Kim Dang

June 9, 2017

Professor Yong Jae Lee

ECS 174

Problem Set 3

I. Short Answer Problems

1. In the case of (a) where we take positions that are local maxima in a scale-space, we can detect blobs or interest points at virtually any scale, whereas in the case of (b) where we take any position whose filter responses exceeded a threshold, we do not look at potential interest points at any scale. Both methods described in (a) and (b) will be robust to rotation and illumination differences. Both are also capable of detecting distinct features, granted that the interest points we're looking for are distinct themselves, like looking for corners. However, method (b) is not as scale invariant as method (a), so method (b) will not have strongly repeatable results compared to method (a).

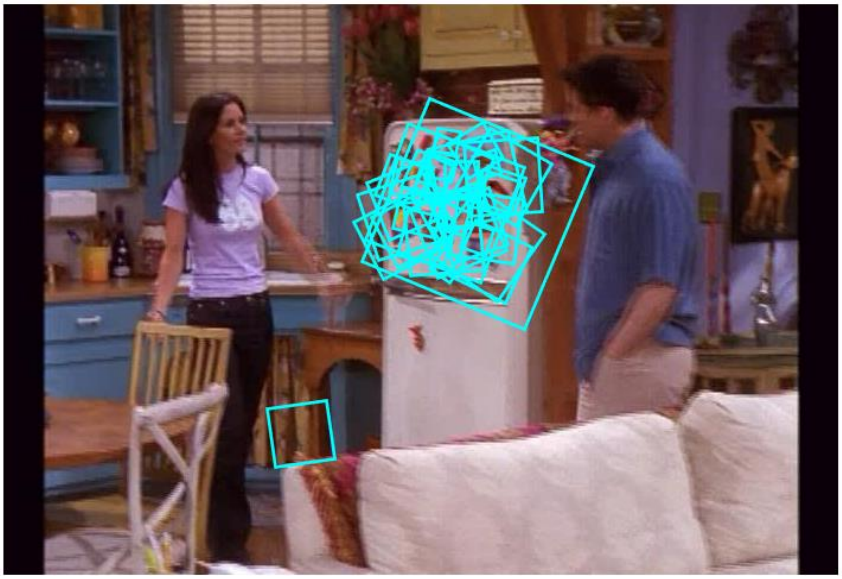
2. In a single dimension of a SIFT keypoint descriptor, each value is a magnitude of a bin of gradient orientations, of sampled image gradients in a sub-patch of a descriptor. These bins together are part of a histogram that describe each sub-patch in the keypoint descriptor.

3. For a Generalized Hough Transform for object instance recognition using SIFT, Hough parameter space has four dimensions: scale, orientation, and x and y translation. SIFT is scale, rotation, and translation invariant, so it's given that the Generalized Hough Transform must have these properties as well and take them into account.

II. Programming Problems

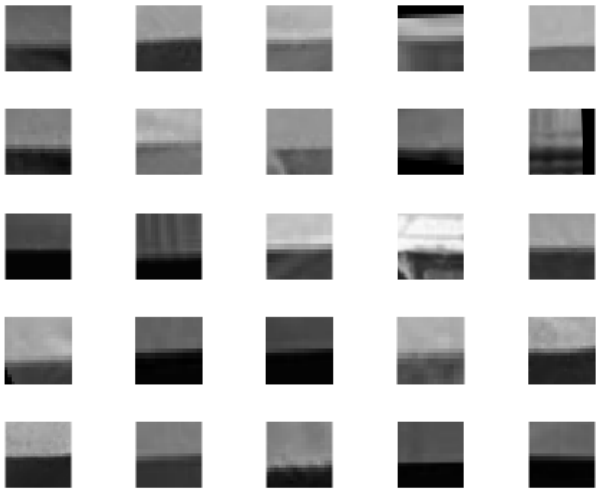
1. The script `rawDescriptorMatches.m` takes user input to select a region of one image to extract SIFT descriptors and find close matches in another image provided in `twoFrameData.mat`.

