Gupta, Ajitesh
Choi, Jean
Mavalankar, Aditi Ashutosh
Department of Electrical Engineering and Computer Science
University of California, San Diego

November 7, 2016

HOMEWORK 2

Problem 1. **Epipolar Geometry Theory** : We are given a transformation (R, T) for a camera calibration that maps a world point to the camera by the equation: ${}^{C}P = R^{W}P + T$

1. We are given calibrations of two cameras i.e. a stereo pair to a common external coordinate system, represented by R_1, T_1, R_2, T_2. We are required to give an expression that maps points expressed in the coordinate system of camera 1 to that of camera 2.

   Solution

   In order to map a point from camera 1 to camera 2, we take the following approach:

   Let P be a point in the world.

   ${}^{C}P_1 = R_1P + T_1$

   $\therefore R_1^{-1}({}^{C}P_1 - T_1) = P$

   Substituting this value of P in the equation of coordinates for camera 2 i.e.

   ${}^{C}P_2 = R_2P + T_2$

   we get

   ${}^{C}P_2 = R_2R_1^{-1}({}^{C}P_1 - T_1) + T_2$

2. Find the length of the baseline (b) of the stereo pair.

   Solution

   We are given the coordinates in terms of the two cameras. We now want to convert them into actual world coordinates. In order to do this, we will have to apply the reverse of the transformation that was applied initially.

   We know that a rotation was applied first, followed by a translation. So here, to convert them to the world coordinates, we will first perform an inverse of the translation operation and then an inverse of the rotation operation.

   The inverse of the translation matrix T_1 is given by $-T_1$, and that of the translation matrix T_2 is given by $-T_2$.

   The inverse of the rotation matrix R_1 is given by its transpose $(R_1)^T$ and that of the rotation matrix R_2 is given by its transpose $(R_2)^T$.

   Thus, after we have transformed into the world coordinates, we find the length of the baseline in the following way:

   $b = \| - R_1^TT_1 - (-R_2^TT_2)\|$

   $\therefore b = \| - R_1^TT_1 + R_2^TT_2\|$

3. Find the essential matrix (E) in terms of R_1, R_2, T_1, T_2

   Solution We know that the formula to find the essential matrix E is given by

   $E = [t_X]R$

   $\therefore E = [T_2 - R_2R_1^{-1}T_1]_xR_2R_1^{-1}$

Problem 2. **Epipolar Geometry** : We are given two camera planes, which have the image plane $z = 1$. The focal point of the first camera is at $(-12, 0, 0)$ and that of the second camera is at $(12, 0, 0)$. The notation we will use is:

(x, y) : A point in the first camera

(u, v) : A point in the second camera

These points are relative to their respective camera centers. So, for instance, if we have $(x, y) = (0, 0)$, it actually refers to the world coordinates $(-12, 0, 1)$. Similarly, $(u, v) = (0, 0)$ refers to the world coordinates $(12, 0, 1)$.

1. $(x, y) = (8, 7)$ is mapped to the point $(u, v) = (2, 7)$ with a disparity of 6. Find the 3-D location of this point.

   Solution

   One way to look at this problem is that we need the 3-D coordinates of the point that is obtained when we map $(8, 7)$ to $(2, 7)$.

   The actual 3-D coordinates for $(x, y) = (8, 7)$ are $(-4, 7, 1)$.

   The actual 3-D coordinates for $(u, v) = (2, 7)$ are $(14, 7, 1)$.

   The equation of a line joining two points in 3-D is given by:

   $\frac{x-x_1}{x_2-x_1} = \frac{y-y_1}{y_2-y_1} = \frac{z-z_1}{z_2-z_1} = constant$

   If we find the equation of the line joining $(-12, 0, 0)$ and $(-4, 7, 1)$, and that of the line joining $(12, 0, 0)$ and $(14, 7, 1)$, and find their point of intersection, that point will be the desired point.

   For line l_1,

   $\frac{x-(-4)}{-12-(-4)} = \frac{y-7}{0-7} = \frac{z-1}{0-1} = k$

   $\therefore \frac{x+4}{-8} = \frac{y-7}{-7} = \frac{z-1}{-1} = k$

   Thus, we get the coordinates (x, y, z) of line l_1 in the form:

   $x = -4 - 8k$

   $y = 7 - 7k$

   $z = 1 - k$

   Similarly, for line l_2,

   $\frac{x-12}{14-12} = \frac{y-0}{7-0} = \frac{z-0}{1-0} = m$

   $\therefore \frac{x-12}{2} = \frac{y}{7} = \frac{z}{1} = m$

   Thus, we get the coordinates (x, y, z) of line l_2 in the form:

   $x = 2m + 12$

   $y = 7m$

   $z = m$

   Now, to find the point of intersection, $x_1 = x_2$, $y_1 = y_2$ and $z_1 = z_2$.

   Comparing x,

   $-4 - 8k = 2m + 12$

   $\therefore 2m + 8k = -16$

   $\therefore m + 4k = -8$

   Comparing y,

   $7 - 7k = 7m$

   $\therefore m + k = 1$

   We now have two equations for solving two unknowns.

Subtracting the second equation from the first, we get

$3k = -9$

$\therefore k = -3$

Substituting this value of k in the second equation, we get

$m = 4$

Thus, the coordinates of the point of intersection are:

$x = 2m + 12 = 2(4) + 12 = 8 + 12 = 20$

$y = 7m = 7(4) = 28$

$z = m = 4$

Thus, the 3-D coordinates of the point of intersection are $(20, 28, 4)$.

2. We are given the 3-D line $X + Z = 2, Y = 0$. Using the same stereo set up as before, we are supposed to give an expression for the disparity of a point on this line after projecting it onto the two images, as a function of its position in the right image. This means that the expression obtained will contain only the terms u and d. Consider only points where $Z > 1$.

   Solution

   Let the coordinates of the point on the line be:

   $x = a$

   $y = 0$

   $z = 2 - a$

   Using the equation of a line in 3-D,

   $\frac{x-x_1}{x_2-x_1} = \frac{y-y_1}{y_2-y_1} = \frac{z-z_1}{z_2-z_1} = constant$

   We have the equations of the first line given by:

   $\frac{x-(-12)}{a-(-12)} = \frac{y-0}{0-0} = \frac{z-0}{(2-a)-0} = \alpha$

   $\therefore \frac{x+12}{a+12} = \frac{y}{0} = \frac{z}{2-a} = \alpha$

   Thus, we get the coordinates on this line as:

   $x_1 = \alpha(a + 12) - 12$

   $y_1 = 0$

   $z_1 = \alpha(2 - a)$

   The equation of the second line is given by:

   $\frac{x-12}{a-12} = \frac{y-0}{0-0} = \frac{z-0}{(2-a)-0} = \beta$

   $\therefore \frac{x-12}{a-12} = \frac{y}{0} = \frac{z}{2-a} = \beta$

   Thus, we get the coordinates on this line as:

   $x_2 = 12 + \beta(a - 12)$

   $y_2 = 0$

   $z_2 = \beta(2 - a)$

   Now, to find the coordinates of these points on the image plane $Z = 1$, we substitute this value in z, so we get

   $\alpha = \beta = \frac{1}{2-a}$

   We want the entire expression in terms of u and d.

   So, we will express a in terms of u.

   $u = x_2 - 12$

   $\therefore u = 12 + \beta(a - 12) - 12$

$\therefore u = \frac{a-12}{2-a}$
$\therefore 2u - au = a - 12$
$\therefore au + a = 2u + 12$
$\therefore a(u+1) = 2(u+6)$
$\therefore a = 2\frac{u+6}{u+1}$
Now, we find the disparity d.
We know that $d = X_L - X_R$.
$X_L = x_1 - (-12) = x_1 + 12$
$\therefore X_L = \frac{a+12}{2-a} - 12 + 12$
$\therefore X_L = \frac{a+12}{2-a}$
$X_R = x_2 - 12$
$\therefore X_R = 12 + \frac{a-12}{2-a} - 12$
$\therefore X_R = \frac{a-12}{2-a}$
Now, from these two values, we get
$d = X_L - X_R$
$\therefore d = \frac{a+12-a+12}{2-a}$
$\therefore d = \frac{24}{2-a}$
Now, substituting the value of a in terms of u obtained earlier,
$d = \frac{24}{2-2\frac{u+6}{u+1}}$
$\therefore d = \frac{24(u+1)}{2u+2-2u-12}$
$\therefore d = \frac{24(u+1)}{-10}$
$\therefore d = -2.4u - 2.4$

Problem 3. **Reconstruction Accuracy** : Given a 2-D stereo system where camera 1 is at the origin, and camera 2 is thus, at $(b, 0)$, where b is the baseline, and the image plane is given by $Z = f$, assume that the only source of error is the disparity. Discuss the dependence of the error in depth estimation $\Delta Z'$ as a function of the focal length f, the baseline length b, and the depth Z'.

