Today we will look at a few important topics in scale space in computer vision, in particular, coarse-to-fine approaches, and the SIFT feature descriptor. I will present only the main ideas here to give you a sense of the problem and possible solutions. You should consider these ideas to be the tip of the iceberg. I'm hoping that some of you will be curious and will choose these as topics for your term paper or further readings on your own.

Coarse-to-fine image registration

Lucas-Kanade image registration made use the second moment matrix.

$$\mathbf{M} = \sum_{(x,y)\in Ngd(x_0,y_0)} \begin{bmatrix} (\frac{\partial I}{\partial x})^2 & (\frac{\partial I}{\partial x})(\frac{\partial I}{\partial y}) \\ (\frac{\partial I}{\partial x})(\frac{\partial I}{\partial y}) & (\frac{\partial I}{\partial y})^2 \end{bmatrix}$$

Let's begin by building a scale space out of this matrix. Suppose that each of the partial derivatives is computed with a normalized Gaussian derivative where the Gaussian has parameter σ. This defines a scale space of 2nd moment matrices, namely a family of second moment matrices that are defined over the domain (x, y, σ).

Notice that the summations over the neighborhoods themselves define a convolution. For example, if we were to define

$$w(x,y) \equiv \begin{cases} 1, & (x,y) \in Ngd(0,0) \\ 0, & \text{otherwise} \end{cases}$$

then we could write

$$\mathbf{M} = w(x,y) * \begin{bmatrix} (\frac{\partial I}{\partial x})^2 & (\frac{\partial I}{\partial x})(\frac{\partial I}{\partial y}) \\ (\frac{\partial I}{\partial x})(\frac{\partial I}{\partial y}) & (\frac{\partial I}{\partial y})^2 \end{bmatrix}$$

where it is understood that each term in the 2×2 matrix is a function of (x, y) and the convolution is applied to each of the terms.

Often in practice, one wishes to give more weight to closer neighbors of (x_0, y_0) and one uses a Gaussian for $w(x, y)$. In this common case, we have one Gaussian $G_{\sigma_D}(x, y)$ to blur the noisy image and a second Gaussian $G_{\sigma_I}(x, y)$ to integrate the second moment matrix. The D and I refer to the *derivative scale* and the *integration scale*, respectively. They are also sometimes called the *inner scale* and *outer scale*. Typically the ratio $\sigma_I : \sigma_D$ is constant, for example, 3.

Recall the version of the problem in which we were trying to find a translation (h_x, h_y) such that $I(x + h_x, y + h_y) \approx J(x, y)$. In deriving the Lucas-Kanade method, we used a first order model for the intensities of $I(x, y)$ in local neighborhoods. This required that the translation distance $h = |(h_x, h_y)|$ is much less than the radius of the neighborhood.[1] This implies that we need to choose a neighborhood scale σ_I that is much greater than h. But what do we do if the actual translation distance h is large ? In that case, it would seem that our estimate of (h_x, h_y) should give a poor match (since it is computed under the assumption that h is small).

Assume our scale space for the second moment matrix is defined by keeping the ratio $\sigma_I : \sigma_D$ constant, so by "σ" here let's just say we mean σ_D. Blurring the image by σ now removes not just the image noise, but also smooths out variations in the image signal at distances similar to σ. The

[1] Recall: The first order model says that the intensity is linear. But if the intensity is linear over the whole neighborhood, then we have the aperture problem and cannot uniquely determine (h_x, h_y).

blurred image is now smooth (roughly linear) at these distances and so the linear model applies at these distances, and we can estimate (h_x, h_y) up to around the distance σ.

Note, however, that this estimate of (h_x, h_y) often will be give us only a rough approximation since the LK method *also* assumes that (h_x, h_y) is constant over the integration neighborhood $Ngd(x_0, y_0)$. But using a larger σ_D means that we need to use a larger σ_I too[2] since we need the gradient ∇I to be constant over the σ_D neigbhorhood, but we need it to vary over the σ_I neighborhood. But the larger we make σ_I, the less likely it is that (h_x, h_y) will be constant over this neighborhood. What can we do?

The solution is to use a *coarse-to-fine* approach. Suppose we wish to estimate (h_x, h_y) near pixel (x_0, y_0). (We will repeat the following for all pixels, so let's just work with a single pixel.) We compute the scale spaces $I(x, y, \sigma)$ and $J(x, y, \sigma)$. We first estimate (h_x, h_y) at the largest scale, and then proceed iteratively to the smaller and smaller scales.

