CSE 252A – Computer Vision – Homework 2
Instructor: Ben Ochoa
Revision 1

Instructions:

- This homework should be done in pairs.
- Submit your homework electronically by email to nkinkade@eng.ucsd.edu AND akdave@eng.ucsd.edu with the subject line *CSE 252A Homework 2*. The email should have one file attached. Name this file: `CSE_252A_hw2_lastname1_lastname2.zip`. The contents of the file should be:

  1. A pdf file with your writeup. This should have all code attached in the appendix. Name this file: `CSE_252A_hw2_lastname1_lastname2.pdf`.
  2. All of your source code in a folder called code.

  The code is thus attached *both* as text in the writeup appendix and as source-files in the compressed archive.
- No physical hand-in for this assignment.
- Coding is to be done only in MATLAB.
- In general, MATLAB code does not have to be efficient. Focus on clarity, correctness and function here, and we can worry about speed in another course.

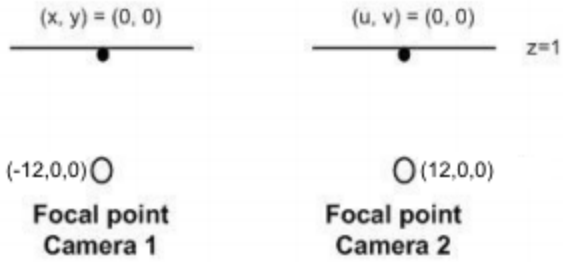
1 Epipolar Geometry Theory [3pt]

Suppose a camera calibration gives a transformation (R, T) such that a point in the world maps to the camera by ${}^{C}P = R{}^{W}P + T$.

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 the right camera to that of the left.
2. What is the length of the baseline of the stereo pair.
3. Give an expression for the Essential Matrix in terms of $R1$, $T1$, $R2$, $T2$.

2 Epipolar Geometry [3pt]

Consider two cameras whose image planes are the $z = 1$ plane, and whose focal points are at $(-12, 0, 0)$ and $(12, 0, 0)$. We'll call a point in the first camera (x, y), and a point in the second camera (u, v). Points in each camera are relative to the camera center. So, for example if $(x, y) = (0, 0)$, this is really the point $(-12, 0, 1)$ in world coordinates, while if $(u, v) = (0, 0)$ this is the point $(12, 0, 1)$.

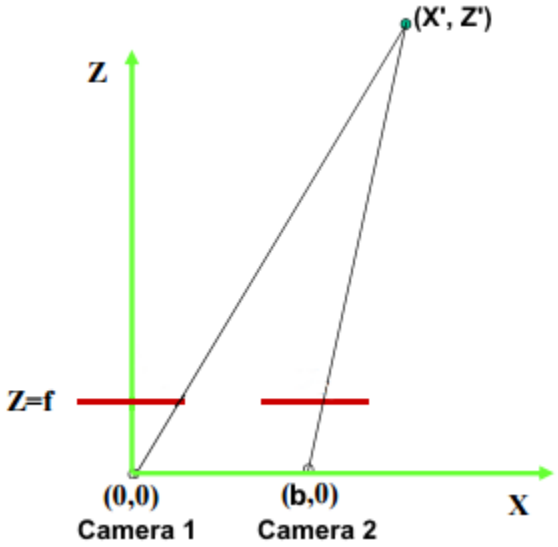


1. Suppose the points $(x, y) = (8, 7)$ is matched with disparity of 6 to the point $(u, v) = (2, 7)$. What is the 3D location of this point?

2. Consider points that lie on the 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 $z > 1$.

3 Reconstruction Accuracy [2pt]

Characterize the accuracy of the 2D stereo system below. Assume the only source of noise is the localization of corresponding points in the two images (in other words, the only source of error is the disparity). Discuss the dependence of the error in depth estimation ($\Delta Z'$) as a function of baseline width (b), focal length (f), and depth (Z').