Solution

We know that
$Z' = \frac{bf}{X_L - X_R}$
i.e. $Z' = \frac{bf}{d}$
Now, to find $\Delta Z'$,
$\frac{dZ'}{dd} = -\frac{bf}{d^2}$
$\therefore \Delta Z' = -\frac{bf}{d^2}$
Since we want the value in terms of Z', b and f, we substitute d using
$d = \frac{bf}{Z'}$
$\therefore \Delta Z' = -\frac{bf}{(\frac{bf}{Z'})^2}$
$\therefore \Delta Z' = -\frac{Z'^2}{bf}$

Here, we see that $\Delta Z'$ has a direct proportionality with Z' and an inverse proportionality with b and f. However, due to the negative sign, we get the following results:

- As Z' increases, $\Delta Z'$ decreases.
- As b increases, $\Delta Z'$ increases.
- As f increases, $\Delta Z'$ increases.

Problem 4. **Filters as templates** : This problem deals with convolution filters. When we convolve filters with an image, they will fire strongest on locations of an image that resemble the filter. Thus, we can use filters as object templates to identify specific objects inside an image. In this problem, we will find cars inside an image by convolving a car template with that image. This is not a very good way to do object detection, but this teaches us some of the steps necessary to create a good object detector. We will learn some preprocessing steps to make vision algorithms successful, and the benefits and drawbacks of filters. In each problem, we are required to analyze and explain the results.

4.1. **Warmup** : This problem involves convolution of a filter with a synthetic image. The filter is in 'filter.jpg' and the synthetic image is 'toy.png'. We could modify the filter image and the original image slightly. One possible modification could be

```
filter img = filter img - mean(filter img(:))
```

The conv2 function in Matlab will be used to convolve the filter image with the toy example. The output of the convolution will be an intensity image. In the original image, we have to draw a bounding box of the same size as the filter image around the top 3 intensity value locations in the convolved image. How well will this technique work on more realistic images? What problems are involved when we use this algorithm on more realistic images?

Solution

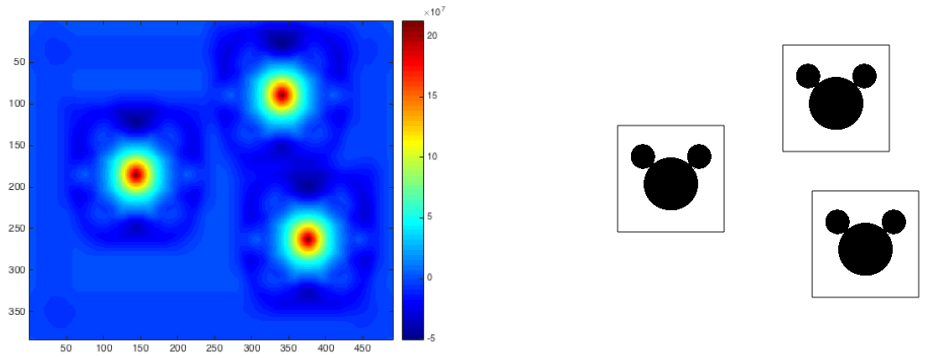

Figure 1: (a) Heat map (b) Bounding boxes

Both the filter and the image were really simple in shape and color in this case. However, more realistic images will be more complicated as in shape and color. Unless either the filter or the image are modified, it would be hard to find the most intense locations because there may be too many or none.

4.2. **Detection Error** : In the previous problem, we implemented the algorithm that produces a bounding box around a detected object. Now, we need to know if the bounding box is good or not. Given a ground truth bounding box (g) and a predicted bounding box

(p), the bounding box quality can be measured by $\frac{p \cap g}{p \cup g}$. Intuitively, this is the number of overlapping pixels between the bounding boxes divided by the total number of unique pixels of the two bounding boxes combined. Assuming that all bounding boxes will be axis-aligned rectangles, we have to implement this error function and try it on the toy example in the previous section. We have to choose 3 different ground truth bounding box sizes around one of the Mickey 3 silhouettes. In general, if the overlap is 50% or more, the detection did a good job.

Solution

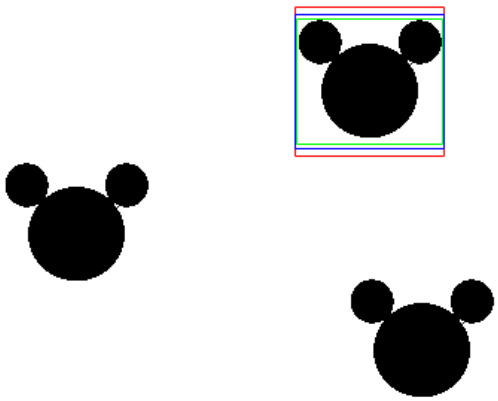

Figure 2: Bounding boxes detection in comparison with ground truth

For the red groundtruth, the overlap is 0.6404. For the blue groundtruth, the overlap is 0.5767. For the green groundtruth, the overlap is 0.5277. Overall, the performance is good since they are all over 50%.

4.3. **More Realistic Images** : We created an algorithm for matching templates and a function to determine the quality of the match. Now, we shift to more realistic images. The file, 'cartemplate.jpg', will be the filter to convolve on each of the 5 other car images ('car1.jpg', 'car2.jpg', 'car3.jpg', 'car4.jpg', 'car5.jpg'). Each image will have an associated text file that contains 2 (x, y) coordinates. These coordinates will be the ground truth bounding box for each image. For each car image, we have to give the following:

- A heat map image
- A bounding box drawn on the original image
- The bounding box overlap percent
- A description of the pre-processing steps needed to achieve this overlap
- An explanation of the importance of these steps

To formulate the algorithm, we could rescale the car template, rotate it, or blur out the test image/template, or even both.

Solution

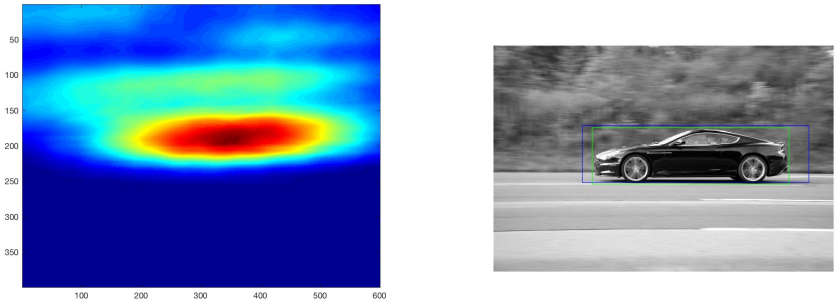

Figure 3: (a) Heat map (b) Bounding box for car 1

The overlap in the first car is 0.8201. Its quite a straightforward test case as the car is clearly visible and is oriented completely horizontally. The filter was flipped for convolution and resized into a smaller image in order to match the size of the car in the test image. Also both the image and the filter were inversed in color as initially both had dark cars and white surroundings which meant the filter was looking for a similar surrounding rather than a similar car. So to achieve maximum detection we inversed the colors in both so that both cars were in high intensity.

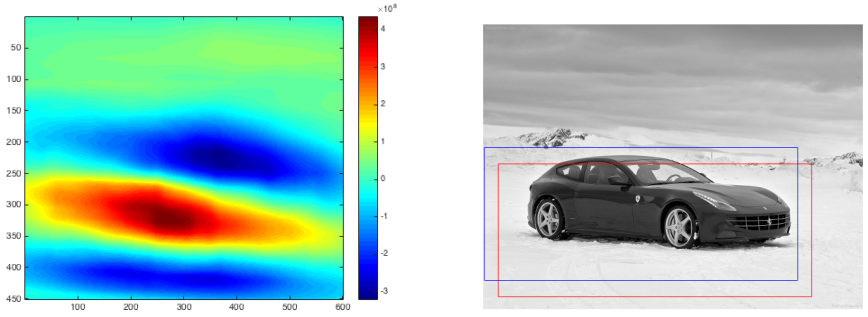

Figure 4: (a) Heat map (b) Bounding box for car 2

We need to flip, rotate, crop, and then resize the car template for the best match. This is because the car in the image is going in the other way and slightly rotated. Also, the car template has blank space which needs to be cropped and it needs to be resized to match that of the image. We designed the algorithm such that it draws a bounding box centered at the averaged position of the maximum intensity values. The overlap is 0.7295.

