

Lecture 10

Detectors and descriptors



• Properties of detectors

- Edge detectors
- Harris
- DoG

• Properties of descriptors

- SIFT
- HOG
- Shape context

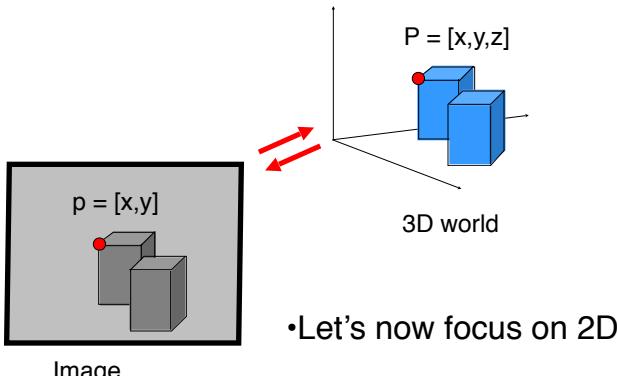
Silvio Savarese

Lecture 10 -

16-Feb-15

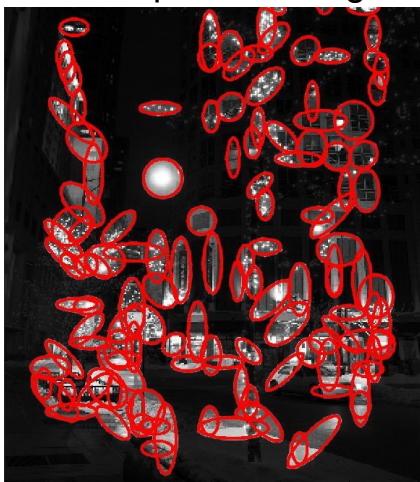
This lecture is about detectors and descriptors, which are the basic building blocks for many tasks in 3D vision and recognition. We'll discuss some of the properties of detectors and descriptors and walk through examples.

From the 3D to 2D & vice versa



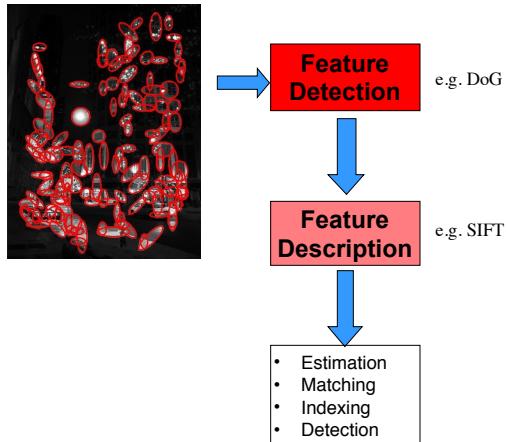
Previous lectures have been dedicated to characterizing the mapping between 2D images and the 3D world. Now, we're going to put more focus on inferring the visual content in images.

How to represent images?



The question we will be asking in this lecture is - how do we represent images? There are many basic ways to do this. We can just characterize them as a collection of pixels with their intensity values. Another, more practical, option is to describe them as a collection of components or features which correspond to "interesting" regions in the image such as corners, edges, and blobs. Each of these regions is characterized by a descriptor which captures the local distribution of certain photometric properties such as intensities, gradients, etc.

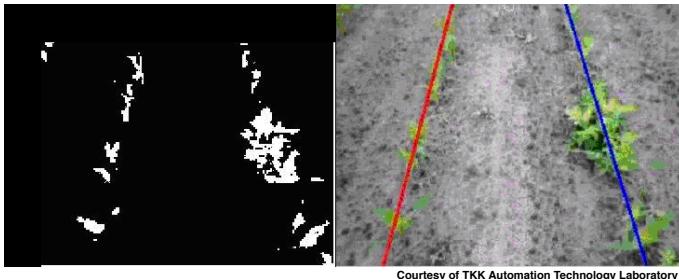
The big picture...



Feature extraction and description is the first step in many recent vision algorithms. They can be considered as building blocks in many scenarios where it is critical to:

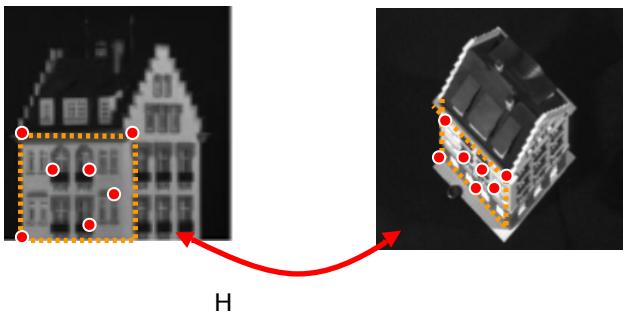
- 1) Fit or estimate a model that describes an image or a portion of it
- 2) Match or index images
- 3) Detect objects or actions from images

Estimation



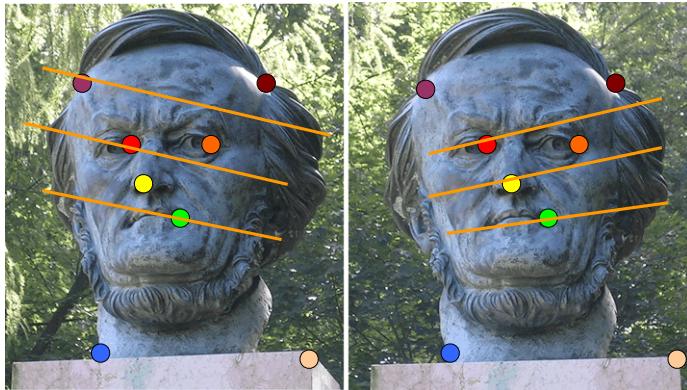
Here are some examples we've seen in the class before. In earlier lectures, we took for granted that we could extract out keypoints to fit a line to. This lecture will discuss how some of those keypoints are found and utilized.

Estimation



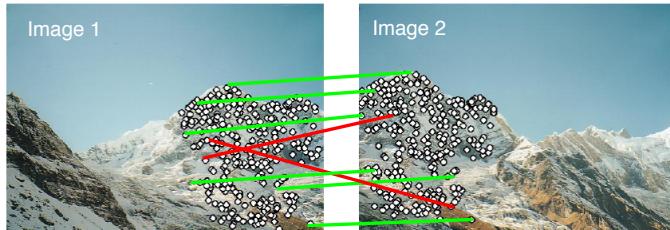
This is an example where detectors/descriptors are used for estimating a homographic transformation.

Estimation



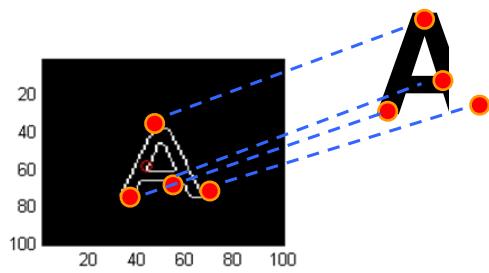
Here's another example from earlier in the class. When we were looking at epipolar geometry and the fundamental matrix F relating an image pair, we just assumed that we had matching point correspondences across images. Keypoint detectors and descriptors allow us to find these keypoints and related matches in practice.

Matching



Here's an example using detectors/descriptors for matching images in panorama stitching

Object modeling and detection



We can also use detectors/descriptors for object detection. For example, later in this lecture, we discuss the shape context descriptor and describe how it can be used for solving a shape matching problem.

