ---
graph = figure('visible', 'off');
set(gcf,'units','points','position',[0,0,2000,1000]);

subplot(2,3,1);
imshow(filter_img);
axis on;
title('(a) Filter');

subplot(2,3,2);
imshow(car_img_mean);
axis on;
title('(b) Car with Blur');

subplot(2,3,3);
imagesc(ans_img);
title('(c) Heat map');

subplot(2,3,4);
imshow(car_img_box);
axis on;
title('(d) Bounding box');

subplot(2,3,5);
imshow(car_img_box_err);
axis on;
title(strcat('(e) Overlapping box: OLA%= ', string(overlappingArea)));

saveas(graph, 'Output/4.3_Car4.jpg');




function [Xstart,Ystart] = box_Pre(inten_img, dx, dy)
[M,I] = max(inten_img(:));
```

```
[I_row, I_col] = ind2sub(size(inten_img),I);
Xstart = I_row - round(dx/2);
Ystart = I_col - round(dy/2);
end

function res_img = boxDrawing(toy_img_box, x, y, dx, dy, color)
toy_img_box(x:x+dx,y) = color;
toy_img_box(x:x+dx,y+dy) = color;
toy_img_box(x,y:y+dy) = color;
toy_img_box(x+dx,y:y+dy) = color;
res_img = toy_img_box;
end

function len = overlapLength(x1,x2,y1,y2)
if (x1<y1) && (y2<x2)
    len = abs(y2-y1);
elseif (x1>y1) && (y2>x2)
    len = abs(x2-x1);
elseif max(y2-x1) > max(x2-y1)
    len = abs(x2-y1);
elseif max(y2-x1) < max(x2-y1)
    len = abs(y2-x1);
end
end

function overlappingArea=errDetecion(x0,y0,x1,y1, p0,q0,p1,q1)
X_intersect = overlapLength(x0,x1, p0,p1);
Y_intersect = overlapLength(y0,y1, q0,q1);

numerator = X_intersect * Y_intersect;
demoniator = abs(x0-x1)*abs(y0-y1) + abs(p0-p1)*abs(q0-q1) - numerator;

overlappingArea = numerator / demoniator *100;
end
```

Listing 5: Code for 4-3 Car 4

```
% ***** 4.3-3 Realistic Image *****

% --- Input image file ---
filter_img = im2double( imread('data/cartemplate.jpg') );
car_img = im2double( imread('data/car5.jpg') );

fileID = fopen('data/car5.txt','r');
Coord = fscanf(fileID,'%d %d',[2 Inf]);
Coord = Coord';
fclose(fileID);


filter_img = filter_img - mean(filter_img(:));
car_img_mean = car_img - mean(car_img(:));

%  --- Rescale ---
filter_img = imresize(filter_img, 0.2);
%filter_img = imresize(filter_img, [186,407]);

% --- Blur with std ---
filter_img = imgaussfilt(filter_img, 2);
car_img_mean = imgaussfilt(car_img_mean, 2);

% --- Rotation ---
filter_img = imrotate(filter_img, 180,'bilinear','crop');

% --- Flip ---
%filter_img = flip(filter_img,2);

% --- Convolution ---
ans_img = conv2(car_img_mean, filter_img, 'same');

%  --- Prediction ---
car_img_box(:,:) = car_img(:,:);
[img_x, img_y] = size(car_img);
[box_x, box_y] = size(filter_img);
```

```
[x1,y1] = box_Pre(ans_img, box_x, box_y);
car_img_box = boxDrawing(car_img_box,x1,y1, box_x, box_y, 0);

% --- Overlapping Area ---
car_img_box_err(:,:) = car_img(:,:);
overlappingArea = errDetecion(Coord(1,2),Coord(1,1), Coord(2,2),Coord(2,1), x1
    , y1, x1+box_x, y1+box_y);
car_img_box_err = boxDrawing(car_img_box_err, Coord(1,2), Coord(1,1), Coord
    (2,2)-Coord(1,2), Coord(2,1)-Coord(1,1), 1);
car_img_box_err = boxDrawing(car_img_box_err,x1,y1, box_x, box_y, 0);

disp(overlappingArea);

% --- Plotting ---
graph = figure('visible', 'off');
set(gcf,'units','points','position',[0,0,2000,1000]);

subplot(2,3,1);
imshow(filter_img);
axis on;
title('(a) Filter');

subplot(2,3,2);
imshow(car_img_mean);
axis on;
title('(b) Car with Blur');

subplot(2,3,3);
imagesc(ans_img);
title('(c) Heat map');

subplot(2,3,4);
imshow(car_img_box);
axis on;
title('(d) Bounding box');

subplot(2,3,5);
imshow(car_img_box_err);
axis on;
title(strcat('(e) Overlapping box: OLA%= ', string(overlappingArea)));

saveas(graph, 'Output/4.3_Car5.jpg');




function [Xstart,Ystart] = box_Pre(inten_img, dx, dy)
[M,I] = max(inten_img(:));
[I_row, I_col] = ind2sub(size(inten_img),I);
Xstart = I_row - round(dx/2);
Ystart = I_col - round(dy/2);
end