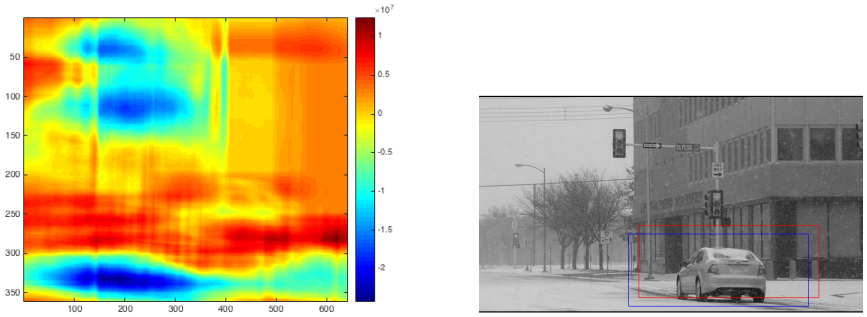

Figure 5: (a) Heat map (b) Bounding box for car 3

We need to inverse the color, fill the dark spots with gray color, rotate, crop, and then resize the car template for the best match. This is because the car in the image if white, rotated, and horizontally smaller. Also, the blank space of the car template needs to be filled with gray or brighter color, because the background of the image is snowy. The same algorithm was used. The overlap is 0.7303.

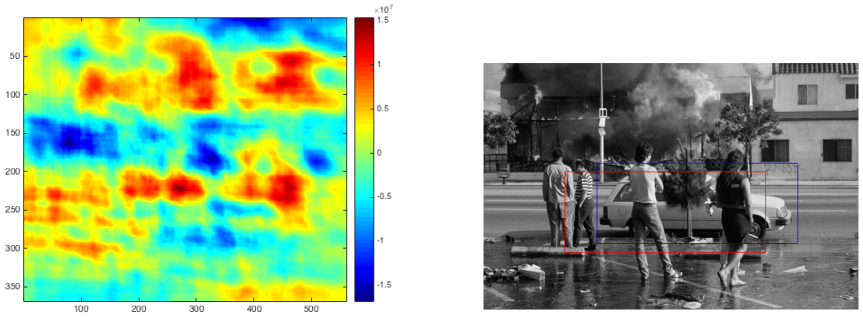

Figure 6: (a) Heat map (b) Bounding box for car 4

We need to flip, inverse the color, fill the dark spots with gray color, crop, and then resize the car template for the best match. This is because the car in the image is going the other way and is white. Also, the car template needs to be cropped and resized to match that in the image. Furthermore, the blank space of the car template needs to be filled with gray or brighter color, because the car is on the gray, asphalt road. The same algorithm was used. The overlap is 0.6116.

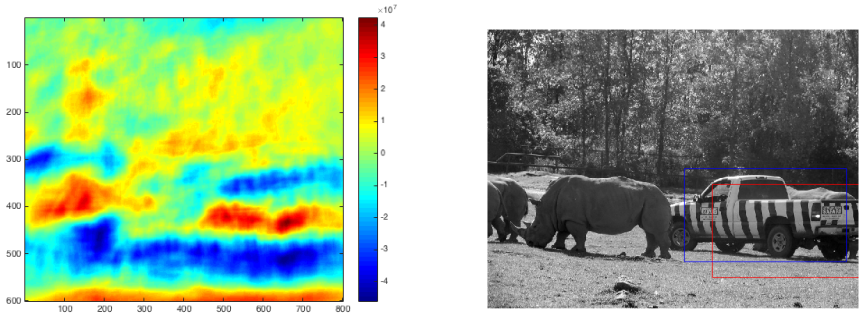

Figure 7: (a) Heat map (b) Bounding box for car 5

We need to inverse the color, fill the dark spots with gray color, crop, and then resize the car template for the best match. This is because the car in the image is going on the other way and the background is composed of trees and grass. Also, the car template needs to be cropped and resized to match that of the image. The same algorithm was used. The overlap is 0.5317.

4.4. **Invariance** : The detection algorithm that was implemented for this problem may have seemed a bit brittle. We have to describe a few things that this algorithm was not invariant to.

Solution

The filter was also not invariant to the color the of the car. Plain colored cars like the first image was easy to detect unlike the last one which had stripes. It depended upon the shape of the car. As again the first car matched pretty well whereas the last one wasn't the same shape as the template and hence had difficulties. The occulusion level also affected the detection as template matching solely depends on matching two given windows. Occlusions cover up the object being matched. The algorithm is not invariant to the orientation of the car. It needs to use a differently oriented filter for each image depending upon the orientation of the car in that image.

Problem 5. **Canny Edge Detection** : This problem requires us to implement a function to perform Canny Edge Detection. The steps to be followed are:

- **Smoothing** : Smoothing is required to remove noise from being considered edges. We have to use a 5×5 Gaussian kernel filter with $\sigma = 1.4$ to smooth the images.

$$K = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} \tag{0-1}$$

- **Gradient Computation** : We have to find the image gradient in the horizontal and vertical directions. We can use Sobel operators as filter kernel to calculate G_x (gradient along the x-axis) and G_y (gradient along the y-axis). s_x and s_y are the corresponding kernels. We have to compute the gradient magnitude image as $|G| = \sqrt{G_x^2 + G_y^2}$. The edge direction at each pixel is given by $G_\theta = tan^{-1}(\frac{G_y}{G_x})$.

$$s_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, s_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{0-2}$$

- **Non maximum suppression** : The desired edges need to be sharp. So, we are required to use non-maximum suppression to preserve the local maxima and discard the rest. The method we will follow is:

  For every pixel,

  - Round off the gradient direction θ to the nearest multiple of 45 deg in a 8-connected neighborhood.
  - Compare the edge strength at the current pixel to the pixels along the positive and negative gradient direction in the 8-connected neighbourhood.
  - Preserve the values of only those pixels which have maximum gradient magnitudes in the neighbourhood along the +ve and ve gradient direction.

- **Hysteresis Thresholding** :

We have to compute the images at each step and select thresholds that retain most of the true edges. The image to be used is 'geisel.jpg'.

Solution

Figure 8: Original image

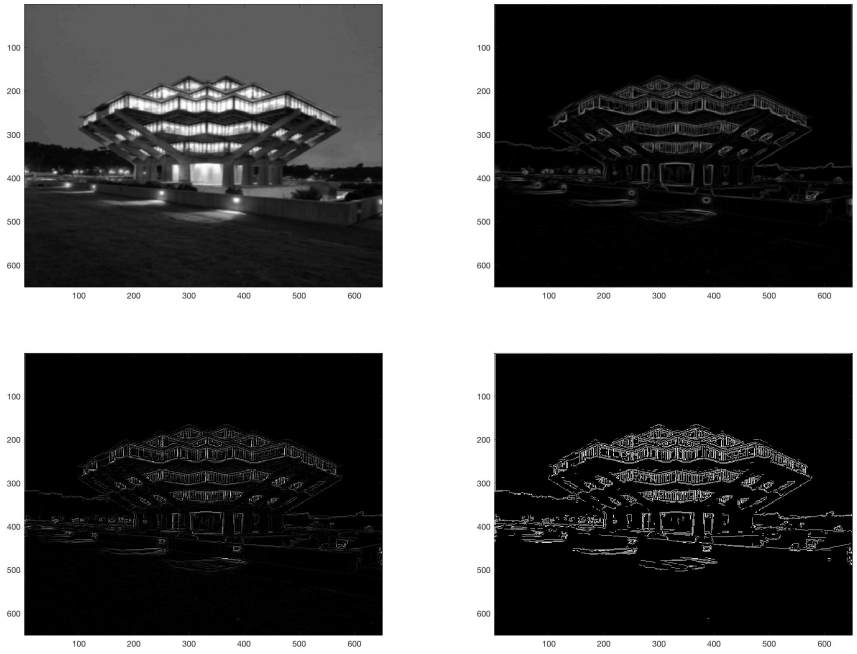

Figure 9: (a) Smoothed Image (b) Gradient Magnitude (c) NMS edges (d) Final edges after thresholding

The optimal threshold values were obtained by using an upper threshold of 40 and a lower threshold of 20.

