# CS 4476: Introduction to Computer Vision, Fall 2018
# PS4

Instructor: Devi Parikh

Due **before**: Wednesday, November 7th, 11:58:59 pm

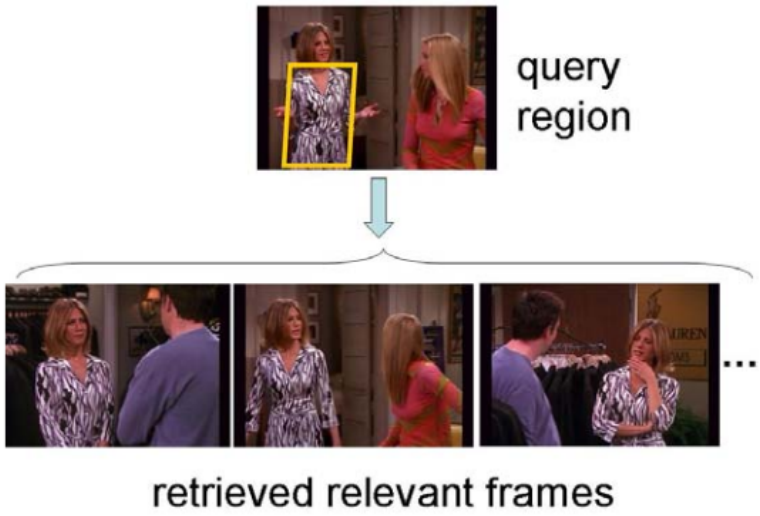
**Instructions**

1. Answer sheets, code and input/output images must be submitted on Canvas. Hard copies will not be accepted.
2. Please provide a pdf version of your answer sheet named: LastName_FirstName_PS4.pdf.
3. If your scripts are in Python, please use '.py' as file extension. If your scripts are in Matlab, please use '.mat' as file extension.
4. Please put all your code, input/output images and answer sheets in a folder (no subdirectories). Make sure your code is bug-free and works out of the box. Please be sure to submit all main and helper functions. Be sure to not include absolute paths. Points will be deducted if your code does not run out of the box.
5. If plots are required, you must include them in your answer sheet (pdf) and your code must display them when run. Points will be deducted for not following this protocol.
6. Your code and plots should use the same filenames mentioned in the question (if present). Variables in your code should use the same names that are mentioned in the question (if present).
7. Please make sure that the folder is named LastName_FirstName_PS4_mat if using Matlab and LastName_FirstName_PS4_py if using Python.
8. Zip the above folder and name the zipped file LastName_FirstName_PS4_mat.zip if using Matlab and LastName_FirstName_PS4_py.zip if using Python. Submit *only* the zip file.

## 1 Short answer questions [25 points]

1. When performing interest point detection with the Laplacian of Gaussian, how would the 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.

## 2 Programming problem [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. We are providing image data and some starter code for this assignment.

**Provided data**

You can access pre–computed SIFT features here:
https://filebox.ece.vt.edu/~F13ECE5554/resources/PS4_material/PS4SIFT.zip.
The associated images are stored here:
https://filebox.ece.vt.edu/~F13ECE5554/resources/PS4_material/PS4Frames.zip.
Please note that the data takes about **6GB**. Each `.mat` file in the provided SIFT data corresponds to a single image, and contains the following variables, where n is the number of detected SIFT features in that image:

| | | |
|---|---|---|
| descriptors | nx128 | 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 | nx1 | double // the orientations of the patches |
| positions | nx2 | double // the positions of the patch centers |
| scales | nx1 | double // the scales of the patches |

**Provided code**

The following are the provided code files. You are not required to use any of these functions, but you will probably find them helpful. You can access the code from here:

**Matlab:** https://filebox.ece.vt.edu/~F15ECE5554ECE4984/resources/PS4_material/PS4CodeMatlab.zip

- `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

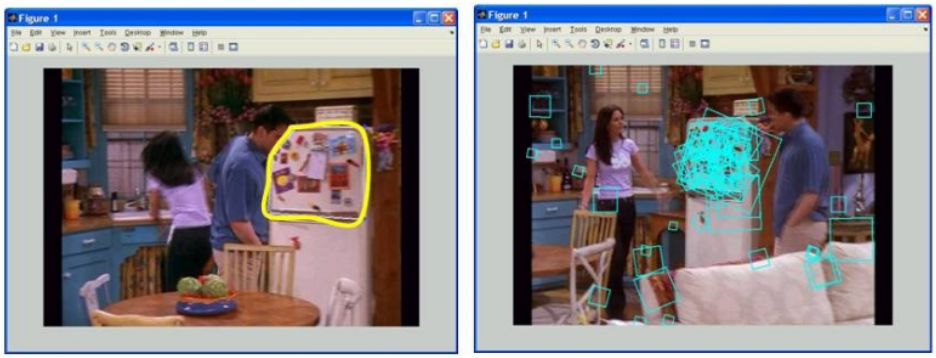
- `getPatchFromSIFTParameters.m`: given SIFT descriptor info, it extracts the image patch itself and returns as a single image
- `selectRegion.m`: given an image and list of feature positions, it allows a user to draw a polygon showing a region of interest, and then returns the indices within the list 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.

**Python:** https://filebox.ece.vt.edu/~F15ECE5554ECE4984/resources/PS4_material/PS4CodePython.zip

- `loadDataExample.py (ipynb)`: 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. You can also run the ipyhon notebook version code.
- `displaySIFTPatches.py`: given SIFT descriptor info, it draws the patches on top of an image
- `getPatchFromSIFTParameters.py`: given SIFT descriptor info, it extracts the image patch itself and returns as a single image
- `selectRegion.py`: 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.py`: a fast implementation of computing pairwise distances between two matrices for which each row is a data point

**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.m(py)`) 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(py)`
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.m(py)`. Explain what you see. Name your script `visualizeVocabulary.m(py)`
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(py)`



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(py)`

**Tips: overview of framework requirements**
The basic framework will require these components:

- 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.
- 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 [For Matlab, see Matlab's kmeans function, or provided `kmeansML.m` code. For Python, see kmeans function in `sklearn, scipy, opencv etc.`]. *Note:* You may run out of memory if you use all the provided SIFT descriptors to build the vocabulary.
- Map a raw SIFT descriptor to its visual word. The raw descriptor is assigned to the nearest visual word. [see provided `dist2.m(py)` 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:
$$F(I_j) = [freq_{1,j}, freq_{2,j}, ..., freq_{k,j}]$$
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.
- Compute similarity scores. Compare two bag-of-words histograms using the normalized scalar product:
$$S(I_i, I_j) = \frac{F(I_i) \cdot F(I_j)}{\|F(I_i)\|\|F(I_j)\|} = \frac{1}{\|F(I_i)\|\|F(I_j)\|} \sum_{m=1}^{k} freq_{m,i} freq_{m,j}$$
where $S()$ is the similarity score. $\|F(I_i)\|$ is the L2 norm of $F(I_i)$.
- 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.

- 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. You may weight it with tf-idf. [see provided `selectRegion.m(py)` code]

# 3 OPTIONAL: Extra credit (up to 10 points each, max 20 points total)

1. **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.
2. **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.

This assignment is adapted from PS4 of Kristen Grauman's CS 376: Computer Vision at UT Austin.