Lecture Notes, Week 8: SIFT

1 Correspondence Problem

A very basic problem in Computer Vision is to establish matches (correspondences) between images. In other words, finding a location of point (x, y) from a *source* image in the *target* image, assuming that point exists in both images. The correspondences could be *dense* or *sparse*. Dense correspondences allow finding such matches for all points/pixels in a given image in another image. This is something we will use in stereo, where the goal is to find a dense depth map. Sparse correspondences only find matches for some (typically locally salient or distinct) points/pixels which we will call *keypoints*.

We will for the time-being focus on sparse correspondences. In this context, the two main questions are (1) how do we select good locally distinct points to match, and (2) how do we match them robustly across images. In particular, our strategy will be to find candidate points in both source and target images and then to match them according to some criteria (*e.g.*, using a distance metric).

In order to be able to match the keypoints we need to make sure that we are able to locate them and match them across variety of typical transformations. For example, photometric changes – changes in lighting, changes in scale, rotation and viewpoint of the scene. While we have covered some methods that could give us locally distinct locations that would be good potential keypoints (*e.g.*, corners, blobs), matching them purely based on response of the corner/blob detector would result in too many ambiguities. These representations are highly impoverished in capturing local content. Therefore we will instead characterize local pixel neighborhoods around keypoints, with what we will call *keypoint descriptors*.

2 Scale Invariant Feature Transform (SIFT)

SIFT is a very robust keypoint detection and description algorithm developed by David Lowe at UBC. It is a technique for detecting salient and stable feature points in an image and for characterizing a small image region around this point using a 128-dimensional feature vector. SIFT makes use of local coordinate frames that define a position, local scale, and orientation of each feature point. It then computes the descriptor in this local coordinate frame. This ensures that the SIFT descriptor is robust to image and lighting transformations. SIFT can be characterized by four steps. The first three deal with keypoint detection / localization. The last one with keypoint description.

2.1 Step 1: Multi-scale extrema detection

In this step the goal is to find the locally distinct keypoints at their characteristic scale. This is done by first constructing a Difference of Gaussian (approximates a Laplacian) scale space and then looking for extrema in this space. This is very similar to the blob detection we covered before.

First a Gaussian scale pyramid is constructed, where an image is progressively smoothed, multiple times, and then sub-sampled by a factor of 2. The resulting Gaussian levels, before sub-sampling, form an *octave* and each octave corresponds to smoothing with $2\sigma_0$, where σ_0 is the base smoothing parameter of the Gaussian kernel. Within an octave, the adjacent scales differ by a constant factor k. The number of levels in an ocave is a hyperparameter s. If an octave contains $s+1$ stack of images, then $k = 2^{\frac{1}{s}}$. In other words, the first image in an octave is smoothed with Gaussian with σ_0 and effectively has the same scale factor - σ_0; the second image in an octave is convolved with a Gaussian with larger spread and has scale of $k\sigma_0$, the third has scale $k^2\sigma_0$. Note, the last image in the octave will have scale/smoothing of $k^s\sigma_0 = 2\sigma_0$.

Once a Gaussian scale space is constructed, a Difference of Gaussian (DoG) pyramid can be constructed by taking a difference of adjacent Gaussian levels within each octave. Note that for an octave with $s+1$ Gaussian levels, we will get exactly s DoG levels. We then look for spatio-scale extrema by comparing a value at each pixel (x, y) at a level L in DoG with it's 8-spatial neighbors at that scale, and 9 neighbors in the two adjacent scales. In other words, we compare response at level L pixel (x, y) to responses at:

$(x-1, y-1)$, $(x+1, y-1)$, $(x-1, y)$, $(x+1, y)$, $(x-1, y+1)$, $(x+1, y+1)$, $(x, y-1)$, $(x, y+1)$, at level L, $L-1$ and $L+1$ (we also compare to (x, y) at $L-1$ and $L+1$). If (x, y) is larger or smaller than ALL of the 27 neighbors, then it is a keypoint candidate and the location is stored along with the scale corresponding to L in a keypoint location/detection candidate list. Note that this procedure also gives a lower limit on the s, since we need at least three levels in DoG to conduct this operation.

2.2 Step 2: Keypoint Localization

Since the result of Step 1 is effectively maxima / minima in DoG, which corresponds to blobs, we know that some candidate keypoints will reside on edges, as DoG also has high response for the edge pixels. However, these points along the edges should not be considered locally distinct or salient because of the reasons we outlined with Harris corners (recall autocorrelation). Hence in this step we want to filter keypoint candidates such that only strong locally distinct keypoints remain. Hence two filtering criteria is applied. First, low contrast keypoints are removed. This is done by checking the significance of the specific keypoint maxima/minima (*e.g.*, if the keypoint is bigger than its neighbors by only one greyscale level, it is weak and should be filtered out). Second, we compute the ration of the eigenvalues for the covariance / Harris matrix (see Corner Detection) and filter out candidates where this ratio is not close to 1. Note that ratio close to 1 would constitute a corner like locally distinct structure. The output of this procedure is a sub-set of keypoints found in Step 1. Each keypoint at this stage has a position and a scale.

