

CS 477/577 - Introduction to Computer Vision

Assignment Nine

Due: 11:59pm (*) Sunday, November 10

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

Weight: Approximately 5 points

This assignment can be done in groups of 2-3. You should create one PDF for the entire group, but everyone should hand in a copy of the PDF and the code to help the TA keep track.

Make it clear on the first page of your PDF whether you worked as a group, and if so, who your group is, and a brief explanation of how you worked together, and a brief explanation of how you worked together. If the group consists of both graduate and undergraduate students, make it clear to what degree undergraduates contributed to any graduate parts or extra parts. Group reports are expected to be higher quality than what the individuals would have done on their own.

General instructions

You can use any language you like for this assignment, but unless you feel strongly about it, you might consider continuing with Matlab.

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.

30% of the assignment grade is reserved for exposition.

Learning goals

- Understand what the sparse SIFT detector does
- Learn about using local feature vectors to find known planar objects in images
- Reinforce/learn about three different commonly used feature vector distances
- Learn about quantifying edge energy and how a classic corner detector works (grad)

You will find that many of the matches that we find in this assignment are wrong. However, do not give up on this quite yet --- we will fix this assignment 11.

Assignment specification

This assignment has two parts, one of which is required for both undergrads and grads, and a second one that is only required for graduate students (undergrads are eligible for modest extra credit).

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 image format (PGM) is a gray scale image that the SIFT software knows how to read, as does Matlab. I have put these into D2L as well.

<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. I have put these into D2L as well.

<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>. Of course, if you prefer, you can get the latest version of the binaries from there also, which might be worth a try if you run into problems.

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 132 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. You should look at this file and see if it is what you expect. For example, do you expect integers for X and Y? How about the 128 numbers in the descriptor?

Hint about the XY values. The implemented version does not simply find the maxima in $XY\sigma$ space in a loop over discrete cells as described in class, but fits a quadratic function to the data in the neighborhood.

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 every frame keypoint using the nearest neighbor computed from the Euclidean distance of the 128-element feature vector. To visualize results, choose some subset of the N pairs to enable visualizing them without too much confusion. For example, you might choose every 5th 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 the keypoints for frame in frame_kp (\$). Keypoints should be visualized with vectors attached to them showing the scale (via their length which should be proportional to scale) and the orientation. The right two images should have lines connecting the location of the slide keypoint to the location of the frame keypoint. There may be code in vlfeat or other sources to visualize the keypoints in this way, **but you should code it up yourself** (but you can make use of code that draws lines). See Figure 1.

Possible helpful code. If you have not already developed or found a way to draw a line segment into an image, you are free to use the code linked here which does this for color images (also on D2L/Code):

http://kobus.ca/teaching/cs477/code/draw_segment.m

An example of the first kind of visualization (used for the left two images) is the house image in slide 19 of lecture 15, but perhaps you can use a nicer color, and a better way to distinguish the endpoints of the line segments. An example of the second kind of visualization is in slide 26 of lecture 15, except that the pairs are vertical instead of horizontal. However, if you prefer vertical, that is fine.