function res_img = boxDrawing(toy_img_box, x, y, dx, dy, color)
toy_img_box(x:x+dx,y) = color;
toy_img_box(x:x+dx,y+dy) = color;
toy_img_box(x,y:y+dy) = color;
toy_img_box(x+dx,y:y+dy) = color;
res_img = toy_img_box;
end

function len = overlapLength(x1,x2,y1,y2)
if (x1<y1) && (y2<x2)
    len = abs(y2-y1);
elseif (x1>y1) && (y2>x2)
    len = abs(x2-x1);
elseif max(y2-x1) > max(x2-y1)
    len = abs(x2-y1);
elseif max(y2-x1) < max(x2-y1)
    len = abs(y2-x1);
```

```
end
end

function overlappingArea=errDetecion(x0,y0,x1,y1, p0,q0,p1,q1)
X_intersect = overlapLength(x0,x1, p0,p1);
Y_intersect = overlapLength(y0,y1, q0,q1);

numerator = X_intersect * Y_intersect;
demoniator = abs(x0-x1)*abs(y0-y1) + abs(p0-p1)*abs(q0-q1) - numerator;

overlappingArea = numerator / demoniator *100;
end
```

Listing 6: Code for 4-3 Car 5

```
%% Data preparation
addpath('../dat/');

image = imread('geisel.jpeg');
image = im2double(image);
[nrow, ncol] = size(image);

res = figure('visible', 'off');


%% Smoothing [1pt]
K = 1/159 * [2 4 5 4 2; 4 9 12 9 4; 5 12 15 12 5; 4 9 12 9 4; 2 4 5 4 2];
smoothImg = conv2(image, K, 'same');
subplot(2,2,1);
imshow(smoothImg);
title('Gaussian Filter with {\sigma} = 1.4');


%% Gradient Computation [2pt]
sx = [-1 0 1; -2 0 2; -1 0 1];
sy = [-1 -2 -1; 0 0 0; 1 2 1];
Gx = conv2(image, sx, 'same');
Gy = conv2(image, sy, 'same');
G = zeros(nrow, ncol);
Gt = zeros(nrow, ncol);
for i = 1:nrow
    for j = 1:ncol
        G(i,j) = sqrt(Gx(i,j)^2 + Gy(i,j)^2);
        Gt(i,j) = atan2(Gy(i,j), Gx(i,j));
    end
end
subplot(2,2,2);
imshow(G);
title('Gradient Magnitude');


%% Non Maximum suppression [3pt]
dir = [[0, 1]; [-1, 1]; [-1, 0]; [-1, -1]; [0, -1]; [1, -1]; [1, 0]; [1, 1]];
ts = pi/4;

nm = zeros(nrow, ncol);

for i = 2:(nrow-1)
    for j = 2:(ncol-1)
        k = round(Gt(i,j) / ts) + 4;

        idx1 = int32(mod(k, 8)+1);
        idx2 = int32(mod(k+4, 8)+1);
        d1 = dir(idx1, :);
        d2 = dir(idx2, :);

        if G(i,j) > G(i+d1(1), j+d1(2)) ...
                && G(i,j) > G(i+d2(1), j+d2(2))
            nm(i,j) = G(i,j);
        end
    end
end
subplot(2,2,3);
```

```
imshow(nm);
title('Suppressed Image');


%% Hysteresis Thresholding [3pt]
import java.util.LinkedList
q = LinkedList();

th = 80/256;
tl = 50/256;
ht = zeros(nrow, ncol);
for i = 1:nrow
    for j = 1:ncol
        if nm(i,j) >= th
            q.add([i, j]);
            ht(i,j) = nm(i,j);
        end
    end
end

while (q.size() ~= 0)
    p = q.getFirst(); q.remove();
    r = p(1); c = p(2);
    for k = 1:8
        nr = r + dir(k, 1);
        nc = c + dir(k, 2);
        if nr == 0 || nc == 0 || nr > nrow || nc > ncol
            % do nothing
        elseif nm(nr, nc) >= tl && ht(nr, nc) == 0
            ht(nr, nc) = nm(nr, nc);
        end
    end
end
subplot(2,2,4);
imshow(ht);
title('Hystersis Thresholding');


%% Save Image
saveas(res, '../res/prob_5.jpg');
```

Listing 7: Code for problem 5

```
% ***** 6.1-2 Corner Detection *****

% --- Input image file ---
load('data/dino2.mat')
load('data/matrix2.mat')
load('data/warrior2.mat')

% --- Parameters ---
nCorners = 20;
smoothSTD = 5;
windowSize = 10;
%image1 = dino01;
%image2 = dino02;

PlottingCorners(dino01, dino02, 'dino01','dino02', '6-1CornerDetection',
    nCorners, smoothSTD, windowSize);

