## Fall 2016 - CS 477/577 - Introduction to Computer Vision

# Assignment Six

### Due: 11:59pm (*) Tuesday, November 22.

(*) There is grace until 8am the next morning, as the instructor will not grade assignment before then. However, once the instructor starts grading assignments, no more assignments will be accepted.

I will **not** be able to respond to Piazza or email after about 11am on Tuesday, November 22.

### Weight: 9 points

---

**Code development for this assignment can be done in teams**, but each person must provide their own writeup. If you work in a team, please make it clear in your writeup who was in your team. Finally, each team member should submit a copy of the code.

---

Some parts of what you need to do for this assignment is quite close to code available on-line, **but you are responsible for your own implementation**.

---

## General instructions

Unless there is a good reason others, you should use Matlab for this assignment. **If you want to use any other language, you must discuss this with the instructor at lease one week before the due date**. If permission is granted, you will have to sort out any image handling and numerical library support on your own or as a group. (The IVILAB has support for C/C++ , with example files geared towards this course that can be made available on request).

You need to create a PDF document that tells the story of the assignment, copying into it output, code snippets, and images that are displayed when the program runs. Even if the question does not remind you to put the resulting image into the PDF, if it is flagged with ($), you should do so. I should not need to run the program to verify that you attempted the question. See

http://kobus.ca/teaching/assignment-instructions.pdf

for more details about doing a good write-up. While it takes work, it is well worth getting better and more efficient at this. A substantive part of each assignment grade is reserved for exposition.

## Assignment specification

This assignment has two parts, A and B that are required for both undergrads and grads, and one more part (C) which is required for grads. There are some optional parts for those that are interested. Please make it clear at the beginning of the your report if you did any optional questions.

To simplify things, you should hard code the file names in the version of your program that you hand in. You can assume that if the grader needs to run your code, they will do so in a directory that has the files linked from this page.

## Part A (Required for both undergrads and grads)

The following image pairs are a high quality image of a PowerPoint slide, and a low quality image that is a frame of a video of a presentation that used that slide. The format (PGM) is a gray scale image that the SIFT software knows how to read, as does Matlab.

http://kobus.ca/teaching/cs477/data/slide1.pgm http://kobus.ca/teaching/cs477/data/frame1.pgm
http://kobus.ca/teaching/cs477/data/slide2.pgm http://kobus.ca/teaching/cs477/data/frame2.pgm
http://kobus.ca/teaching/cs477/data/slide3.pgm http://kobus.ca/teaching/cs477/data/frame3.pgm

While you need black and white images for the sift program, visualizing results is more fun in color. So here are the corresponding color images that you should use to make all images requested for your report, even though what you mark on them will be determined via SIFT features computed from the black and white ones.

http://kobus.ca/teaching/cs477/data/slide1.tiff http://kobus.ca/teaching/cs477/data/frame1.jpg
http://kobus.ca/teaching/cs477/data/slide2.tiff http://kobus.ca/teaching/cs477/data/frame2.jpg
http://kobus.ca/teaching/cs477/data/slide3.tiff http://kobus.ca/teaching/cs477/data/frame3.jpg

The following links are to executables and corresponding libraries for extracting SIFT features for various OSs and architectures. You only need to download one pair of them. However I have not tested on all architectures (let me know if you come across any problems). Unless your computer is really old, you probably want one of the 64 bit versions. These programs are from the open source VLFeat system (VLFeat stands for Vision Lab Features). You can find out more about VLFeat, and even get the source code from http://www.vlfeat.org.

Newer computers

(Mac, Intel, 64 bit)

http://kobus.ca/teaching/cs477/code/vlfeat/maci64/sift
http://kobus.ca/teaching/cs477/code/vlfeat/maci64/libvl.dylib

(Linux, 64 bit)

http://kobus.ca/teaching/cs477/code/vlfeat/glnxa64/sift
http://kobus.ca/teaching/cs477/code/vlfeat/glnxa64/libvl.so

(Windows, 64 bit)

http://kobus.ca/teaching/cs477/code/vlfeat/win64/sift.exe
http://kobus.ca/teaching/cs477/code/vlfeat/win64/vl.dll

Older computers

(Mac, Intel)

http://kobus.ca/teaching/cs477/code/vlfeat/maci/sift
http://kobus.ca/teaching/cs477/code/vlfeat/maci/libvl.dylib