Problem 6. **Sparse Stereo Matching** : We are given two figures, 'warrior2.mat' and 'matrix2.mat'. These files contain two images, two camera matrices and sets of corresponding points extracted by manually clicking on the images.

6.1. **Corner Detection** : We have to build a corner detector. The file should be named 'CornerDetect.m' and should take in as input the following arguments:

```
corner = CornerDetect(Image, nCorners, smoothSTD, windowSize)
```

Here,

- smoothSTD = standard deviation of the smoothing kernel
- windowSize = size of the smoothing window
- nCorners = number of strongest corners to be returned after non-maximum suppression (Here, we will use nCorners = 20)

Solution

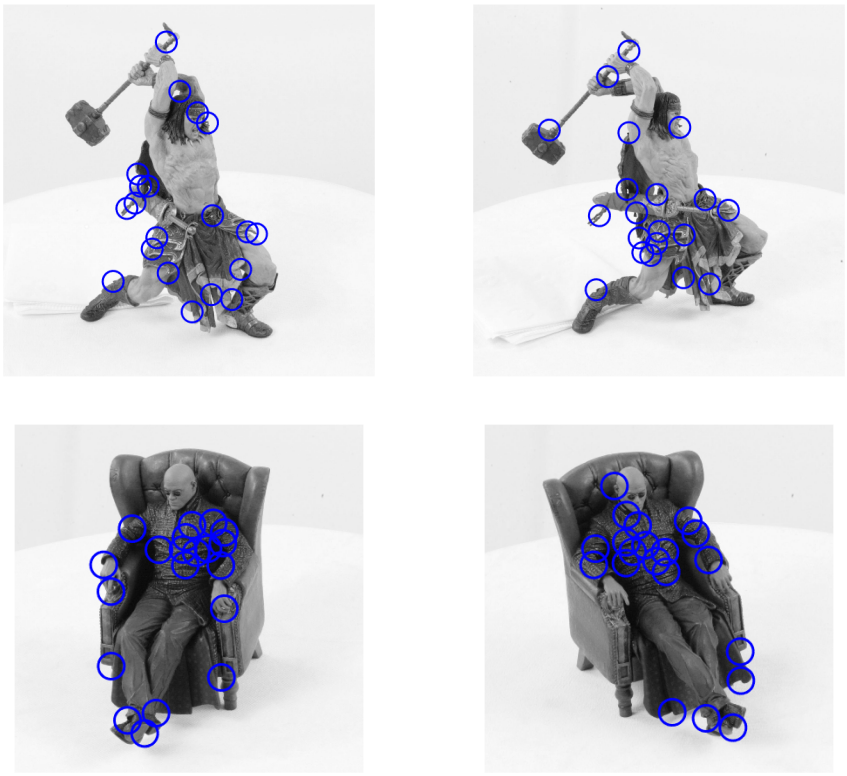

Figure 10: (a) Warrior Image 1 (b) Warrior Image 2 (c) Matrix Image 1 (d) Matrix Image 2

Reference the appendix for the values used for nCorners, smoothSTD, and windowSize.

6.2. **SSD Matching** : We have to write a function that implements the SSD matching algorithm for two input windows.

Solution

Reference the appendix for the values used for the size of the patches for SSD comparison.

6.3. **Naive Matching** : We have to now find correspondences. we will start with trying to find the best match between two sets of corner points. For each corner in image1, we have to find the best match from the detected corners in image2. If the SSD match score is very low, there is no match for that point. We have to find a good threshold value. We have to write a function 'naiveCorrespondenceMatching.m' and use it in the following way:

```
ncorners = 10;
corners1 = CornerDetect(I1, ncorners, smoothSTD, windowSize));
corners2 = CornerDetect(I2, ncorners, smoothSTD, windowSize));
[I, corsSSD] = naiveCorrespondenceMatching(I1, I2, corners1, corners2,
R, SSDth);
```

Solution

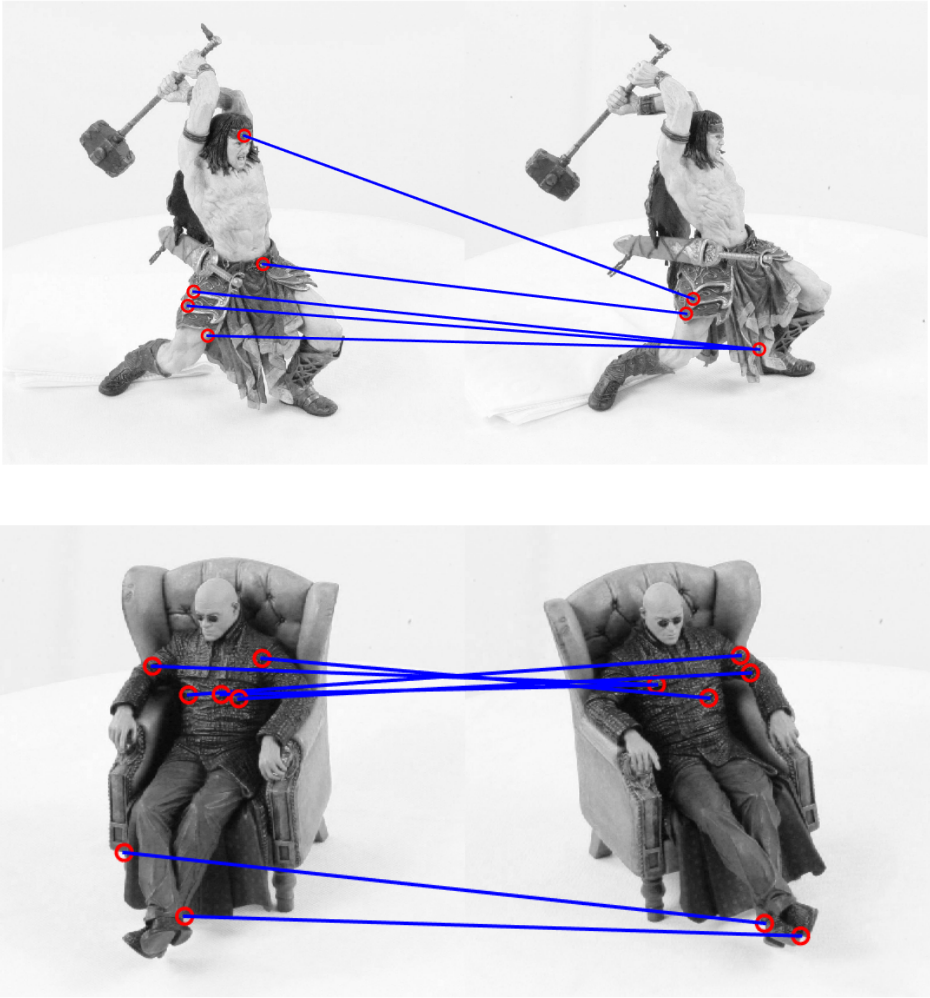

Figure 11: (a) Warrior Image (b) Matrix Image

Reference the appendix for values used for nCorners, smoothSTD, windowSize, R, and SSDth. In general, the matching is terrible. When you look at the matched corners closely, they kind of resemble each other as in color and shape. However, the actual locations of the corners were not considered, so many corners from the left image were matched to corners in totally different places in the right image.