Notice that in this case, descriptors and their locations typically capture local information and don't take into account for the spatial or temporal organization of semantic components in the image. This is usually achieved in a subsequent modeling step, when an object or action – needs to be represented.

Lecture 10

Detectors and descriptors



• Properties of detectors

- Edge detectors
- Harris
- DoG

• Properties of descriptors

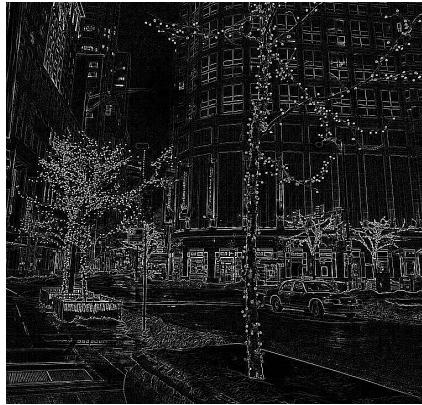
- SIFT
- HOG
- Shape context

Silvio Savarese

Lecture 10 -

16-Feb-15

Edge detection



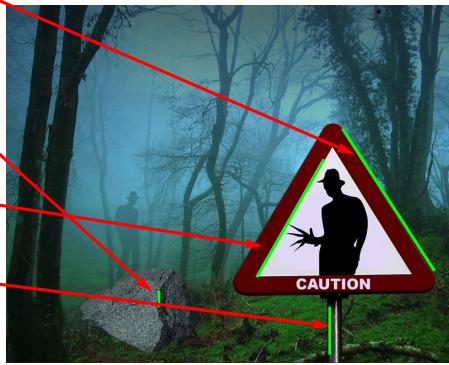
The first type of feature detectors that we will examine today are edge detectors.

What is an edge? An edge is defined as a region in the image where there is a “significant” change in the pixel intensity values (or high contrast) along one direction in the image, and almost no changes in the pixel intensity values (or low contrast) along its orthogonal direction.

What causes an edge?

Identifies sudden changes in an image

- Depth discontinuity
- Surface orientation discontinuity
- Reflectance discontinuity (i.e., change in surface material properties)
- Illumination discontinuity (e.g., highlights; shadows)



Why do we see edges in images? Edges occur when there are discontinuities in illumination, reflectance, surface orientation, or depth in an image. Although edge detection is an easy task for humans, it is often very difficult and ambiguous for computer vision algorithms. In particular, an edge can be induced by:

- Depth discontinuity
- Surface orientation discontinuity
- Reflectance discontinuity (i.e., change in surface material properties)
- Illumination discontinuity (e.g., highlights; shadows)

We will now examine ways for detecting edges in images regardless of where they come in the 3D physical world.

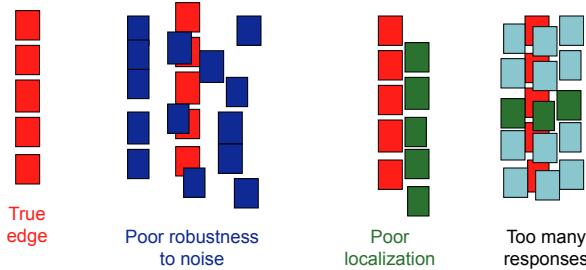
Edge Detection

- Criteria for **optimal edge detection** (Canny 86):
 - **Good detection accuracy:**
 - minimize the probability of false positives (detecting spurious edges caused by noise),
 - false negatives (missing real edges)
 - **Good localization:**
 - edges must be detected as close as possible to the true edges.
 - **Single response constraint:**
 - minimize the number of local maxima around the true edge (i.e. detector must return single point for each true edge point)

An ideal edge detector would have good detection accuracy. This means that we want to minimize false positives (detecting an edge when we don't actually have one) and false negatives (missing real edges). We would also like good localization (our detected edges must be close to the real edges) and single response (we only detect one edge per real edge in the image).

Edge Detection

- Examples:



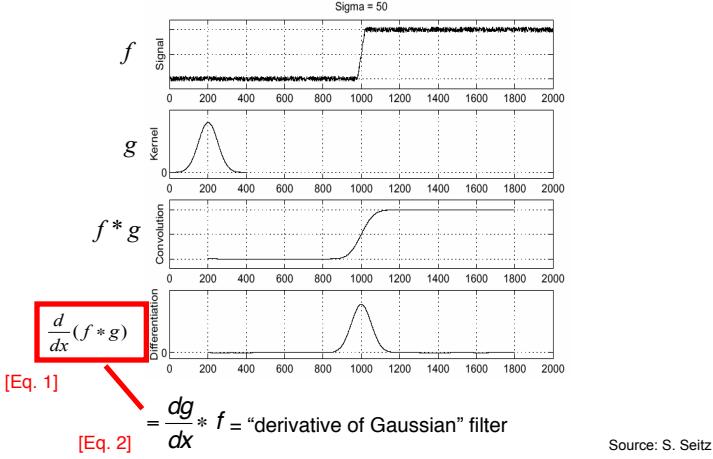
Here are some examples of the edge detector properties mentioned in the previous slide. In each of these cases, the red line represents a real edge. The blue, green and cyan edges represent edges detected by non-ideal edge detectors. The situation in blue shows an edge detector that does not perform well in the presence of noise (low accuracy). The next edge detector (green) does not locate the real edge very well. The last situation shows an edge without the single response property – we detected 3 possible edges for one real edge.

Designing an edge detector

- **Two ingredients:**
 - Use derivatives (in x and y direction) to define a location with high gradient .
 - Need **smoothing** to reduce noise prior to taking derivative

There are two important parts to designing an edge detector. First, we will use derivatives in the x and y direction to define an area w/ high gradient (high contrast in the image). This is where edges are likely to lie. The second component of an edge detector is that we need some smoothing to reduce noise prior to taking the derivative. We don't want to detect an edge anytime there is small amounts of noise (derivatives are very sensitive to noise).

Designing an edge detector



This image shows the two components of an edge detector mentioned in the last slide (smoothing to make it more robust to noise and finding the gradient to detect an edge) in action. Here, we are looking at a 1-d slice of an edge.

At the very top, in plot f, we see a plot of the intensities of this edge. The very left contains dark pixels. The very right has bright pixels. In the middle, we can see a transition from black to white that occurs around index 1000. In addition, we can see that the whole row is fairly noisy: there are a lot of small spikes within the dark and bright regions.

In the next plot, we see a 1-d Gaussian kernel (labeled g) to low pass filter the original image with. We want a low pass filter to smooth out the noise that we have in the original image.

The next plot shows the convolution between the original edge and the Gaussian kernel. Because the Gaussian kernel is a low pass filter, we see that the new edge has been smoothed out considerably. If you are unfamiliar with filtering or convolution, please refer to the CS 131 notes.