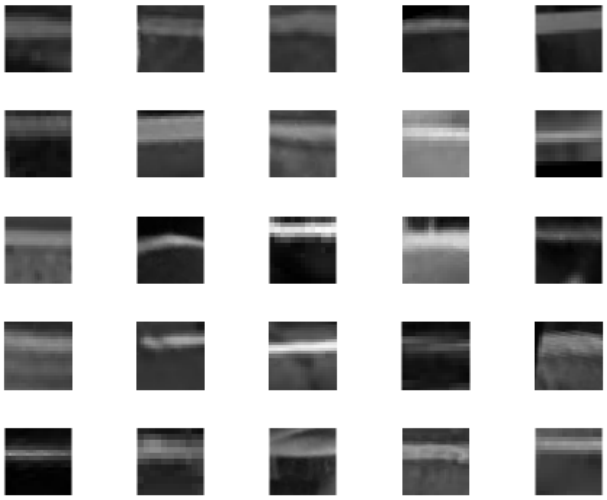
   I modified `selectRegion.m` from provided code so that I can get the points for the boundary plot to display, named `selectRegionMod.m`. I then made a helper function called `getNearestRawDescriptor.m` which implements `dist2.m` which gives the index for the nearest descriptor and the ratio of first best match to second best match to check for irregularities. If the ratio is low, I display that SIFT region in the second image using `displaySiftPatches.m`, else I discard that descriptor and check the next one. I set the threshold to 0.5 because that was a decent number to set and very few of the descriptors end up being wrong. The first image below is a sample input region on the first image, and the second is the second image with matched SIFT descriptors.



2. The script `visualizeVocabulary.m` implements the helper function `buildVocab.m` and the provided code, `getPatchFromSIFTParameters.m` and `dist2.m`. We first build the vocabulary beforehand by sampling 50 random SIFT descriptors from the provided frames in intervals of 20. We then quantize these descriptors into visual words using `kMeansML.m` from the provided code. Then, we choose the two visual words that have the most member descriptors, and display the top 25 patches that resemble the visual word the most (therefore they have the shortest Euclidean distance). The visual words are stored as `kMeans.mat` in the files. Please note that `visualizeVocabulary.m` will take around 2 minutes to execute, so it will print messages for all the images so the script won't look like it's freezing.

   The two figures below show 25 patches for each of the two visual words chosen. The first one is the most common visual word, which is a edge from a light color to a dark one below. The second most common visual word is for patches with a dark background with a single thin lighter line passing through the upper region of the patch. These patches aren't detailed like faces or furniture, but they are very distinct and easy to find in the sampled frames.

