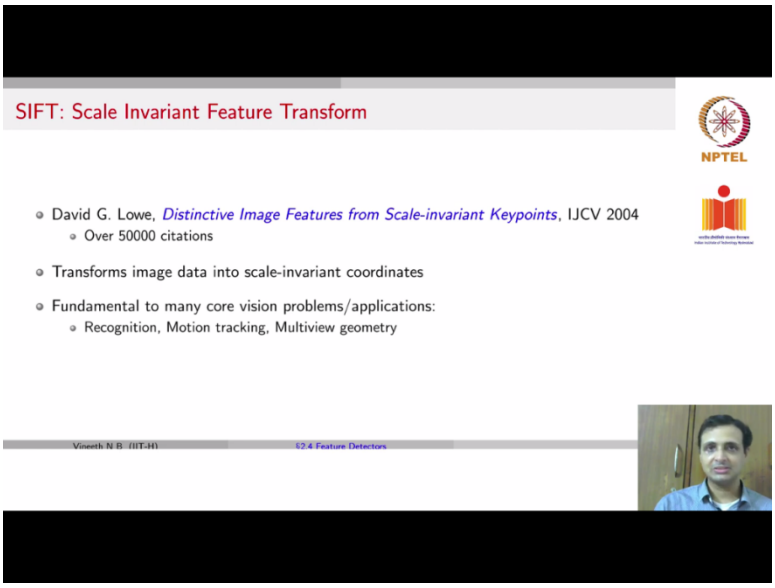
Deep Learning for Computer Vision
Prof. Vineeth N Balasubramanian
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
Lecture 11
Feature Detectors: SIFT and Variants

Last lecture, we saw the filter banks, image pyramids and also a scale invariant Harris corner detector. We will now move on to one of the most significant feature detectors that have been developed in computer vision known as the scale invariant feature transform or SIFT.

(Refer Slide Time: 0:45)



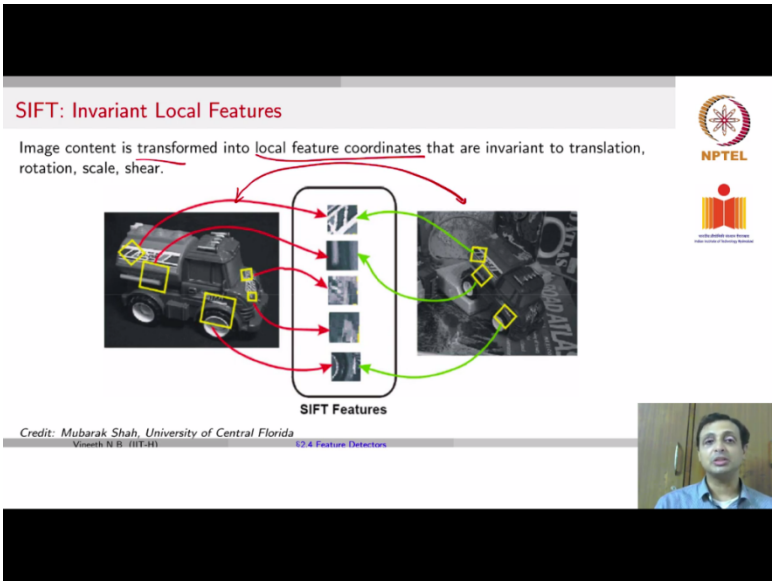
SIFT was developed by a person David Lowe from University of British Columbia. It was developed way back in the late nineties, but by the time it was formerly published, it was close to 2004. To this date, it has over, I think, 56,000 or 57,000 citations. That speaks of the impact that it has had on the community over the last decade and a half.

The main objective of SIFT is to transform image data into scale invariant key point coordinates. So once again, very similar to Harris corner detector. Our goal is to extract key points from images, but we are going to go one step further now. In addition to detecting those key points in the images, we also want to find an effective way of describing those key points. A simple way of describing key points that we have seen so far is X comma Y, which is the coordinate location of that key point. Maybe you can add a scale just such as a sigma that we saw in the previous lecture.

But now with SIFT, you are going to talk about a full-fledged feature descriptive, that describes the local characteristics around that particular key point. SIFT is fundamental to many core vision problems and applications, including recognition, motion tracking, multi-view geometry, so on and so forth.

