Imperial College London

12 – Pattern & Feature Matching

Prof Peter YK Cheung

Dyson School of Design Engineering

URL: www.ee.ic.ac.uk/pcheung/teaching/DE4_DVS/
E-mail: p.cheung@imperial.ac.uk

The materials in this Lecture can be found partially in Chapter 11 of the recommended textbook: "Digital Image Processing" by Gonzalez and Woods. You can access the electronic version of this book via the following Imperial College library link:

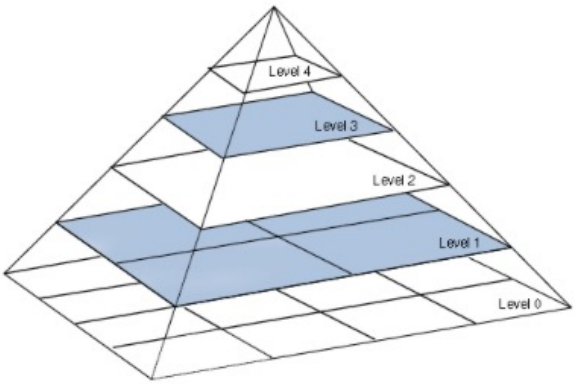
https://imperial.alma.exlibrisgroup.com/leganto/public/44IMP_INST/lists/44618090290001591?auth=SAML

This lecture will discuss three main topics:

1. Image resizing – up-sampling and down-sampling of images;
2. Pattern matching using normalised cross-correlation;
3. Scale Invariant Feature Transform (SIFT) for feature detectors and matching.

Multi-resolution Pyramid

- Important to process image at the **appropriate resolution**.
- Objective: good accuracy with minimal computation.
- Achieved by rescaling the image through **sub-sampling** or **interpolation**.



Images come in all sizes. To get the best results, it is important to perform computation with the appropriate resolution of images.

Using too high an image resolution (i.e. too many pixels) may make algorithms really slow without yield better results. Sometimes one may even make mistakes because too much details are being processed.

Using too low an image resolution may result in the wrong matching or identification due to missed features.

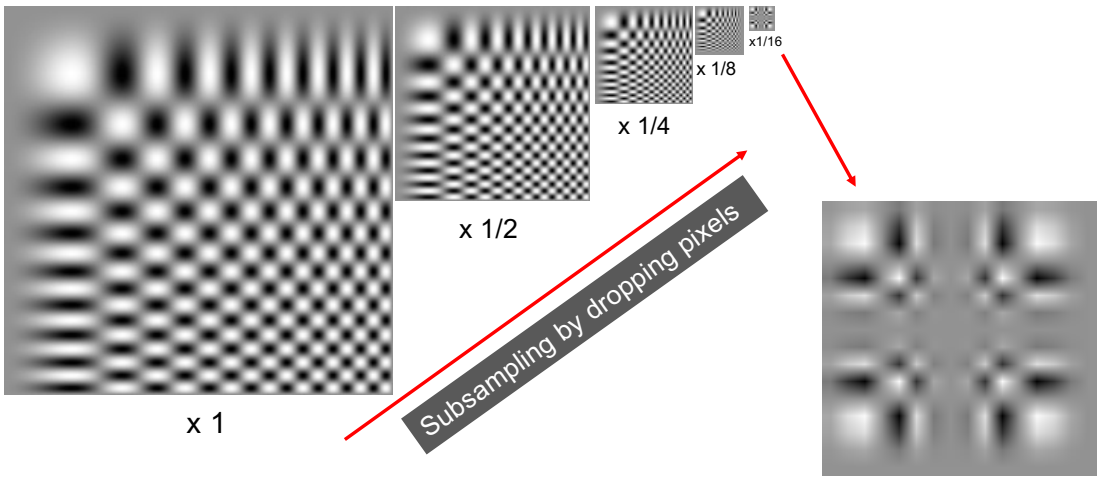
Resizing an image can go either ways: increasing the resolution is called up-sampling; reducing the resolution is called subsampling or down-sampling.

Given an original image, often processing would involve building a stack of the image in different resolutions. This is called an image pyramid.

The large resolution is level 0. Then we reduce the size by ½ each time until we get to a size that makes the image no longer containing useful information.

The question we should ask is: how down-sampling should be done?

Beware of Aliasing



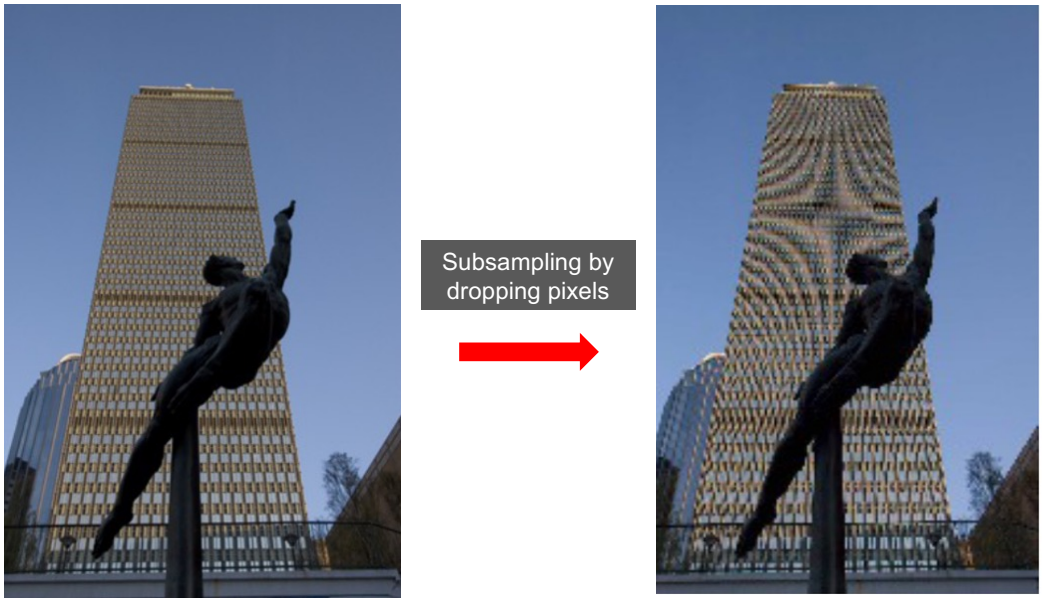
- Image size halved by taking every other pixel in both directions.
- High frequency patten now appears as low frequency.
- This is the result of ALIASING (DE2 Electronics 2).

The obvious approach, but wrong, is to take every other pixel in both x and y directions. However, Nyquist sampling theorem (as covered in DE2 Electronics 2) tells us that doing so can be dangerous.

The slide above shows what happens using this naïve approach. Reducing the sampling rate (i.e. subsampling the image) will reduce the ability to present rapidly changing pattern (or high frequency component in the spatial domain). The checker pattern can no longer be retained and the 1/16 image does not like the original in the sense that the fast changing patterns appear like slowly changing pattern. This effect is known as ALIASING.

Therefore subsampling an image by dropping pixel is in danger of creating artifacts in the image which are not present in the original image.

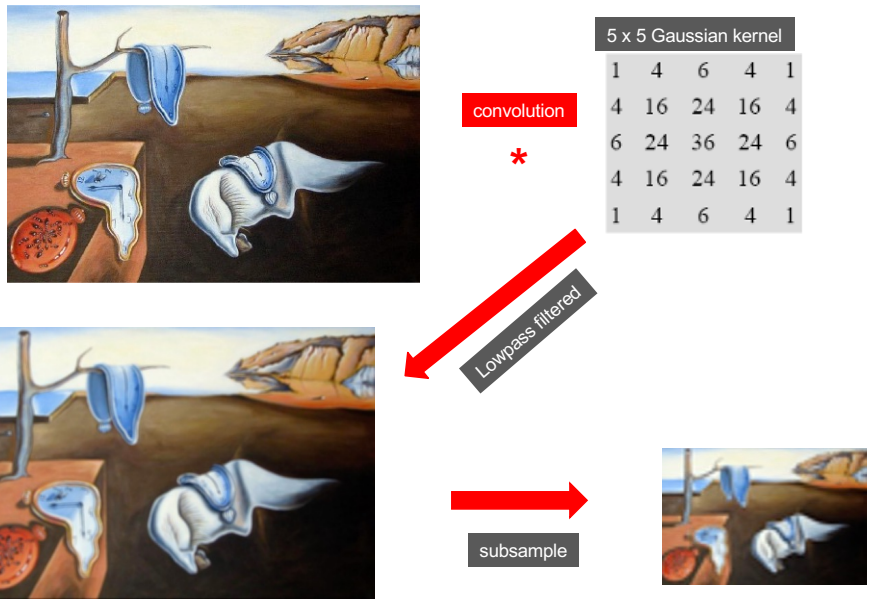
Aliasing creates false pattens



Source: F. Durand

Here is another example. The subsampled image after dropping alternative rows and columns of pixels create an image that has patterns that are NOT in the original.

Right way to Down-sample



To avoid aliasing effect after subsampling, one needs to first limit the maximum frequency contents of the image. Just as we learned in Electronics 2 during the 2nd year, the signal (i.e. image) needs to be filtered to remove high frequency signals.

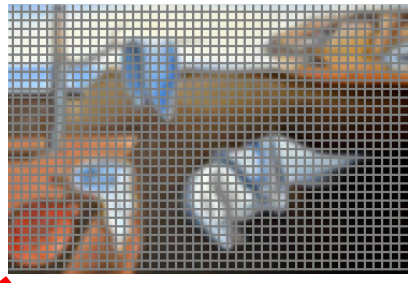
The best way to do so is to convolve the image with a Gaussian filter kernel. This is a lowpass filter which remove high frequency contents.

To choose the correct Gaussian filter, we have to specify the standard deviation σ, which determines the width of the Gaussian curve. The rule of thumb is to use σ = 0.6.