(Linux, 32 bit)

http://kobus.ca/teaching/cs477/code/vlfeat/glnx86/sift
http://kobus.ca/teaching/cs477/code/vlfeat/glnx86/libvl.so

(Windows, 32 bit)

http://kobus.ca/teaching/cs477/code/vlfeat/win32/sift.exe
http://kobus.ca/teaching/cs477/code/vlfeat/win32/vl.dll

The program is relatively easy to use, and the default response to a black and white PGM image is simply a dot "sift" file which is a text file with rows consisting of 134 numbers. These are 4 numbers for x, y, scale, and direction of each key-point followed by 128 numbers for a descriptor for that keypoint. The number of keypoints found in the image is the number of rows in this file. If you run the program with no arguments, it will output some of the options you can play with if you like. More details can be found at: http://www.vlfeat.org.

1. For each slide-frame image pair, find the best slide keypoint for each every frame keypoint using the nearest neighbor computed from the Euclidean distance of the 128 element feature vector.

   *A previous posting of this assignment requested 1-1 matches, which you can do instead, but doing 1-1 for A.3 below is awkward than intended, and so if you use 1-1 here, the comparison with the result in A.3 is not precise (but we will accept it, since this change was made in mid-stream).*

   To visualize results, choose some subset of the N pairs to enable visualizing them without too much confusion. For example you might choose every 5$^{th}$ pair. To visualize your results, for each of the 3 pairs of images, produce a collage of four images, in the following arrangement:
   slide_kp frame_kp slide_match frame_match
   where the left two images should show the chosen subset of the keypoint pairs used in the N matches by drawing the keypoints for the slide on slide_kp, and keypoints for frame in frame_kp. Keypoints should be visualized with vectors attached to them showing the scale (via their length) and the orientation. The right two images should have lines connecting each of the matched keypoint pairs chosen for display (without the vectors).

   *Possible helpful code. If you have not already developed or found a way to draw a line into an image, you are free to use the code linked here which does this for color images:*
   http://kobus.ca/teaching/cs477/code/draw_line.m

   Experiment a bit with N for each image, and see if you can notice that closer matches tend to be better. Choose a value of N that is a fixed percentage P of the number of slide keypoints (P should be the same for each image pair). For example, your should be able to compute the "best" 20% of the matches. Experiment with P to find a value that provides many good matches at the expense of having some bad matches also. Shoot for roughly 2/3 good matches and 1/3 bad matches, **but I do not know if this possible**—perhaps we have to live with a smaller fraction of good matches. Again, if N is too large for clarity, visualize every second or third or fourth pair. Provide P in your report **($)**. Your program should display these collages, and write them out so that you can **put them into your report** (with a caption!) **($)**.

2. Using the same P as above, try measuring the distance between matches as the angle between the two 128 element feature vectors. Again, your program should display the collages and write them out so that you can **put them into your report** (with a caption) **($)**. Comment on what you found in comparison with the basic Euclidean distance **($)**.

3. In the paper, Lowe suggests that the ratio of the distance to the nearest neighbor and the second nearest neighbor can be used to make matching more robust. Lowe describes using the distance ratio to prune matches based on a threshold for the ratio instead of simply using the ratio as a distance measure. In particular he rejected any match with ratio greater than 0.80. For your implementation, you can simply skip finding a match for the frame keypoint if the best match does not satisfy this criterion. Does this provide roughly the same result? **($)**.

4. Using your preferred method, use the results above to set a threshold, T, that seems like a good choice for accepting a match as good. Now, for each method, try matching **all** slides to **all** frames, and report the number of matches in a 3x3 matrix where the element (i,j) is the number of matches T for slide_i with frame_j. Make sure you put these **confusion matrices** into your report **($)**, and comment on what they say. In particular address using the maximum match number as a way to match the frames to its corresponding slide, even as the number of slides and frames grows larger

## Part B (required for both ugrads and grads)

5. Implement K-means clustering. You should make K easy to set. At every iteration you must print out the value of the objective function (which should go down). Recall that the objective function is the sum of the squared errors from the mean of the assigned cluster. Use your implementation to segment the following three images based on color taken as a 3D feature vector, using k=5 and k=10.
   http://kobus.ca/teaching/ista452/data/sunset.tiff .
   http://kobus.ca/teaching/ista452/data/tiger-1.tiff .
   http://kobus.ca/teaching/ista452/data/tiger-2.tiff .