4 Filters as Templates [8pt]

In this problem we will play with convolution filters. Filters, when convolved with an image, will fire strongest on locations of an image that look like the filter. This allows us to use filters as object templates in order to identify specific objects within an image. In the case of this assignment, we will be finding cars within an image by convolving a car template onto that image. Although this is not a very good way to do object detection, this problem will show you some of the steps necessary to create a good object detector. The goal of this problem will be to teach some pre-processing steps to make vision algorithms be successful and some strengths and weaknesses of filters. Each problem will ask you to analyze and explain your results. If you do not provide an explanation of why or why not something happened, then you will not get full credit. Provide your code in the appendix.

4.1 Warmup [2pt]

First you will convolve a filter to a synthetic image. The filter or template is `filter.jpg` and the synthetic image is `toy.png`. These files are available on the course webpage. You may want to modify the filter image and original slightly. I suggest `filter_img = filter_img - mean(filter_img(:))`. To convolve the filter image with the toy example, in Matlab you will want to use `conv2`. The output of the convolution will create an intensity image. Provide this image in the report. In the original image (not the image with its mean subtracted), draw a bounding box of the same size as the filter image around the top 3 intensity value locations in the convolved image. The outputs should look like Figure 1. Describe how well you think this will technique will work on more realistic images? Do you foresee any problems for this algorithm on more realistic images?

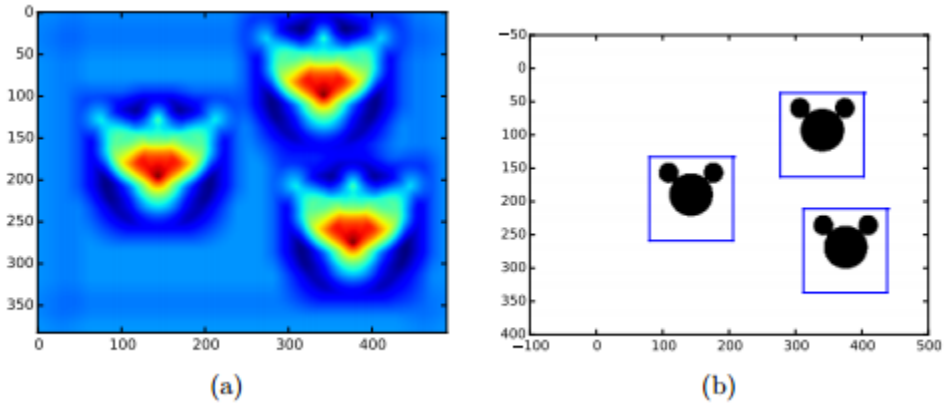


Figure 1: Example outputs for the synthetic example. (a) Heat map. (b) Bounding boxes

4.2 Detection Error [2pt]

We have now created an algorithm that produces a bounding box around a detected object. However we have no way to know if the bounding box is good or bad. In the example images shown above, the bounding boxes look reasonable, but not perfect. Given a ground truth bounding box (g) and a predicted bounding box (p), a commonly used measurement for bounding box quality is $\frac{p \cap g}{p \cup g}$. More 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 squares (and not diamonds), implement this error function and try it on the toy example in the previous section. Choose 3 different ground truth bounding box sizes around one of the Mickey

silhouettes. In general, if the overlap is 50% or more, you may consider that the detection did a good job.

4.3 More Realistic Images [2pt]

Now that you have created an algorithm for matching templates and a function to determine the quality of the match, it is time to try some 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 files that contains 2 x, y coordinates (one pair per line). These coordinates will be the ground truth bounding box for each image. For each car image, provide the following:

- A heat map image
- A bounding box drawn on the original image.
- The bounding box overlap percent.
- A description of what pre-processing steps you needed to do to achieve this overlap.
- An explaination of why you felt these steps made sense.

Here are a few suggestions for your algorithm:

- Rescale the car template to various sizes
- Rotate the car template to various rotations
- Blur either the template or the test image or both

An example output is show for `car1.jpg` in Figure 2. It may not be possible to achieve 50% overlap on all the images. Your analysis of the images will be worth as much as achieving a high overlap percentage.