%PlottingCorners(matrix01, matrix02, 'matrix01','matrix02', '6-1
    CornerDetection_matrix', nCorners, smoothSTD, windowSize);
%PlottingCorners(warrior01, warrior02, 'warrior01','warrior02', '6-1
    CornerDetection_warrior', nCorners, smoothSTD, windowSize);


function finish = PlottingCorners(image1, image2, name1, name2, fileName,
    nCorners, smoothSTD, windowSize);
corners1 = CornerDetect(image1, nCorners, smoothSTD, windowSize);
corners2 = CornerDetect(image2, nCorners, smoothSTD, windowSize);

[NCorners1, dim1] = size(corners1);
[NCorners2, dim2] = size(corners2);
```

```
27
28 % --- Save Corners result to .txt file ---
29 tempName = strcat( strcat( strcat(strcat('Output/Corners', fileName), '_'),
       name1), '.txt' );
30 save( tempName, 'corners1', '-ascii');
31
32 tempName = strcat( strcat( strcat(strcat('Output/Corners', fileName), '_'),
       name2), '.txt' );
33 save( tempName, 'corners2', '-ascii');
34
35
36 % --- Plotting Original Image ---
37 graph = figure('visible', 'off');
38 set(gcf,'units','points','position',[0,0,1000,1000]);
39
40 subplot(2,2,1)
41 imshow(image1);
42 axis on;
43 title( strcat('(a) Original - ', name1 ) );
44
45 subplot(2,2,2)
46 imshow(image2);
47 axis on;
48 title( strcat('(b) Original - ', name2 ) );
49
50 subplot(2,2,3)
51 imshow( smoothts(rgb2gray(image1), 'g', windowSize, smoothSTD) );
52 axis on;
53 title( strcat('(c) with smoothing and gray - ', name1 ) );
54
55 subplot(2,2,4)
56 imshow( smoothts(rgb2gray(image2), 'g', windowSize, smoothSTD) );
57 axis on;
58 title( strcat('(d) with smoothing and gray - ', name2 ) );
59
60 contcatString = strcat( strcat('Output/', fileName), '_Original.jpg');
61 saveas(graph, contcatString);
62
63
64 % --- Plotting Result Image ---
65 graph = figure('visible', 'off');
66 set(gcf,'units','points','position',[0,0,1000,500]);
67
68 subplot (1,2,1);
69 imshow(rgb2gray(image1));
70 axis on;
71 for i = 1:NCorners1
72 viscircles([corners1(i,2), corners1(i,1)],40);
73 end
74 title( strcat('(a) - ', name1) );
75
76 subplot (1,2,2);
77 imshow(rgb2gray(image2));
78 axis on;
79 for i = 1:NCorners2
80 viscircles([corners2(i,2), corners2(i,1)],40);
81 end
82 title( strcat('(b) - ', name2) );
83
84 contcatString = strcat( strcat('Output/', fileName), '_Result.jpg');
85 saveas(graph, contcatString);
86
87 finish = 1;
88 end
89
90 function corners = CornerDetect(Image, nCorners, smoothSTD, windowSize)
91
92 img = rgb2gray(Image);
93 [img_x, img_y] = size(img);
94 img = smoothts(img, 'g', windowSize, smoothSTD);
95
96 % ---Gradient ---
```

```
[Ix, Iy] = imgradientxy(img);
Ix2 = Ix.*Ix;
Iy2 = Iy.*Iy;
IxIy = Ix.*Iy;

EigMat = zeros(img_x, img_y);

start_x = 1 + round(windowSize/2);
start_y = 1 + round(windowSize/2);
end_x = img_x - round(windowSize/2)-1;
end_y = img_y - round(windowSize/2)-1;

for i = start_x : end_x
    for j = start_y : end_y
        start_i = i - round(windowSize/2);
        start_j = j - round(windowSize/2);
        end_i = i + round(windowSize/2);
        end_j = j + round(windowSize/2);

        C11 = sum(sum( Ix2(start_i:end_i, start_j:end_j) ));
        C12 = sum(sum( IxIy(start_i:end_i, start_j:end_j) ));
        C22 = sum(sum( Iy2(start_i:end_i, start_j:end_j) ));

        C = [C11, C12; C12, C22];
        e = eig(C);
        EigMat(i,j) = min(e(1), e(2));

    end
end

corners = zeros(nCorners,2);

i=1;
while (i <= nCorners)
    [M,I] = max(EigMat(:));
    [I_row, I_col] = ind2sub(size(EigMat),I);
    tooCloseFlag = 0;
    for j = 1:i
%        I_row, I_col
        if ( distance(I_row, I_col, corners(j,1), corners(j,2)) < windowSize)
            tooCloseFlag = 1;
            break;
        else
            tooCloseFlag = 0;
            continue
        end
    end

    if (tooCloseFlag == 1)
        EigMat(I_row,I_col) = 0;
        i = i-1;
    else
        corners(i,1) = I_row;
        corners(i,2) = I_col;
        EigMat(I_row,I_col) = 0;
    end

    i=i+1;
end

end

function dist = distance(x1,y1,x2,y2)
    dist = sqrt( (x1-x2)^2 + (y1-y2)^2 );
end
```

Listing 8: Code for problem 6-1

```
% ***** 6.2 SSD Matching *****

function ssd = SSDmatch(w1,w2)
[rows1, cols1] = size(w1);
[rows2, cols2] = size(w2);
```