6.4. **Epipolar Geometry** : We have to use the file 'fund.m' and the provided points cor1 and cor2, to calculate the fundamental matrix and plot the epipolar lines in both image pairs. We could use 'linePts.m'.
Solution

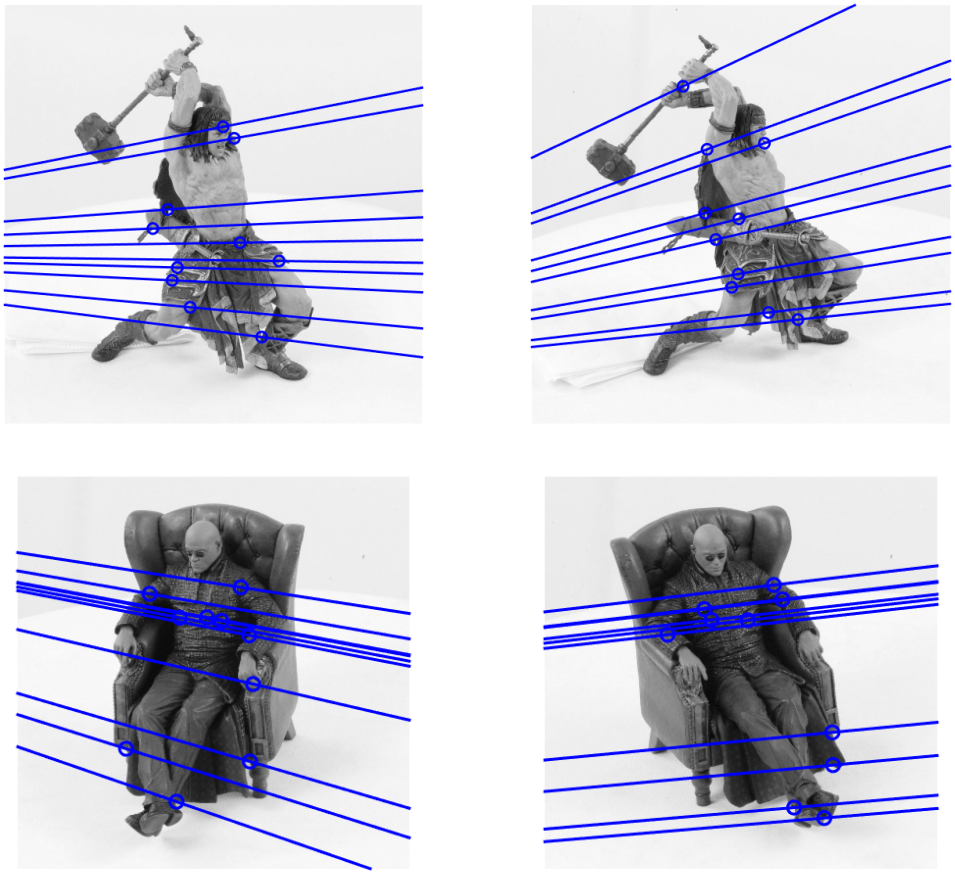

Figure 12: (a) Warrior Image 1 (b) Warrior Image 2 (c) Matrix Image 1 (d) Matrix Image 2

The epipolar lines for the warrior images look like they are running in the similar way, while the epipolar lines for the matrix images look like they are running in the opposite way. This is, presumably, because the pictures of the warrior were taken at camera positions with relatively small angle variation, while the pictures of the matrix figure were taken at camera positions with relatively large angle variation.

6.5. **Epipolar Geometry Based Matching** : We will now build a better matching algorithm using epipolar geometry. We first have to detect 10 corners in image1, and for each corner, do a linesearch along the corresponding epipolar line in image2. We then evaluate the SSD score for each point along this line and return the best match, or no match if the scores are too low. R is the radius of the SSD patch.

```
ncorners = 10;
F = fund(cor1, cor2);
corners1 = CornerDetect(I1, ncorners, smoothSTD, windowSize));
corsSSD = correspondenceMatchingLine( I1, I2, corners1, F, R, SSDth);
```

Solution

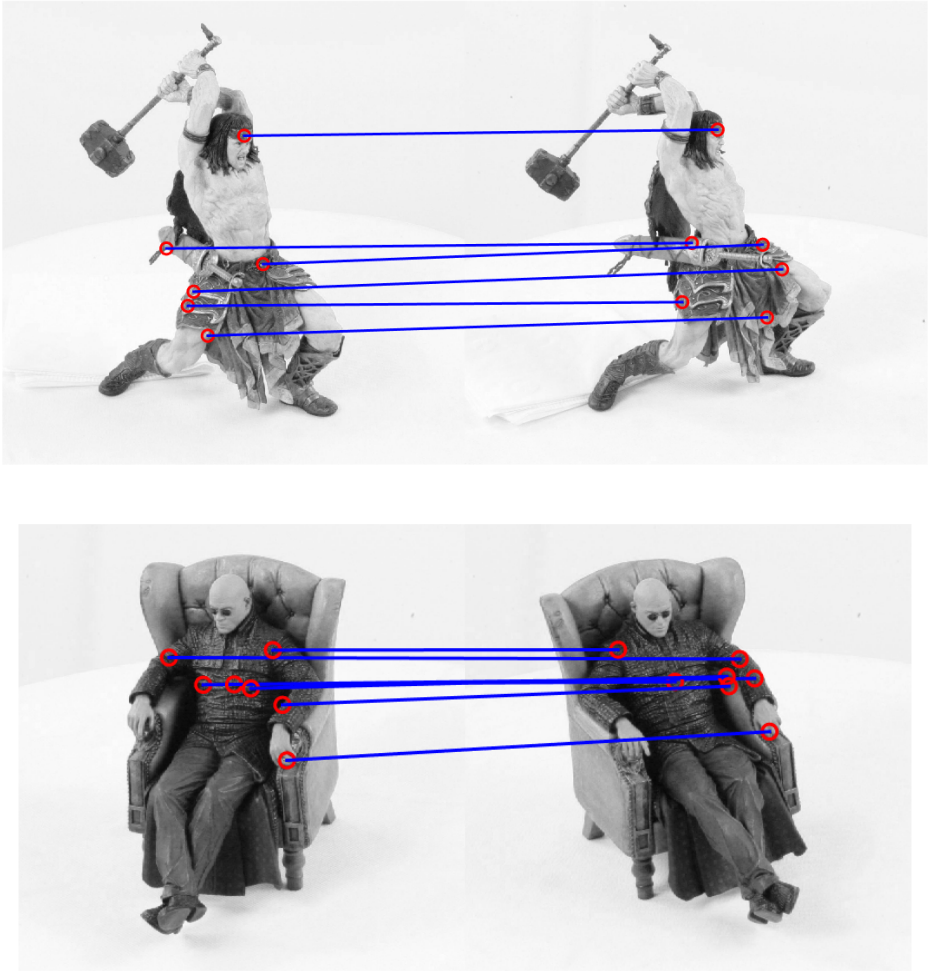

Figure 13: (a) Warrior Image (b) Matrix Image

Now the matching is significantly better than that of 6.3. However, there are still mismatched corners. Using epipolar geometry prevents the corners from being mapped to a totally different positions(e.g. head to foot or arm to foot), but it can result in matching similar looking corners given that they are on the same epipolar line, as you can see in Figure 13(b).

6.6. **Triangulation** : After we have found the correspondences between the pairs of images, we can now proceed to the triangulation of the corresponding 3-D points. The correspondences may contain noise and many outliers as we have not enforced the ordering constraint. We have to triangulate a 3-D point for each corresponding pair of points using the provided camera matrices. P1 and P2 are the camera matrices. We also have to write a function 'findOutliers.m' that reprojects the world points to image2 and then deterines which points are outliers and inliers. We will call a point an outlier if the distance between the true location, and the reprojected point location is more than 20 pixels.

```
outlierTH = 20;
F = fund(cor1, cor2);
ncorners = 50;
corners1 = CornerDetect(I1, ncorners, smoothSTD, windowSize));
corsSSD = correspondanceMatchingLine( I1, I2, corners1, F, R, SSDth);
points3D = triangulate(corsSSD, P1, P2);
[ inlier, outlier ] = findOutliers(points3D, P2, outlierTH, corsSSD);
```

The results should be displayed as:

- original points - black circles
- inliers - blue plus points
- outliers - red plus signs

Do the detected outliers correspond to false matches?

Solution

The occurrence of outliers may be due to the inaccuracies in feature detection, false matching, and several errors in the estimation of fundamental matrices and projection matrices. A common source of error is mismatches satisfying the epipolar constraint by coincidence, which in turn lead to the reconstruction of erroneous 3D points that yield small reprojection errors.