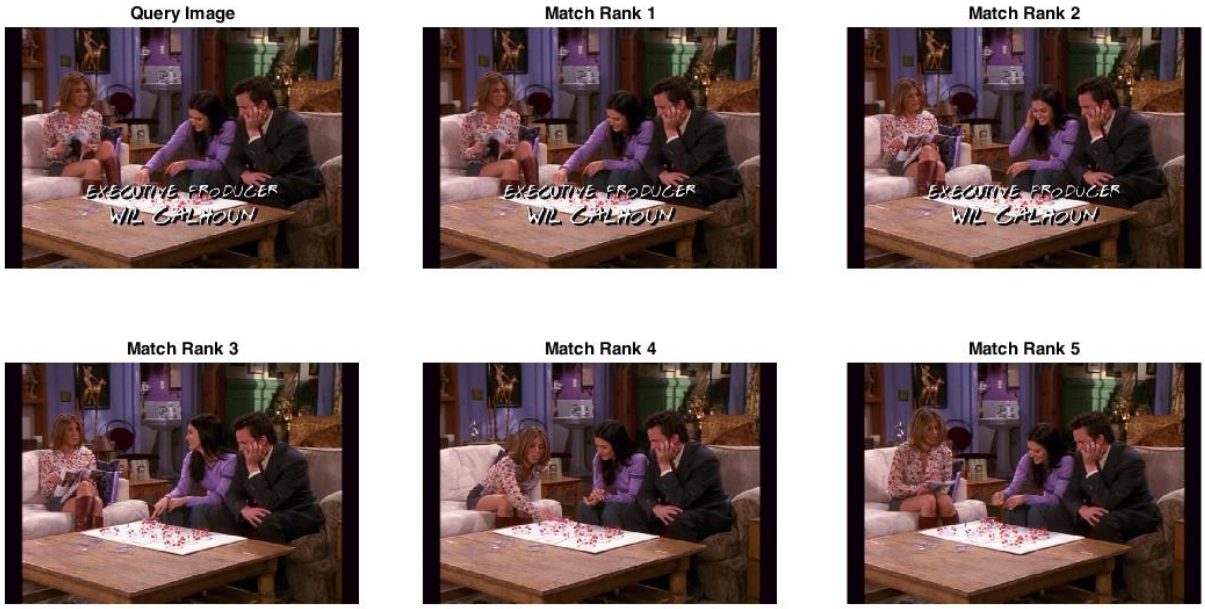




3. The script `fullFrameQueries.m` takes 3 certain frames from the video dataset and finds the five most similar frames. The helper functions `makeHistogram.m` and `generateHistogram.m` are used to convert all frames into histograms of bag of words compared to `kMeans1.mat` made from an attempt of part 2 by `dist2.m`. We use a different `kMeans.mat` from the one generated in part 2 to ensure consistency for part 3 and 4 because `buildVocab.m` randomly samples descriptors.

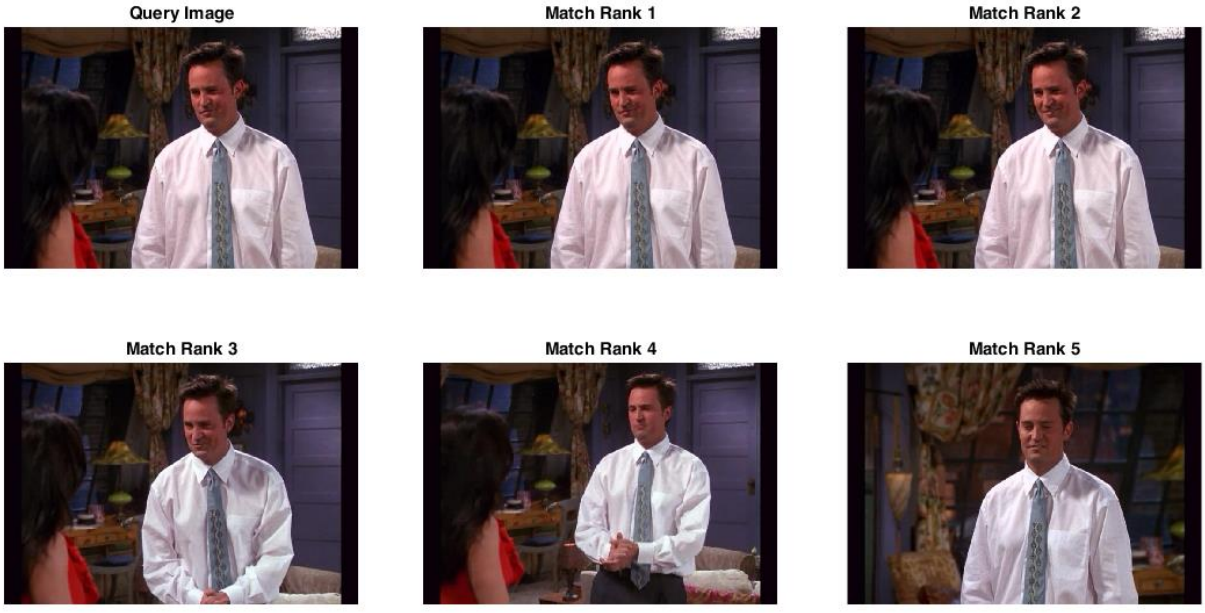
   The generated histograms are made beforehand and placed in a file called `histograms.mat`. The histograms are loaded onto `fullFrameQueries.m` and compared to the histograms of the chosen 3 frames. Similarity is calculated by a normalized scalar product between all the histograms and the histograms of the image in `scoreHistogram.m`, then ranked by highest score in `scoreTopImages.m`. Then, the results are displayed side by side with the original image for comparison. The figures below are the results of the three queries.

