CS 231A Computer Vision (Winter 2014)
Problem Set 3

Due: Feb. 18th, 2015 (11:59pm)

1 Single Object Recognition Via SIFT (45 points)

In his 2004 SIFT paper, David Lowe demonstrates impressive object recognition results even in situations of affine variance and occlusion. In this problem, we will explore a similar approach for recognizing and locating a given object from a set of test images. It might be useful to familiarize yourself with sections 7.1 and 7.2 of the paper[1]. The code and data necessary to solve this problem can be found at `www.stanford.edu/class/cs231a/hw/hw3/PS3_data.zip`.

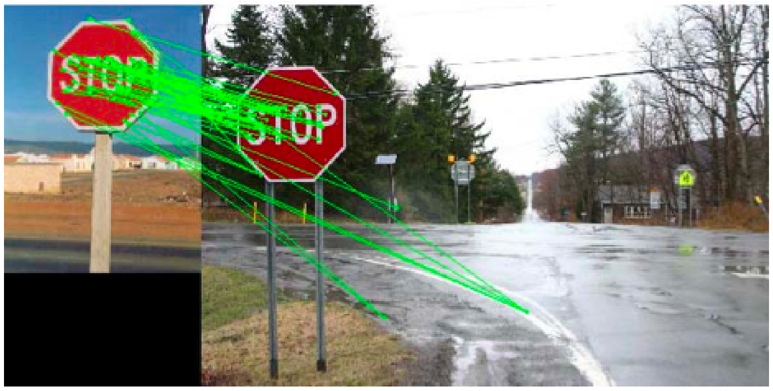


Figure 1: Sample Output, showing training image and keypoint correspondences.

(a) Given the descriptor g of a keypoint in an image and a set of keypoint descriptors from another image $f_1...f_n$, write the algorithm and equations to determine which keypoint in $f_1...f_n$ (if any) matches g. Implement this matching algorithm in the given function `matchKeypoints.m` and test its performance using the `matchObject.m` skeleton code. The matching algorithm should be based on the one used by Lowe in his paper, using the ratio of the two closest matches to determine if a keypoint has a match. Please read the section on object recognition for more details. Load the data in `PS3_Prob3.mat` and run the system with the following line:

```
>>matchObject(stopim{1}, sift_desc{1}, keypt{1}, obj_bbox, stopim{3}, ...
 sift_desc{3}, keypt{3});
```

[1]`http://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf`

Note that the SIFT keypoints and descriptors are given to you in `PS3_Prob3.mat` file. Your result should match the sample output in Fig. 1. [Turn in your code and a sample image similar to Fig. 1]

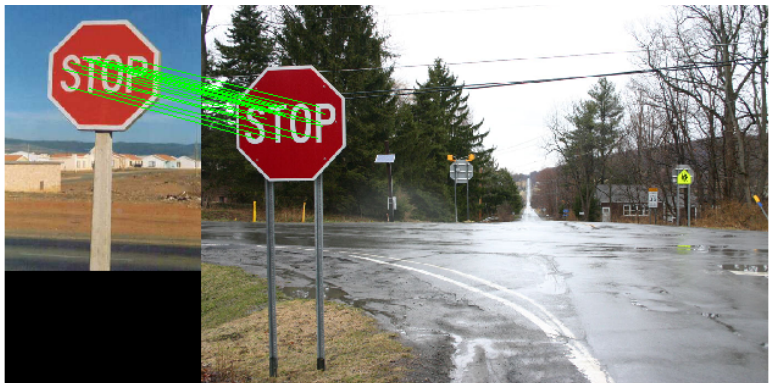


Figure 2: Sample Output, showing training image and keypoint correspondences with RANSAC filtering.

(b) From Figure 1, we can see that there are several spurious matches that match different parts of the stop sign with each other. To remove these matches, we can use a technique called RANSAC to find matches that are consistent with a homography between the locations of the matches in the two images.