2.3 Step 3: Orientation Assignment

In this final keypoint localization / detection stage we additionally endow each keypoint with an estimated rotation. Keypoint orientation is estimated by forming a histogram of gradient orientations from pixels around the keypoint. Typically 36 bins are used, each corresponding to 10 degrees. The contribution of each pixel in the neighborhood are weighted by gradient magnitude and a Gaussian weighting function which emphasizes pixels closer to the keypoint. In other words, pixels whos gradient magnitude is larger are contributing more; pixels who are closer to the given keypoint (x, y) are also contributing more. Note that a different histogram is formed for each candidate keypoint from Step 2. The dominant orientation, correspondent to the bin with largest value, is assigned as the orientation for the keypoint in question. If multiple peaks, or histogram bins, are more than a large fraction of the peak (0.8) peak, then keypoint is duplicated with corresponding orientations. For example, consider a keypoint candidate at position (x, y) with scale σ for which orientation histogram has three peaks of value 20.4 at θ_1, 20.5 at θ_2 and 20.4 at θ_3 and all other histogram bins < 0.6. In this case, this keypoint will be duplicated three times, resulting in final keypoint detections / locations of: (x, y, σ, θ_1), (x, y, σ, θ_2), (x, y, σ, θ_3).

2.4 Step 4: Keypoint descriptor

The result of the previous three steps is a list of keypoints, each represented by a 4-dimensional vector. For example, if 478 SIFT keypoints are detected, then the output of the previous three stages is effectively a 478×4 matrix, each row of which will describe where, at which scale and orientation the keypoint was found. The goal now is to characterize the image region around this keypoint such that it can be matched.

The first step of the keypoint description is to conceptually define a small region of "pixels" that is rotated and scaled with the keypoint detection. Specifically, assuming we have a keypoint at position (x, y), scale σ and orientation θ. We can define a region of size 16×16 "pixels", centered around (x, y) that is scaled appropriately and divide it into non-overlapping cells. Typically, in SIFT, we divide the region into 4×4 cells, each of which consists of 4×4 pixels. For each of these cells we form a histogram of gradient orientations. Typically this histogram is quantized at 45 degrees, which results in 8 bins. Each of the 4×4 pixels vote into this histogram. The votes are weighted by a Gaussian weighting to account for larger

contribution of locations closer to the keypoint location and also by the gradient magnitude. The process is similar to the one in Step 3, however, while in Step 3 the patch which is considered is image aligned, here the alignment follows the keypoint orientation. Overall this procedure results in 8-dimensional histogram for each of the $4 \times 4 = 16$ cells; these are concatenated to obtain a 128-dimensional SIFT descriptor.

The final step is normalizing this 128-dimensional vector to unit norm. The motivation for this last step is to make the descriptor invariant to scaling (multiplicative change) in brightness. If brightness values are scaled by a constant, the gradients will be similarly scaled. Since we are using these to vote into the histograms, the histograms themselves will also be scaled. Dividing the vector by the norm alleviates this effect. Note that the SIFT descriptor is already resilient to additive increase/decrease in brightness as gradients are invariant to this transformation. In other words, after normalization, the SIFT descriptor is robust to most typical lighting (or photometric) changes.

2.5 Alternatives to SIFT

SIFT is both a keypoint detector and a keypoint descriptor. Other alternatives exist for both of these. For example, keypoint detection can be obtained using Harris Corner detection or Blob detection. SIFT keypoint detection is actually closely related to both of those, but details do differ slightly. Keypoint description alternatives include HoG and SURF. HoG, which stands for the Histogram of Oriented Gradients, is very similar to SIFT in that it also takes a region around a keypoint and builds a histogram of gradients from it. The difference is mainly in that the region is larger (64×128) and the cells are set to be overlapping, and the histogram quantizes absolute value of gradient directions. This results in a HoG descriptor that is 3780-dimensional. HoG tends to encode more detailed information but at the same time is slower; both are artifacts of HoG's higher dimensionality. SURF, which stands for Speeded Up Robust Features, instead of building a histogram of gradients for each cell, simply computes first order statistics, *e.g.*, average x- and y- derivatives and average absolute value of x- and y- derivatives. The result is a descriptor feature vector that is 64-dimensional, which is faster to use in practice.

3 Finding Matches Across Images

Given a SIFT descriptor, solving a Matching Problem amounts to finding correspondences among SIFT keypoints found in *source* image and SIFT keypoints found in *target* image. This task can be solved using a simple nearest-neighbor matching. Specifically, for a keypoint descriptor in one image, lets call it $\mathbf{x}_i \in \mathbb{R}^{128}$ we can compute the Euclidian distance to all keypoint descriptors in another image, lets call these $\{\mathbf{y}_j \in \mathbb{R}^{128}\}$. The matching can then be expressed as follows:

$$NN(i) = \arg\min_j |\mathbf{x}_i - \mathbf{y}_j|$$

where $NN(i)$ will produce an index of the matched keypoint.