To down-sample by a factor of 2, filter the image with the Gaussian filter kernel and then drop every other row and column. Repeat this at each stage of down-sampling and you can safely build the multi-resolution pyramid for the image.

Right way to Up-sample

Expand image



interpolation

- Nearest neighbour
- Bilinear
- Bicubic

To perform up-sampling, we first expand the image by creating a large image grid and fill them with the original image. There will be pixel locations that are mapped directly to the original small image.

To fill the gaps, one approach is to duplicate the value of the nearest neighbour. This works to a certain degree but the image will be grainy.

A better approach is to determine the value of the missing pixel using interpolation.

Two interpolation methods are commonly used. The best is 'bilinear' interpolation. It is simply linear interpolation in both x and y direction – simply and fast to apply and gives good results. It only makes use of values from the nearest four neighbouring pixels.

Another is to use 'bicubic' interpolation. This uses cubic polynomial in the x and y directions. This requires a lot more neighbouring pixel data and take much longer to calculate.

After interpolation, we have larger image with many more pixels.

Comparison of Right and Wrong way of resizing image

Dropping pixels

Gaussian filter then dropping pxiels

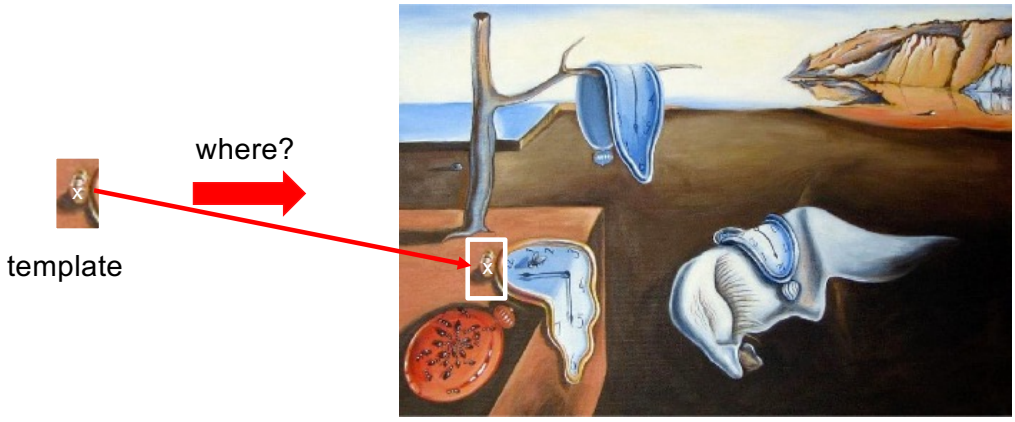
Source: S. Seitz

This slide compares the results when we downsize an image using the wrong way of just dropping pixels, and that of the right of filtering before down-sampling (which is sometimes called decimation).

It is clear that dropping pixel creates artifacts while filtering before down-sampling simply blur the image. Don't forget that the down-sampled image is much ¼ the size of the original. Therefore the blurring effect is not really noticeable.

Template matching Problem

- Given a template (e.g. image of an object), find the location in a large image.
- Usually template is small, image is large.
- Assumption: template is an exact copy of a part of the large image.
- Particularly useful for image alignment (registration).



The goal now is to match a given template to an image. Here is another painting, this time not by van Gogh, but by Salvador Dali. The goal is to match the "watch crown" template to that in the image.

If successful, it will find the crown as highlighted in the rectangle and return the location.

Template matching by Normalized Cross Correlation

- Given an image $f(x,y)$, and a template $t(u,v)$, cross correlation is the same as performing image filter with $t(u,v)$ as the kernel (see Lecture 5, slides 2-7.
- However, here we use normalized cross correlation (NCC) γ, where γ is normalised to the range of +1 to -1.
- The definition of γ is:

$$\gamma(x,y) = \frac{\sum_{x,y}(f(x,y) - \bar{f}_{u,v})\,(t(x-u,y-v) - \bar{t})}{\sqrt{\sum_{x,y}\left(f(x,y) - \bar{f}_{u,v}\right)^2 \sum_{x,y}(t(x-u,y-v) - \bar{t})^2}}$$

- where:

  $\bar{f}_{u,v}$ is the mean value of $f(x,y)$ within the area of the template $t(u,v)$, and $\bar{t}$ is the mean value of the template.
- Using this normalization, $\gamma(x,y)$ is independent of changes in brightness (mean) or contrast (standard deviation) of the image.

A common method to perform such a matching task is to use cross correlation.

We have already studied cross correlation in Lecture 5 when we study the process of filtering an image with a filter kernel. The filtering process can be done either through correlation or convolution.

For template matching, we use the template as the kernel and perform correlation between it and the image at every pixel location to obtain a cross correlation function.

However, the magnitude of the result is very much image and template dependent. A high intensity image will have maximum values that are much larger than a low intensity image.

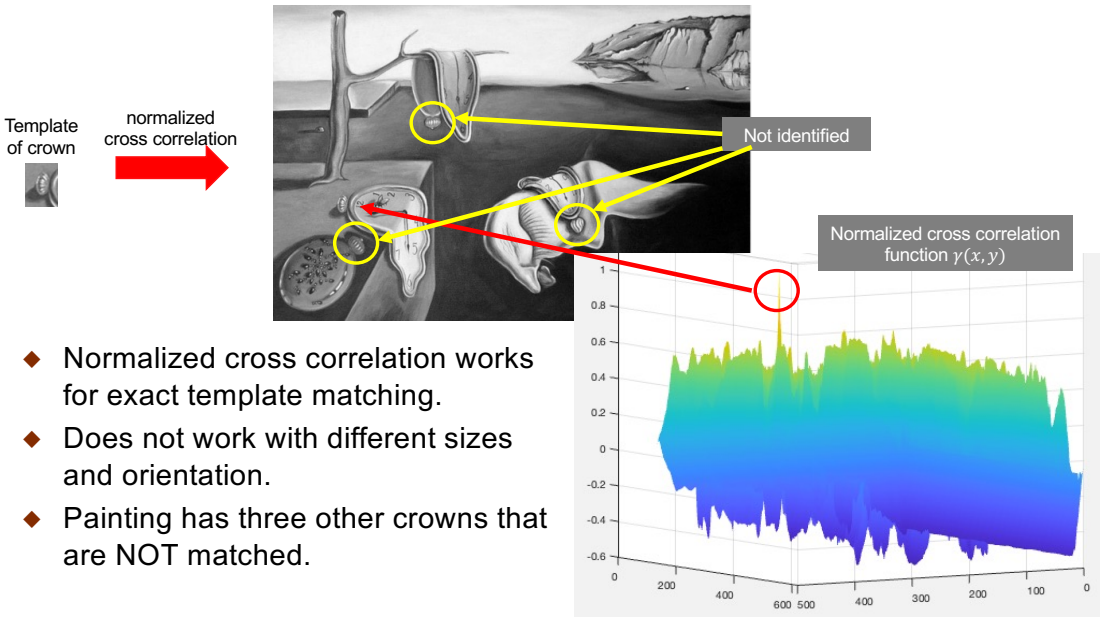
The solution to this problem is to normalize the results to a range of -1 to +1. This is achieved using the formulae above.

Essentially we remove the mean value of both the image patch (within the kernel window) and the template, before performing the multiplication and add.

Finally, we scale the results to the product of variance between the image and the template.

As can be seen, if the template is an exact match to the image (within the template window), the result of this calculation is +1.

The Dali painting example



- Normalized cross correlation works for exact template matching.
- Does not work with different sizes and orientation.
- Painting has three other crowns that are NOT matched.

This is an example of matching the "watch crown" template to the Dali painting using normalized cross correlation as explained in the previous slide.

The resulting function γ is plot here. The peak of the function (with value of +1 indicating an exact match) provide the location of the template in the image.

Note that this method requires the template being exactly the same in size and orientation as that found in the image. There are three more "watch crowns" in this painting as highlighted. These are not found using the NCC method because they are not an exact match to the template.

General feature matching problem

Where?

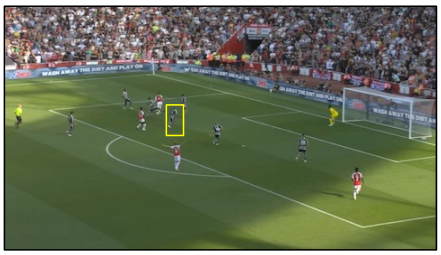
- Problems in matching objects in general:
  1. Different size (or scale)
  2. Different orientation
  3. Different brightness and contrast
  4. Partially covered (occlusion)

So, we need a better method to do matching. One approach is not just working with pixels, but extract features from a collection of pixels in an image. Then perform matching of the feature based on its characteristics.

We want a method that can match features that are different in size, orientation, brightness, and even when it is partially covered.

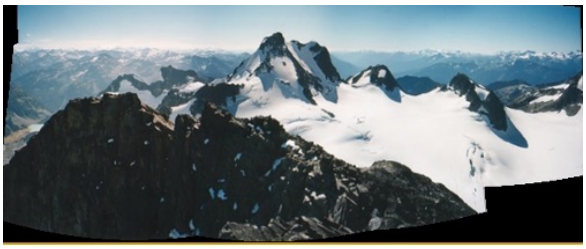
The remaining slides in this lecture is to explain a ground-breaking technique introduced by Lowe in 2004, known as the Scale Invariant Feature Transform (SIFT).

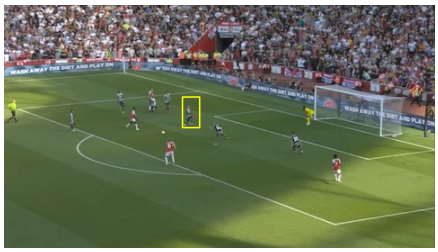
Harder Visual Processing



Stitching

Tracking





Here are two examples of where SIFT can be used to solve harder visual processing tasks.