Suppose we have estimated $(h_x, h_y)^{k+1}$ at scale σ^{k+1}. To estimate $(h_x, h_y)^k$ at scale σ^k, we shift the pixels $(x, y) \in Ngd(x_0, y_0)$ and within scale σ^k

$$I^*(x, y) \leftarrow I(x + h_x^{k+1}, y + h_y^{k+1}, \sigma^k).$$

Note that we copy the values into a temporary variable $I^*(x, y)$ – that is, we don't actually modify the scale space!

We then apply LK registration to match $I^*(x, y)$ to $J(x, y)$ in $Ngd(x_0, y_0)$. This match gives an estimate (h_x, h_y), and so our net estimate at scale σ^k is:

$$(h_x, h_y)^k \leftarrow (h_x, h_y)^{k+1} + (h_x, h_y)$$

Notice that at each scale σ^k, the blur used to compute the gradient is less and less and the size of the neighborhood over which we integrate the gradients is less and less. This means that we ultimately are using all the information in the image, and we are (at the end) assuming that the translation (h_x, h_y) is constant in a small neighbhorhood only.

One final note: you should realize that there are two iterations going on here. There is the coarse-to-fine iteration mentioned above. And there is also the iteration *within* each scale σ^k which we discussed in lecture 10. I did not mention the latter iteration above – to avoid confusion. But now I mention it to remind you it is still there.

Local descriptors

Difference of Gaussians (DOG)

Let's now turn to our second topic for today's lecture. Recall the problem of blob detection from last lecture. I presented a simple solution to it which was based on the normalized Laplacian filter. One observation is that is "historically" very important in vision research is that the Laplacian of a Gaussian function has a similar shape to a difference of Gaussians (called a DOG), namely a difference of two Gaussians with different standard deviations.[3] You can see this by sketching the two functions by by hand (as I did in class - do it for yourself now). A more formal proof goes as follows:

[2] as mentioned earlier, often we are hold $\sigma_I : \sigma_D$ constant

[3] e.g. D. Marr, E. Hildreth, "Theory of Edge Detection", Proc. Royal Soc. Lond. (Series B), 1980.

First, one can show just by taking derivatives that

$$\sigma \frac{\partial^2 G_\sigma(x)}{\partial^2 x} = \frac{\partial G_\sigma(x)}{\partial \sigma}.$$

(This happens to be a version of the *heat equation* in physics.) Then[4] we can approximate

$$\frac{\partial G_\sigma(x)}{\partial \sigma} \approx \frac{G_{s\sigma}(x) - G_\sigma(x)}{s\sigma - \sigma}$$

where s is slightly bigger than 1, and so

$$\sigma \frac{\partial G_\sigma(x)}{\partial \sigma} \approx \frac{G_{s\sigma}(x) - G_\sigma(x)}{s - 1}$$

and substituting from the heat equation above gives what we want

$$\sigma^2 \frac{\partial^2 G_\sigma(x)}{\partial^2 x} \approx \frac{G_{s\sigma}(x) - G_\sigma(x)}{s - 1}.$$

Similarly in 2D,

$$\frac{G_{s\sigma}(x,y) - G_\sigma(x,y)}{s - 1} \approx \sigma^2 \nabla^2 G_\sigma(x,y).$$

Thus, the DOG and Laplacian of a Gaussian have similar shapes, as claimed. (Note that $s - 1$ is a constant, and so doesn't affect any claims about scale invariance.)

Often one approximates the normalized Laplacian scale space by computing a Gaussian scale space $(G_\sigma * I)(x, y)$ over a discrete set of scales $s^i \sigma_0$ and then computing the difference of Gaussians at neighboring scales. This is done by the SIFT method, which I will discuss below. (You will also use this scheme in Assignment 2.)

Keypoints

Suppose that we have computed a scale space using the normalized Laplacian (or a difference of Gaussians). I argued last lecture that such a scale space will give local maxima and local minima over (x, y, σ) when there is a "blob" present. In 2D, a blob is a square that is brighter than its background (this gives a local minimum) or a square that is darker than its local background (this gives a local maximum). Recall that the maxima and minima occur at specific scales which are related to the size of the squares. Also, the height of the local maxima and minima will depend on the difference of the intensity between the blob and its background. Last class I assumed the background intensity was zero, but since the DOG gives no response to a constant image all that matters is the difference in intensity (recall Question 7 on the midterm exam).