   For visualizing the segmentations, set each segment to its mean color. You could spice it up adding region borders. One way to do so is to make any pixel that is **not** surrounded by pixels from the same cluster a distinctive color.

   Hand in code to generate the segmentations (6 total), and put the original and segmented images (with captions) into your PDF report **($)**.

6. Add $\lambda$*X and $\lambda$*Y to your feature vector, where (X,Y) are the coordinates of the pixel, and $\lambda$ is constant that you can play with. If $\lambda$ is zero, then you would have the same as before. For at least one image, try several values including $\lambda$=1 and $\lambda$=10. Do the results make sense? You may have to push $\lambda$ higher to have a significant effect.

   Describe what you did in your PDF with some result images **($)**, any code, and some segmented images. Your report should explain the result of varying $\lambda$.

## Part C (Required for grads, optional for undergrads)

7. Construct some texture features from a black and white versions of the above three images. To get texture features, use your ability to find dots and horizontal and vertical edges at various blurring sigmas. Consider a window size, W, centered on each image pixel in turn. Compute the mean squared response of some filters over the pixels of W, for some sigmas. This will result in a feature vector that captures some notion of texture for the location of W.

   Explore **(a)** using this feature vector alone; **(b)** together with the full color (RGB), and **(c)** with the spatial location feature in the previous question, for some choice for W. You do not need to provide images for more than one choice of W, but make it easy to experiment with so you can provide results for a choice of W that you like. Provide results for multiple images for the three different feature vectors just described **($)**. If the three images that are provided do not behave interestingly enough, try some other ones!

   *Don't forget that K-means is sensitive to the range of your data, and so you will want to scale your features accordingly.*

*For more interesting results, you may want to apply the scaling trick to the color features (e.g., multiply the color features by some constant to tune the degree that color is used (much like tuning k above to tune the degree that spatial adjacency is used).*

Describe what you did and what you found, together with some images, in your PDF report **($)**. The code you hand in should generate the images in your report and display them.

## Part D (Optional for both grads and undergrads)

8. Create a texton representation of texture from a modest size data base of images. If you have trouble gathering suitable images, let me know, and I can provide some. Recall that you want to cluster feature vectors that are the result of convolving images with a set of filters in a "filter bank". You can use K-means for clustering.

   Having found your textons, the characterization of texture in your test image at a given point is the histogram of textons in a window W around the point. You can then cluster image points using K-means again to segment the image. Experiment with segmenting images based on **(a)** this texture characterization alone, **(b)** together with RGB, and **(c)** together with RGB and the spatial location feature (as above). Provide results for the same set of images that you did in the previous question **($)**. Comment on any differences with the more naïve one from the previous question **($)**.

   *Note. For the segmentation phase you might wonder if you should use the chi-squared measure for the texture elements instead of sum of squares implied by Euclidean distance. However, K-means is essentially tied to Euclidean distance because the minimum error occurs when the cluster centers are the mean, which is conveniently true with Euclidean distance, but not chi-squared.*

## Part E (Optional)

If you were keen on doing HW3 part E, or a second of the two choices for HW5 part B, but did not quite have time, they will be considered for grade as part of this assignment.

## What to Hand In

As usual, the main deliverable will be PDF document that tells the story of your assignment as described above. Ideally the grader can focus on that document, simply checking that the code exists, and seems up to the task of producing the figures and results in the document. But you need hand in your code as well as follows.

**If you are working in Matlab (recommended)**: You should provide a Matlab program named hw6.m, as well any additional dot m files if you choose to break up the problem into multiple files. Do not package up the files into a tar or zip file–this messes up the D2L conventions.

**If you are working in C/C++ or any other compiled language you need to discuss this with the instructor at least one week before the due date**: You should provide a Makefile that builds a program named hw6, as well as the code. The grader will type:

```
make    ./hw6
```

You can also hand in hw4-pre-compiled which is an executable pre-built version that can be consulted if there are problems with make. However, the grader has limited time to figure out what is broken with your build. In general a C/C++ solution will require nonstandard libraries, and you should discuss with

the instructor how they can be provided as part of your submission, or assumed to exist on the system that is used for testing.

**If you are working in any other interpreted/scripting language (again you need to discuss this with the instructor at least one week before the due date)**: Hand in a script named hw4 and any supporting files. The grader will type:

```
./hw6
```