```
6
7  if (rows1 ~= rows2 || cols1 ~= cols2)
8      ssd = 0;
9  else
10     Diff = w1 - w2;
11     ssd = sum(sum(Diff.*Diff));
12 end
13
14 end
```

Listing 9: Code for problem 6-2

```
1  % ***** 6.3 Naive Matching *****
2
3  % --- Input image file ---
4  load('data/dino2.mat')
5  load('data/matrix2.mat')
6  load('data/warrior2.mat')
7
8  % --- Image ---
9  %I1 = dino01;
10 %I2 = dino02;
11
12 % --- FileName ---
13 %fileName = '6-3NaiveMatching_Dino';
14
15 % --- Parameters ---
16 ncorners = 10;
17 smoothSTD = 5;
18 windowSize = 5;
19 R = 10;
20 SSDth = 1.2;
21
22
23 PlottingNaiveMatching(dino01, dino02, '6-3NaiveMatching_Dino', ncorners,
       smoothSTD, windowSize, R, SSDth);
24 %PlottingNaiveMatching(matrix01, matrix02, '6-3NaiveMatching_matrix', ncorners
       , smoothSTD, windowSize, R, SSDth);
25 %PlottingNaiveMatching(warrior01, warrior02, '6-3NaiveMatching_waerrior',
       ncorners, 5, 10, 10, 2);
26
27
28 function finish = PlottingNaiveMatching(I1,I2, fileName, ncorners, smoothSTD,
       windowSize, R, SSDth)
29 corners1 = CornerDetect(I1, ncorners, smoothSTD, windowSize);
30 corners2 = CornerDetect(I2, ncorners, smoothSTD, windowSize);
31
32 [I, corsSSD] = naiveCorrespondanceMatching(I1, I2, corners1, corners2, R,
       SSDth);
33
34
35 % --- Combine the two image ---
36 I1 = rgb2gray(I1);
37 I2 = rgb2gray(I2);
38 [I1_x, I1_y] = size(I1);
39 [I2_x, I2_y] = size(I2);
40 I_combine = zeros(I1_x, I1_y *2);
41 I_combine(1:I1_x, 1:I1_y)= I1(1:I1_x, 1:I1_y);
42 I_combine(1:I2_x, I1_y + 1 : I1_y + I2_y)= I2(1:I2_x, 1 : I2_y);
43 [Nrows, I_width] = size(I);
44
45 graph = figure('visible', 'off');
46 %set(gcf,'units','points','position',[0,0,1000,500]);
47
48 imshow(I_combine);
49 axis on;
50 for i = 1:Nrows
51 viscircles([I(i,2), I(i,1)],40);
52 viscircles([I(i,4) + I1_y , I(i,3)],40);
53
54 % --- usage of line: line([col1,col2],[row1,row2])
55 line( [I(i,2), I(i,4) + I1_y], [I(i,1), I(i,3)] );
56 end
```

```
57
58 %contcatString = 'Output/6-3Naive Matching.jpg';
59 contcatString = strcat( strcat('Output/', fileName), '.jpg');
60 saveas(graph, contcatString);
61
62 finish = 1;
63 end
64
65 function [I,corsSSD] = naiveCorrespondanceMatching(I1, I2, corners1, corners2,
       R, SSDth);
66
67 [N1, width1] = size(corners1);
68 [N2, width2] = size(corners2);
69
70 I = [];
71 corsSSD = [];
72 for i = 1:N1
73     temp = zeros(1,N2);
74     halfR = round(R/2);
75     w1 = I1( corners1(i,1)-halfR :corners1(i,1)+halfR , corners1(i,2)-halfR:
           corners1(i,2)+halfR );
76
77     for j = 1:N2
78         w2 = I2( corners2(j,1)-halfR:corners2(j,1)+halfR , corners2(j,2)-halfR
               :corners2(j,2)+halfR );
79         temp(j) = SSDmatch(w1,w2);
80     end
81
82     [minSSD,index_min] = min(temp(:));
83     if (minSSD<SSDth)
84         continue;
85     else
86         tempI = [corners1(i,1), corners1(i,2), corners2(index_min,1), corners2
               (index_min,2)];
87         I = [I;tempI];
88         corsSSD = [corsSSD;minSSD];
89     end
90
91
92 end
93
94 end
95
96 function ssd = SSDmatch(w1,w2)
97 [rows1, cols1] = size(w1);
98 [rows2, cols2] = size(w2);
99
100 if (rows1 ~= rows2 || cols1 ~= cols2)
101     ssd = 0;
102 else
103     Diff = w1 - w2;
104     ssd = sum(sum(Diff.*Diff));
105 end
106
107 end
108
109 function corners = CornerDetect(Image, nCorners, smoothSTD, windowSize)
110 img = rgb2gray(Image);
111 [img_x, img_y] = size(img);
112 img = smoothts(img, 'g', windowSize, smoothSTD);
113
114 % ---Gradient ---
115 [Ix, Iy] = imgradientxy(img);
116 Ix2 = Ix.*Ix;
117 Iy2 = Iy.*Iy;
118 IxIy = Ix.*Iy;
119
120 EigMat = zeros(img_x, img_y);
121
122 start_x = 1 + round(windowSize/2);
123 start_y = 1 + round(windowSize/2);
124 end_x = img_x - round(windowSize/2)-1;
```