   This first query image has various subjects in the scene, making it likely ripe with many descriptors spread out around the image. The frames retrieved are nearly identical to each other and the query image. The script takes note of the credits on some of the frames as well, since the frames with credits are ranked the highest. Despite the pose of the characters, the script still manages to retrieve the images.



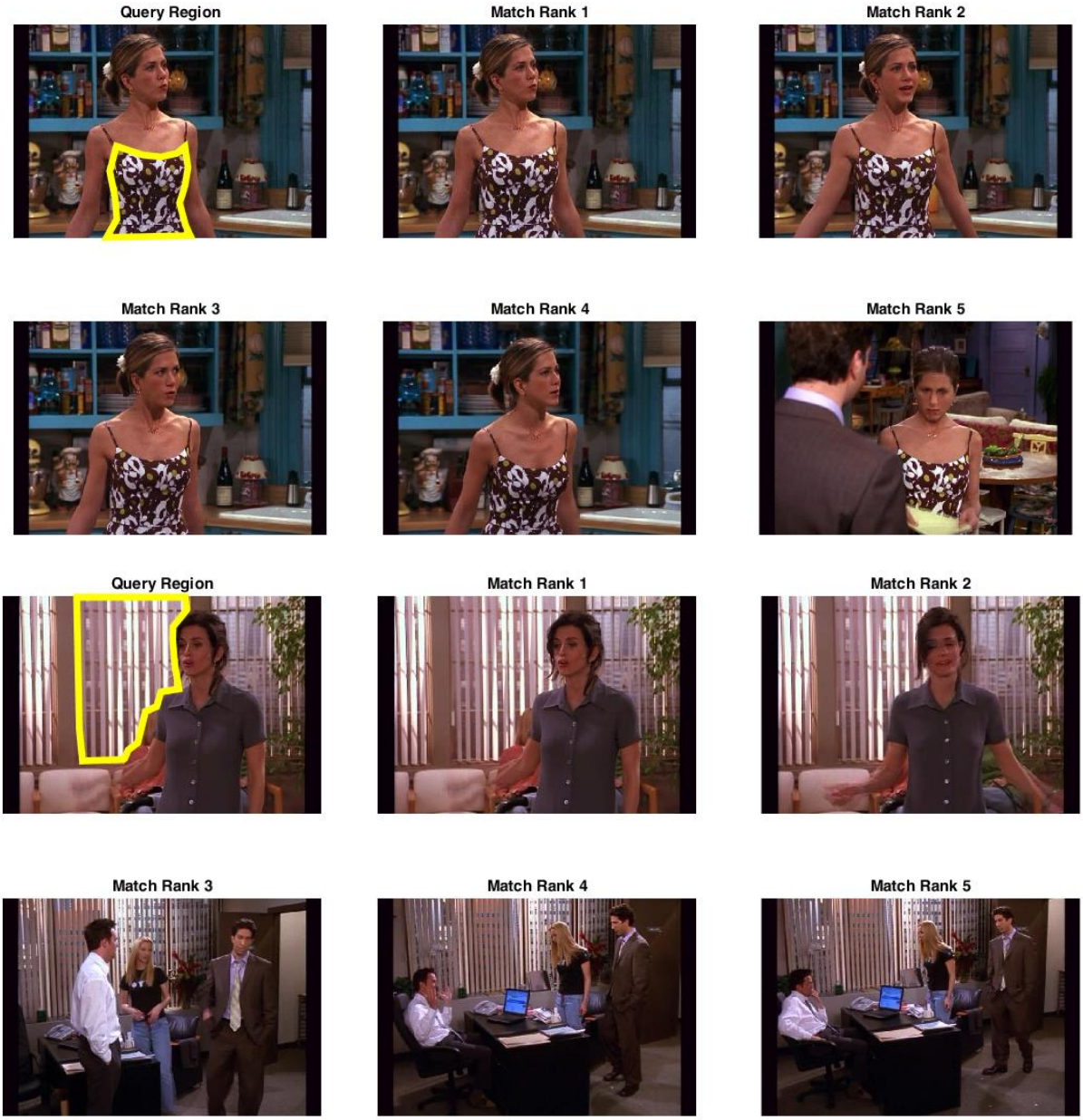
The second query is less varied, where the focus of the descriptors is likely on the woman in the figure and her clothes. Unfortunately, the script does not grab all the frames, nor does it rank the frames correctly. This can be attributed to perhaps a limited word vocabulary, or the limited number of descriptors provided by the image.