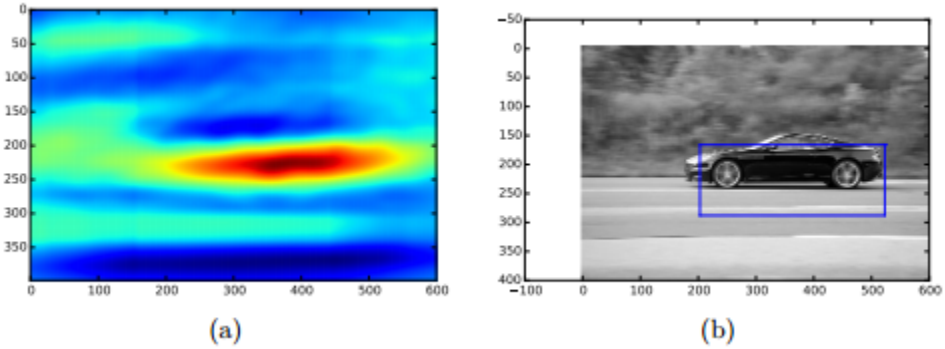


Figure 2: Example outputs for the car example. (a) Heat map. (b) Bounding boxes

4.4 Invariance [2pt]

In computer vision there is often a desire for features or algorithms to be invariant to X. One example briefly described in class was illumination invariance. The detection algorithm that was implemented for this problem may have seemed a bit brittle. Can you describe a few things that this algorithm was not invariant to? For example, this algorithm was not scale-invariant, meaning the size of the filter with respect to the size of the object being detected mattered. One filter size should not have worked on everything.

5 Sparse Stereo Matching [10pt]

In this problem we will play around with sparse stereo matching methods. You will work on two image pairs, a warrior figure and a figure from the Matrix movies (`warrior2.mat` and `matrix2.mat`). These files both contain two images, two camera matrices, and set sets of corresponding points (extracted by manually clicking the images). For illustration, I have run my code on a third image pair (`dino2.mat`). This data is also provided on the webpage for you to debug your code, but you should only report results on warrior and matrix. In other words, where I include one (or a pair) of images in the assignment below, you will provide the same thing but for BOTH matrix and warrior. Note that the matrix image pair is harder, in the sense that the matching algorithms we are implementing will not work quite as well. You should expect good results, however, on warrior. To make the TAs extra happy, make the line width and marker sizes bigger than the default sizes.

5.1 Corner Detection [2pt]

The first thing we need to do is to build a corner detector. This should be done according to `http://cseweb.ucsd.edu/classes/fa15/cse252A-a/lec7.pdf`. Your file should be named `CornerDetect.m`, and take as input

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

where `smoothSTD` is the standard deviation of the smoothing kernel, and `windowSize` the size of the smoothing window. In the lecture the corner detector was implemented using a hard threshold. Do not do that but instead return the `nCorners` strongest corners after non-maximum suppression. This way you can control exactly how many corners are returned. Run your code on all four images (with `nCorners` = 20) and show outputs as in Figure 3.

Figure 3: Result of corner detection

5.2 SSD Matching [1pt]

Write a function `SSDmatch.m` that implements the SSD matching algorithm for two input windows. Include this code in your report (in appendix as usual).

5.3 Naive Matching [2pt]

Equipped with the corner detector and the SSD matching code, we are ready to start finding correspondances. One naive strategy is to try and find the best match between the two sets of corner points. Write a script that does this, namely, for each corner in image1, find the best match from the detected corners in image2 (or, if the SSD match score is too low, then return no match for that point). You will have to figure out a good threshold (SSDth) value by experimentation. Write a function `naiveCorrespondanceMatching.m` and call it as below. Examine your results for 10, 20, and 30 detected corners in each image. In your report, only include your results for 10 corners, so that the figure will not be cluttered. `naiveCorrespondanceMatching.m` will call your SSD mathching code. Include a figure like Figure 4 in your report. The parameter `R` below, is the radius of the patch used for matching.