```
end_y = img_y - round(windowSize/2)-1;

for i = start_x : end_x
    for j = start_y : end_y
        start_i = i - round(windowSize/2);
        start_j = j - round(windowSize/2);
        end_i = i + round(windowSize/2);
        end_j = j + round(windowSize/2);

        C11 = sum(sum( Ix2(start_i:end_i, start_j:end_j) ));
        C12 = sum(sum( IxIy(start_i:end_i, start_j:end_j) ));
        C22 = sum(sum( Iy2(start_i:end_i, start_j:end_j) ));

        C = [C11, C12; C12, C22];
        e = eig(C);
        EigMat(i,j) = min(e(1), e(2));

    end
end

corners = zeros(nCorners,2);

i=1;
while (i <= nCorners)
    [M,I] = max(EigMat(:));
    [I_row, I_col] = ind2sub(size(EigMat),I);
    tooCloseFlag = 0;
    for j = 1:i
%        I_row, I_col
        if ( distance(I_row, I_col, corners(j,1), corners(j,2)) < windowSize)
            tooCloseFlag = 1;
            break;
        else
            tooCloseFlag = 0;
            continue
        end
    end

    if (tooCloseFlag == 1)
        EigMat(I_row,I_col) = 0;
        i = i-1;
    else
        corners(i,1) = I_row;
        corners(i,2) = I_col;
        EigMat(I_row,I_col) = 0;
    end

    i=i+1;
end

end

function dist = distance(x1,y1,x2,y2)
    dist = sqrt( (x1-x2)^2 + (y1-y2)^2 );
end
```

Listing 10: Code for problem 6-3

```
% ***** 6.4 Epipolar Geometry *****

% --- Input image file ---
load('data/dino2.mat')
%load('data/matrix2.mat')
%load('data/warrior2.mat')

% --- Gray ---
image1 = rgb2gray(dino01);
image2 = rgb2gray(dino02);


% --- Fundamental matrix ---
F = fund(cor1, cor2);

```

```
plotting(F, image1, image2, 'dino01', 'dino02', '6-4EpipolarGeometry_dino', cor1, cor2);
%plotting(F, rgb2gray(matrix01), rgb2gray(matrix02), 'matrix01', 'matrix02', '6-4EpipolarGeometry_matrix', cor1, cor2);
%plotting(F, rgb2gray(warrior01), rgb2gray(warrior02), 'warrior01', 'warrior02', '6-4EpipolarGeometry_warrior', cor1, cor2);



function finish = plotting(F, image1, image2, name1, name2, fileName, cor1, cor2)
[nRow1, wid1] = size(cor1);
[nRow2, wid2] = size(cor2);

[H1,L1] = size(image1);
[H2,L2] = size(image2);

% --- Plotting ---
graph = figure('visible', 'off');
set(gcf,'units','points','position',[0,0,4000,2000]);

imshow(image1);
axis on;
hold on;
for i = 1:nRow1
viscircles([cor1(i,1), cor1(i,2)],40);

line = F*[cor1(i,1), cor1(i,2),1]';
xrange = [1, L1];
yrange = [1, H1];
PtMat = round( linePts(line, xrange, yrange) );

for i = 1:2
    for j = 1:2
        if (PtMat(i,j) == 0)
            PtMat(i,j) = 1;
        elseif (PtMat(i,j) >= H1 && j == 2)
            PtMat(i,j) = H1;
        elseif (PtMat(i,j) > L1 && j == 1)
            PtMat(i,j) = L1;
        end
    end
end

% --- usage of plot line: plot([x1;x2],[y1;y2])
plot( [PtMat(1,1);PtMat(2,1)], [PtMat(1,2);PtMat(2,2)],'b-' );
end

title( strcat('(a) - ', name1) );
contcatString = strcat( strcat('Output/', fileName), '_Result1.jpg');
saveas(graph, contcatString);



graph = figure('visible', 'off');
%set(gcf,'units','points','position',[0,0,1000,500]);

imshow(image2);
axis on;
hold on;
for i = 1:nRow2
viscircles([cor2(i,1), cor2(i,2)],40);

line = F*[cor1(i,1), cor1(i,2),1]';
xrange = [1, L2];
yrange = [1, H2];
PtMat = round( linePts(line, xrange, yrange) );

for i = 1:2
    for j = 1:2
        if (PtMat(i,j) == 0)
            PtMat(i,j) = 1;
```

```
        elseif (PtMat(i,j) >= H2 && j == 2)
            PtMat(i,j) = H2;
        elseif (PtMat(i,j) > L2 && j == 1)
            PtMat(i,j) = L2;
        end
    end
end