The last plot shows the derivative of the filtered edge. Because we are working with a 1-d slice, the gradient is simply the derivative in the x direction. We can use this as a tool to locate where the edge is (at the maximum of this derivative). As a side note, because of the linearity of gradients and convolution, we can re-arrange the final result $d/dx(f * g) = d/dx(g) * f$, (Equations 1 and 2 are equivalent) where the “*” represents

Edge detector in 2D

- Smoothing

$$I' = g(x, y) * I \quad [\text{Eq. 3}]$$

$$g(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad [\text{Eq. 4}]$$

- Derivative

$$S = \nabla(g * I) = (\nabla g) * I =$$

$$\nabla g = \begin{bmatrix} \frac{\partial g}{\partial x} \\ \frac{\partial g}{\partial y} \end{bmatrix} = \begin{bmatrix} g_x \\ g_y \end{bmatrix}$$

[Eq. 6]

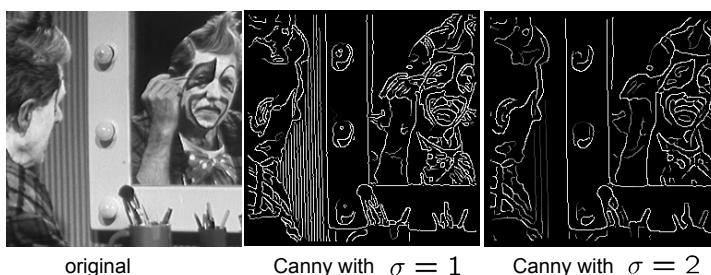
$$= \begin{bmatrix} g_x \\ g_y \end{bmatrix} * I = \begin{bmatrix} g_x * I \\ g_y * I \end{bmatrix} = \begin{bmatrix} S_x & S_y \end{bmatrix} = \text{gradient vector}$$

[Eq. 5]

In 2-d, the process is very similar. We smooth by first convolving the image with a 2-d Gaussian filter. We denote this by Eq. 3 (convolving the image w/ a Gaussian filter to get a smoothed out image). The 2d Gaussian is defined for your convenience in Eq. 4. This still has the property of smoothing out noise in the image. Then, we take the gradient of the smoothed image. This is shown in Eq. 5, where Eq. 6 is the definition of the 2d gradient for your convenience. Finally, we check for high responses which indicate edges.

Canny Edge Detection (Canny 86):

See CS131A for details



A popular edge detector that is built on this basic premise (smooth out noise, then find gradient) is the Canny Edge Detector. Here, we vary the standard deviation of the Gaussian σ to define how granular we want our edge detector to be. For more details on the Canny Edge Detector, please see the CS 131A notes.

- The choice of σ depends on desired behavior

- large σ detects large scale edges
- small σ detects fine features

Other edge detectors:

- Sobel
- Canny-Deriche
- Differential

Some other edge detectors include the Sobel, Canny-Deriche, and Differential edge detectors. We won't cover these in class, but they all have different trade-offs in term of accuracy, granularity, and computational complexity.

Corner/blob detectors



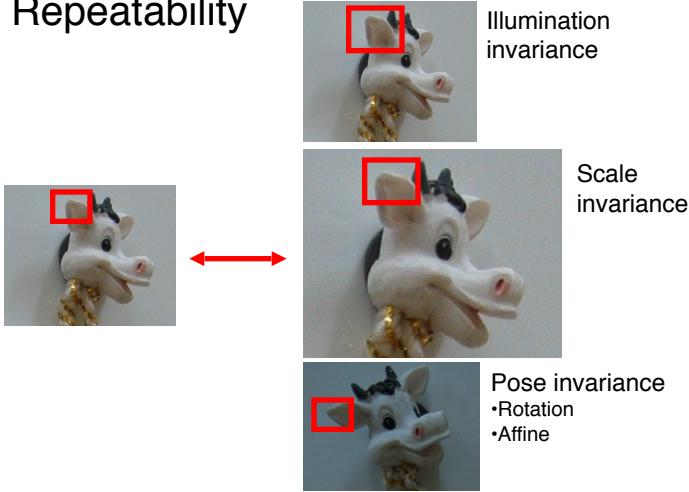
Edges are useful as local features, but corners and small areas (blobs) are generally more helpful in computer vision tasks. Blob detectors can be built by extending the basic edge detector idea that we just discussed.

Corner/blob detectors

- Repeatability
 - The same feature can be found in several images despite geometric and photometric transformations
- Saliency
 - Each feature is found at an “interesting” region of the image
- Locality
 - A feature occupies a “relatively small” area of the image;

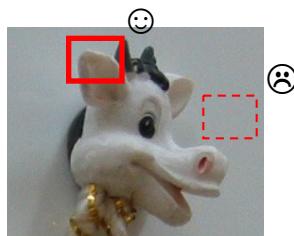
We judge the efficacy of corner or blob detectors on three metrics: repeatability (can we find the same corners under different geometric and photometric transformations), saliency (how “interesting” this corner is), and locality (how small of a region this blob is).

Repeatability

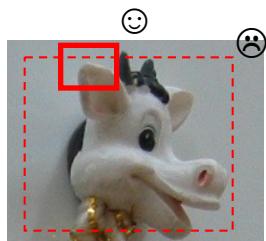


Ideally, we want our corner detector to consistently find the same corners in different images. In this example, we have detected the corner of this cow's ear. We would like our corner detector to be able to detect the same corner regardless of lighting conditions, scale, perspective shifts, and pose variations.

• Saliency



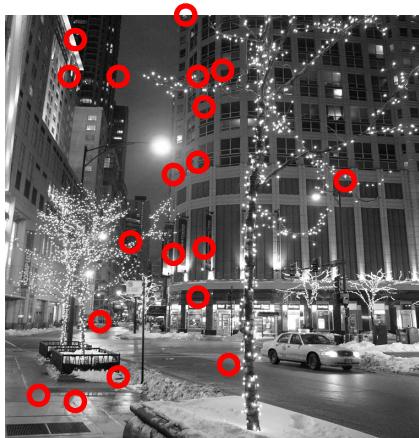
• Locality



Our corner/blob detector should also pick up on interesting (salient) keypoints. In the top image, an example of an area with poor saliency is indicated by the dashed bounding box. There is not much texture or interesting substance in this area.

We also would like our interesting features to have good locality. That is, it should take up a small portion of the whole image. If our blob detector returned the whole image (or nearly the whole image) – e.g., see region indicated by the dashed bounding box— that would defeat the purpose of keypoint detection.

Corners detectors

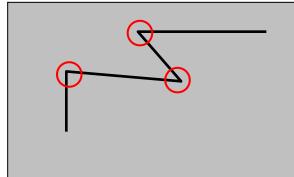


Here is an example of some corners that are detected in an image.

Harris corner detector

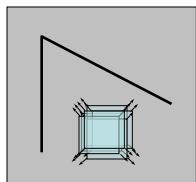
C.Harris and M.Stephens. ["A Combined Corner and Edge Detector."](#) *Proceedings of the 4th Alvey Vision Conference*: pages 147–151.

See CS131A for details

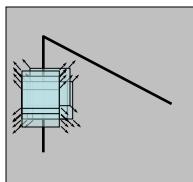


Harris Detector: Basic Idea

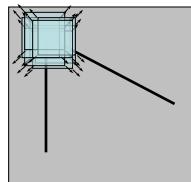
Explore intensity changes within a window as the window changes location



"flat" region:
no change in
all directions



"edge":
no change along
the edge
direction



"corner":
significant
change in all
directions

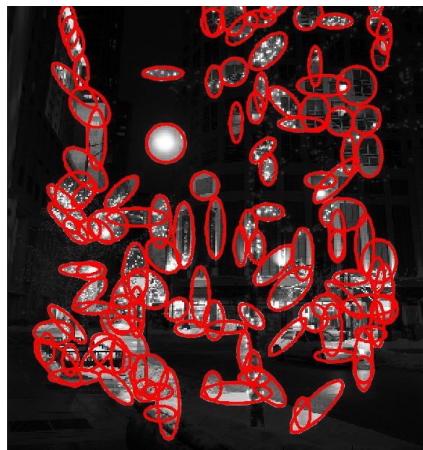
The basic idea of the Harris Corner Detector is to slide a small window across an image and observe changes of intensity values of the pixels within that window. If we slide our window in a flat region (as shown on the left) we shouldn't get much variation of intensity values as we slid the window. If we moved along an edge, we would see a large variation as we slid perpendicular to the edge, but not parallel to it. At a corner, we would see large variation in any direction.