The RANSAC algorithm generates a model from a random subset of data and then calculates how much of the data agrees with the model. In the context of this problem, a random subset of matches and their respective key point locations in each image is used to generate a homography in the form of $x_i^{'} = Hx_i$ where x_i and $x_i^{'}$ are the homogeneous coordinates of matching key points of the first and second images respectively. We then calculate the per-keypoint reprojection error by applying H to x_i:

$$error_i = ||x_i' - Hx_i||_2,$$

where x_i' and Hx_i should be converted back to nonhomogeneous coordinates. The inliers of the model are those with an error smaller than a given threshold.

We then iterate by choosing another set of random matches to find a new H and repeat the process, keeping track of the model with the most inliers. This is the model and inliers returned by `refineMatch.m`. The result of using RANSAC to filter the matches can be seen in Figure 2.

Implement this RANSAC algorithm in the given function `refineMatch.m` and test its performance using `matchObject.m`. A skeleton for the code as well as parameters such as the number of iterations and allowable error for inlier detection have been included.

(c) We will now explore some theoretical properties of RANSAC.

(i) Suppose that e is the fraction of outliers in your dataset, i.e.

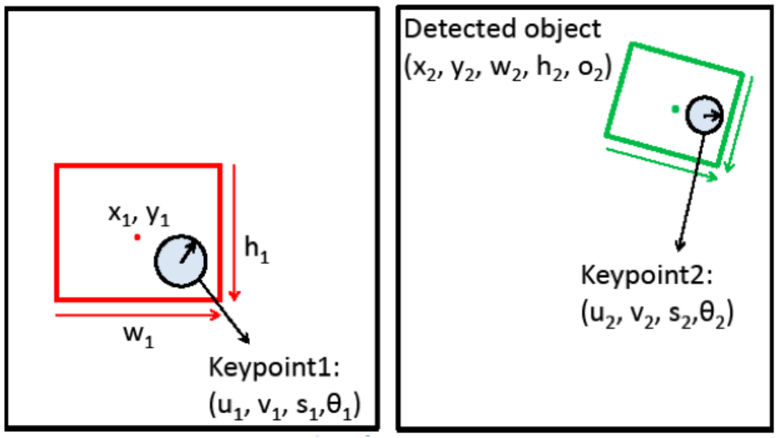
$$e = \frac{\#\text{ outliers}}{\#\text{total correspondences}}$$

If we choose a single random set of matches to compute a homography, as we did above, what is the probability that this set of matches will produce the correct homography?

(ii) Let p_s be your answer from above, i.e. p_s is the probability of sampling a set of points that produce the correct homography. Suppose we sample N times. In terms of p_s, what is the probability p that at least one of the samples will produce the correct homography? Remember, sets of points are sampled with replacement, so models are independent of one another.

(iii) Combining your answers for the above, if 15% of the samples are outliers and we want at least a 99% guarantee that at least one of the samples will give the correct homography, then how many samples do we need?

(d) Now given an object in an image, we want to explore how to find the same object in another image by matching the keypoints across these two images.


x_1, y_1
h_1
w_1
Keypoint1:
$(u_1, v_1, s_1, \theta_1)$
Detected object
$(x_2, y_2, w_2, h_2, o_2)$
Keypoint2:
$(u_2, v_2, s_2, \theta_2)$

(a) Image 1 (b) Image 2

Figure 3: Two sample images for part (c)

(i) A keypoint is specified by its coordinates, scale and orientaion (u, v, s, θ). Suppose that you have matched a keypoint in the bounding box of an object in the first image to a keypoint in a second image, as shown in Figure 3. Given the keypoint pair and the red bounding box in Image 1, which is specified by its center coordinates, width and height (x_1, y_1, w_1, h_1), find the predicted green bounding box of the same object in Image 2. Define the center position, width, height and relative orientation $(x_2, y_2, w_2, h_2, o_2)$ of the predicted bounding box. Assume that the relation between a bounding box and a keypoint in it holds across rotation, translation and scale.

(ii) Once you have defined the five features of the new bounding box in terms of the two keypoint features and the original bounding box, briefly describe how you would utilize the Hough transform to determine the best bounding box in Image 2 given n correspondences.