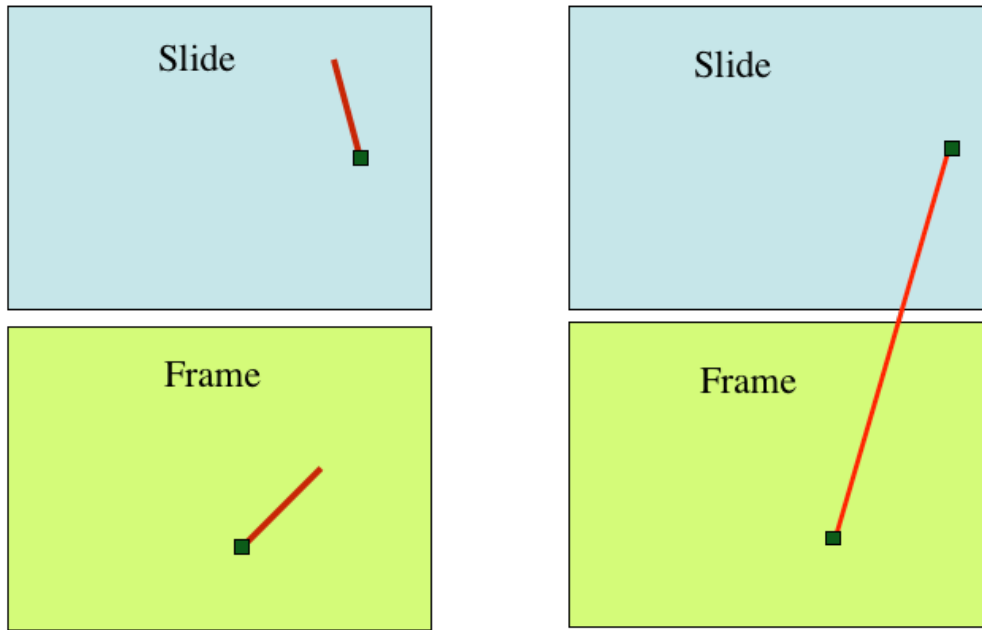


Figure 1. Left column. Slide and frame keypoints indicated by green boxes, and attached red lines showing scale via their length and orientation via their angle. Right column. Frame keypoint matches for the slide keypoints are shown by connecting lines.

2. This problem is set up to remind us that there are multiple ways to measure error between features. Your code should implement all three measures (only one line of code difference between them) and display the results. Having done so, report the best one, or declare that they are all roughly the same, and use the first one, for results for your writeup.
 - a. Euclidean: Consider the best P percentage of the matches as defined by the Euclidean distance (P should be the same for each image pair). For example, you should be able to compute the “best” $P=20\%$ of the matches. See if you can notice that closer matches tend to be better. Experiment a little with P to find a value that provides many good matches at the expense of having some bad matches. Try at least one value of P less than 20% and value greater than 20% . You want to set things up so that it is easy to play with this parameter, but do not spend a lot of time searching for a *really good* one. Visualize your results by putting images side by side and drawing lines between the matches. Again, if the number of points is too large for clarity, visualize every second or third or fourth pair.
 - b. Cosine: As above, but use angular distance (or the cosine) as the measure of how close two feature vectors are.
 - c. χ^2 : As above, but use the χ^2 based measure introduced in class when we were discussing texture. The reason for trying this is that we know that the 128 features are analogous to histogram counts (they are counts of edges in directional bins weighted by edge strengths).

Provide your chosen P in your report (\$). Your program should display these collages and write them into files so that you can **put them into your report** (with a good caption!) (\$).

Briefly summarize your findings in your report (\$), including which one you thought worked best (\$), the corresponding value of P (\$), and provide the visualization along the lines of the preceding problem (1) with an informative caption (\$).

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. You may want to experiment a bit with the ratio threshold. Does the approach help (i.e., do you agree with the suggestion?) or does it provide roughly the same result? (§).
4. Using your preferred error measure (and ratio of distances if you found that to be helpful), use the results above to set a threshold, T , that seems like a good choice for accepting a match as reasonably likely to be good. Now, try matching keypoints from **all** slides to keypoints from **all** frames. If things are working well, many keypoints from slide 1 would find their true match in the corresponding frame, 1, and similarly for keypoints from slide 2 and 3. Thus, if we were trying to find which frame corresponds to a particular slide, we could ask which frame yields the most matches.

Report the number of matches in a 3x3 matrix where the element (i,j) is the number of matches of keypoints from slide i with those in 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. If you experiment with some alternative strategies, be sure to summarize what you found in your report.