Results



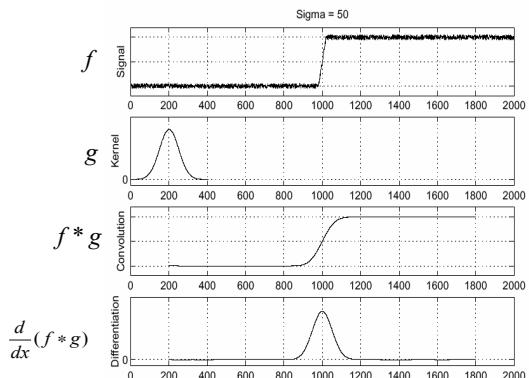
Here is an example result from the Harris Corner detector. It detected small corners throughout the cow figurine and was even able to pick up many of the same corners in an image with much different illumination.

Blob detectors



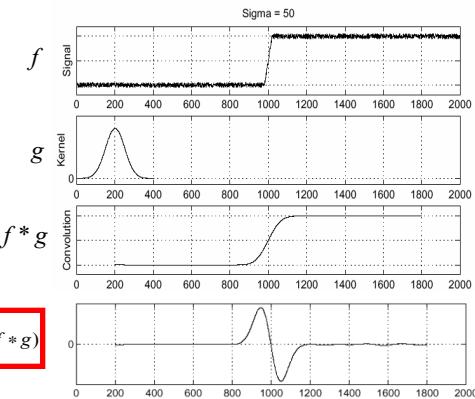
Now we will talk a bit about general blob detectors.

Edge detection



Blob detectors build on a basic concept from edge detection. This is the series of pictures from our earlier discussion about edge detection. From top to bottom, the plots are our edge, the Gaussian kernel, the edge convolved with the kernel, and the first derivative of the smoothed out edge.

Edge detection



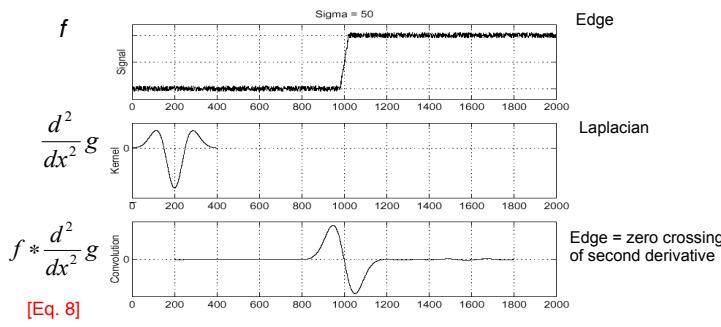
We can extend the basic idea of an edge detector. Here we show the second derivative of the smoothed edge in the last plot. This is equivalent to the Laplacian of the Gaussian filter (second derivative in this one dimensional example) convolved with the original edge, shown on the next slide.

[Eq. 7]

$$\frac{d^2}{dx^2}(f * g)$$

[Eq. 8] $f * \frac{d^2}{dx^2} g$ = "second derivative of Gaussian" filter = Laplacian of the gaussian

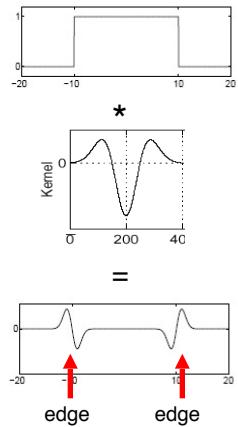
Edge detection as zero crossing



Here, we can see that the result from convolving the second derivative of the Gaussian filter (second plot) with the edge (first plot) is the same as convolving the original Gaussian filter and taking the second derivative. Similar to the previous case where we were examining the first derivative, the second derivative and convolution are both linear operations and therefore interchangeable. Thus, Eq. 7 from the previous slide and Eq. 8 are equivalent statements.

We also see that the zero crossing of the final plot (Laplacian of Gaussian convolved with edge) corresponds well with the edge in the original image.

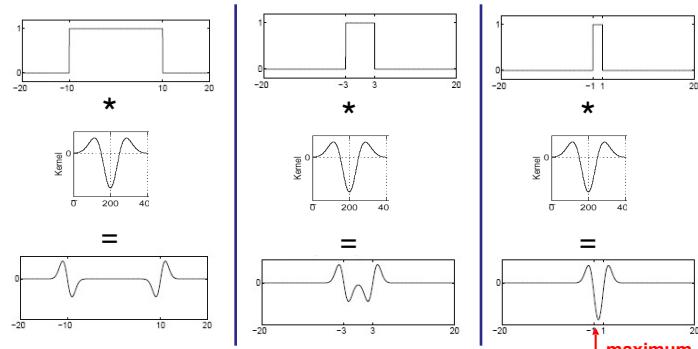
Edge detection as zero crossing



Now, we know that convolving the Laplacian of Gaussian with an edge gives us a zero crossing where an edge occurs. Here is an example where the original image has two edges; if we convolve it with the Laplacian of Gaussian, we get two zero crossings as expected.

From edges to blobs

- Can we use the laplacian to find a blob (RECT function)?

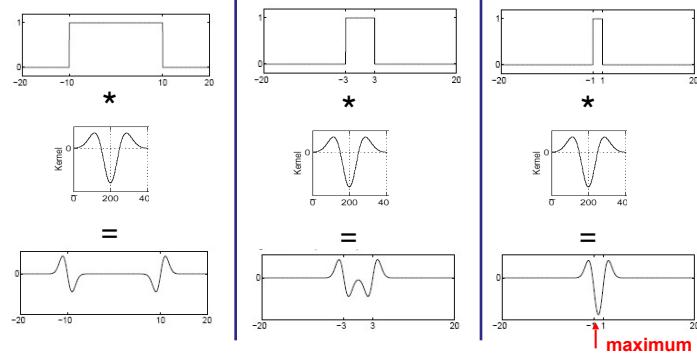


Magnitude of the Laplacian response achieves a maximum at the center of the blob, provided the scale of the Laplacian is "matched" to the scale of the blob

Now, let's generalize from the edge detector to the blob detector. The question is: can we use the Laplacian to find a blob or, more formally speaking, a RECT function? Let's see what happens if we change the scale (or width) of the blob. The central panel shows the result of convolving the Laplacian kernel with a blob with smaller scale. The right panel shows the result of convolving the Laplacian kernel with a blob with an even smaller scale. As we decrease the scale, the zero crossings of the filtered blob come closer together until they superimpose to produce a peak in the response curve: the magnitude of the Laplacian response achieves a maximum at the center of the blob, provided the scale of the Laplacian is "matched" to the scale of the blob.

From edges to blobs

- Can we use the laplacian to find a blob (RECT function)?