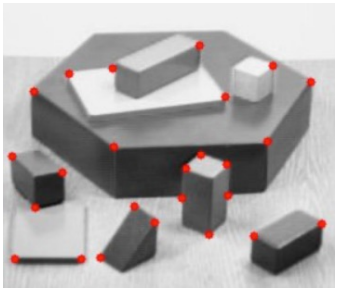
The first is to take a collate of images of the same scene and stitch them together automatically to form a panoramic photo. This is fundamentally an image registration problem. If we can find the same features in different small photos, we can align them together by stitching.

Another problem is that of feature tracking for video sequences. Here are two frames from a football game video recording. SIFT can be used to track the highlighted player from one frame to the next.

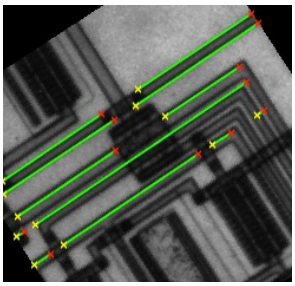
What are interesting features?

- To handle a scene, need to identify and locate interesting features.
- These could be:
  1. Points, particularly corners
  2. Lines or shapes (e.g. circles)
  3. Blobs or patches

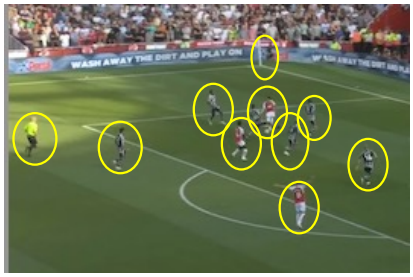
Corners



Line and edges



Blobs



Before we consider how to identify features, let us examine the different types of features that are useful to detect.

We have already study the detection of points and corners. We have also learned about line detection using Hough Transforms.

There is one last type of interesting features – that of a “blob”. It turns out that in most images, the most interesting features are not lines or edges, but blobs – a collection of pixel around a certain pixel location the includes interesting characteristics.

The right most image here shows that the players in a football games are the interesting items. The centres of the blobs are the locations, and the yellow shapes that enclose the blobs gives their sizes. Finally, each blob will have a principal orientation, although they are all pointing up in this case (because the players are all standing).

SIFT – A Blob Feature Detection Method

- **SIFT** stands for **Scale Invariant Feature Transform.**
- Useful for image alignment (registration), object tracking and 2D object recognition.
- Proposed by Lowe in 2004 to identify interesting blob features that are independent of their size, orientation and intensity (paper on webpage).
- Output from SIFT detector are these properties of features:
  1. **Locations** of the blobs
  2. **Scales** (or sizes) of the blobs
  3. **Orientations** of the blobs
  4. **Signatures** or descriptors for the blobs

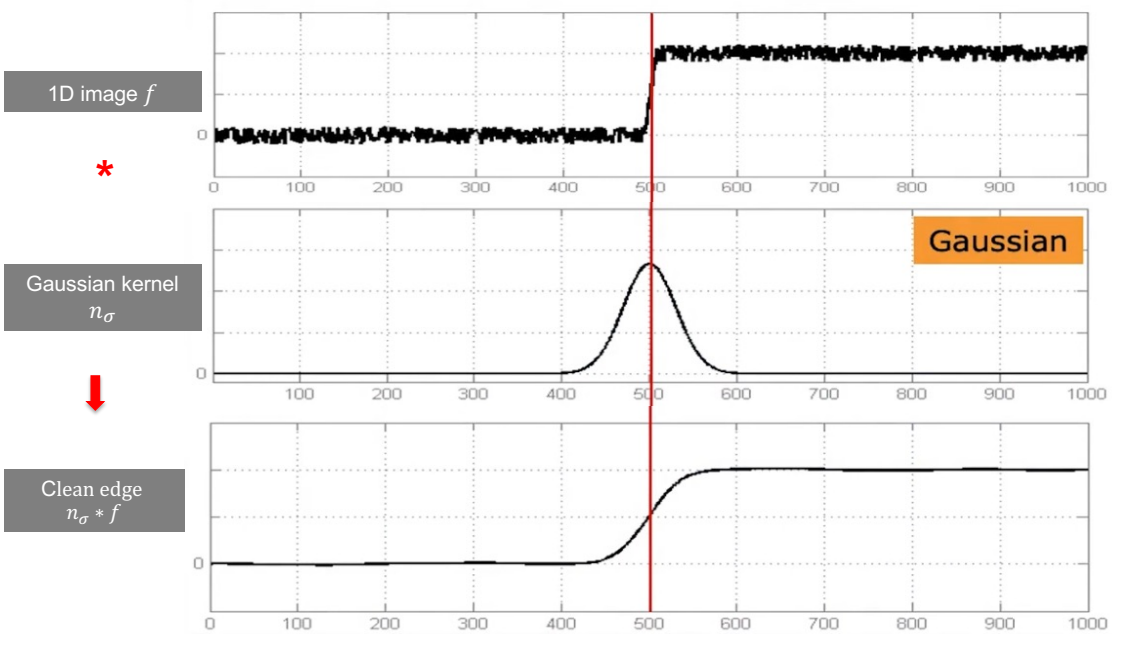
Descriptor or signature

SIFT is probably the most significant method in detecting "blobs" in an image.

The output results from the SIFT detector is a list (or array) of features called interest points, each has four useful pieces of information:

1. Location – the coordinate of the centre of a blob gives the location of that blob.
2. Scale – as shown in the image above, there are many packs of sweets, and they are of different sizes.
3. Orientations – different packets are oriented in different directions.
4. Signature (or Descriptor) – to identify different blobs, SIFT provides each a "unique" signature. So if two blobs are the same, but at different locations and of different size and rotation, SIFT will give them the same signature. If they appear differently, then SIFT will give them different signatures.

Recap on Gaussian Filter – noise removal

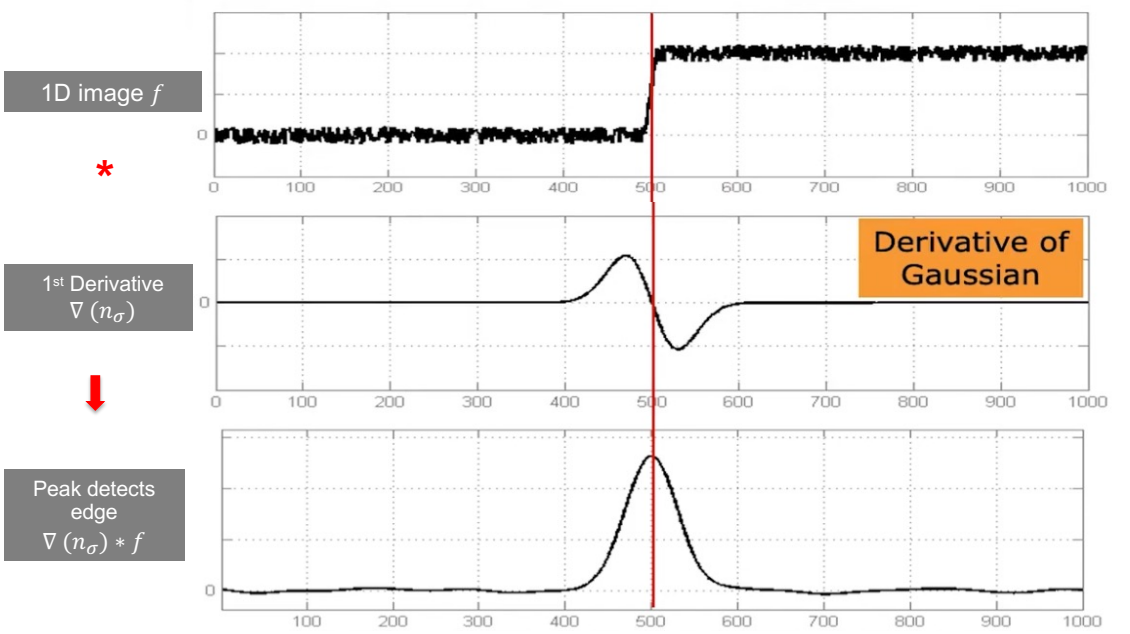


Before we explain how SIFT achieve scale invariant (i.e. insensitive to size), let us consider the Gaussian filter because SIFT relies heavily on the Gaussian function.

To simplify the explanation, let us consider a 1D image (such as the pixel along a line of an image). Show here is an edge corrupted by noise.

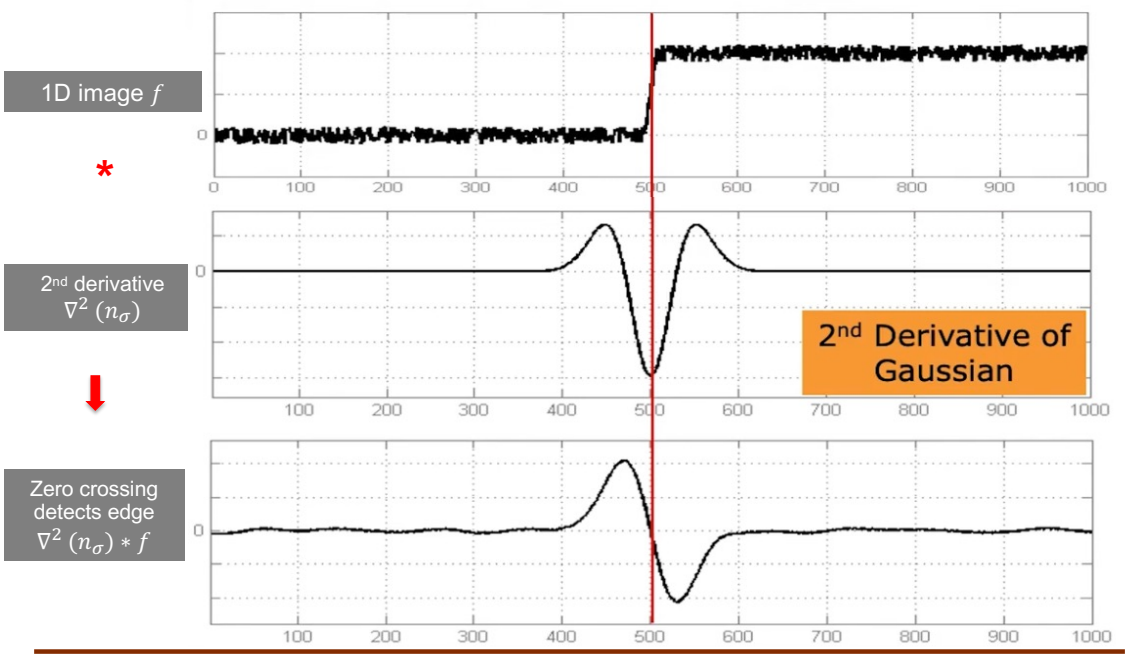
Applying a Gaussian filter through convoluting the image $f(x)$ with a Gaussian function n_σ. The end result is a blurred edge but with noise significantly removed.