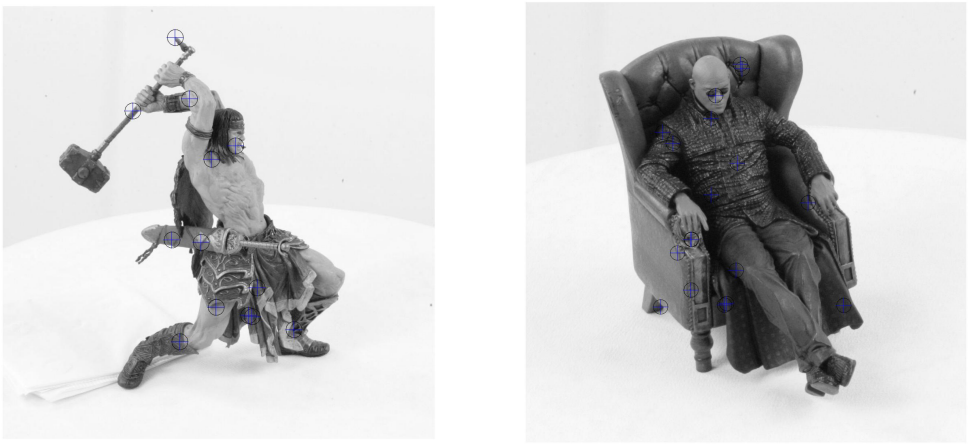

Figure 14: (a) Warrior Image (b) Matrix Image

Appendix

```
%% 4.1
% import data
filter = double(imread('/Users/junelee/Desktop/CSE252/hw2/data/filter.jpg'));
toy = double(imread('/Users/junelee/Desktop/CSE252/hw2/data/toy.png'));

% (1) Warmup
filter_ = filter - mean(filter(:));
toy_ = toy - mean(toy(:));

```

```
A = conv2(toy_, filter_, 'same');

%figure('Position', [100, 100, 500, 400]);
figure
colormap('jet');
imagesc(A);
colorbar;

[h w d] = size(A);
[h1 w1 d1] = size(filter);

counter = 0;
Max = max(A(:));

figure
imshow(toy);
hold on
for j = 1:h
    for i = 1:w
        if A(j, i) >= Max
            disp(j);disp(i);
            rectangle('Position',[i-w1/2 j-h1/2 w1 h1]);
        end
    end
end
```

Listing 1: MATLAB source for 4.1

```
%% 4.2
% (2) Detection error
figure
imshow(toy);
hold on
dx1 = 100; dy1 = 100;
x1 = 341 - dx1/2; y1 = 89 - dy1/2;
rectangle('Position', [x1 y1 dx1 dy1], 'EdgeColor', 'red');

dx2 = 100; dy2 = 90;
x2 = 341 - dx2/2; y2 = 89 - dy2/2;
rectangle('Position', [x2 y2 dx2 dy2], 'EdgeColor', 'blue');

dx3 = 98; dy3 = 84;
x3 = 341 - dx3/2; y3 = 89 - dy3/2;
rectangle('Position', [x3 y3 dx3 dy3], 'EdgeColor', 'green');
hold off

error1 = error_func(341, 89, w1, h1, x1, y1, dx1, dy1);
error2 = error_func(341, 89, w1, h1, x2, y2, dx2, dy2);
error3 = error_func(341, 89, w1, h1, x3, y3, dx3, dy3);
```

Listing 2: MATLAB source for 4.2

```
%% 4.3
% (3) More realistic images
% filter
temp = imread('/Users/junelee/Desktop/CSE252/hw2/data/cartemplate.jpg');

% car2
car2 = imread('/Users/junelee/Desktop/CSE252/hw2/data/car2.jpg');
temp2 = fliplr(temp);
temp2 = imrotate(temp2, -3);
temp2 = imcrop(temp2, [70 300 1650 700]);
```

```
temp2 = imresize(temp2, [70*3 165*3]);
figure
imshow(temp2);
e2 = conv_func(temp2, car2);
disp(e2);

%% car3
car3 = imread('/Users/junelee/Desktop/CSE252/hw2/data/car3.jpg');
%car3 = car3 - mean(car3(:));
temp3 = imcomplement(temp);
%temp3 = temp3 - mean(temp3(:));
[h w d] = size(temp3);
for i = 1:w
    for j = 1:h
        if temp3(j, i) < 20
            temp3(j, i) = 230;
        end
    end
end
temp3 = imrotate(temp3, -5);
temp3 = imcrop(temp3, [70 400 1800 600]);
temp3 = imresize(temp3, [120, 300]);
figure
imshow(temp3);
e3 = conv_func(temp3, car3);
disp(e3);

%% car4
car4 = imread('/Users/junelee/Desktop/CSE252/hw2/data/car4.jpg');
temp4 = fliplr(temp);
temp4 = imcomplement(temp4);
for i = 1:w
    for j = 1:h
        if temp4(j, i) < 20
            temp4(j, i) = 200;
        end
    end
end
temp4 = imcrop(temp4, [70 400 1800 800]);
temp4 = imresize(temp4, [120, 300]);
figure
imshow(temp4);
e4 = conv_func(temp4, car4);
disp(e4);

%% car5
car5 = imread('/Users/junelee/Desktop/CSE252/hw2/data/car5.jpg');
temp5 = imcomplement(temp);
[h w d] = size(temp5);
for i = 1:w
    for j = 1:h
        if temp5(j, i) < 20
            temp5(j, i) = 200;
        end
    end
end
temp5 = imcrop(temp5, [40 400 1800 400]);
temp5 = imresize(temp5, [200, 350]);
figure
imshow(temp5);
e5 = conv_func(temp5, car5);
```

```
disp(e5);
```

Listing 3: MATLAB source for 4.3

```
template = imread('cartemplate.jpg');
template = mean2(template) - template;
template = imresize(template,[100, 400]);
template = imrotate(template,180);
templates{count} = imresize(flip(template,2), [h w]);

im = imread('car1.jpg');
im2 = im;
im = mean2(im) - im;
heatmap = conv2(double(im),double(template),'same');
ma = max(heatmap(:));
mi = min(heatmap(:));
heatmap = (heatmap-mi)/(ma-mi)*255;
figure,imagesc(heatmap);
colormap('jet');
[values, indices] = sort(heatmap(:), 'descend');
[p, q] = ind2sub(size(heatmap), indices);
figure,imshow(im2);
hold on;
rectangle('Position',[q(1)-size(template,2)/2, p(1)-size(template,1)/2, size(template
    ,2), size(template,1)], 'EdgeColor', 'b');
rectangle('Position',[175, 145, 522-175, 245-145],'EdgeColor', 'g');
detect_rect = [q(1)-size(template,2)/2, p(1)-size(template,1)/2, size(template,2),
    size(template,1)];
gt_rect = [175, 145, 522-175, 245-145];
intersection_area = rectint(gt_rect, detect_rect);
detect_area = detect_rect(3)*detect_rect(4);
gt_area = gt_rect(3)*gt_rect(4);
iou = intersection_area/(detect_area+gt_area-intersection_area);
disp(iou);
```

Listing 4: MATLAB source for 4.3(car 1)

```
function e = error_func( i, j, w1, h1, x1, y1, dx1, dy1 );

% initial values
x = i - w1/2; y = j - h1/2;

minX = x;minY = y;
if x < x1
    minX = x1;
end
if y < y1
    minY = y1;
end
maxX = x + w1;maxY = y + h1;
if x + w1 > x1 + dx1
    maxX = x1 + dx1;
end
if y + h1 > y1 + dy1
    maxY = y1 + dy1;
end

pNq = (maxX-minX + 1) * (maxY - minY + 1);
p = w1 * h1;
q = dx1 * dy1;
pUq = p + q - pNq;
e = pNq/pUq;
```

```

end
```

Listing 5: MATLAB source for error function

```
function result = conv_func( temp, car_, x1, y1, x2, y2 );

% conversion
temp = double(temp);
temp = temp - mean(temp(:))
car = double(car_);
car = car - mean(car(:))

% convolution
A = conv2(car, temp, 'same');

% draw heatmap
figure
colormap('jet');
imagesc(A);
colorbar;

% find MAX position
[h w d] = size(A);
[h1 w1 d1] = size(temp);

Max = max(A(:));
Avg = mean(A(:));
disp(Max);
disp(Avg);
thres = 2*(Max+Avg)/3;
counter = 0;
avgX = 0;
avgY = 0;
% calculate bounding box
for j = 1:h
    for i = 1:w
        if A(j, i) >= Max%thres for car2&3, max for car4&5
            avgX = avgX + i;
            avgY = avgY + j;
            counter = counter + 1;
        end
    end
end

% draw bouding box
avgX = avgX / counter;
avgY = avgY / counter;
%disp(avgX);disp(avgY);
figure
imshow(car_);
hold on
% my calculation
rectangle('Position',[avgX-w1/2 avgY-h1/2 w1 h1], 'EdgeColor', 'red');
% groundtruth car2
%rectangle('Position', [69 205 w1 h1], 'EdgeColor', 'blue');
% groundtruth car3
%rectangle('Position', [400-w1/2 290-h1/2 w1 h1], 'EdgeColor', 'blue');
% groundtruth car4
%rectangle('Position', [320-w1/2 210-h1/2 w1 h1], 'EdgeColor', 'blue');
% groundtruth car5
rectangle('Position', [600-w1/2 400-h1/2 w1 h1], 'EdgeColor', 'blue');
hold off;
```

```
59
60 %result = error_func( avgX, avgY, w1, h1, 69, 205, 488-69, 357-205);
61 %result = error_func( avgX, avgY, w1, h1, 400-w1/2, 290-h1/2, w1, h1);
62 %result = error_func( avgX, avgY, w1, h1, 320-w1/2, 210-h1/2, w1, h1);
63 result = error_func( avgX, avgY, w1, h1, 600-w1/2, 400-h1/2, w1, h1);
64 end
```