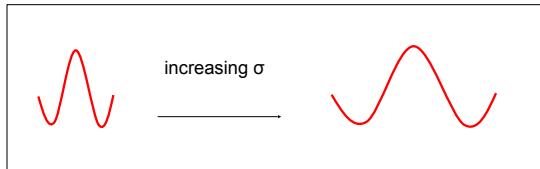


What if the blob is slightly thicker or slimmer?

So what if the blob is slightly thicker or slimmer? What if we don't know the exact size of the blob (which is what happens in practice)? We discuss a solution to this problem on the next couple of slides.

Scale selection

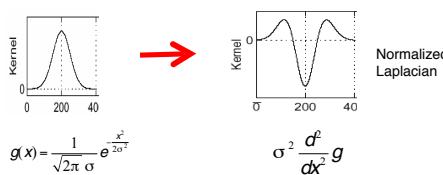
Convolve signal with Laplacians at several sizes and looking for the maximum response



To be able to identify blobs of all different sizes, we can convolve our candidate blob with Laplacians at different scales and only keep the scale where we achieve the maximum response. How do we obtain Laplacians with different scales? By varying the variance σ of the Gaussian kernel. The figure shows that the scale of the Laplacian kernel increases as we increase σ .

Scale normalization

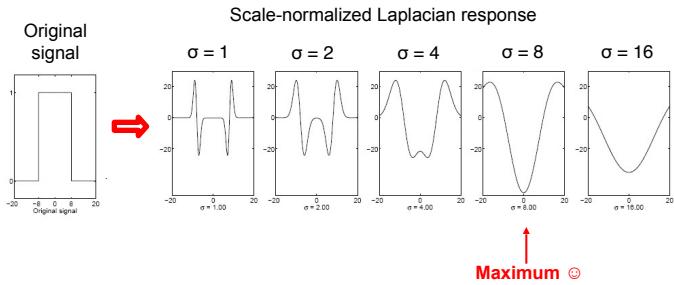
- To keep the energy of the response the same, must multiply Gaussian derivative by σ
- Laplacian is the second Gaussian derivative, so it must be multiplied by σ^2



To be able to compare responses at different scale, we must normalize the energy of the responses to each Gaussian kernel. This means that each Gaussian derivative must be multiplied by sigma and each Laplacian must be multiplied by sigma² so that the responses from different kernels are comparable (calibrated).

Characteristic scale

We define the **characteristic scale** as the scale that produces peak of Laplacian response



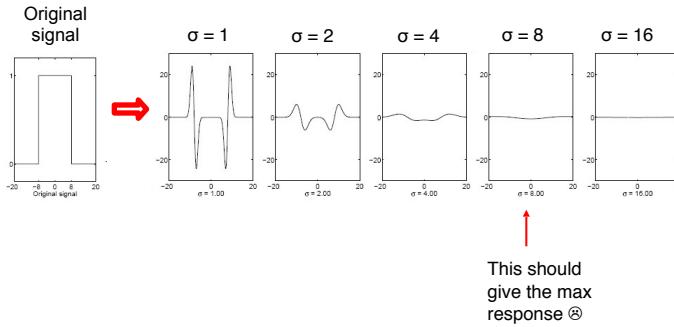
T. Lindeberg (1998). "Feature detection with automatic scale selection." *International Journal of Computer Vision* 30 (2): pp 77–116.

We define the characteristic scale as the scale that produces maximum response at the location of the blob. In this image, we show our original candidate blob signal on the left. On the right, we show its Laplacian response after scale normalization for a series of different sigmas. We see that the maximum response comes at sigma = 8, so we say the characteristic scale is at sigma = 8.

The concept of characteristic scale was introduced by Lindeberg in 1998.

Characteristic scale

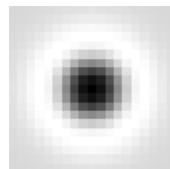
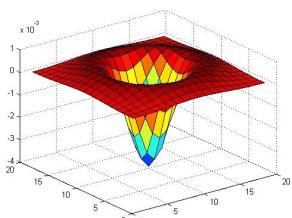
Here is what happens if we don't normalize the Laplacian:



Here we see the same results when the Laplacian kernels we convolve the blob with are not normalized. Notice that, because the energy of the kernel decreases as sigma increases, the response tapers off and the response for sigma=8 is significantly lower than before.

Blob detection in 2D

- Laplacian of Gaussian: Circularly symmetric operator for blob detection in 2D



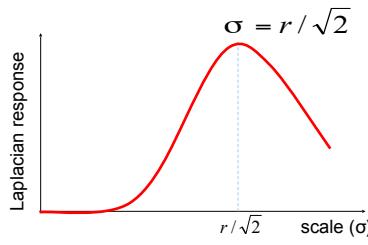
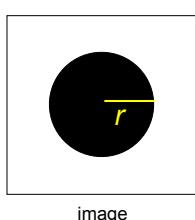
[Eq. 9]

$$\text{Scale-normalized: } \nabla_{\text{norm}}^2 g = \sigma^2 \left(\frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2} \right)$$

In 2D, the same idea applies. Here, we use a 2D Gaussian distribution to design the normalized 2D Laplacian kernel (Eq. 9).

Scale selection

- For a binary circle of radius r , the Laplacian achieves a maximum at

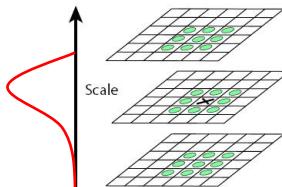


Just like in the 1-d case, we define the scale with the maximum response as the characteristic scale. It is possible to prove that for a binary circle of radius r , the Laplacian achieves a maximum at $\sigma = r/\sqrt{2}$

Scale-space blob detector

- Convolve image with scale-normalized Laplacian at several scales
- Find maxima of squared Laplacian response in scale-space

The maxima indicate that a blob has been detected and what's its intrinsic scale



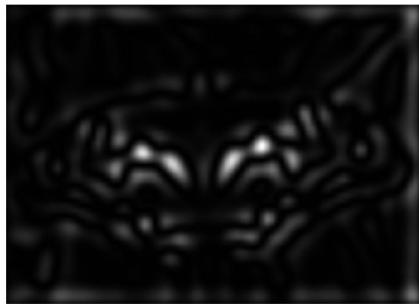
This slide summarizes our process of finding blobs at different scales. First, we convolve the image with scale-normalized Laplacian of Gaussian filters. Next, we find the maximum response to the scale normalized Laplacian. If the max response is above a threshold, the location in the image where the max response appears returns the location of the blob, and the corresponding scale returns the scale of the blob.

Scale-space blob detector: Example



Here's an example of a blob detector run on an image.

Scale-space blob detector: Example



sigma = 11.9912

This is the response after convolving the image with a single Laplacian of Gaussian kernel (with $\sigma = 11.9912$). Different responses can be found if σ is smaller or bigger.

Scale-space blob detector: Example



After convolving with many different sizes of Laplacian of Gaussian kernels, we can threshold the image and see where blobs are detected. This image shows all of the blobs detected over several scales.