1st Derivative of Gaussian – Edge detection



If we apply a kernel which is the 1st derivative of the Gaussian function, we have an output where the peak is aligned with the location of the edge as shown.

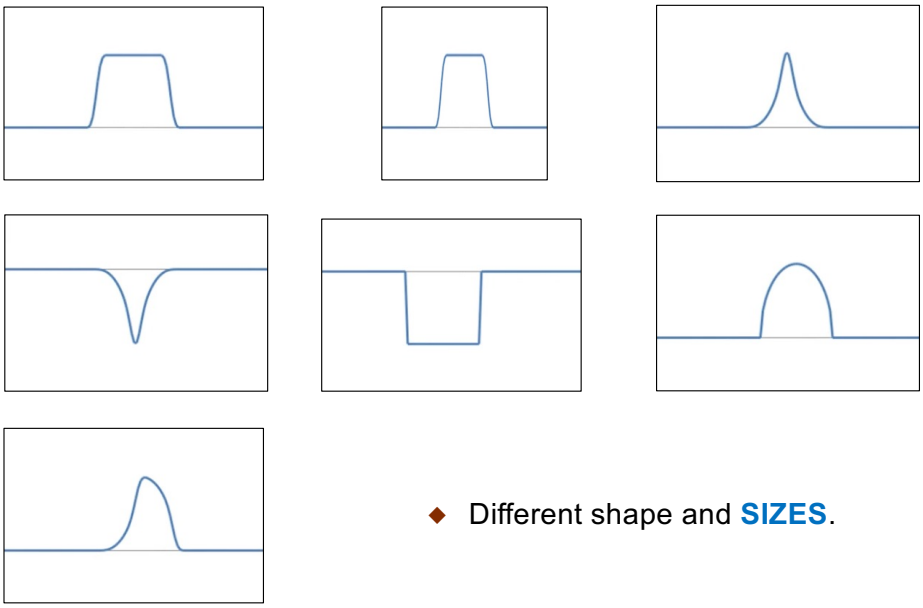
2nd Derivative of Gaussian – Edge detection



Finally, if we apply the 2nd derivative of the Gaussian as the kernel to filter the image, the result is shown at the bottom plot. Here the **zero-cross** of the output is aligned with the edge of the 1D image.

Remember these three slides because they become very useful later.

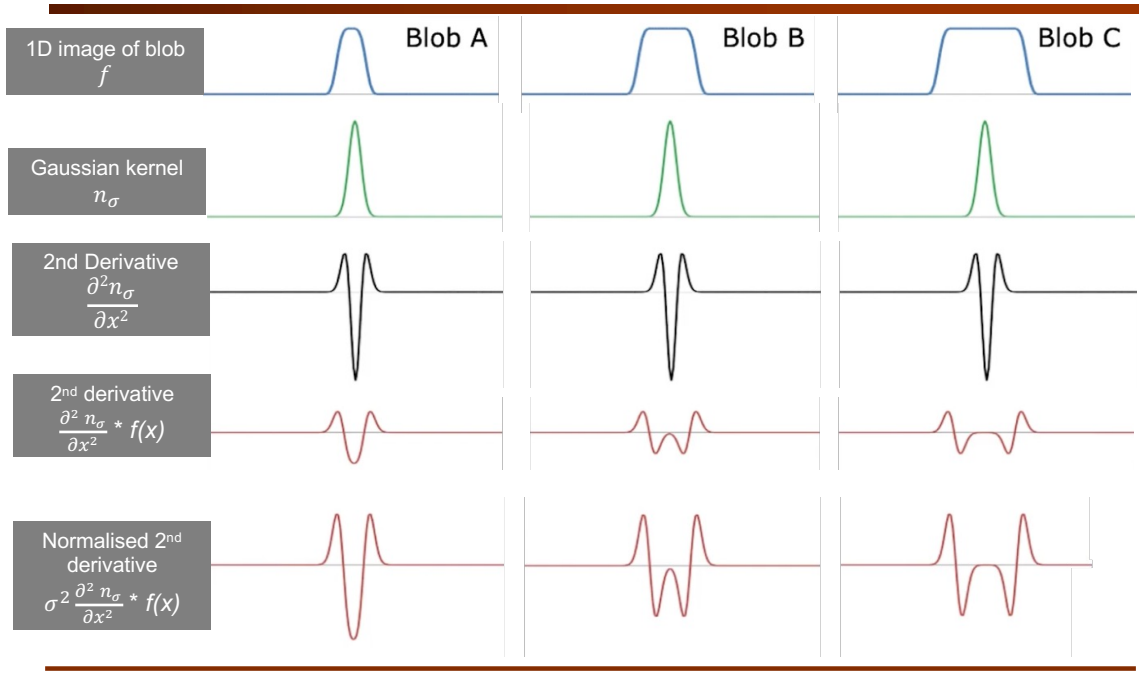
Different types of Blobs in 1D



- Different shape and **SIZES**.

Our goal is to detect blobs. However, there are many type and shape of blobs as shown here. Most importantly, they are of different sizes. 1D case, size is equivalent to the width of the “blob”.

Normalized 2nd Derivative of Gaussian



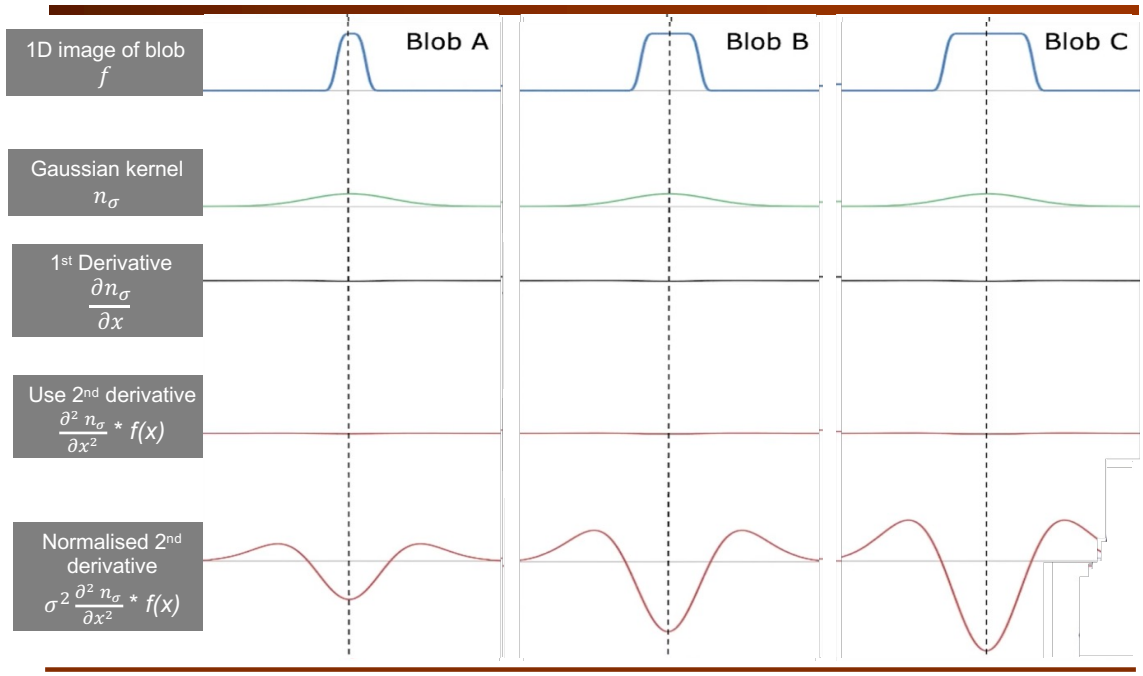
Let us consider the three blobs here: A is a narrow blob, B is a wider blob, and C is the widest of the three. We will focus on the use of 2nd derivatives to detect the blobs. That is, we shall focus on the waveforms on the fourth row in the slide.

If we apply the 2nd derivative of the Gaussian to the blob, we found that they have different zero-crossing as shown – some wider and some narrower. More importantly the negative peak of the output after filtering with the 2nd derivative is of different height (or magnitude).

If we use a Gaussian kernel with smaller σ (a narrow Gaussian), the magnitude is larger. If we widen the Gaussian kernel with a larger σ, the magnitude is smaller.

To make the magnitude not dependent of σ, we can **normalize** the 2nd derivative of the Gaussian by multiplying the kernel with σ^2. This is known as the ***σ*-normalized 2nd derivative of the Gaussian**.