The third query image is, as expected, able to retrieve frames that show the character with a white shirt and tie, both of which would likely be the major focus of the descriptors for the image. The script retrieves frames that are nearly identical to the original query image, but it has also remarkably retrieved an image that is scaled back. This demonstrates the power of the SIFT descriptors to be scale invariant.

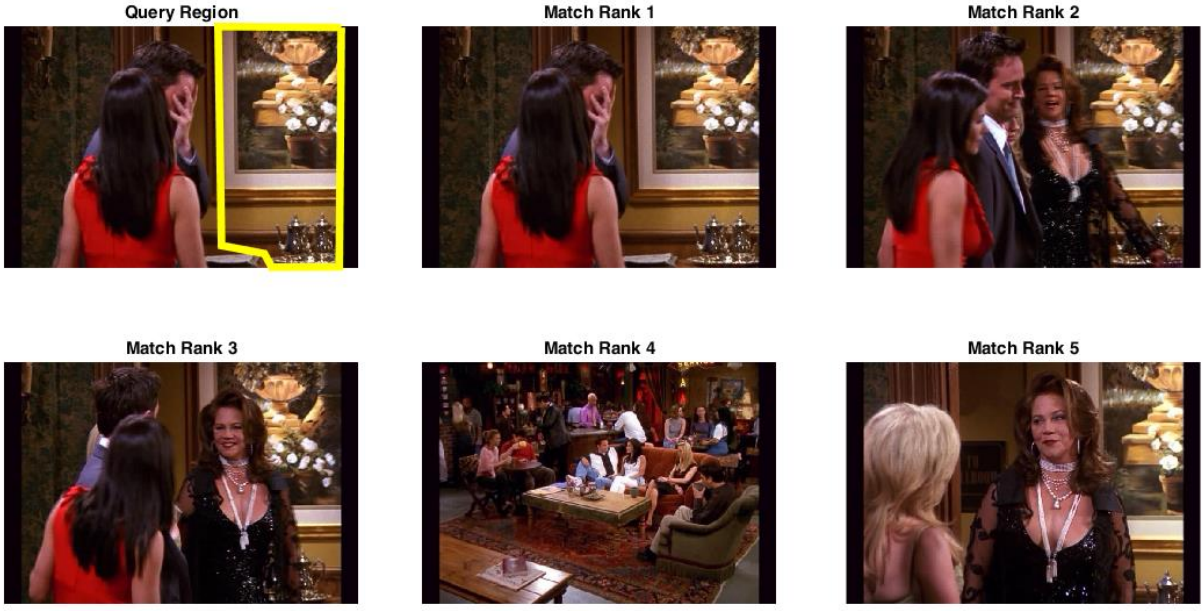


4. The script `regionQueries.m` retrieve frames from the entire video dataset given selected query regions from within 4 frames. We use `selectRegion.m` again to retrieve the descriptors contained inside a user selected region. We then create histograms of these descriptors with `makeHistograms.m`, then compare these histograms to our already generated histograms from part 3. Like in part 3, we score and rank the best frames by similarity in histograms and display the most similar 5 frames.

   The first two sets are easy success cases. By focusing on regions containing distinct features, we can retrieve frames that are from the same TV show set. For cases like patterns, large regions should be sampled to gather more descriptors to evaluate and compare to all the frames.



This third set is a bit broader. By focusing on a wider array of distinct features, not just patterns, we are still successful. We've even found a frame where the region of interest is not in the focus of the scene and scaled smaller, once again demonstrating that the SIFT descriptors are scale and orientation invariant.



This example is a failure case, where we attempt the first and second sets of highlighting patterns, but the retrieved frames are of the wrong scene and object. This is because the window blinds and the chair rungs in isolation are very similar and are likely to have patches of stripes, which can be mistaken for each other quite easily. This example failed because the query region was too generic, and that the two kinds of objects have similar features. This demonstrates why it is critical to look for distinct features in an image.