[These lecture notes are meant to complement the slides. They can be read on their own, but I've left out some details and all the figures.]

Features 2

We have discuss several local features so far in the course: edges, corners (locally distinctive points), RGB histograms. Today we'll look at another classic method for describing local image structure, known as SIFT. SIFT was invented by David Lowe (a CS prof at UBC). It was first published in 1999, and a longer publication appeared in 2004. The two publications have together have been cited over 75,000 times.

SIFT: Scale invariant feature transform

SIFT stands for "scale invariant feature transform". Scale is a fundamental aspect of SIFT, as we'll see, and SIFT builds on the scale space ideas from lecture 9. So let's return to those ideas briefly.

Given an image, suppose that we have computed a scale space using a normalized Laplacian of Gaussian filter $\sigma^2 \nabla^2 G_\sigma(x,y)$ filter. I argued that such a scale space will give local maxima and local minima over (x,y,σ) when there is a box-like intensity pattern present (sometimes called a "blob"). The box either can be brighter than its background (this gives a local minimum) or darker than its local background (this gives a local maximum). The maxima and minima occur at specific scales which are related to the size of the box. Local maxima and minima arise from other image intensity patterns – not just 2D boxes.

Local maxima and minima in the 3D scale space define the positions (x,y,σ) of the SIFT *keypoints*. However, as we'll see next, SIFT does not use a normalized Laplacian of a Gaussian filter. Instead it creates a scale space that is essentially equivalent.

Difference of Gaussians scale space

It has been know for many years[1] that a Laplacian of a Gaussian function has a nearly equivalent shape as another function – called a "difference of Gaussians" – provided that the two Gaussians have similar standard deviations. For completeness, let's examine where this equivalence comes from.

First, one can show using basic calculus that the 2D Gaussian formula satisfies:

$$\frac{\partial G(x,y,\sigma)}{\partial \sigma} = \sigma \nabla^2 G(x,y,\sigma).$$

Now, approximate the left side $\frac{\partial G(x,y,\sigma)}{\partial \sigma}$ as follows:

$$\frac{\partial G(x,y,\sigma)}{\partial \sigma} \approx \frac{G(x,y,k\sigma) - G(x,y,\sigma)}{k\sigma - \sigma}$$

where k is some number that is slightly larger than 1. The approximation approaches equality as $k \to 1$. Rewrite that approximation as follows:

$$(k-1)\ \sigma\ \frac{\partial G(x,y,\sigma)}{\partial \sigma} \approx G(x,y,k\sigma) - G(x,y,\sigma).$$

[1]For example, Marr and Hildreth discussed this in their 1980 paper.

Substituting from the first equation above gives our result:

$$(k-1)\ \sigma^2\ \nabla^2 G(x,y,\sigma) \approx G(x,y,k\sigma) - G(x,y,\sigma).$$

The right side is called a *difference of Gaussians* or DOG

$$DOG(x,y;\sigma) \equiv G(x,y,k\sigma) - G(x,y,\sigma).$$

We can build a scale space by filtering an image with a DOG. Since the DOG is nearly identical to a normalized Laplacian of a Gaussian times a fixed constant $k-1$, the DOG scale space has similar properties as the normalized Laplacian of a Gaussian scale space. In particular, box images produce local minima or maxima across scales in a DOG scale space. If we convolve a 2D box with $DOG(x,\sigma)$ of different scales σ, then we will get a minimum at scale $\sigma = \sigma_0$ that corresponds to the halfwidth of the box.

SIFT uses a Gaussian pyramid. When we introduced a Gaussian pyramid in lecture 9, we sampled the scales by successively doubling the σ values and halving the pixel sampling rate. However, this sampling of the scale dimension is too coarse for our purposes here. We would like to more accurately say at which scale the maxima or minima occur. Therefore, SIFT divides each doubling of scale (called an octave) into m steps that are equal on a log scale, for example,

$$\sigma = 1, 2^{\frac{1}{m}}, 2^{\frac{2}{m}}, \ldots, 2^{\frac{m-1}{m}}, 2, 2^{1+\frac{1}{m}}, \ldots, 4, \ldots, 8, etc$$

So in this case, we are using $k = 2^{\frac{1}{m}}$. In the figures shown in the lecture slides, $m = 4$.

SIFT step 1: Find the position and scale of keypoints

SIFT keypoints are defined by the local minima and maxima in the DOG scale space. It finds these by brute force search over $3 \times 3 \times 3$ neighborhood of each point in the Gaussian pyramid. Note that for points (x, y, σ) such that σ lies on the boundary of two octaves, one needs to take care about what one means by a $3 \times 3 \times 3$ neighborhood, since the spatial sampling rate is different for the higher versus lower scales $\sigma 2^{\frac{1}{m}}$ and $\sigma 2^{-\frac{1}{m}}$.

Another issue is that one might not want to keep all minima and maxima. For example, one may want to ensure that this local peak is sufficiently well defined or well localized. This can be done using ideas similar to the Harris-Stevens operator. (I did not go into this in the lecture and I will omit the details here as well.)