Effect of changing σ on Normalized 2nd Derivatives



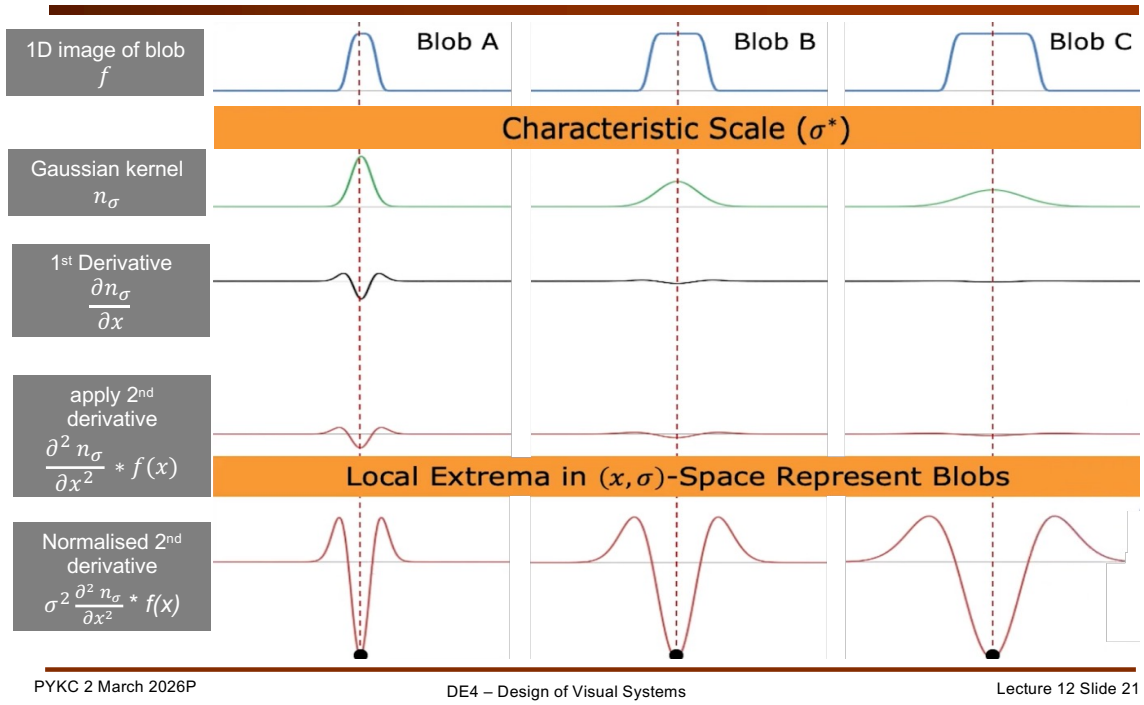
Let us now apply the normalized 2nd derivative to the three sizes of blobs.

This slides is animated using normalized 2nd derivative of a Gaussian kernel having increasing value of σ. It is difficult to see what happens without the animation. (Therefore, the slide is broken up into many frames on the PDF version.)

Basically, with a given σ value, the largest negative peak of bottom waveform happens for blob A. When we double σ, blob B yields the largest negative peak. Finally, the widest σ results in blob C producing the largest peak. (Sometime, negative peak is also called an extrema, which is either positive or negative.)

In other words, if we apply the normalized 2nd derivatives to every location of the image and vary the σ values, we can find which value of σ will give the largest extrema. This means that we can successfully deduce the size of the blob because it is related to the value of σ.

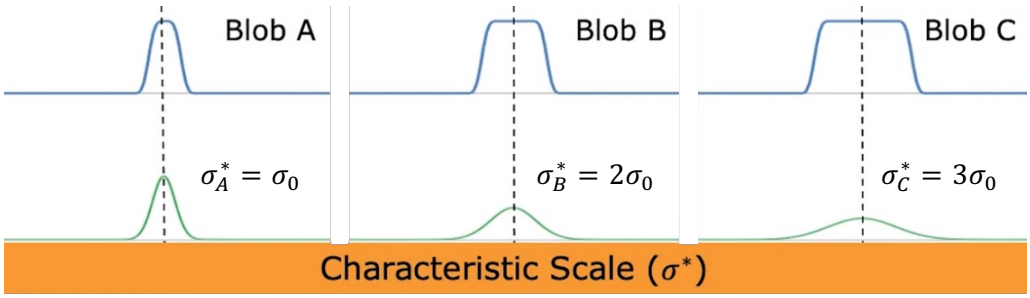
Characteristic Scale of Blobs



To recap, as we change the value of σ, we keep looking for the local extrema (negative peak) in the normalized 2nd derivative of the image.

The values of σ that gives the maximum negative normalized 2nd derivative, we denote as σ^*, provides a measure of the size of possible blobs. This therefore characterizes the size of potential "blobs". This collection of σ^* values is known as the **Characteristic Scale**.

Characteristic Scale Measures Size of a Blob



- Characteristic Scale: The σ value at which σ-normalised 2nd derivative reaches its peak value.
- Characteristic Scale is a valid measure of the SIZE of the blob. That is:

$$\frac{size\ of\ blob\ A}{size\ of\ blob\ B} \approx \frac{\sigma_A^*}{\sigma_B^*}$$

This slides shows the characteristic scale for the three blobs as σ_A^*, σ_B^* and σ_C^*.

Since the characteristic scale σ_A^* is a measure of the size of the blob, we can make an approximation in assuming that the ratio of blobs is approximately equal to the ratio of their characteristic scale value.

Instead of using a continuous varying value of σ, which is obviously not possible, we use a base value of σ, denoted as σ_0. We then increase σ by multiply it with a scaling factor s or s^2 or s^3 and so on. That is:

$$\sigma_k^* = s^k\ \sigma_0, \text{ where } k = 1, 2, 3, \ldots..$$

Summary on Steps of Blob Detection in 1D

1. Given a 1D signal $f(x)$, convolve it with σ-normalized 2nd derivative function:

   Compute: $\sigma^2 \frac{\partial^2 n_\sigma}{\partial x^2} * f(x)$ at different scales $(\sigma_0, \sigma_1, \ldots, \sigma_k)$.

2. Find $(x^*, \sigma^*) = \max_{(x,\sigma)} \left| \sigma^2 \frac{\partial^2 n_\sigma}{\partial x^2} * f(x) \right|$
3. Blob position = x^*
4. Blob size = σ^*

Let us recap how we detect the location and size of a 1D blob.

We first convolve the 1D signal $f(x)$ with a set of normalized 2nd derivative with different σ scales, $(\sigma_0, \sigma_1, \ldots, \sigma_k)$.

The location of the blobs is given by the local extrema (or negative peaks) of the output of that convolution.

The local extrema at different σ scales are compared. If there exists an extremum whose peak value is significantly larger than others at different values of σ, it indicates that there is a candidate for a blob. In this way, we can detect the potential existence of a blob.

The location of the extremum, x^*, is the centre of the blob.

The characteristic scale σ^* is the size of the blob.

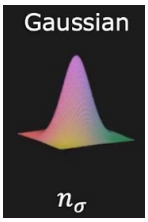
If all local extrema are more or less the same, it indicates that there is no interesting feature there.

Blob Detection in 2D

- For 2D image $I(x, y)$, use **Normalized Laplacian of Gaussian** (NLoG) for blob detection:

Laplacian

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$



- Location of blobs found by **Local Extrema** after applying **NLoG at many scales**.

Now let us extend the 1D case to 2D.

The 2nd derivative of a 2D image is also called the Laplacian as shown in the slide.

The 2D Gaussian function is:

$$n_\sigma(x, y) = Ke^{\frac{x^2+y^2}{2\sigma^2}}$$

The shape of this function is as shown in the slide.

The Laplacian of the Gaussian (LoG) (Lecture 8 slide 9) is:

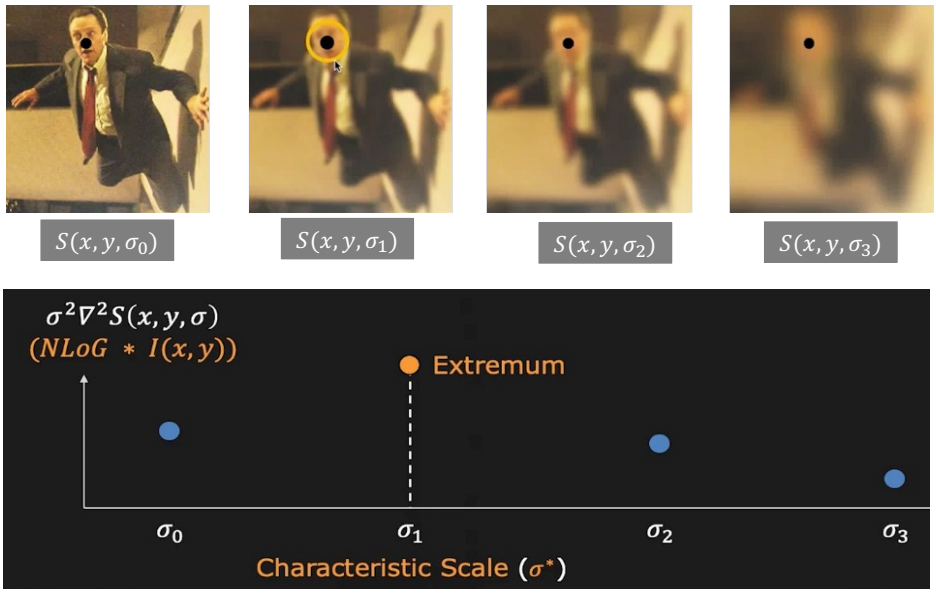
$$\nabla^2 n_\sigma(x, y) = \left(\frac{x^2 + y^2 - 2\sigma^2}{\sigma^4}\right) e^{-\frac{x^2+y^2}{2\sigma^2}}$$

and its shape is as shown which is often called the inverted Mexican hat.

Again, just like the 1D case, if we use the LoG as a kernel, the range of values of the output is dependent on the value of σ. Therefore, we use the normalized form of LoG by multiplying the LoG function with σ^2 to give us the NLoG, which is the same shape as LoG, but scaled appropriately.