Part B (Required for grad students only)

In this part we will study quantifying edge energy and diversity by building a corner detector that can detect image locations where there is good edge strength in two orthogonal directions. The following is meant to be self contained, but a reference for it is Forsyth and Ponce 2nd edition page 150.

Consider the smoothed gradient operator ∇ response to an image I as you have implemented in the previous assignment. So ∇I is a column vectors with two components, one for the smoothed X derivative, and one for the smoothed Y derivative. The vector outer product $(\nabla I)(\nabla I)^T$, is the reverse of the dot product, and, in this case, is a 2x2 matrix. We sum up these matrices over a fixed sized window to get a symmetric matrix \mathcal{H} . Following Forsyth and Ponce:

$$H = \sum_{\text{window}} \left\{ (\nabla I)(\nabla I)^T \right\} \approx \sum_{\text{window}} \left\{ \begin{pmatrix} \left(\frac{\partial G_\sigma}{\partial x} \otimes I \right) \left(\frac{\partial G_\sigma}{\partial x} \otimes I \right) & \left(\frac{\partial G_\sigma}{\partial x} \otimes I \right) \left(\frac{\partial G_\sigma}{\partial y} \otimes I \right) \\ \left(\frac{\partial G_\sigma}{\partial x} \otimes I \right) \left(\frac{\partial G_\sigma}{\partial y} \otimes I \right) & \left(\frac{\partial G_\sigma}{\partial y} \otimes I \right) \left(\frac{\partial G_\sigma}{\partial y} \otimes I \right) \end{pmatrix} \right\} \quad (1)$$

The elements of the added matrices are products of elements of the gradient found as in the previous assignment. The only new part so far is taking the sum (you can think average if you like) of these self-outer products.

\mathcal{H} is like a covariance matrix with energies for edge directions X, Y, and the correlation of the two. (Take a minute to understand this). As in PCA, we can compute a rotation that puts the maximal energy direction in the first eigenvector direction and the rest in the second eigenvector direction. However, unlike PCA, where we would be happy if most of the variance is in one direction, here we want edge

diversity. This means that we want the both eigenvalues to be large. Neither of them being large means we do not have much edge energy at all, and only one of them being large means we have strong edge energy in only one direction.

A simple expression that gets at this criterion forms the basis of the Harris corner detector:

$$\det(\mathcal{H}) - k \left(\frac{\text{trace}(\mathcal{H})}{2} \right)^2, \quad (2)$$

where k is parameter you can tune (1/2 was used in the original paper), $\det()$ is the determinant, and $\text{trace}()$ is the sum of the diagonal elements. This mysterious formula is easier to understand if you know that:

$$\det(\mathcal{H}) = \lambda_1 \lambda_2 \quad \text{and} \quad \text{trace}(\mathcal{H}) = \lambda_1 + \lambda_2, \quad (3)$$

and you can use those two expressions directly. (The original paper used $\det()$ and $\text{trace}()$ simply because you can get them faster for a 2x2 matrix than the eigenvalues λ_1 and λ_2).

If you like playing with linear algebra, you might want to amuse yourself by deriving (3).

The Harris corner detector outputs local maxima of (2) that are larger than a threshold. Notice that this means the corner is detected at a pixel, as similar responses nearby are suppressed. (You are doing non-maximal suppression). For this assignment, you are to implement this detector, experimenting a bit with thresholds, Gaussian scales / window sizes (you can make window size proportional to scale), and perhaps k in (2). For your experiments, you should find a few images that are different with respect to corners (e.g., indoor, outdoor city, outdoor natural). You should summarize what you tried and what you found (e.g., perhaps k did not seem to matter much, but the effect of changing the scale (sigma) did what you expected, which was ...). You not need to provide results for everything you tried, but you should provide visualizations of several variations of your corner detector on several images (§). The variations you choose should illustrate a point, and this should be clear from the captions.

Finally (§), is this detector rotationally invariant? Convince the reader of your answer, ideally with an analytic argument.

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. So, you need hand in your code as well.