Difference of Gaussians (DoG)

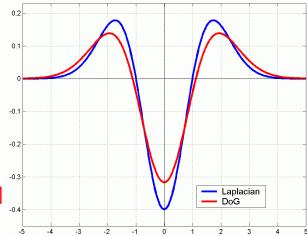
David G. Lowe. "Distinctive image features from scale-invariant keypoints." IJCV 60 (2), 04

- Approximating the Laplacian with a difference of Gaussians:

$$L = \sigma^2 (G_{xx}(x, y, \sigma) + G_{yy}(x, y, \sigma)) \quad (\text{Laplacian}) \quad [\text{Eq. 10}]$$

$$DoG = G(x, y, 2\sigma) - G(x, y, \sigma) \quad [\text{Eq. 11}]$$

Difference of gaussian with scales 2σ and σ



In general:

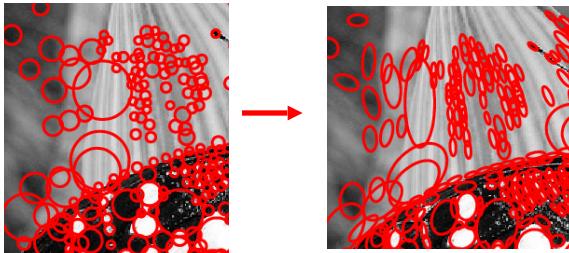
$$DoG = G(x, y, k\sigma) - G(x, y, \sigma) \approx (k-1)\sigma^2 L \quad [\text{Eq. 12}]$$

The scale normalized Laplacian of Gaussian (Eq. 10) works well if one wants to detect blobs, but computing a Laplacian can be very computationally expensive. In practice, we often use the difference of two Gaussian distributions at different variances (e.g. 2σ and σ) to approximate a Laplacian (Eq. 11). Eq. 12 illustrates this approximation for a generic scale k . In the plot, the blue curve is a Laplacian and the red curve is the difference of Gaussian approximation which illustrates that the approximation given in Eq. 12 is fairly close. The difference of Gaussians (DoG) scheme is used in the SIFT feature detector proposed by D Lowe in 2004

Affine invariant detectors

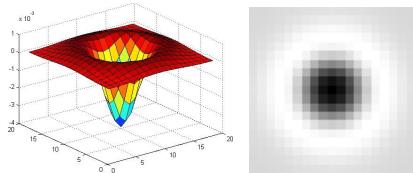
K. Mikolajczyk and C. Schmid, [Scale and Affine invariant interest point detectors](#), IJCV 60(1):63-86, 2004.

Similarly to characteristic scale, we can define the **characteristic shape** of a blob



Properties of detectors

Detector	Illumination	Rotation	Scale	View point
Lowe '99 (DoG)	Yes	Yes	Yes	No



$$\text{Scale-normalized: } \nabla_{\text{norm}}^2 g = \sigma^2 \left(\frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2} \right)$$

We can summarize the feature detectors we have seen so far along with their properties (i.e., robustness to illumination changes, rotation changes, scale changes and view point variations) in a table. For instance, the Lowe's Difference of Gaussian blob detector is invariant to shifts in illumination, rotation, and scale (but not view point).

Properties of detectors

Detector	Illumination	Rotation	Scale	View point
Lowe '99 (DoG)	Yes	Yes	Yes	No
Harris corner	Yes	Yes	No	No
Mikolajczyk & Schmid '01, '02	Yes	Yes	Yes	Yes
Tuytelaars, '00	Yes	Yes	No (Yes '04)	Yes
Kadir & Brady, 01	Yes	Yes	Yes	no
Matas, '02	Yes	Yes	Yes	no

Here we see a more extensive list of feature detectors along with relevant properties.

Lecture 10

Detectors and descriptors



• 18 Properties of detectors

- Speed detectors
 - Harris
 - DoG

• Properties of descriptors

- SIFT
- HOG
- Shape context

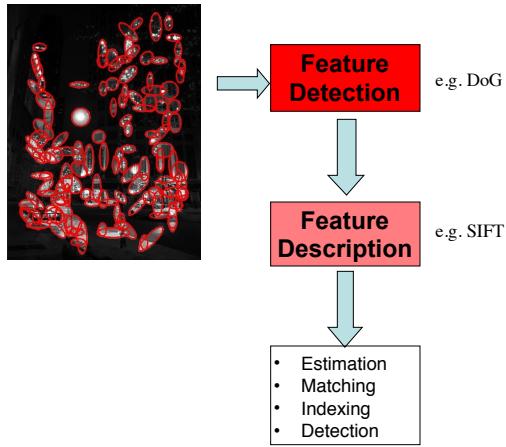
Silvio Savarese

Lecture 10 -

16-Feb-15

Until this point, we have been concerned with finding interesting keypoints in an image. Now, we will describe some of the properties of keypoints and how to describe them with descriptors.

The big picture...



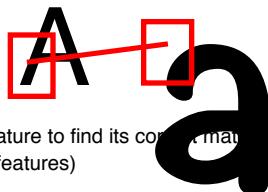
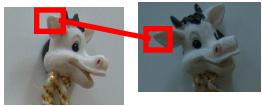
Let's take a step back and look at the big picture. After detecting keypoints in images, we need ways to describe them so that we can compare keypoints across images or use them for object detection or matching.

Properties

Depending on the application a descriptor must incorporate information that is:

• Invariant w.r.t:

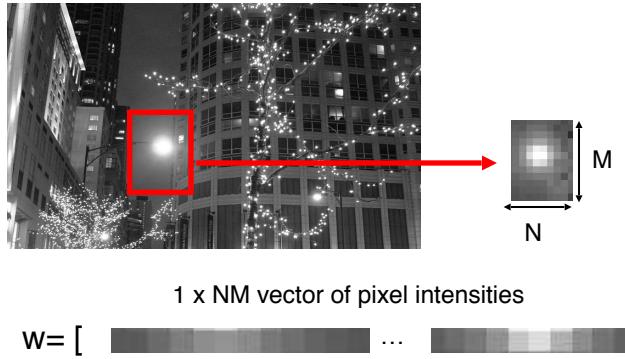
- Illumination
- Pose
- Scale
- Intraclass variability



- Highly distinctive (allows a single feature to find its correct match with good probability in a large database of features)

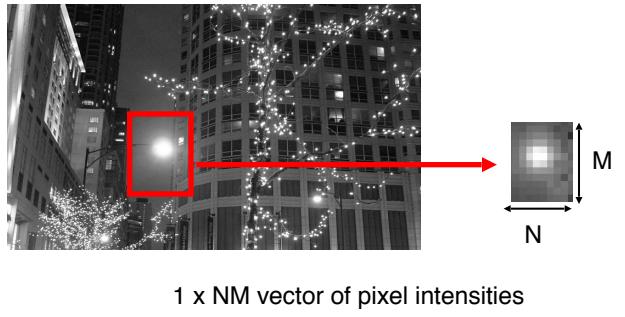
As we did for feature detectors, let's analyze some of the properties we want a feature descriptor to have. These include the ability of the descriptor to be invariant with respect to illumination, pose, scale, and intraclass variability. We also would like our descriptors to be highly distinctive, which allows a single feature to find its correct match with good probability in a large database of features. In the next few slides, we are going to run through some descriptors and discuss how well each descriptor meets these standards.

The simplest descriptor



The simplest, naïve descriptor we could make is just a $1 \times NM$ vector w of pixel intensities that are obtained by considering a small window (say $N \times M$ pixels) around the point of interest.

Normalized vector of intensities



$$w = [w_1 \quad \dots \quad w_{NM}]$$

$$w_n = \frac{(w - \bar{w})}{\|(w - \bar{w})\|} \quad \text{[Eq. 13]}$$

Makes the descriptor invariant with respect to affine transformation of the illumination condition

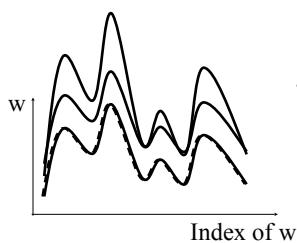
We could also normalize this vector to have zero mean and norm 1 to make it invariant to affine illumination transformations (see Eq. 13).