The location of blobs can now be found by convolving our 2D image with the NLoG kernel.

Example of Detecting an Interesting Blob



Source: Lindeberg

Here is an example of using NLoG to detect the location AND scale of potential blobs.

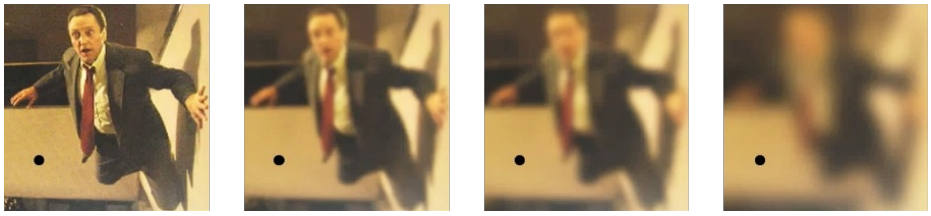
The image $I(x_i, y_i)$ is a man falling. The current location of interest is that shown with a black dot – his nose.

Convolving this part of the image with the NLoG kernel at different scale space $(\sigma_0, \sigma_1, ..)$ produces the output vs σ plot as shown. It is clear that there is an extremum at σ_1.

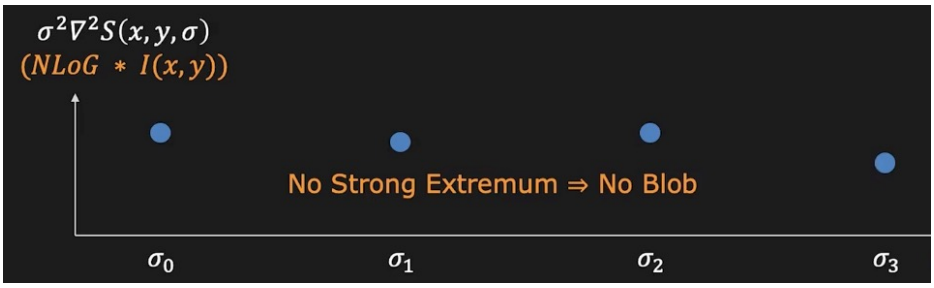
This indicates two things:

1) there is indeed a potential interesting blob at this location $\mathrm{x}^* = (x_i, y_i)$;
2) the characteristic scale σ^* , and hence the size, of this blob is σ_1.

Example of Detecting a non-blob



$S(x, y, \sigma_0)$ $S(x, y, \sigma_1)$ $S(x, y, \sigma_2)$ $S(x, y, \sigma_3)$



Source: Lindeberg

Let us now consider the new location at the black dot which is on the uniform background. Here the plot of $NLog * I(x, y)$ against the different values of σ is very flat. Therefore, it indicates that there is no blob present at this location.

Summary on Steps of Blob Detection in 2D

1. Given an image $I(x, y)$, convolve it with $NLoG$ at many scales of σ .

   Compute: $(\sigma^2 \nabla^2 n_\sigma) * I(x, y)$ at different scale $(\sigma_0, \sigma_1, \ldots, \sigma_k)$.

2. Find $(x^*, y^*, \sigma^*) = \max_{(x,y,\sigma)} \left|(\sigma^2 \nabla^2 n_\sigma) * I(x, y)\right|$
3. Blob position = (x^*, y^*)
4. Blob size = σ^*

This slides summarizes how blobs in an imaged can be detected, located and sized using NloG kernels of different σ values.

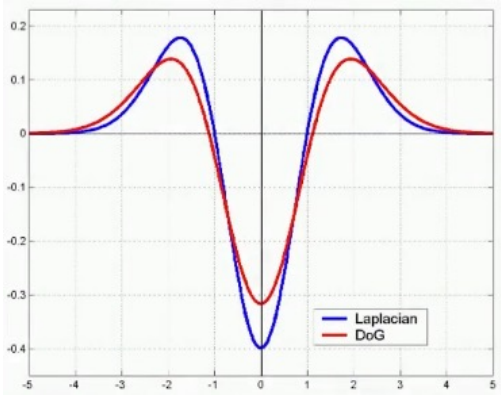
DoG is fast approximation of NLoG

- Is there a faster way to compute NLoG?
- Difference of Gaussian (DoG):

$$DoG = (n_{s\sigma} - n_{\sigma}) \approx (s-1)\sigma^2\nabla^2 n\_\sigma$$

NLoG

- s is different multipliers (octave) of σ.



$$DoG \approx (s-1)NLoG \qquad s > 1$$

Now before we start considering the SIFT detector, we need to learn about a good approximation to the NLoG function.

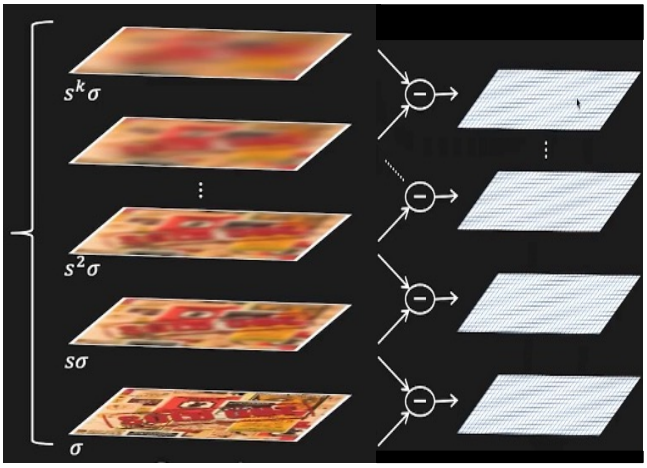
If we plot the function NLoG in 1D, which is the Laplacian of the Gaussian, we found the curve in blue above.

However, if we plot the **Difference of Gaussian** (DoG) which is defined as the difference between two Gaussian functions, one with the standard deviation σ, and another with $s\sigma$, we have the red curve. As can be seen, the two are very similar to each other.

In other words, instead of calculating the normalized Laplacian of the Gaussian (NLoG), we can approximate it by simply taking the difference of the Gaussians (DoG). This therefore bypass both the Laplacian operation and the normalization calculation – much simpler to do!

Extracting SIFT Interest Points (1)

Image
$I(x, y)$



Gaussian Scale Space
$S(x, y, \sigma)$

Difference of Gaussians (DoG)
$\approx (s-1)\sigma^2\nabla^2 S(x, y, \sigma)$

Source: Lowe

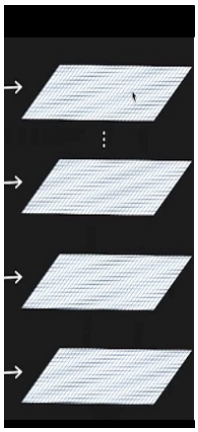
Now we are ready to explain how SIFT extracts interesting points (centres of blobs) from an image. Remember, convolving an image with a Laplacian kernel is the same as convolving the image with a Gaussian kernel then take the Laplacian because both convolution and differentiation operations are linear and the order of the operations does not matter.

Given an image $I(x, y)$, we first produce a pyramid of images, each filtered with a Gaussian filter with increasing values of σ: $(\sigma, s\sigma, s^2\sigma, \ldots, s^k\sigma)$. This gives us the Gaussian Scale Space $S(x, y, \sigma)$ which a stack of Gaussina filtered images.

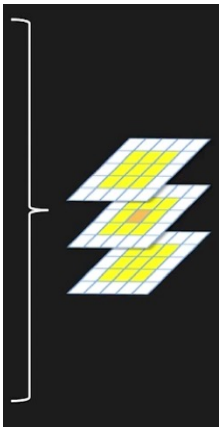
We now calculate the Difference of Gaussians (DoG) between successive layers, as shown in the slide, as an approximation of the NLoG.

We now have a stack of DoG "images" which contains information relating to potentially interesting blobs!

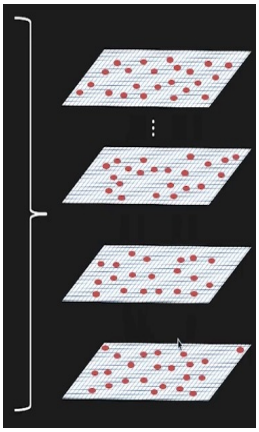
Extracting SIFT Interest Points (2)



Difference of Gaussians (DoG)
$\approx (s-1)\sigma^2\nabla^2 S(x,y,\sigma)$



Find peaks (extrema) in every 3x3 grid



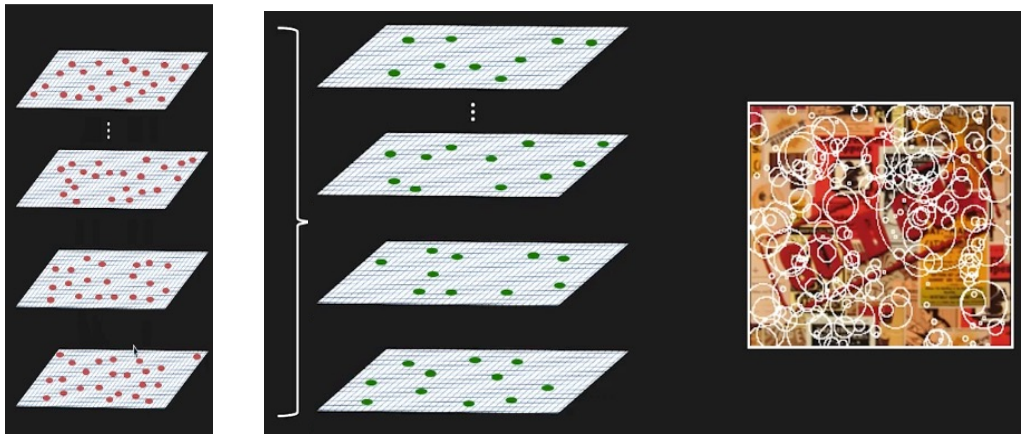
Candidates for Interest Point

Source: Lowe

From the DoG pyramid (of increasing value of σ), we can now search for extrema at each location ACROSS the stack within a 3x3 grid. These extrema provide potential interesting points, i.e. centre of potential blobs.