% --- usage of plot line: plot([x1;x2],[y1;y2])
plot( [PtMat(1,1);PtMat(2,1)], [PtMat(1,2);PtMat(2,2)],'b-');

end
title( strcat('(b) - ', name2) );

contcatString = strcat( strcat('Output/', fileName), '_Result2.jpg');
saveas(graph, contcatString);

end

function F = fund(x1, x2)
% Computes the fundamental matrix from a set of image correspondences
%
% INPUTS
%   x1    - Image coordinates for reference camera 1 (one per row)
%   x2    - Image coordinates for moved camera 2 (one per row)
%
% OUTPUTS
%   F     - Fundamental matrix (relates pts in cam 1 to lines in cam 2)
%
% DATESTAMP
%   21-May-07 4:16pm
%
% See also SVD

%% 0. Error-checking for input
n = size(x1,1);
if (size(x2,1) ~= n)
    error('Invalid correspondence: number of points don''t match!');
end

%% 1. Perform Hartley normalization (p. 212 of MaSKS)
avg1 = sum(x1,1) / n;
avg2 = sum(x2,1) / n;
diff1 = x1 - repmat(avg1,n,1);
diff2 = x2 - repmat(avg2,n,1);
std1 = sqrt(sum(diff1.^2,1) / n);
std2 = sqrt(sum(diff2.^2,1) / n);
H1 = [1/std1(1), 0, -avg1(1)/std1(1);
      0, 1/std1(2), -avg1(2)/std1(2);
      0, 0, 1];
H2 = [1/std2(1), 0, -avg2(1)/std2(1);
      0, 1/std2(2), -avg2(2)/std2(2);
      0, 0, 1];

norm1 = ([x1 ones(n,1)])*H1';
norm2 = ([x2 ones(n,1)])*H2';

%% 2. Compute a first approximation of the fundamental matrix
design = zeros(n,9);

for i=1:n
    design(i,:) = kron(norm1(i,:), norm2(i,:));
end

[U,S,V] = svd(design);
Fs = V(:,9);
F = reshape(Fs, 3, 3);

%% 3. Impose the rank constraint
[U,S,V] = svd(F);
S(3,3) = 0;
F = U*S*V';
```

```
156
157 %% 4. Undo the normalization
158 F = H2'*F*H1;
159
160 end
161
162 function pts = linePts(line, xrange, yrange)
163 % Returns the endpoints of a line clipped by the given bounding box.
164 %
165 % INPUTS
166 %   line    - Homogeneous coordinates of the line
167 %   xrange  - X-coordinate range of the bounding box [xmin, xmax]
168 %   yrange  - Y-coordinate range of the bounding box [ymin, ymax]
169 %
170 % OUTPUTS
171 %   pts     - The endpoints of the line in the bounding box.
172 %
173 % DATESTAMP
174 %   18-May-07 12:10am
175 %
176
177 xmin = xrange(1); xmax = xrange(2);
178 ymin = yrange(1); ymax = yrange(2);
179
180 % Find the four intersections with the bounding box limits
181 allPts = [xmin, -(line(1)*xmin + line(3))/line(2);
182           xmax, -(line(1)*xmax + line(3))/line(2);
183           -(line(2)*ymin + line(3))/line(1), ymin;
184           -(line(2)*ymax + line(3))/line(1), ymax];
185
186 % Clip testing: find the two intersections inside the bounding box
187 count = 0;
188 pts = zeros(2,2);
189 for i=1:4
190     if ((allPts(i,1) >= xmin) && (allPts(i,1) <= xmax) && ...
191         (allPts(i,2) >= ymin) && (allPts(i,2) <= ymax))
192         % add it to the list of endpoints
193         count = count + 1;
194
195         if (count == 1)
196             pts(count,:) = allPts(i,:);
197         elseif (count > 1)
198             addPoint = logical(1);
199             for j=1:count-1
200                 % Check to see that we're not adding a duplicate point
201                 diff = sum(abs(pts(j,:) - allPts(i,:)));
202                 if (diff < 1e-5)
203                     % Don't add the point
204                     addPoint = 0;
205                 end
206             end
207
208             if (~addPoint)
209                 count = count - 1;
210             elseif (count > 2)
211                 % This is an error
212                 error('Bug: more than two points on the line intersect the
                        bounding box!');
213             else
214                 % Add the point to the list
215                 pts(count,:) = allPts(i,:);
216             end
217         end
218     end
219 end
220 end
```

Listing 11: Code for problem 6-4

```
1 % ***** 6.5 Epipolar Geometry Based Matching *****
2
3 % --- Input image file ---
4 load('data/dino2.mat');
```

```
%load('data/matrix2.mat')
%load('data/warrior2.mat')

% --- Parameters ---
ncorners = 10;
smoothSTD = 5;
windowSize = 5;
R = 10;
SSDth = 1.2;


% --- Gray ---
image1 = rgb2gray(dino01);
image2 = rgb2gray(dino02);

%image1 = rgb2gray(matrix01);
%image2 = rgb2gray(matrix02);

%image1 = rgb2gray(warrior01);
%image2 = rgb2gray(warrior02);


corners1 = CornerDetect(image1, ncorners, smoothSTD, windowSize);

fileName = '6-5_dino';
%fileName = '6-5_matrix';
%fileName = '6-5_warrior';

