CS 378 Computer Vision
Problem set 4
Out: Thursday, Nov 5
Due: Tuesday, Nov 24, 11:59 PM

See the end of this document for submission instructions.

I. Short answer problems [30 points]

1. The SIFT descriptor is formed using a 4 x 4 grid of histograms, each of which counts the underlying pixels' gradient directions for each of 8 possible orientations, yielding a 4*4*8 = 128-dimensional descriptor for an image patch. If we were to instead use an 8 x 8 grid of histograms for the same image patch, we would have an 8*8*8 = 512-dimensional descriptor. What impact would this adjustment have on the feature descriptor's invariance properties?

2. RANSAC is useful for robustly estimating the homography matrix (e.g., for the mosaic stitching application) and for robustly estimating the fundamental matrix (for stereo with weakly calibrated cameras). Explain what defines an "inlier" correspondence in either scenario.

3. Given a world point measured in the *world* coordinate system, what parameters are needed to compute the point's associated coordinates in the *camera's* coordinate system?

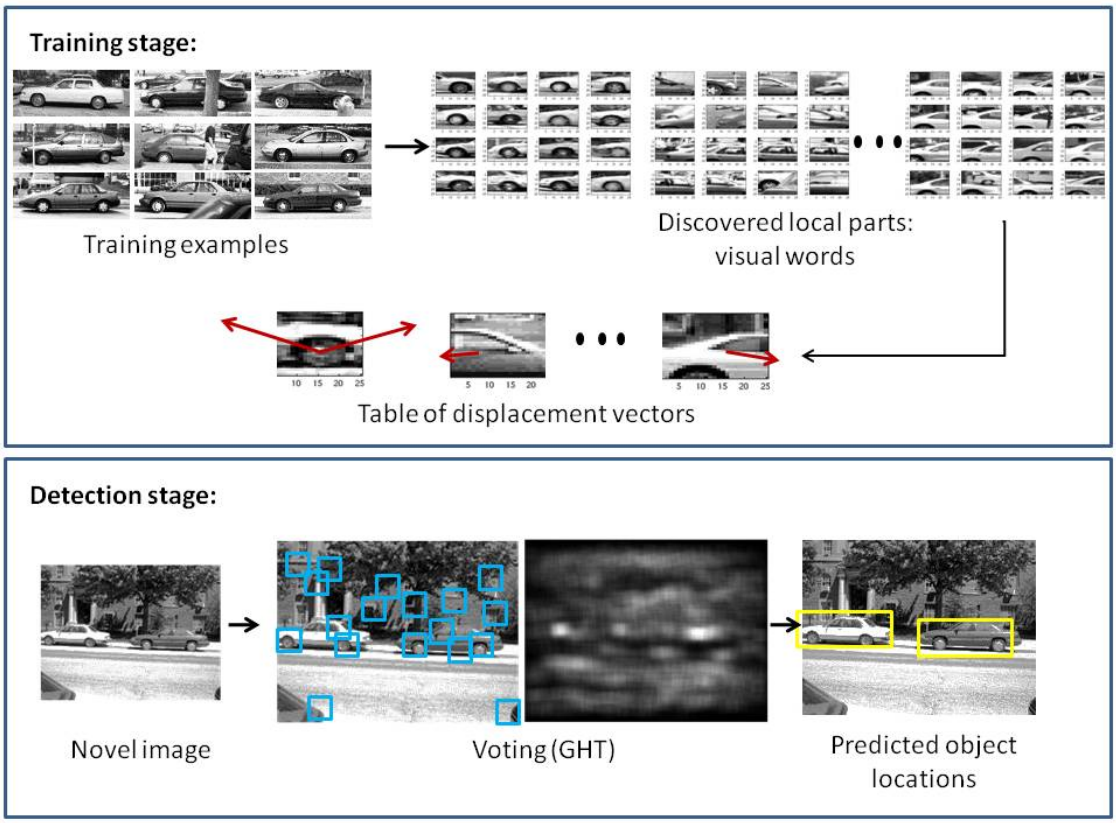
4. Briefly describe two tradeoffs (i.e., pros/cons) between performing dense stereo matching (where we attempt to find correspondences for every pixel) vs. sparse stereo matching (where we attempt to find correspondences only at interest points).