Listing 6: MATLAB source for conv function

```
1  im = imread('geisel.jpeg');
2  [h, w] = size(im);
3  gaussian = 1/159*[2, 4, 5, 4, 2;...
4                    4, 9, 12, 9, 4;...
5                    5, 12, 15, 12, 5;...
6                    4, 9, 12, 9, 4;...
7                    2, 4, 5, 4, 2;];
8  im = conv2(double(im), double(gaussian), 'same');
9  figure,imagesc(im);colormap(gray);
10 sobel_x = [-1, 0, 1;...
11           -2, 0, 2;...
12           -1, 0, 1;];
13 sobel_y = [-1, -2, -1;...
14           0, 0, 0;...
15           1, 2, 1;];
16 gx = conv2(im, sobel_x, 'same');
17 gy = conv2(im, sobel_y, 'same');
18 figure,imagesc(gx);
19 colormap(gray);
20 figure,imagesc(gy);
21 colormap(gray);
22 gmag = sqrt(gx.*gx+gy.*gy);
23 figure,imagesc(gmag);
24 colormap(gray);
25 gtheta = atan2(gy,gx);
26 coords = zeros(2);
27 gmag = padarray(gmag,[1, 1]);
28 gtheta = padarray(gtheta,[1, 1]);
29 for i=2:h+1
30     for j=2:w+1
31         if gmag(i,j)>0
32             dir = gtheta(i,j)/(pi/8);
33             if (dir>=-1 && dir<=1 || dir<=-7 && dir>=7)
34                 coords = [i,j+1;i,j-1];
35             elseif (dir>1 && dir<=3 || dir>-7 && dir<=-5)
36                 coords = [i-1,j+1;i+1,j-1];
37             elseif (dir>3 && dir<=5 || dir>-5 && dir<=-3)
38                 coords = [i-1,j;i+1,j];
39             else
40                 coords = [i+1,j+1;i-1,j-1];
41             end
42             if (gmag(i,j)<gmag(coords(1,1),coords(1,2)) || gmag(i,j)<gmag(coords(2,1)
   ,coords(2,2)) || i==2 || j==2 || i==h+1 || j==w+1)
43                 gmag(i,j) = 0;
44             end
45         end
46     end
47 end
48 gmag = gmag(2:h+1,2:w+1);
49 ma = max(gmag(:));
50 mi = min(gmag(:));
51 gmag = (gmag-mi)/(ma-mi)*255;
52 figure,imagesc(gmag);
53 colormap(gray);
```

```
T2 = 40;
T2mask = gmag>T2;
T1 = 20;
[T1r, T1c] = find(gmag>T1);
bw = bwselect(T2mask, T1c, T1r, 8);
figure,imagesc(bw);
colormap(gray);
```

Listing 7: MATLAB source for 5

```
function corners = CornerDetect( Image, nCorners, smoothSTD, windowSize );

% grayscale
A = rgb2gray(Image);

% gaussian filter g_sig and g_size
g_size = windowSize;
g_sig = smoothSTD;
B = imgaussfilt(A, g_sig, 'FilterSize', g_size);

% compute derivatives
dx = [-1 0 1; -1 0 1; -1 0 1];
dy = dx';

Ix = conv2(A, dx, 'same');
Iy = conv2(A, dy, 'same');

% find products of derivatives
g_size = 3;
Ix2 = imgaussfilt(Ix.^2, g_sig, 'Filtersize', g_size);
Iy2 = imgaussfilt(Iy.^2, g_sig, 'Filtersize', g_size);
Ixy = imgaussfilt(Ix.*Iy, g_sig, 'Filtersize', g_size);


% construct matrix C and find eigenvalues
% using det and trace is much faster
[h w d] = size(Ix);
E = zeros(h, w);
for i = 10:w-10
    for j = 10:h-10
        % https://courses.cs.washington.edu/courses/cse455/09wi/Lects/lect6.pdf
        E(j, i) = (Ix2(j, i) * Iy2(j, i) - Ixy(j, i).^2) - 0.04 * (Ix2(j, i) + Iy2(j, i)).^2;
    end
end

% find local maxima
dia = g_size*20;
Max = ordfilt2(E, dia^2, ones(dia, dia));

% make an image of local maxima and above threshold
for i = 1:w
    for j = 1:h
        if E(j, i) ~= Max(j, i)% || E(j, i) <= 0.01%0.1 for dino
            E(j, i) = 0;
        end
    end
end

% sort by maxima
S = sort(E(:), 'descend');

% find the 20th maximum value
```

```
value = S(nCorners);

% create corner array
corners = zeros(nCorners, 2);
counter = 0;

figure
imshow(A);
hold on

for i = 10:w-10
    for j = 10:h-10
        if E(j, i) >= value && counter < nCorners
            counter = counter + 1;
            rectangle('Position',[i-windowSize*5 j-windowSize*5 windowSize*10
    windowSize*10],'Curvature', [1 1], 'EdgeColor', 'blue', 'LineWidth',3);
            corners(counter, 1) = j; corners(counter, 2) = i;
        end
    end
end
hold off;

end
```

Listing 8: MATLAB source for 6.1

```
function sum = SSDmatch(x1, y1, x2, y2, I1, I2, windowSize);

% add up
sum = 0;

for i = 1:windowSize
    for j = 1:windowSize
        Y1 = y1 - windowSize/2 + j;
        X1 = x1 - windowSize/2 + i;
        Y2 = y2 - windowSize/2 + j;
        X2 = x2 - windowSize/2 + i;
        if X1 > 0 && Y1 > 0 && X2 > 0 && Y2 > 0
            sum = sum + (I1(Y1, X1) - I2(Y2, X2)).^2;
        else
            sum = 1000;
        end
    end
end


end
```

Listing 9: MATLAB source for 6.2

```
function I = naiveCorrespondanceMatching( I1, I2, corners1, corners2, R, SSDth );

[num coord] = size(corners1);

% find best match
A = zeros(num, 3);

minSum = 1000;
minInd_m = 0;

for n = 1:num
    for m = 1:num
```

```
        sum = SSDmatch(corners1(n, 2), corners1(n, 1), corners2(m, 2), corners2(m, 1)
    , I1, I2, R);
        if sum < minSum
            minSum = sum;
            minInd_m = m;
        end
    end

    A(n, 1) = n;
    A(n, 2) = minInd_m;
    A(n, 3) = minSum;
    minSum = 1000;minInd_m = 0;
end

% create new image I
[h1 w1 d1] = size(I1);
[h2 w2 d2] = size(I2);

h = max(h1, h2);
w = max(w1, w2);

I = zeros(h, 2*w);

for j = 1:h1
    for i = 1:w1
        I(j, i) = I1(j, i);
    end
end

for j = 1:h2
    for i = 1:w2
        I(j, i + w1) = I2(j, i);
    end
end

% draw circles and lines
figure
imshow(I);
hold on
for it = 1:num
    if A(it, 3) <= SSDth
        y1 = corners1(it, 1);
        x1 = corners1(it, 2);
        y2 = corners2(A(it, 2), 1);
        x2 = corners2(A(it, 2), 2);
        rectangle('Position',[x1-R y1-R R*2 R*2],'Curvature', [1 1], 'EdgeColor', '
    red', 'LineWidth',3);
        rectangle('Position',[x2+w1-R y2-R R*2 R*2],'Curvature', [1 1], 'EdgeColor',
    'red', 'LineWidth',3);
        line([x1,x2+w1],[y1,y2], 'LineWidth', 3, 'Color', 'blue');
    end
end
hold off;
```

Listing 10: MATLAB source for 6.3

```
% load data
% dino
%{
data = load('/Users/junelee/Desktop/CSE252/hw2/data/dino2.mat');
[h1 w1 d1] = size(data.dino01);
[h2 w2 d2] = size(data.dino02);