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

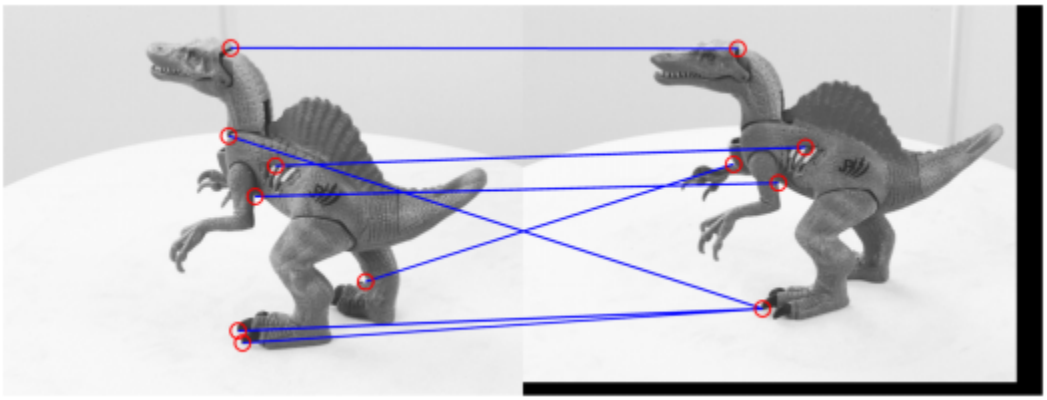


Figure 4: Result of naive matching

5.4 Epipolar Geometry [1pt]

Using the provided `fund.m` together with the provided points `cor1`, and `cor2`, calculate the fundamental matrix, and then plot the epipolar lines in both images pairs as shown below. Plot the points and make sure the epipolar lines go through them. Show your output as in Figure 5. You might find the supplied `linePts.m` function useful when you are working with epipolar lines.

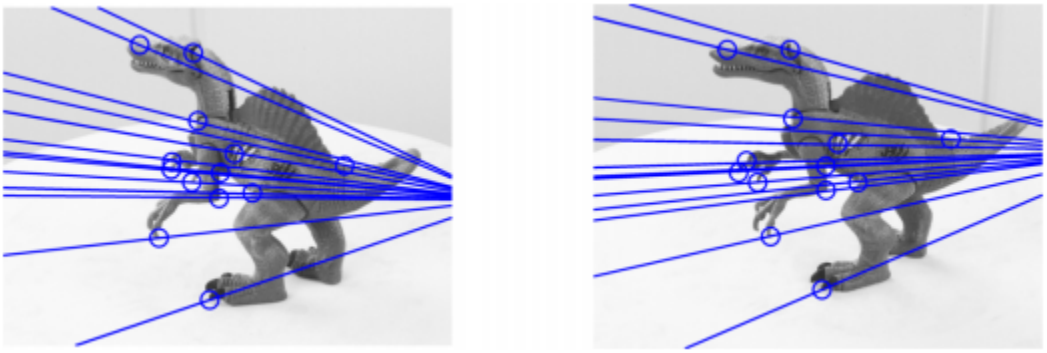


Figure 5: Epipolar lines

5.5 Epipolar Geometry Based Matching [2pt]

We will now use the epipolar geometry to build a better matching algorithm. First, detect 10 corners in Image1. Then, for each corner, do a linesearch along the corresponding epipolar line in Image2. Evaluate the SSD score for each point along this line and return the best match (or no match if all scores are below the SSDth). R is the radius (size) of the SSD patch in the code below. You do not have to run this in both directions, but only as indicated in the code below.

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

Your resulting plots should look like Figure 6.