5. To perform automatic scale selection with the Laplacian of Gaussian filter, we first filter the image with a Laplacian with increasing values of sigma. Then we look at the magnitude of the response at each pixel, and take as an interest point any point at which the response is a *local maximum* in both position and scale. Why take the local maxima, rather than simply take any responses that exceed a threshold?

II. Programming problem: object category detection [70 points]

The goal of this assignment is to perform object detection using a voting-based Generalized Hough Transform approach. Given a set of cropped training images showing the class of interest, the method first generates a set of prototypical object parts. We'll use *visual words* formed from local image patches extracted at Harris corner interest points to represent the parts. That is, object part = visual word. Having identified the object parts, the method stores the possible displacement vectors (translations) between each part and the object's center, as observed in all the training examples. Given a novel test image, the local patches are first mapped to object parts, by assigning the patch around each interest point to the nearest prototype (visual word). Then each part uses the stored displacement vectors to cast votes for the position of the object. By looking for peaks in that vote space, one or more detections are made.

Note that the object of interest may appear multiple times in a test image, at any location. However, we will assume that the scale and orientation of the object is fixed; that is, the scale and orientation at which it appears in the training images are both constant, and will also be similar in the test images. Thus the vote space is only 2-dimensional.



To summarize the main steps required:

- Training stage - Prepare the visual vocabulary and GHT displacement vectors:
    - Run the Harris corner detector on each training example to collect its interest points.
    - At each interest point, extract a fixed size image patch (e.g., 25 x 25 pixels), and use the vector of raw pixel intensities as the descriptor.
    - Cluster the patches from the training images with k-means to form a visual vocabulary, where each mean represents an object part discovered in the training data.
    - Having found the vocabulary, go back over the training examples and assign their local patches to visual words (i.e., the prototype object parts). An image patch is assigned to the visual word to which it is closest using Euclidean distance.
    - For each visual word occurrence in the training examples, record the possible displacement vectors between it and the object center. Let the object center be the center of the cropped training image. Note that a given visual word may have multiple displacement vectors, depending on its appearance. (For example, see the "wheel-like" word in the figure above.)

- Testing stage - Given a novel test image, detect instances of the object:
    - Run the Harris corner detector on the test image to collect its interest points.
    - At each interest point, extract a fixed size image patch, using the same scale selected above. Use the vector of raw pixel intensities as the descriptor.

- Assign each patch to a visual word.
- Let each visual word occurrence vote for the position of the object using the stored displacement vectors.
- After all votes are cast, analyze the votes in the accumulator array, threshold, and predict where the object occurs. (To predict the fixed-size bounding box placement, assume object center = bounding box center.) Note that the object of interest may occur multiple times in the test image.
- Compute the accuracy of the detector's predictions, based on the overlap between the predicted and true bounding boxes. Specifically, a predicted detection is counted as correct if the area of the intersection of the boxes, normalized by the area of their union, exceeds 0.5.

The following data and code are provided on the course website:

- `CarTrainImages/`: directory containing 550 cropped training images of cars, each 40 x 100 pixels. (The images are small, but if you encounter memory issues, it's fine to reduce the training set size as needed.)

- `CarTestImages/`: directory containing 100 test images. Each has one or more cars amidst cluttered backgrounds.

- `Code/harrisDetector.m`: function to compute *cornerness* response and return Harris interest points. See documentation for usage.