```

```
I1 = data.dino01;
I2 = data.dino02;
%}

% matrix
%{
data = load('/Users/junelee/Desktop/CSE252/hw2/data/matrix2.mat');
[h1 w1 d1] = size(data.matrix01);
[h2 w2 d2] = size(data.matrix02);

I1 = data.matrix01;
I2 = data.matrix02;
%}

% warrior

data = load('/Users/junelee/Desktop/CSE252/hw2/data/warrior2.mat');
[h1 w1 d1] = size(data.warrior01);
[h2 w2 d2] = size(data.warrior02);

I1 = data.warrior01;
I2 = data.warrior02;


smoothSTD = 3;
windowSize = 9;
R = 20;
SSDth = 10;%5;

%% 6.1
nCorners = 20;
CornerDetect(I1, nCorners, smoothSTD, windowSize);
CornerDetect(I2, nCorners, smoothSTD, windowSize);

%% 6.3
nCorners = 10;
C1 = CornerDetect(I1, nCorners, smoothSTD, windowSize);
C2 = CornerDetect(I2, nCorners, smoothSTD, windowSize);
I = naiveCorrespondanceMatching( I1, I2, C1, C2, R, SSDth);%10^11

%% 6.4
cam1 = data.cor1;
cam2 = data.cor2;
F = fund(cam1, cam2);

% right null space
er = null(F);
er = er./er(3, 1);

% left null space
el = (null(F.'));
el = el./el(3, 1);

figure
imshow(rgb2gray(I1));
hold on
for i = 1:10
    x1 = C1(i, 2);
    y1 = C1(i, 1);
    rectangle('Position',[x1-R y1-R R*2 R*2],'Curvature', [1 1], 'EdgeColor', 'blue',
     'LineWidth',3);
    One = [x1 y1 1];
```

```
    Two = [er(1, 1) er(2, 1) 1];
    Three = cross(One, Two);
    Three = Three.';
    pts = linePts(Three,[1, w1],[1, h1]);
    line([pts(1, 1), pts(2, 1)], [pts(1, 2), pts(2, 2)], 'Color', 'blue', 'LineWidth'
    ,3);
end
hold off

figure
imshow(rgb2gray(I2));
hold on
for i = 1:10
    x2 = C2(i, 2);
    y2 = C2(i, 1);
    rectangle('Position',[x2-R y2-R R*2 R*2],'Curvature', [1 1], 'EdgeColor', 'blue',
     'LineWidth',3);
    One = [x2 y2 1];
    Two = [el(1, 1) el(2, 1) 1];
    Three = cross(One, Two);
    Three = Three.';
    pts = linePts(Three,[1, w2],[1, h2]);
    line([pts(1, 1), pts(2, 1)], [pts(1, 2), pts(2, 2)], 'Color', 'blue', 'LineWidth'
    ,3);
end
hold off

%% 6.5
ncorners = 10;
F = fund(cam1, cam2);
corners1 = CornerDetect(I1, ncorners, smoothSTD, windowSize);
corsSSD = correspondanceMatchingLine( I1, I2, corners1, F, R, SSDth);
```

Listing 11: MATLAB source for 6.4

```
function corsSSD = correspondanceMatchingLine( I1, I2, corners1, F, R, SSDth);

[h2, w2, d2] = size(I2);
[h, w] = size(corners1);
result = zeros(h, 3);
counter = 0;

figure
imshow(I2);
hold on
for i = 1:h
    x1 = corners1(i, 2);
    y1 = corners1(i, 1);

    lr = F*[x1; y1; 1];

    minSum = 1000;
    minInd = 0;
    for x = 1:w2
        px = x;
        py = round((-lr(1)*x - lr(3)*1)./lr(2));
        if py >= 1+R && py <= h2-R
            sum = SSDmatch(x1, y1, px, py, I1, I2, R);
            if sum < minSum
                minSum = sum;
                minInd = x;
            end
        end
```

```
        %x = x + 5;
    end

    if minSum < SSDth
        coordX = minInd;
        coordY = round((-lr(1)*coordX - lr(3)*1)./lr(2));
        %rectangle('Position',[coordX-20 coordY-20 40 40],'Curvature', [1 1], '
    EdgeColor', 'blue');
        counter = counter + 1;
        result(counter, 1) = coordY;result(counter, 2) = coordX;result(counter, 3) =
    i;
    end


end

hold off

final = result(1:counter, :);
corsSSD = final;

% create new image I
[h1 w1 d1] = size(I1);
[h2 w2 d2] = size(I2);

h = max(h1, h2);
w = max(w1, w2);

I = zeros(h, 2*w);

for j = 1:h1
    for i = 1:w1
        I(j, i) = I1(j, i);
    end
end

for j = 1:h2
    for i = 1:w2
        I(j, i + w1) = I2(j, i);
    end
end

% draw circles and lines
figure
imshow(I);
hold on
for it = 1:counter
    x1 = corners1(corsSSD(it, 3), 2);
    y1 = corners1(corsSSD(it, 3), 1);
    x2 = corsSSD(it, 2);
    y2 = corsSSD(it, 1);
    rectangle('Position',[x1-R y1-R R*2 R*2],'Curvature', [1 1], 'EdgeColor', 'red',
    'LineWidth',3);
    rectangle('Position',[x2+w1-R y2-R R*2 R*2],'Curvature', [1 1], 'EdgeColor', 'red
    ', 'LineWidth',3);
    line([x1,x2+w1],[y1,y2], 'Color', 'blue', 'LineWidth',3);
end
hold off;

end
```

Listing 12: MATLAB source for 6.5

```
%% Triangulation
outlierTH = 20;
[~, ~, v] = svd(F');
e = v(:,3)/v(3,3);
ex = [0 -e(3) e(2);...
      e(3) 0 -e(1);...
      -e(2) e(1) 0];
P1 = eye(3,4);
P2 = [ex*F, e];
points3D = triangulate(corsSSD, P1, P2);
[ inlier, outlier ] = findOutliers(points3D, P2, outlierTH, corsSSD);

figure,imshow(I2);
hold on;
for i=1:n
    plot(matchedPoints2(i,1),matchedPoints2(i,2),'ko','MarkerSize',20);
end
n = size(inlier,1);
for i=1:n
    plot(inlier(i,1),inlier(i,2),'b+','MarkerSize',20);
end
n = size(outlier,1);
for i=1:n
    plot(outlier(i,1),outlier(i,2),'r+','MarkerSize',20);
end
```

Listing 13: MATLAB source for 6.6

```
function [ inlier, outlier ] = findOutliers(points3D, P2, outlierTH, corsSSD)
n = size(corsSSD,1);
inlier = [];
outlier = [];
for i=1:n
    p = P2*points3D{i};
    p = floor(p./p(3));
    dist = pdist2([p(1) p(2)],[corsSSD(i,3), corsSSD(i,4)]);
    if(dist<outlierTH)
        inlier = [inlier; p(1:2)'];
    else
        outlier = [outlier; p(1:2)'];
    end
end
```

Listing 14: MATLAB source code for findOutliers function

```
function [points3D] = triangulate(corsSSD, P1, P2)
n = size(corsSSD,1);
points3D = cell([1,n]);
for i=1:n
    p1 = [corsSSD(i,1:2) 1];
    p2 = [corsSSD(i,3:4) 1];
    x1 = p1(1);
    y1 = p1(2);
    x2 = p2(1);
    y2 = p2(2);
    A = [x1.*P1(3,:) - P1(1,:);...
         y1.*P1(3,:) - P1(2,:);...
         x2.*P2(3,:) - P2(1,:);...
         y2.*P2(3,:) - P2(2,:)];
    A(1,:) = A(1,:)./norm(A(1,:));
    A(2,:) = A(2,:)./norm(A(2,:));
    A(3,:) = A(3,:)./norm(A(3,:));
```

```
    A(4,:) = A(4,:)./norm(A(4,:));
    [~,~,v] = svd(A);
    points3D{i} = v(:,4);
    points3D{i} = points3D{i}./points3D{i}(4);
end
```

Listing 15: MATLAB source for triangulate function

```
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';

%% 4. Undo the normalization
```

```
F = H2'*F*H1;
```

Listing 16: MATLAB source for fund function

```
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
```

```
58 end
```

Listing 17: MATLAB source for linePts function

Submitted by Gupta, Ajitesh
Choi, Jean
Mavalankar, Aditi Ashutosh on November 7, 2016.