SIFT step 2: Find a dominant direction

We next describe the intensities in the (x, y) neighborhood of the keypoint at that particular scale σ. Rather than working with the image intensities themselves, SIFT constructs a local descriptor of the image gradient vectors in the neighborhood of the keypoint. More weight is given to gradient vectors of positions near the keypoint, namely the magnitudes of gradient vectors around a keypoint are multipled by a Gaussian with standard deviation proportional the scale σ of the keypoint. These weighted gradient vectors are then binned by their direction. 36 bins are used, each 10 degrees wide. An *orientation histogram* is defined which adds up the magnitudes of the gradients at each (binned) orientation:

```
//  Let the keypoint be (x0, y0, sig0).
//  Let the number of angular bins be numAngleBins.
//  Lowe 2004 used 36 angular bins, so binWidth was 10 degrees.
//  gradient vectors around keypoint are grad(I*G(x,y,sig0))

orientationHist = zeros(numAngleBins)
for each gradient vector in (x,y) ngd of keypoint
  theta = direction of gradient vector at (x,y)
  binTheta = round(theta / binWidth)
  orientationHist[ binTheta ] +=  length of gradient vector at (x,y)
                                  * W(x,y; sig0)
```

where the weight `W(x,y, sig0)` gives more weight to positions `(x,y)` that are closer to the keypoint. The neighborhood width is proportional to the scale of the keypoint, when distance is measured in a non-pyramid scale space. In a pyramid scale space, the neighborhood distance is a constant fraction of the width of the image at the keypoint's scale.

The orientation histogram will have maximum at some direction, called the *dominant orientation.* If there are multiple maxima that are similar in value then there are multiple dominant orientations. In that case, one creates multiple SIFT keypoints, namely one for each dominant orientation.

Feature descriptor

The orientation histogram is useful for defining a dominant orientation for the intensity gradients near the keypoint. However, the orientation histogram does not carry any information about the spatial distribution of the gradients in the neighborhood of the keypoint. SIFT keeps some of the information about the spatial arrangement of the gradients, as follows.

Once a dominant orientation for a keypoint has been found, SIFT defines a rotated square around the keypoint which is oriented with two sides parallel to the dominant direction. The width of the square is the same as (or similar to) the width used for the neighborhood to define the dominant direction. A 16×16 grid of intensity gradient vectors are then defined on this square. The intensity gradient vectors are interpolated from the gradient vectors defined in the Gaussian pyramid.

This 16×16 grid is then partitioned into a 4×4 array of 4×4 subgrids. (See slides.) For each of the subgrids, an orientation histogram with 8 orientations (each 45 degrees wide) is constructed – namely the magnitudes of the gradient vectors falling in each bin are summed. These $4 \times 4 = 16$ orientation histograms, each with 8 orientation bins, define a 128 dimensional "feature descriptor". Note that we have reduced the number of dimensions from 512 ($16 \times 16 \times 2$) to 128.

The most important property of these feature descriptors is that – in theroy at least – they should not change much when the image is resized or rotated. In this sense, we can say that SIFT features are both scale and rotation invariant. These are important properties, since it often happens that two images of the same scene might be somewhat rotated or scaled relative to each other. For example, the images might be shot with different cameras from different positions or orientations. While rotational invariance was used by several techniques prior to SIFT. SIFT was the first feature descriptor that was designed to be scale invariant.

Using SIFT features for image indexing (or recognition)

[Here I will give just a loose overview. I will not examine you on this section.]

Suppose we have a database of images, sometimes referred to as "training" images. We are given a new image and we would like to find whether the new image is similar to one of the images in the database (trainig set). How can we do this?

For each image in the database (training set), we compute the keypoints and the SIFT feature descriptor for each keypoint. For each of these SIFT feature, we have a vector (imageID, $x, y, \sigma, \theta_{dominant}$,descriptor). We create a data structure that represents all these SIFT features – namely points in a 128-dimensional space – and algorithms for indexing into this data structure.

We can think of a mapping (key, value) pair, where the key is the 128-D descriptor and the value is the above tuple vector. We can imagine searching for keypoints that match a given discriptor: given a (new) SIFT feature descriptor, find a set of similar SIFT features in the training set, i.e. SIFT features that were in the training set and whose 128-D descriptor is similar. (We might find the k most similar SIFT features in the training sets. There are many algorithms and data structures for this "k-nearest neighbor" problem. Details elsewhere.)

How are the SIFT features used for indexing? Given a new image, compute the SIFT features of that image. For each SIFT feature F_i in our new image, we find all the features in the training set that are with a threshold distance τ from the new feature (where the definition of distance is omitted here). Each of these nearby features F_j came from some image in the database (the value – see above). So, for each feature in the new image, we can index a set of candidate training set images that contain a SIFT feature that is similar to F_i.

What to do next? One naive idea is to take these sets of data base images (one set for each F_i) and vote, namely cast one vote for image J if F_i is close to feature F_j and F_j belongs to (maps to) image J. Then, the database image with the most votes is the one that is chosen.

The above scheme is simple. However, it doesn't work well since it ignores all spatial relations between keypoints. One way to improve the above scheme is to use clusters of keypoints in the new image, and match these clusters to images in the training set. For example, take a triplet of nearby keypoints in the new image. Each of the three keypoints will generate a set of matching features descriptors in the database, with each matching feature belonging to a image. For that triplet to cast a vote for an image in the database, the three sets of matching features must have that image in common. Moreover, because each of the three features in the triplet of features has a position, an orientation, and a scale, the matching features in the training set must obey consistent geometric relationships between the features. For example, if the features scales F_1, F_2, F_3 are related by some ratio $s_1 : s_2 : s_3$ in the original image, then this constrains the scales of the matching features in the database image. Similarly, the locations and orientations are constrained for a valid match. Such constraints allow one to prune away many of the potential matches.
[ASIDE: For more details, see Lowe's 2004 paper.]

Using SIFT features for panoramas

See slides. We will return to this problem in a few weeks.