- `GroundTruth/CarGroundTruthBoundingBoxes.mat`: data file containing ground truth bounding boxes for the test images. Load with "`load`". The variable '`groundtruth'` is a struct, and `groundtruth(i)` contains the box for the i-th test image. The box is specified in terms of the top left corner position and the width and height of the box. Each box gets a row in the list of locations: `groundtruth(i).topLeftLocs = [x1, y1; x2, y2,…, xn yn]` if there are n boxes in image i. For example,

```
>> load CarsGroundTruthBoundingBoxes
>> groundtruth(5)

ans =

    topLeftLocs: [2x2 double]
           boxW: 100
           boxH: 40

>> groundtruth(5).topLeftLocs

ans =

     5    43
   110    45
```

  means that for test image #5, there are two cars present, and their boxes have the top left corners x=5, y=43 and x = 110, y=45, respectively. Both have size 40 x 100.

- `Code/exampleLoadingGroundTruth.m`: function to illustrate the format explained above.

- Other useful Matlab functions: `subplot, rectangle, kmeans, im2double, dist2`

Note that you should be able to re-use a portion of your code from Pset2 (since clustering filter bank responses with k-means to form textons ↔ clustering local image patches to form visual words).

After implementing your detector and adjusting the parameters to values that produce reasonable results, answer each of the following:

1. **Visualizing the discovered prototypical object parts [15 points]**: Display example image patches associated with each of three visual words within the vocabulary. Choose three words that are distinct to illustrate what the different words are capturing, and display enough sample patches so that the word content is evident (e.g., say 10-20 patches per word displayed). Explain what you see.

2. **Overall accuracy [15 points]**: Report the average accuracy over all the test images. Measure both the percentage of correct detections per image, as well as the average number of false positives per image. A false positive occurs when the detector fires, but the object is not there. Use the same set of parameters (vote space bin size, vocabulary size, patch size, etc.) for all test images.

3. **Show example detections [15 points]**: Pick five test image examples, and display the images with the bounding box detections made by your system. Include some good detection results, as well as at least one interesting error case. Briefly explain.

4. **Impact of patch size [15 points]**: Run your full system twice – once with a smaller patch size, once with a larger patch size. Analyze and describe the impact on the visual vocabulary and the detector. Use images to illustrate your point as needed.

5. **Multi-object detection [10 points]**: Explain how you could extend your implementation to detect multiple types of objects in a single image (e.g., cars and people). You do not need to implement this.

III. [OPTIONAL] Extra credit [up to 10 points each, max 30 points]

1. Train and run your detector on the Face data provided (see provided `FaceTrainImages`, `FaceTestImages`, and `GroundTruth/FacesGroundTruthBoundingBoxes.mat`.) Show the results and briefly explain, noting any contrasts with the Car data experiment.

2. Add a *non-maximum suppression* step to the detector that removes highly overlapping detection windows. Describe and illustrate the results.

3. Design and implement a *verification* stage to be used after voting for detections. The verification should look for support beyond the sparse points that voted for that translation of the model.

4. Extend your code from Pset3 to do automatic interest point detection and feature matching. That is, remove the part where you provided manually clicked correspondences, and insert feature detection, local descriptors, and matching with RANSAC. You can use the Harris corner detector again, and/or check out software linked from the class webpage.

Acknowledgements:

The car images we are using for this assignment are from the UIUC Image Database for Car Detection, courtesy of Shivani Agarwal and Dan Roth. http://l2r.cs.uiuc.edu/~cogcomp/Data/Car/

The face images provided are from the Caltech-101 Database, courtesy of Fei-Fei Li et al. http://www.vision.caltech.edu/Image_Datasets/Caltech101/

Submission instructions: what to hand in

Electronically:

- Your well-documented Matlab code .m files.
- A pdf file containing:
  - Your name and CS login ID at the top.
  - Your answers to Section I, numbered.
  - Your image results and accompanying explanations for Section II, numbered.
  - (optional): any results and descriptions for extra credit portions in Section III.

Submit all the above with one call to turnin:

```
>> turnin --submit jaechul pset4 pset4.pdf codeFileXYZ.m codeFileABC.m
etc.
```

Hardcopy: Print out the pdf files, and either bring it to class or drop off at Jaechul's office in CSA on Thurs 11/24. Do not print out code. As usual, the hardcopy must be the same as what is submitted electronically.