CS 376 Computer Vision
Problem Set 4
Out: Wednesday, April 6
Due: Wednesday, April 20, 11:59 PM

Please see the end of this document for submission instructions. Visit us during office hours to discuss any questions on the assignment. Or, if sending a question via email, please submit to cv-spring2011@cs.utexas.edu with CS376 in the subject line, and please be specific.

I. Short answer questions [25 points]

1. When performing interest point detection with the Laplacian of Gaussian, how would results differ if we were to (a) take any positions that are local maxima in scale-space, or (b) take any positions whose filter response exceeds a threshold? Specifically, what is the impact on *repeatability* or *distinctiveness* of the resulting interest points?

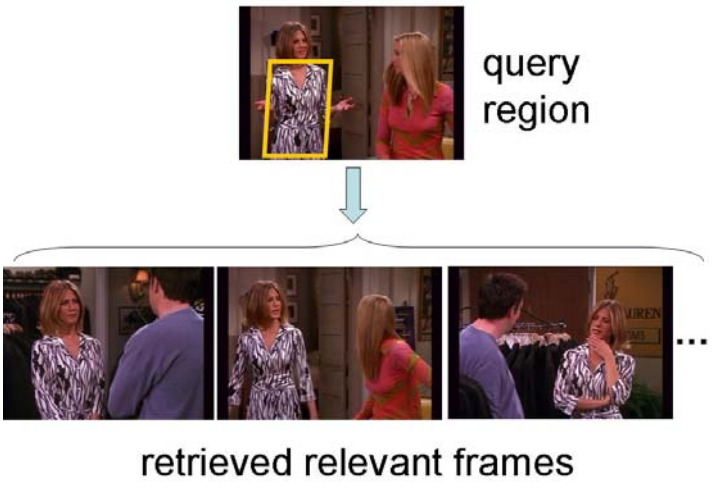
2. What is an "inlier" when using RANSAC to solve for the epipolar lines for stereo with uncalibrated views, and how do we compute those inliers?

3. Name and briefly explain two possible failure modes for dense stereo matching, where points are matched using local appearance and correlation search within a window.

4. What exactly does the value recorded in a single dimension of a SIFT keypoint descriptor signify?

5. If using SIFT with the Generalized Hough Transform to perform recognition of an object instance, what is the dimensionality of the Hough parameter space? Explain your answer.

II. Programming problem: Video search with bags of visual words [75 points]



For this problem, you will implement a video search method to retrieve relevant frames from a video based on the features in a query region selected from some frame.

Data

We are providing image data and some starter code for this assignment. You can access pre-computed SIFT features here: /v/filer4b/v26q010/pset4data/sift/. The associated images are stored here: /v/filer4b/v26q010/pset4data/frames/. The data takes about 6 G, so you should *not* try to copy it to your home directory. Just point to the data files directly in your code.

Each `.mat` file in /v/filer4b/v26q010/pset4data/sift/ corresponds to a single image, and contains the following variables, where *n* is the number of detected SIFT features in that image:

descriptors	*n*x128	double	// the SIFT vectors as rows
imname	1x57	char	// name of the image file that goes with this data
numfeats	1x1	double	// number of detected features
orients	*n*x1	double	// the orientations of the patches
positions	*n*x2	double	// the positions of the patch centers
scales	*n*x1	double	// the scales of the patches

Provided code

These are the provided code files, available on the class website (`provided_code.tar.gz`):

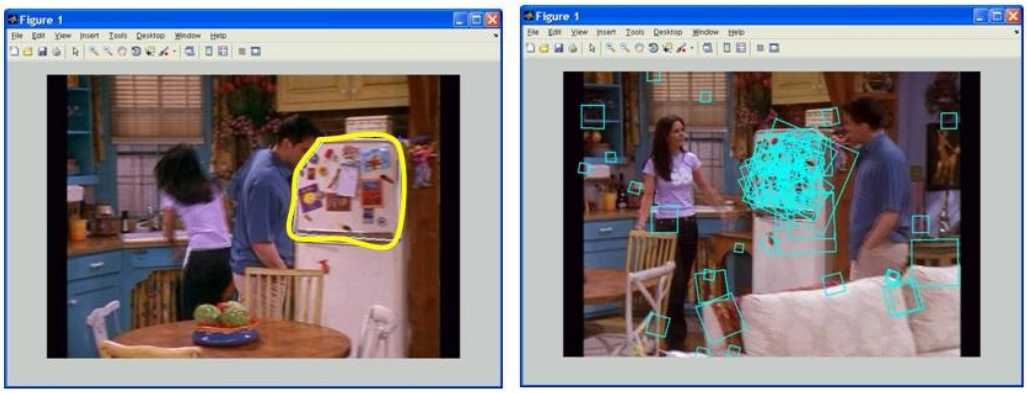
- `loadDataExample.m`: Run this first and make sure you understand the data format. It is a script that shows a loop of data files, and how to access each descriptor. It also shows how to use some of the other functions below.
- `displaySIFTPatches.m`: given SIFT descriptor info, it draws the patches on top of an image
- `getPatchFromSIFTParameters.m`: given SIFT descriptor info, it extracts the image patch itself and returns as a single image
- `selectRegion.m`: given an image and list of feature positions, it allows a user to draw a polygon showing a region of interest, and then returns the indices within the list of positions that fell within the polygon.
- `dist2.m`: a fast implementation of computing pairwise distances between two matrices for which each row is a data point
- `kmeansML.m`: a faster k-means implementation that takes the data points as columns

You are not required to use any of these functions, but you will probably find them helpful.