% --- Fundamental matrix ---
F = fund(cor1, cor2);

%corners1 = corners1(1:10,:);
[I, corsSSD] = correspondanceMatchingLine(image1, image2, corners1, F, R,
    SSDth);


I1 = image1;
I2 = image2;
[I1_x, I1_y] = size(I1);
[I2_x, I2_y] = size(I2);
I_combine = zeros(I1_x, I1_y *2);
I_combine(1:I1_x, 1:I1_y)= I1(1:I1_x, 1:I1_y);
I_combine(1:I2_x, I1_y + 1 : I1_y + I2_y)= I2(1:I2_x, 1 : I2_y);
[Nrows, I_width] = size(I);


% --- Plotting ---
graph = figure('visible', 'off');
imshow(I_combine);
axis on;
for i = 1:Nrows
viscircles([I(i,2), I(i,1)],40);
viscircles([I(i,4) + I1_y , I(i,3)],40);
line( [I(i,2), I(i,4) + I1_y], [I(i,1), I(i,3)] );
end

contcatString = strcat( strcat('Output/', fileName), '_Result.jpg');
saveas(graph, contcatString);




function [I, corsSSD] = correspondanceMatchingLine(I1, I2, corners1, F, R,
    SSDth)
[N1, width1] = size(corners1);
[H1,L1] = size(I1);
[H2,L2] = size(I2);

I = [];
corsSSD = [];

for i = 1:N1
```

```
75
76      line = F*[corners1(i,2), corners1(i,1),1]';
77      xrange = [1, L2];
78      yrange = [1, H2];
79
80      PtMat = linePts(line, xrange, yrange);
81
82      % --- mirroring ---
83      tempPtMat = [PtMat(1,2),PtMat(1,1) ; PtMat(2,2), PtMat(2,1)];
84      PtMat = tempPtMat;
85
86      a = (PtMat(1,2) - PtMat(2,2)) / (PtMat(1,1) - PtMat(2,1));
87      b = PtMat(1,2) - a*PtMat(1,1);
88
89      corners2 = [];
90      for j = R:H2-R
91          if (round(a*j+b) < round(R/2) || round(a*j+b)>L2-round(R/2))
92              continue;
93          else
94              corners2 = [corners2;[j,round(a*j+b)]];
95          end
96      end
97
98      [N2, width2] = size(corners2);
99
100     temp = zeros(1,N2);
101
102     halfR = round(R/2);
103     w1 = I1( corners1(i,1)-halfR :corners1(i,1)+halfR , corners1(i,2)-halfR:
            corners1(i,2)+halfR );
104
105     for j = 1:N2
106         %[corners2(j,1)-halfR, corners2(j,1)+halfR ; corners2(j,2)-halfR,
                corners2(j,2)+halfR]
107         w2 = I2( corners2(j,1)-halfR:corners2(j,1)+halfR , corners2(j,2)-halfR
                :corners2(j,2)+halfR );
108         temp(j) = SSDmatch(w1,w2);
109     end
110
111     [minSSD,index_min] = min(temp(:));
112     if (minSSD<SSDth)
113         continue;
114     else
115         tempI = [corners1(i,1), corners1(i,2), corners2(index_min,1), corners2
                (index_min,2)];
116         I = [I;tempI];
117         corsSSD = [corsSSD;minSSD];
118     end
119 end
120
121
122
123 end
124
125
126 function F = fund(x1, x2)
127 % Computes the fundamental matrix from a set of image correspondences
128 %
129 % INPUTS
130 %   x1    - Image coordinates for reference camera 1 (one per row)
131 %   x2    - Image coordinates for moved camera 2 (one per row)
132 %
133 % OUTPUTS
134 %   F     - Fundamental matrix (relates pts in cam 1 to lines in cam 2)
135 %
136 % DATESTAMP
137 %   21-May-07 4:16pm
138 %
139 % See also SVD
140
141 %% 0. Error-checking for input
142 n = size(x1,1);
```

```
if (size(x2,1) ~= n)
    error('Invalid correspondence: number of points don''t match!');
end

%% 1. Perform Hartley normalization (p. 212 of MaSKS)
avg1 = sum(x1,1) / n;
avg2 = sum(x2,1) / n;
diff1 = x1 - repmat(avg1,n,1);
diff2 = x2 - repmat(avg2,n,1);
std1 = sqrt(sum(diff1.^2,1) / n);
std2 = sqrt(sum(diff2.^2,1) / n);
H1 = [1/std1(1), 0, -avg1(1)/std1(1);
      0, 1/std1(2), -avg1(2)/std1(2);
      0, 0, 1];
H2 = [1/std2(1), 0, -avg2(1)/std2(2);
      0, 1/std2(2), -avg2(2)/std2(2);
      0, 0, 1];

norm1 = ([x1 ones(n,1)])*H1';
norm2 = ([x2 ones(n,1)])*H2';