As you will see in Q2 of Assignment 2, for general images you will lots of local maxima and minima and these typically are *not* due to isolated square regions on a uniform background. Rather these local maxes and mins arise from other image intensity patterns. We can refer to a local maxima and minima of the filtered image as a *keypoint*. These are similar to Harris corner points

[4] This argument was taken from D. Lowe "Distinctive Image Features from Scale Invariant Keypoints, International Journal of Computer Vision 2004", but the basic idea is much older than that.

mentioned in lecture 9, but now we are specifically defining them to be local maxes and mins of the DOG filtered image in a 3D scale space, rather than in terms of a maximum of the Harris corner measure at a particular scale.

For each keypoint, we have a position (x, y) and a scale σ. If we had a second image of the same scene but the image was scaled relative to the first, the scale space of that image will be stretched relative to the first, and it should have corresponding maxima and minima. (Moreover, these maxima and minima should still be present if the second image was 2D translated and 2D rotated relative to the first image.)

If we have a set of several hundred keypoints in one image, and we want to match these to keypoints in another image, it is sometimes convenient to have a *descriptor* of the image in the neighborhood of each keypoint. Otherwise any keypoint in the first image could potentially be matched to any keypoint in the second image. Each keypoint has a scale σ so it makes sense to use the intensity structure in the scale space at that scale σ and in a local neighborhood of the keypoint position, where by "local" we mean a neighborhood that depends on σ. For example, as mentioned earlier, if σ is scale at which we compute the derivative (σ_D), then the neighborhood could be a Gaussian window defined by σ_I which is a fixed multiple of σ_D.

How can we describe the local intensity structure ? Let's briefly sketch out one solution, called SIFT, which has been extremely popular in the last decade.[5]

SIFT: Scale invariant feature transform

Assume we have found a keypoint at (x_0, y_0, σ) which is a local maximum (or minimum) in a difference of Gaussian scale space. We ensure that this local peak is sufficiently different from its neighbors and sufficiently different from zero, i.e. it is it is well localized and it is a large peak. We now want to describe the intensities in scale space in the neighborhood of the keypoint.

SIFT constructs a local descriptor of the normalized gradients vectors $\sigma \nabla G_\sigma * I(x, y)$ at the scale σ and in the (x, y) neighborhood of the feature point. Suppose we sample a square of width say 4σ with 64×64 grid of samples. Note that the larger is σ, the wider the spacing of the samples relative to the original image.

The first step is to find any *dominant orientations*. For example, a square has four dominant orientations corresponding to the four edges. A triangle with long side and two short sides would have one dominant orientation, whereas a triangle with two long sides and one short side would have two dominant orientations.

To find the dominant orientations, take the sampled gradient vectors in the neighborhood of (x_0, y_0) and treat them as a "set", that is, apart from the weighting we ignore their spatial locations. Then make direction bins and drop these vectors into their appropriate bin, as follows.

[5] `http://people.cs.ubc.ca/~lowe/keypoints`

```
//  Let theta(x,y) and mag(x,y) be the direction and length
//  of the image gradient at (x,y,sigma).
//
angleStep = 360/numAngleBins
for ctTheta = 0:numAngleBins-1
  hist(ctTheta) = 0
for (x,y) in Ngd(x0,y0, sig)
  hist(theta(x,y) mod angleStep) += mag(x,y)
```

Lowe uses 24 angle bins of 15 degree width.

This gives a histogram[6] of orientations. The bin with the maximum value is taken to be the dominant orientation of the keypoint. If there is more than one dominant orientation, i.e. there may be more than one peak in the histogram that has roughly the same (max) height, then make multiple local descriptors.

There are several ways one could imaging building a descriptor, once one has a dominant orientation. (For example, one could use the raw image intensities $I(x, y, \sigma)$ in some σ-neighborhood. This turns out to work very poorly, though, for matching between images, since the intensities tend to change enormously when either the lighting changes and or camera parameters change.)

What SIFT does (in a nutshell, ee his 1999 or 2004 paper for more details if you want to probe a bit deeper) is the following. Define a rotated square that is oriented parallel to the dominant direction found above. The width of the square is some multiple of the scale σ and the square is partitioned into a 16×16 grid. The gradient vector is chosen/computed for each of these 256 grid elements. This 16×16 grid is then partitioned in a 4×4 array of 4×4 subgrids. For each of the subarrays, an orientation histogram with 8 orientations of 45 degrees each is constructed (rather than 24 orientations of 15 degrees, which is what was used to find the dominant orientation for the whole neigbhorhood). These 16 orientation histograms, with 8 bins each, define at 128 dimensional "feature descriptor". (In Lowe's 2004 paper, he carries out several experiments comparing performance for various choices of the number of subgrids and number of direction bins.)

Of course, there are several technical details that I have left out here, but that is the main idea.

[6] http://en.wikipedia.org/wiki/Histogram