(e) Implement the function `getObjectRegion.m` to recover the position, scale, and orientation of the objects (via calculating the bounding boxes) in the test images. You can use a coarse Hough transform by setting the number of bins for each dimension equal to 4. Your Hough space should be four dimensional.

Use the line in (a) to test your code and change all the 3's to 2, 4, 5 to test on different images. If you are not able to localize the objects (this could happen in two of the test images), explain what makes these cases difficult. [Turn in your `getObjectRegion.m` and matching result images.]

2 Single Object Matching Via Shape Context (20 points)

Depending on a given problem, some feature descriptors may be better suited than others for object recognition. In contrast to other detectors studied in class, Shape Context [Belongie et al 2002] measures similarity between shapes and exploits it for classification. The methodology also calculates a transformation that would maximally align two potential matches. It will be useful to familiarize yourself with section 3.1 as well as the introduction of section 3 in the paper[2].

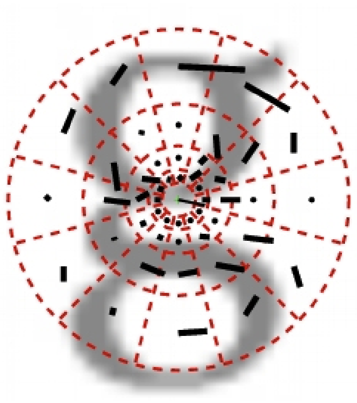


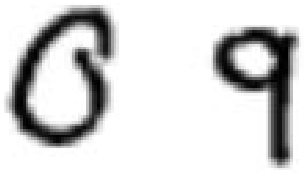
Figure 4: Visualization of polar-coordinate binning

(a) One natural application of Shape Context is to match handwritten digits. In this problem we have provided the data, interest point detection, and matching code for you.

 (i) Write a function `compute_shape_context.m` to generate the shape context feature to be used when matching. Specifically your function should take in the minimum radius, maximum radius, number of bins for radius, number of bins for angle, the input data from the edge detection, the mean radius and outliers. Your code should output the mean radius and the shape context feature for each data point. Write your code in `compute_shape_context.m`. In this file there is a description of the input and output data structures, as well as detailed skeleton code on what to do. This code will take in data from the interest point detector and output data to be used in the matching process for shape context. Run the `test_shape_context.m` script to verify your implementation. Turn in your code for `compute_shape_context.m`.

 (ii) Run the `compare_digits.m` script and turn in the error for each of the 3 matches. The error for each match is the sum of the squared distances. We calculate the error for you and print it out to a figure after running the `compare_digits.m` script.

[2]`http://www.cs.berkeley.edu/~malik/papers/BMP-shape.pdf`

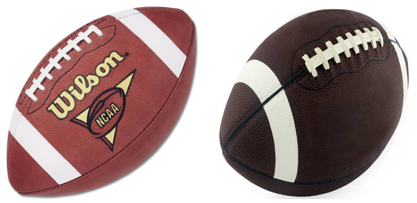


(a) (b)

Figure 5: Example data from handwritten digit dataset

(b) Now we consider different images and matching scenarios other than handwriting to test how shape context performs.

   (i) We will use preprocessed versions of the images in Fig 6 to test your shape context descriptor. Run the `extend_shape_context.m` script. Turn in the resulting error values and warped figure (figure 3 generated from the code) and indicate what properties of shape context yield this performance.



(a) (b)

Figure 6: Other test images for shape context

   (ii) Considering the invariances of Shape Context when would you expect Shape Context descriptor to have poor performance? Give specific circumstances citing the invariances of Shape Context, and use the previous football result as evidence. Given that Shape Context performs well for the digit data, when else would we expect shape context to perform well?

3 Voxel Coloring (35 points)

One technique for 3D reconstruction from photos is voxel coloring[3]. Here, we implement a lightweight version of voxel coloring. Starter code is available in the PS3Prob3 subfolder of starter code distribution.