Illumination normalization

- Affine intensity change:

$$\begin{aligned} w &\rightarrow w + b \\ &\rightarrow a w + b \end{aligned} \quad \text{[Eq. 14]} \quad w_n = \frac{(w - \bar{w})}{\|(w - \bar{w})\|}$$

- Make each patch zero mean: remove b
- Make unit variance: remove a



What does “affine illumination change” mean? It means that the w before and after the illumination change are related by Eq. 14. So the effect of imposing the constraint that the mean of w is zero corresponds to removing b and the effect of imposing the constraint that the variance of w is 1, corresponds to removing a .

Why can't we just use this?

- Sensitive to small variation of:
 - location
 - Pose
 - Scale
 - intra-class variability
- Poorly distinctive

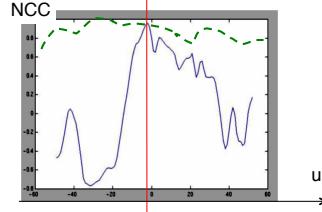
This method is simple, but it has many drawbacks. It is very sensitive to location, pose, scale, and intra-class variability. In addition, it is poorly distinctive. That is, keypoints that are described by this normalized illumination vector may be easily (mis)matched even if they are not actually related.

Sensitive to pose variations



Normalized Correlation:

$$w_n \cdot w'_n = \frac{(w - \bar{w})(w' - \bar{w}')}{\|(w - \bar{w})(w' - \bar{w}')\|}$$



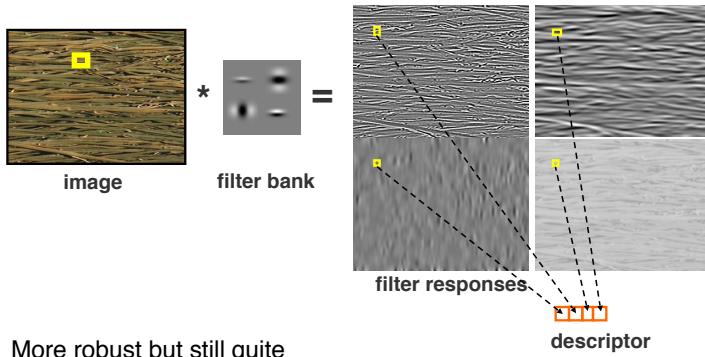
For instance, this slide illustrates an example where a similarity metric based on cross-correlation is sensitive to small pose variations. As already analyzed in lecture 6, the normalized cross-correlation (NCC) (blue plot in the figure) may be used to solve the correspondence problem in rectified stereo pairs – that is, the problem of measuring the similarity between a patch on the left and corresponding patches in the right image for different values of u along the scanline (dashed orange line). The u value that corresponds to the max value of NCC is shown by the red vertical line and indicates the location of the matched corresponding patch (which is correct in this case). As we apply a very minor rotation to the patch on the left, the corresponding NCC as function of u is shown in green (dashed line). Notice that max value of NCC is found at different value of u (which is not correct in this case) and, overall, the green dashed line no longer provides a meaningful measurement of similarity.

Properties of descriptors

Descriptor	Illumination	Pose	Intra-class variab.
PATCH	Good	Poor	Poor

Similar to what we introduced for feature detectors, we can summarize descriptors along with their properties (i.e., robustness to illumination changes, pose variations and intra-class variability) in a table. A descriptor based on normalized pixel intensity values, indicated as **patch**, is robust to illumination variations (when normalized), but not robust against pose and intra-class variation.

Bank of filters



More robust but still quite sensitive to pose variations

<http://people.csail.mit.edu/bill/papers/steerpaper91FreemanAdelson.pdf>
A. Oliva and A. Torralba. Modeling the shape of the scene: a holistic representation of the spatial envelope. IJCV, 2001.

An alternative approach is to use a filter bank to generate a descriptor around the detected key point in the image. The idea is to record the responses to different filters of the filter bank as our descriptor. In the example in the figure, the filter bank comprises 4 "gabor -like" filters (2 horizontal and two vertical). The result of convolving the image (that depicts a horizontal texture in this example) with each of these filters are shown on the right and are denoted as the 4 *filter responses*. The descriptor associated to, say, the key-point shown in yellow in the original image is obtained by concatenating the responses at the same pixel location (also shown in yellow) for each of the 4 filter responses. In this example the descriptor is a 1x4 dimension because we have 4 filters in the filter bank. In general, the dimension of the descriptor is equal to the number of filters in the filter bank. A descriptor based on filter banks can be designed to be computationally efficient and less sensitive to view point transformations than the "patch" descriptor is. This concept led to the GIST descriptor proposed by Oliva and Torralba in 2001

Properties of descriptors

Descriptor	Illumination	Pose	Intra-class variab.
PATCH	Good	Poor	Poor
FILTERS	Good	Medium	Medium

We summarize these results in the table.

SIFT descriptor

David G. Lowe. ["Distinctive image features from scale-invariant keypoints."](#) IJCV 60 (2), 04

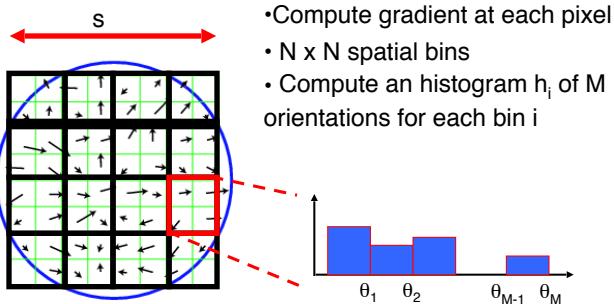
- Alternative representation for image regions
- Location and characteristic scale s given by DoG detector



A very popular descriptor that is widely used in many computer vision applications, from matching to object recognition, is the SIFT descriptor which was proposed by David Lowe in 1999. SIFT is often used to describe the image around a key point that is detected using the Difference of Gaussian detector. Because DoG returns the characteristic scale s of the keypoint, SIFT is typically computed within a window W of size s .

SIFT descriptor

- Alternative representation for image regions
- Location and characteristic scale s given by DoG detector

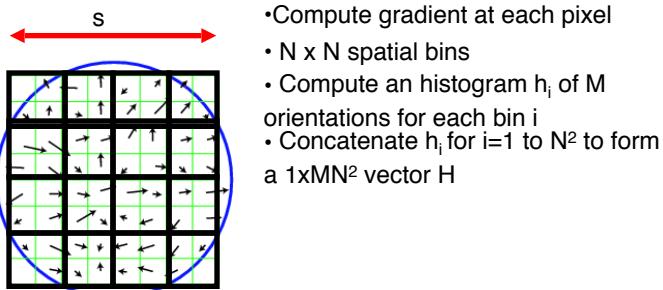


The sift descriptor is computed by following these steps:

1. Compute the gradients at each pixel within the window W
2. Divide the window into $N \times N$ rectangular areas (bins). A bin i is indicated by a black square in the image. In each bin i , we compute the histogram h_i of the orientations of the gradient of each pixel in the bin. That is, within each bin, we keep track of how many gradients are between 0 and θ_1 , θ_1 and θ_2 , etc. This count of orientations within each area forms the basis for the SIFT descriptor vector. Typically $N=8$, which means we are discretizing the orientations from 0 to 45 degrees, from 45 to 90 degrees, etc...