%% 2. Compute a first approximation of the fundamental matrix
design = zeros(n,9);

for i=1:n
    design(i,:) = kron(norm1(i,:), norm2(i,:));
end

[U,S,V] = svd(design);
Fs = V(:,9);
F = reshape(Fs, 3, 3);

%% 3. Impose the rank constraint
[U,S,V] = svd(F);
S(3,3) = 0;
F = U*S*V';

%% 4. Undo the normalization
F = H2'*F*H1;

end

function pts = linePts(line, xrange, yrange)
% Returns the endpoints of a line clipped by the given bounding box.
%
% INPUTS
%   line    - Homogeneous coordinates of the line
%   xrange  - X-coordinate range of the bounding box [xmin, xmax]
%   yrange  - Y-coordinate range of the bounding box [ymin, ymax]
%
% OUTPUTS
%   pts     - The endpoints of the line in the bounding box.
%
% DATESTAMP
%   18-May-07 12:10am
%

xmin = xrange(1); xmax = xrange(2);
ymin = yrange(1); ymax = yrange(2);

% Find the four intersections with the bounding box limits
allPts = [xmin, -(line(1)*xmin + line(3))/line(2);
          xmax, -(line(1)*xmax + line(3))/line(2);
          -(line(2)*ymin + line(3))/line(1), ymin;
          -(line(2)*ymax + line(3))/line(1), ymax];

% Clip testing: find the two intersections inside the bounding box
count = 0;
pts = zeros(2,2);
for i=1:4
    if ((allPts(i,1) >= xmin) && (allPts(i,1) <= xmax) && ...
        (allPts(i,2) >= ymin) && (allPts(i,2) <= ymax))
```

```
        % add it to the list of endpoints
        count = count + 1;

        if (count == 1)
            pts(count,:) = allPts(i,:);
        elseif (count > 1)
            addPoint = logical(1);
            for j=1:count-1
                % Check to see that we're not adding a duplicate point
                diff = sum(abs(pts(j,:) - allPts(i,:)));
                if (diff < 1e-5)
                    % Don't add the point
                    addPoint = 0;
                end
            end

            if (~addPoint)
                count = count - 1;
            elseif (count > 2)
                % This is an error
                error('Bug: more than two points on the line intersect the
                    bounding box!');
            else
                % Add the point to the list
                pts(count,:) = allPts(i,:);
            end
        end
    end
end
end

function ssd = SSDmatch(w1,w2)
[rows1, cols1] = size(w1);
[rows2, cols2] = size(w2);

if (rows1 ~= rows2 || cols1 ~= cols2)
    ssd = 0;
else
    Diff = w1 - w2;
    ssd = sum(sum(Diff.*Diff));
end

end

function corners = CornerDetect(Image, nCorners, smoothSTD, windowSize)
img = rgb2gray(Image);
[img_x, img_y] = size(img);
img = smoothts(img, 'g', windowSize, smoothSTD);

% ---Gradient ---
[Ix, Iy] = imgradientxy(img);
Ix2 = Ix.*Ix;
Iy2 = Iy.*Iy;
IxIy = Ix.*Iy;

EigMat = zeros(img_x, img_y);

start_x = 1 + round(windowSize/2);
start_y = 1 + round(windowSize/2);
end_x = img_x - round(windowSize/2)-1;
end_y = img_y - round(windowSize/2)-1;

for i = start_x : end_x
    for j = start_y : end_y
        start_i = i - round(windowSize/2);
        start_j = j - round(windowSize/2);
        end_i = i + round(windowSize/2);
        end_j = j + round(windowSize/2);

        C11 = sum(sum( Ix2(start_i:end_i, start_j:end_j) ));
        C12 = sum(sum( IxIy(start_i:end_i, start_j:end_j) ));
        C22 = sum(sum( Iy2(start_i:end_i, start_j:end_j) ));
```

```
286
287         C = [C11, C12; C12, C22];
288         e = eig(C);
289         EigMat(i,j) = min(e(1), e(2));
290
291     end
292 end
293
294 corners = zeros(nCorners,2);
295
296 i=1;
297 while (i <= nCorners)
298     [M,I] = max(EigMat(:));
299     [I_row, I_col] = ind2sub(size(EigMat),I);
300     tooCloseFlag = 0;
301     for j = 1:i
302 %         I_row, I_col
303         if ( distance(I_row, I_col, corners(j,1), corners(j,2)) < windowSize)
304             tooCloseFlag = 1;
305             break;
306         else
307             tooCloseFlag = 0;
308             continue
309         end
310     end
311
312     if (tooCloseFlag == 1)
313         EigMat(I_row,I_col) = 0;
314         i = i-1;
315     else
316         corners(i,1) = I_row;
317         corners(i,2) = I_col;
318         EigMat(I_row,I_col) = 0;
319     end
320
321     i=i+1;
322 end
323
324 end
325
326 function dist = distance(x1,y1,x2,y2)
327     dist = sqrt( (x1-x2)^2 + (y1-y2)^2 );
328 end
```

Listing 12: Code for problem 6-5

Submitted by Hao-en Sung (A53204772)
Tsung-han Lee (A53219632) on November 8, 2016.