A dev dataset is provided with ground truth included. In addition, a test dataset of a different object is provided without ground truth.

While the voxel coloring problem is often posed in full 3D, here we will only use a slice of a 3D object so that the problem is more tractable to iterate on. All voxels will have x=0, so images are now 1-d slices (eg. 1x400 pixels) and the object lies fully in the YZ plane. World points are still described in 3D, so the solution for the slice case can easily be adapted to full 3D.

[3] `http://www.cs.cmu.edu/\~seitz/papers/ijcv99.pdf`

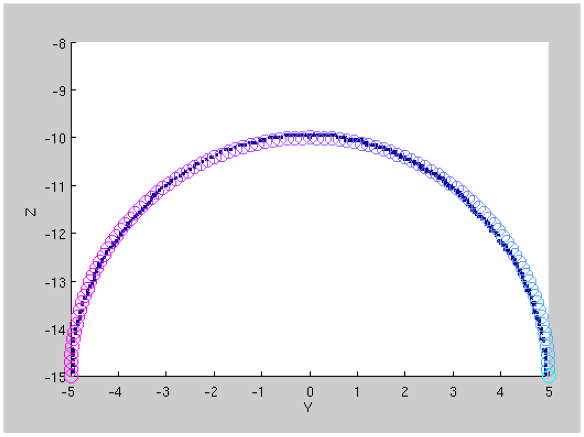


Figure 7: Voxel coloring of dev dataset with grid spacing 0.05. Circles are ground truth, selected voxels are blue squares.

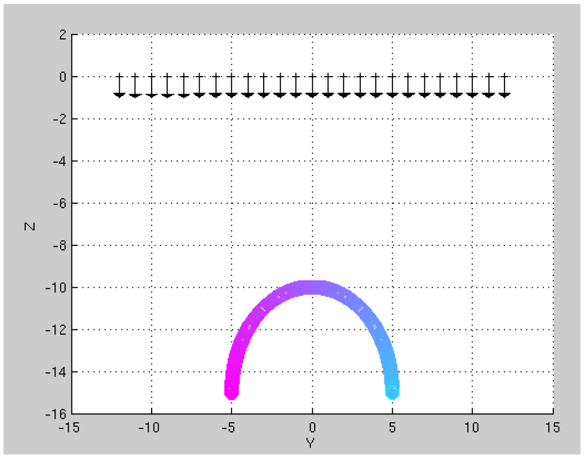


Figure 8: Camera setup. Camera normal vectors are in black, with the target object pictured below. Camera and object have x=0 for all points

(a) Implement the remaining portions of the voxel coloring algorithm including voxel projection, photoconsistency and dealing with occlusion masks in `voxel_projections.m`, `photoconsistency.m`, and `voxel_coloring.m` respectively.

Remember to ignore points that are occluded or outside the image boundary for a given camera. A point must be visible in at least two cameras for it to be triagulated.

Grid traversal and plotting functions have already been provided for you. You may find it helpful to use the 'dev' param settings to iterate quickly on your code, however, please report results using 'dev-highres' and 'test' param settings, respectively. In total, you should submit 2 plots (1 for the dev dataset and 1 for the test dataset).

(b) In addition, please experiment with grid spacing of [0.05, 0.1, 0.25] and describe the effect on the performance of voxel coloring. Why do you think this is? Must other parameters be changed when changing voxel sizing to maintain good performance? In your answer, please consider aspects of the input images which may or may not make certain voxel sizes better than others.

(c) Experiment with the 'dev-two-color' dataset, in which the object is colored with only two different colors. Provide the generated plot of your results. Please explain any differences between these results and the results on the dev dataset in part (a).

(d) Voxel coloring can be contrasted with space carving, which does not make use of color information. Describe the additional ambiguities that space carving introduces and draw the contours of the expected space carving result on the dev dataset using the same camera setup. As a reminder, the camera setup for our images is pictured in Figure 8. Describe changes to our setup (parameters, camera locations, etc.) that might mitigate some of these ambiguities.