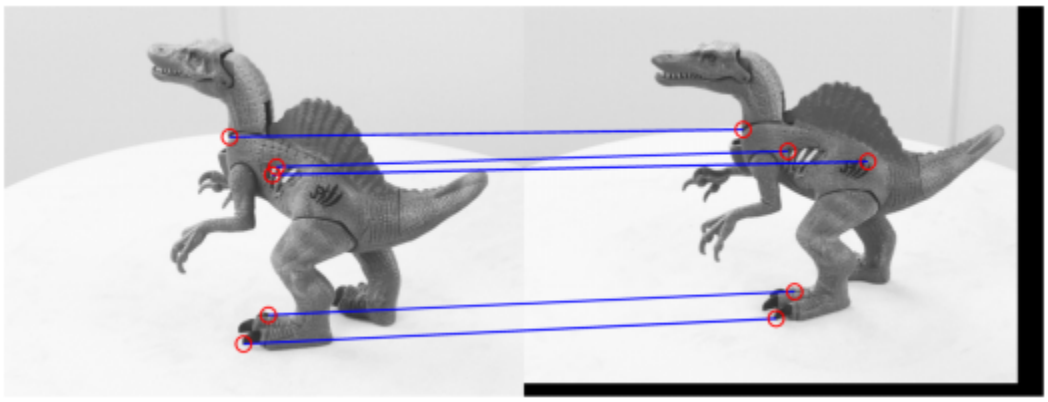


Figure 6: Result of epipolar matching

5.6 Triangulation [2pt]

Now that you have found correspondences between the pairs of images we can triangulate the corresponding 3D points. Since we do not enforce the ordering constraint the correspondences you have found are likely to be noisy and to contain a fair amount of outliers. Using the provided camera matrices you will triangulate a 3D point for each corresponding pair of points. Then by reprojecting the 3D points you will be able to find most of the outliers. You should implement the linear triangulation method described in lecture (For additional reference, see section 12.2 of "Multiple View Geometry" by Hartley and Zisserman. The book is available electronically from any UCSD IP address.). `P1` and `P2` below are the camera matrices. Also write a function, `findOutliers.m`, that reprojects the world points to Image2, and then determines which points are outliers and inliers respectively. For this purpose, 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);
```

Display your results by showing, for Image2, the original points in black circles, the inliers as blue plus signs, and the outliers as red plus signs, as shown in Figure 7. Compare this outlierplot with Figure 6. Does the detected outliers correspond to false matches?

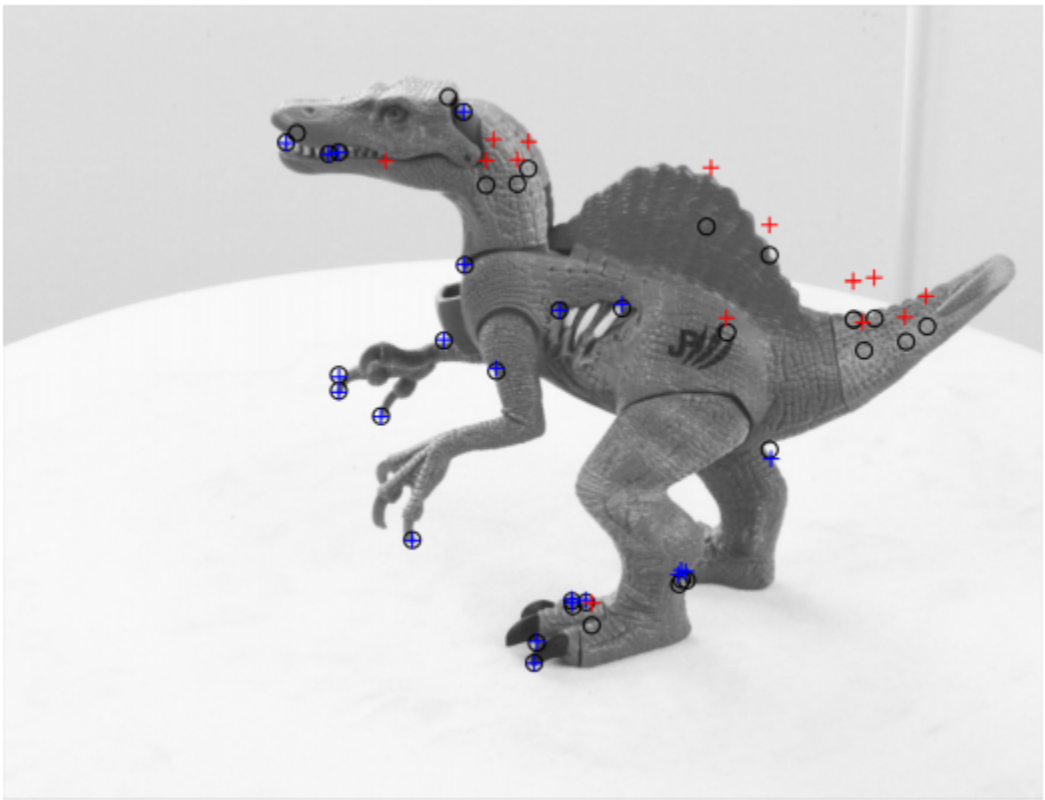


Figure 7: Result of reprojection