This gives us a stack of candidates for interesting points (where blobs could be located) for each scale.

Extracting SIFT Interest Points (3)



Candidates for Interest Point

SIFT Interest Points after removing weak peaks

Annotated SIFT Features

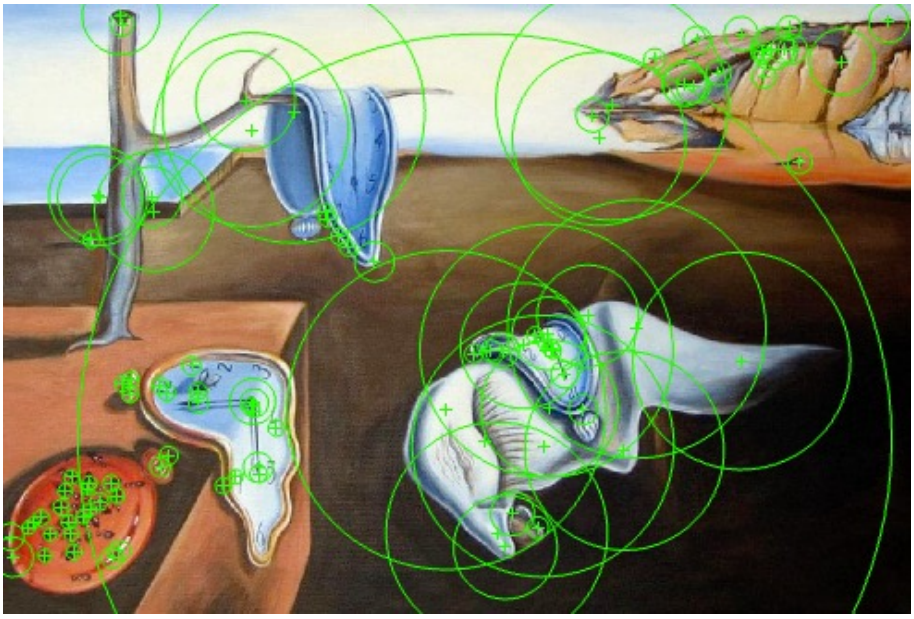
Source: Lowe

Let us examine the pyramid stack on the left. Each layer represents potential candidates of blobs and interesting points AT THAT SCALE of σ. There can be many potential candidates in each layer.

The next step is to eliminate "weak" interesting points. This could be done by thresholding at each layer. Then merge these together as shown in the final image overlay with these blob features. The centre of each white circle gives us the location of the blob. The radius of the circle gives us the size of that blob.

We have now successfully identified interesting features (blobs) in an image, knowing both their locations and sizes. Next, we need to find their orientations.

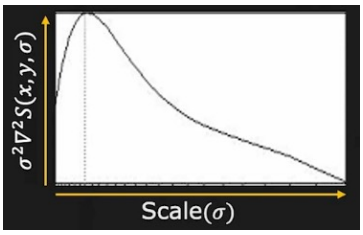
Example of SIFT Interest Points Detector

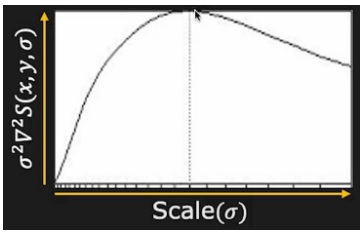


Here is the result of applying the SIFT detector to the Salvador painting. In applying the SIFT feature detector, there are several parameters that a user can specify to affect the results:

1. σ_0 - the width of the Gaussian for the base layer of the scale space. The different layer will have the sigma value increased as $s^k\sigma_0$, where $k = 1,2,3$ ….
2. s – the scaling factor between the successive layers of the scale space pyramid.
3. N – the maximum number of most ”interesting” features. SIFT detectortends to produce a very large number of potential interesting points (or blobs). For each blob, the extremum value provides a metric to indicate how “interesting” that blob is. After rank ordering the entire list with descending order of the metric, one can choose the most significant N features to process and discard the rest.

SIFT Scale Invariance





Ratio of Blob Sizes

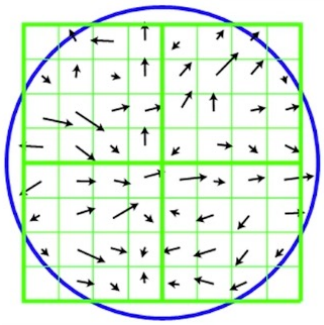
$$= \frac{\sigma_1^*}{\sigma_2^*}$$

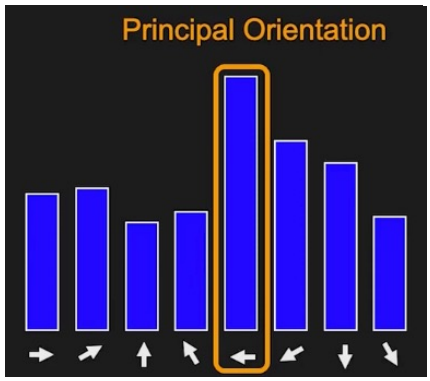
Source: Mikolajczyk

Here is an example of SIFT being scale invariant.

A blob is located at the centre of the flame for both images dicated by the black dot. The two images will have different scale for this flame blob as shown. With the knowledge of the scale σ_1^* and σ_2^*, they can be rescaled to be the same size.

Detect Feature Orientation





- Image gradient directions is calculated by:

$$\theta = \tan^{-1}\frac{\partial I/\partial y}{\partial I/\partial x}$$

- Build histogram of direction for every pixel (8 directions).
- Principle Orientation is the one with highest count.

We now have features that we know are potentially interesting. We know their locations and their sizes. Now we will find the blob's orientation.

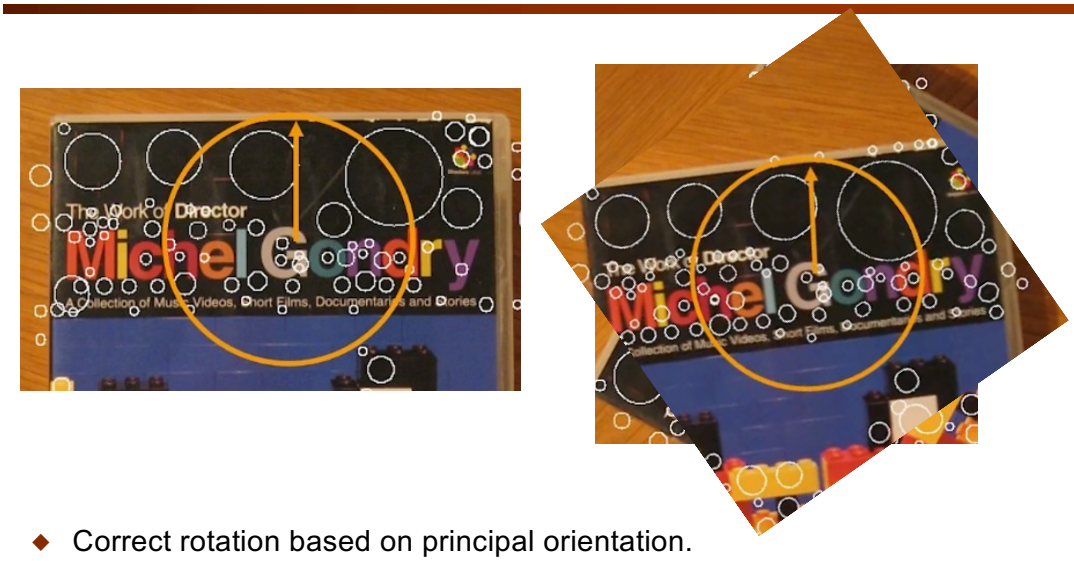
Shown here is a blob at some location of the image. The interesting point is the centre of the blue circle which has a bounding square shown in the green grid.

For each pixel within this square, we can compute the gradient in x and y direction (i.e. the 1st derivatives). The orientation of each pixel is given by the arc-tangent of the ratio of $\partial I/\partial y$ and $\partial I/\partial x$.

This angle is then quantized into eight orientations: pointing up, down, left, right, and four more for the two diagonals.

We can now build a histogram of orientations within this patch (blob). From the histogram, we deduce that the principal orientation of this feature being the angle that has the highest count value in the histogram.

SIFT Rotation Invariance

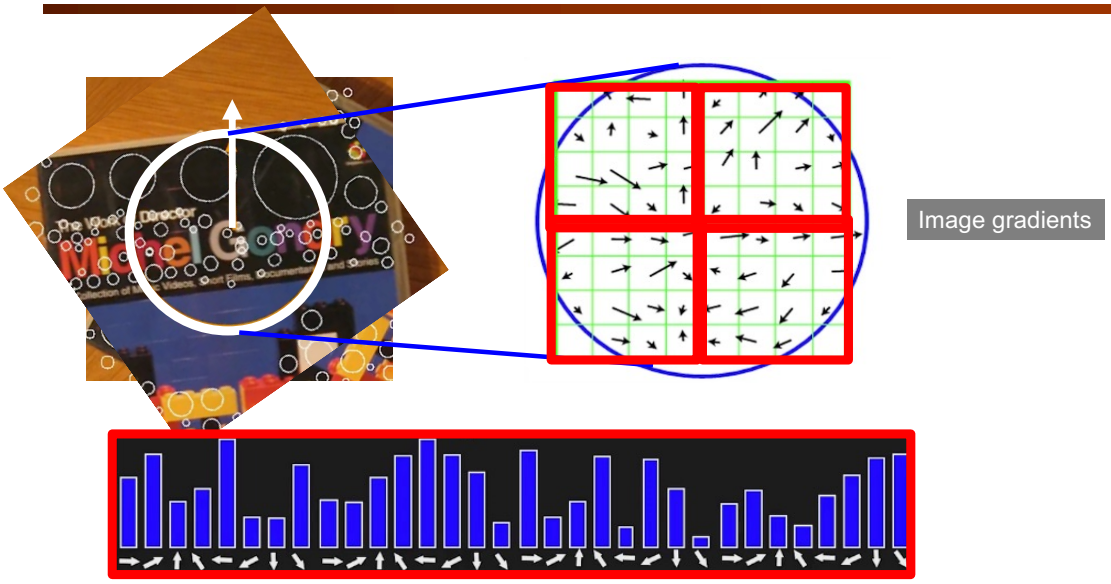


- Correct rotation based on principal orientation.
- We can now match objects of different scale and different orientation.