What to implement and discuss in the writeup

Write one script for each of the following (along with any helper functions you find useful), and in your pdf writeup report on the results, explain, and show images where appropriate.

1. **Raw descriptor matching [15 pts]:** Allow a user to select a region of interest (see provided `selectRegion`) in one frame, and then match descriptors in that region to descriptors in the second image based on Euclidean distance in SIFT space. Display the selected region of interest in the first image (a polygon), and the matched features in the second image, something like the below example. Use the two images and associated features in the provided file `twoFrameData.mat` (in the gzip file) to demonstrate. Note, no visual vocabulary should be used for this one. Name your script `rawDescriptorMatches.m`



2. **Visualizing the vocabulary [20 pts]:** Build a visual vocabulary. Display example image patches associated with two of the visual words. Choose two words that are distinct to illustrate what the different words are capturing, and display enough patch examples so the word content is evident (e.g., say 25 patches per word displayed). See provided helper function `getPatchFromSIFTParameters`. Explain what you see. Name your script `visualizeVocabulary.m`

3. **Full frame queries [20 pts]:** After testing your code for bag-of-words visual search, choose 3 different frames from the entire video dataset to serve as queries. Display the *M*=5 most similar frames to each of these queries (in rank order) based on the normalized scalar product between their bag of words histograms. Explain the results. Name your script `fullFrameQueries.m`

4. **Region queries [20 pts]**: Select your favorite query regions from within 4 frames (which may be different than those used above) to demonstrate the retrieved frames when only a portion of the SIFT descriptors are used to form a bag of words. *Try to include example(s) where the same object is found in the most similar M frames but amidst different objects or backgrounds, and also include a failure case.* Explain the results, including possible reasons for the failure cases. Name your script `regionQueries.m`

Tips: overview of framework requirements

The basic framework will require these components:

- **Form a visual vocabulary.** Cluster a large, representative random sample of SIFT descriptors from some portion of the frames using k-means. Let the k centers be the visual words. The value of k is a free parameter; for this data something like k=1500 should work, but feel free to play with this parameter [see Matlab's `kmeans` function, or provided `kmeansML.m` code]

- **Map a raw SIFT descriptor to its visual word.** The raw descriptor is assigned to the nearest visual word. [see provided `dist2.m` code for fast distance computations]

- **Map an image's features into its bag-of-words histogram.** The histogram for image I_j is a *k*-dimensional vector: $F(I_j) = \lfloor freq_{1,j}, freq_{2,j}, \ldots, freq_{k,j} \rfloor$, where each entry $freq_{i,j}$ counts the number of occurrences of the *i*-th visual word in that image, and *k* is the number of total words in the vocabulary. In other words, a single image's list of *n* SIFT descriptors yields a *k*-dimensional bag of words histogram. [Matlab's `histc` is a useful function]

- **Compute similarity scores.** Compare two bag-of-words histograms using the normalized scalar product.

- **Sort the similarity scores** between a query histogram and the histograms associated with the rest of the images in the video. Pull up the images associated with the *M* most similar examples. [see Matlab's `sort` function]

- **Form a query from a region within a frame.** Select a polygonal region interactively with the mouse, and compute a bag of words histogram from only the SIFT descriptors that fall within that region. Weight it with tf-idf. [see provided `selectRegion.m` code]

- **Compute nearest raw SIFT descriptors.** Use the Euclidean distance between SIFT descriptors to determine which are nearest among two images' descriptors. That is, "match" features from one image to the other, without quantizing to visual words.

III. OPTIONAL : Extra credit (up to 10 points each, max 20 points total)

- **Stop list and tf-idf**. Implement a stop list to ignore very common words, and apply tf-idf weighting to the bags of words. Discuss and create an experiment to illustrate the impact on your results.

- **Spatial verification**. Implement a spatial consistency check to post-process and re-rank the shortlist produced based on the normalized scalar product scores. Demonstrate a query example where this improves the results.

- **Feature matching and RANSAC for mosaic stitching**. If you didn't already implement this for Pset 3 extra credit, develop matching code together with RANSAC to robustly and automatically identify correspondences to use with your Pset 3 image warping program.

Want to generate DoG+SIFT descriptors for your own images? Check out implementations available online:

- http://www.vlfeat.org/
- http://www.robots.ox.ac.uk/~vgg/research/affine/

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 for Section I
  - A brief explanation of your implementation strategy for Section II: paragraph or two describing in English how you implemented the programs.
  - Your image results and accompanying explanations for Section II.
  - (optional): Any results and descriptions for extra credit portions. For max credit explain what you implemented and also interpret the results.

Submit all the above with one call to turnin:

1. `>> turnin --submit shalini pset4 file.pdf visualizeVocabulary.m rawDescriptorMatches.m fullFrameQueries.m regionQueries.m codeFileXYZ.m codeFileABC.m etc.`

Hardcopy: Also drop a hardcopy in the CS homework dropbox in PAI 5.38. Attach a cover page with the course number CS376 and your name, to make it easy to find in the dropbox. Please save paper by concatenating shorter functions into a single page before printing.