SIFT descriptor

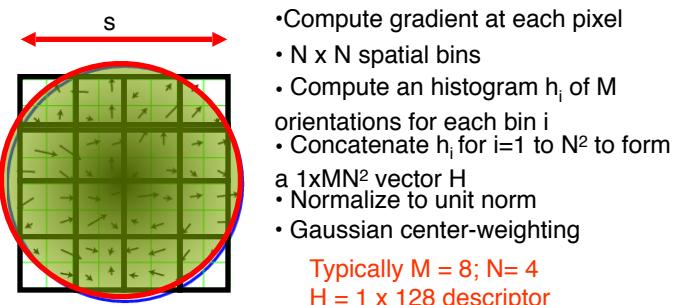
- Alternative representation for image regions
- Location and characteristic scale s given by DoG detector



3. Concatenate h_i for $i=1$ to N^2 to form a $1 \times MN^2$ vector H

SIFT descriptor

- Alternative representation for image regions
- Location and characteristic scale s given by DoG detector



4. Normalize the norm of H – that is divide H by the total number of elements of H , such that the area of the histogram is 1.
5. Reweight each histogram h_i by a gaussian centered at the center of the window W and with a sigma proportional to the size of W .

SIFT descriptor

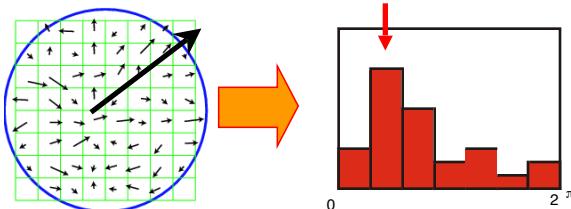
- Robust w.r.t. small variation in:

- Illumination (thanks to gradient & normalization)
- Pose (small affine variation thanks to orientation histogram)
- Scale (scale is fixed by DOG)
- Intra-class variability (small variations thanks to histograms)

The SIFT descriptor is very popular because it is robust to small variations in all of the categories we have mentioned. It is robust to changes in illumination because we are calculating the direction of gradients in normalized blobs. It is invariant to small changes in pose and small intra-class variations because we are binning the direction of gradients (rather than keeping the exact values) in our orientation histograms. It is invariant to scale because we are using scale-normalized DOGs.

Rotational invariance

- Find dominant orientation by building a orientation histogram
- Rotate all orientations by the dominant orientation



This makes the SIFT descriptor rotational invariant

Finally, it is possible to make SIFT rotationally invariant. This is done by finding the dominant gradient orientation (e.g., black arrow in the figure) by computing a histogram of gradient orientations within the entire window – the peak of such histogram (shown by the red arrow) returns the dominant gradient orientation. Once such dominant gradient orientation is computed, we can rotate all the gradients in the window such that the dominant gradient orientation is aligned along a direction that is fixed beforehand (e.g. the vertical direction).

Properties of descriptors

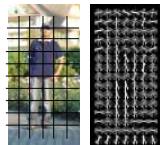
Descriptor	Illumination	Pose	Intra-class variab.
PATCH	Good	Poor	Poor
FILTERS	Good	Medium	Medium
SIFT	Good	Good	Medium

SIFT's properties are summarized and compared to other descriptors in this table

HoG = Histogram of Oriented Gradients

Navneet Dalal and Bill Triggs, Histograms of Oriented Gradients for Human Detection, CVPR05

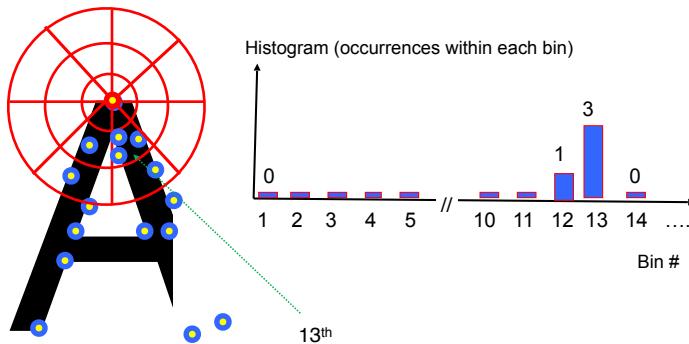
- Like SIFT, but...
 - Sampled on a dense, regular grid around the object
 - Gradients are contrast normalized in overlapping blocks



An extension of the SIFT is the **histogram of oriented gradients** (HOG) which has become commonly used for describing objects in object detection tasks. The gradients of the HOG descriptor are sampled on a dense, regular grid around the object of interest and gradients are contrast normalized in overlapping blocks. For details, please refer to the original paper [Histograms of Oriented Gradients for Human Detection by Dalai and Triggs](#).

Shape context descriptor

Belongie et al. 2002



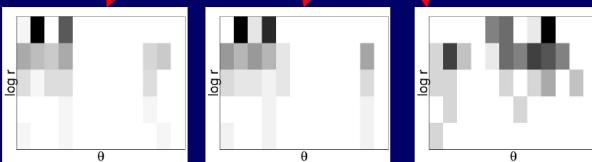
Another popular descriptor is the **shape context** descriptor. It is very popular for optical character recognition (among other things). To use the shape context descriptor, we first use our favorite key-point detector to locate all of the keypoints in an image. Then, for each keypoint, we build circular bins of different radius with the keypoint at the center of the bins. We then divide the circular bins by angle as well as shown in the image. We then count how many keypoints fall within each bin and use this as our descriptor for the keypoint at the center.

The figure shows an example of such an histograms computed over a keypoint on the letter "A". Notice that the 13th bin of the histogram contains 3 keypoints.

Shape context descriptor

Here is an example of the shape context descriptor and relevant representations of descriptors for a few of the keypoints. The image shows 2 letters A (which are similar up to some degree of intraclass variability). Notice that shape context descriptors associated to key points that are located on a similar regions of the letter A do look very similar (descriptor 1 and 2), whereas shape context descriptors associated to key points that are located on a different regions of the letter A do look different (descriptor 1 and 3)

Courtesy of S. Belongie and J. Malik



Other detectors/descriptors

- **HOG: Histogram of oriented gradients**

Dalal & Triggs, 2005

- **SURF: Speeded Up Robust Features**

Herbert Bay, Andreas Ess, Tinne Tuytelaars, Luc Van Gool, "SURF: Speeded Up Robust Features", Computer Vision and Image Understanding (CVIU), Vol. 110, No. 3, pp. 346–359, 2008

- **FAST (corner detector)**

Rosten. Machine Learning for High-speed Corner Detection, 2006.

- **ORB: an efficient alternative to SIFT or SURF**

Ethan Rublee, Vincent Rabaud, Kurt Konolige, Gary R. Bradski: ORB: An efficient alternative to SIFT or SURF. ICCV 2011

- **Fast Retina Key- point (FREAK)**

A. Alahi, R. Ortiz, and P. Vandergheynst. FREAK: Fast Retina Keypoint. In IEEE Conference on Computer Vision and Pattern Recognition, 2012.
CVPR 2012 Open Source Award Winner.

Here is a list of some other popular detector/descriptors. They all have trade-offs in computational complexity, accuracy, and robustness to different kinds of noise. Which one is the best to use is dependent on your application and what is important to your performance.

Next lecture:

Image Classification by Deep Networks by Ronan Collobert (Facebook)