This is an example of applying the SIFT orientation method. The left image has a feature centred at the letter 'G'. The right image has the same image rotated.

After determine the orientation of the feature in the right image, we can apply the affine transform (Lab 1) to re-orient the image so that the feature is now also pointing up.

SIFT Descriptor (signature)



- Normalized Histogram is the featured descriptor or signature.
- It is invariant to Rotation, Scaling and Brightness.

The final step of the SIFT detector is to somehow give find a signature to the blob. This is called the feature's **DESCRIPTOR**.

In SIFT, we ignore the pixel colour or intensity because we want the feature detector to be invariant to lighting and contrast. Instead, we only use the histogram of orientations.

The procedure involve dividing the orientations of pixels within the blob (in a square grid) into four quadrants. For each quadrant, we compute the histogram of orientations for the quadrant's pixels. We then concatenate these four sub-histograms together to form one histogram with 8x4 = 32 bins. Now the profile of these bins forms the signature or descriptor for this blob.

Matching of SIFT Descriptors

- Goal, match two SIFT features from two images, with descriptors $H_1(k)$ and $H_2(k)$. (These are histograms of orientations.)
- Possible measures:
  1. L2 Distance:

$$d(H_1, H_2) = \sqrt{\sum_k \left(H_1(k) - H_2(k)\right)^2}.$$ (Smaller d = better match.)

  2. Normalized Correlation:

$$d(H_1, H_2) = \frac{\sum_k [(H_1(k) - \overline{H_1})(H_2(k) - \overline{H_2})]}{\sqrt{\sum_k (H_1(k) - H_1)^2} \sqrt{\sum_k (H_2(k) - H_2)^2}},$$

  where $\overline{H_i} = \frac{1}{N} \sum_{k-1}^{N} H_i(k)$. (Larger d = better match.)

  3. **Intersection:**

$$d(H_1, H_2) = \sum_k \min(H_1(k), H_2(k))$$

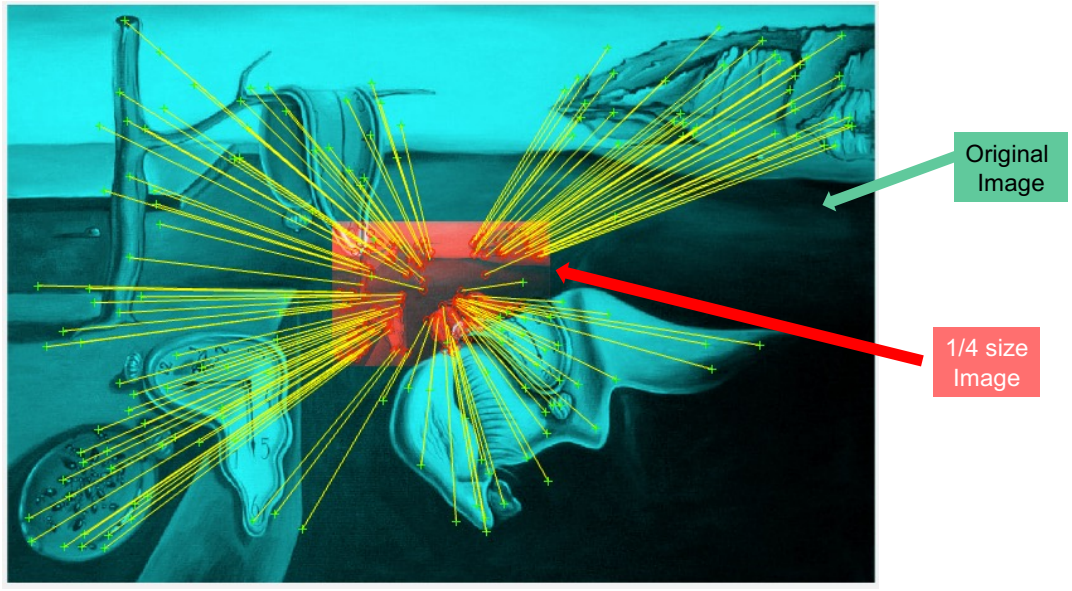
  (Larger d = better match.)

Once we have the descriptors of features, we can match them between two images.

How? We can quantify how similar is a feature in one image to that of another feature in a second image. This can be achieved using one of three popular measures of "distance".

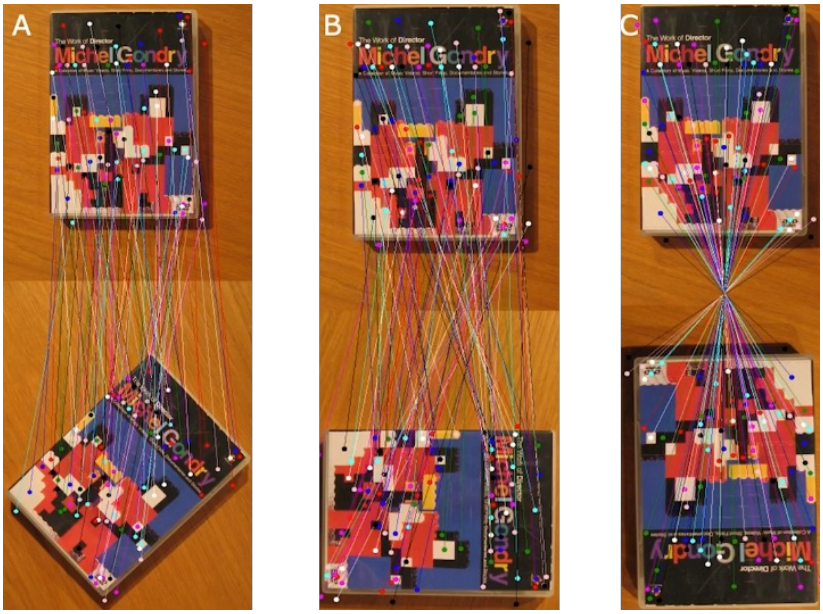
1. **L2 Distance** – this is common Euclidean distance calculation between the histogram H_1 to another feature descriptor (histogram) H_2. The shorter the distance, the better the match. Exact match is indicated by *d = 0.*
2. **Normalized cross correlation** – we have done NCC before. This is now applied to the two histograms for the two features. The larger the distance, the better the back. Since this is normalized, *d = 1* indicates a perfect match.
3. **Intersection** – this computes the bin-to-bin overlaps between the two histogram. The large the distance, the higher the overlap of the histogram and better the match.

SIFT Results: Scale Invariance



This is the results of matching the Dali painting at the original scale to that ¼ the size. As you can see the features are successfully detected and the matching is very good. It demonstrates that SIFT is indeed scale invariant, and the descriptors for features fromt the two images were matched very well.

SIFT Results: Rotation Invariance



Here are three set of images with different orientations.

All feature points are successfully identified and matched.

Photo Stitching using SIFT (1)

This is another example of SIFT successfully identifying the feature points in two separate photos of the same mountain scene.

Photo Stitching using SIFT (2)

These two images can then be stitched together to form a panoramic photo.

SIFT for tracking

Finally, this is a sequence of frames in a video where the green person is successfully tracked from frame to frame. Further, the other individuals are all tracked showing that the SIFT algorithm is useful in tactking multiple features (blobs) in images simultaneously.

Matlab Support for Feature Detection

- Many other feature detection methods have been proposed since Lowe's paper in 2004.
- Here are the different algorithms that are implemented in Matlab, some of these are extensions to SIFT.

Detector	Feature Type	Function	Scale Independent
FAST [1]	Corner	detectFASTFeatures	No
Minimum eigenvalue algorithm [4]	Corner	detectMinEigenFeatu	No
Corner detector [3]	Corner	detectHarrisFeatures	No
SIFT [14]	Blob	detectSIFTFeatures	Yes
SURF [11]	Blob	detectSURFFeatures	Yes
KAZE [12]	Blob	detectKAZEFeatures	Yes
BRISK [6]	Corner	detectBRISKFeatures	Yes
MSER [8]	Region with uniform intensity	detectMSERFeatures	Yes
ORB [13]	Corner	detectORBFeatures	No

After Lowe published his seminal paper in 2004, there have been many modified methods building on Lowe's idea of identifying features in scale, orientation and intensity independent manner, producing the signature of features and then perform matching of these features.

There is not sufficient time to cover any of these improved algorithm developed from the SIFT detector since its introduction. The slide here shows the variety of feature detection methods implemented in Matlab which can be used for different applications. Of particular importance in addition to SIFT is the SURF method.

By now you should realize that in processing visual information, there is not a single killer algorithm or method that works for ANY images. Instead, depending of the nature of the image, one would need to deploy tools from a large collections of methods and tricks, depending on the goals that the designer want to achieve.

Here are a number of Matlab documentation pages that you may find useful:

- https://uk.mathworks.com/help/vision/ug/local-feature-detection-and-extraction.html
- https://uk.mathworks.com/help/vision/ug/feature-based-panoramic-image-stitching.html
- https://uk.mathworks.com/help/vision/ug/point-feature-types.html