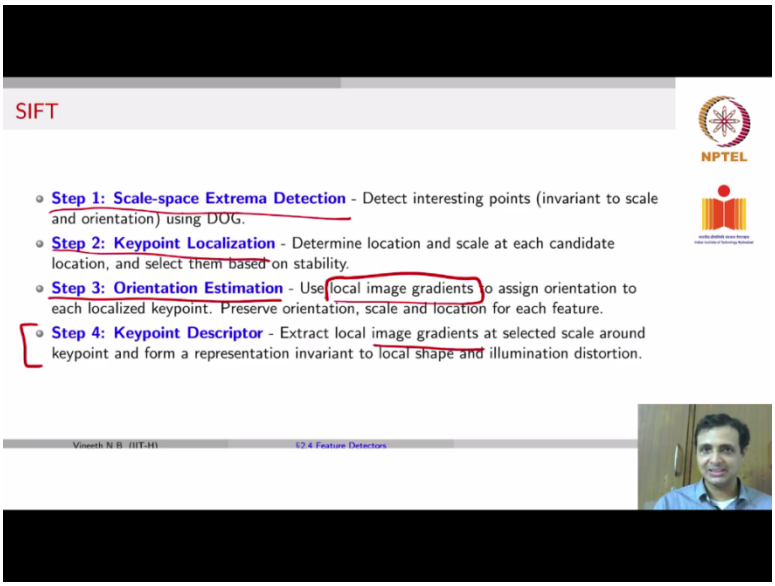
For many years, until deep learning became popular, SIFT was extensively used for simple object recognition in images.

(Refer Slide Time: 02:30)



As I just mentioned, in SIFT, image content is transformed to local feature coordinates, that are ideally invariant to translation, rotation, scale and shear. So for example, once again, two images of the same object taken from different angles, different poses, perhaps different illuminations. But we ideally want to ensure that the same key points are detected in both images irrespective of variations in translation, irrespective of variations in rotation, irrespective of changes in scale, as well as shear, shear is a perspective transformation.

(Refer Slide Time: 03:15)



SIFT consists of four steps and we will go through each of them individually. But before we cover them, I should mention that SIFT is an excellent engineering effort. Each step has a proper motivation as to why it was developed and what part of the objective did it help fulfil. I will highly advise you to read the SIFT paper, just to understand how a beautiful engineering paper in computer vision has been written.

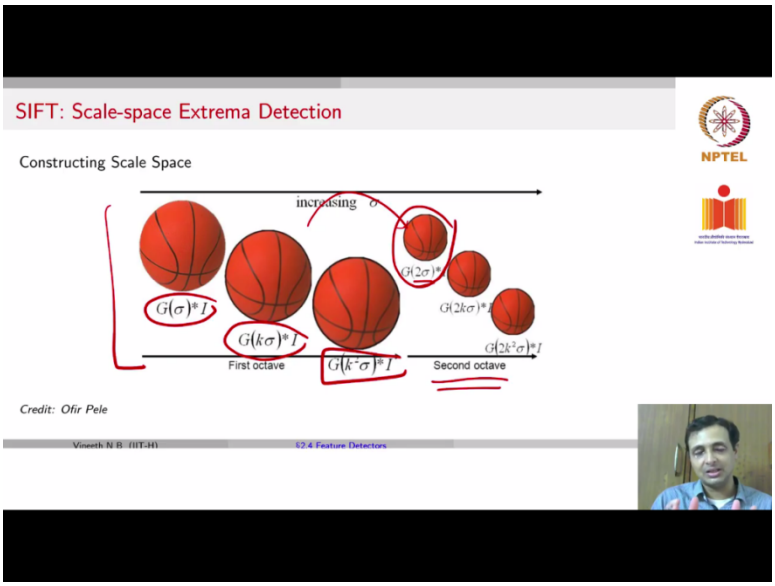
The first step in SIFT is to do what is known as scale space extrema detection. This is very similar to the scale invariant Harris Corner detector. Although there is a little bit more than that, which I will clarify in some time.

The second step is about key point localization. Once you have found a key point, can we try to find out exactly where is that key point? I mean, is it maybe half a pixel, a little further, so on and so forth, could be in location, could be in scale too.

Third step is to determine the orientation of that key point. There is a reason why we are going to try to determine the orientation of that key point. We are going to try to use the local image gradients around that key point to assign the orientation. And we ideally want to preserve the orientation scale and location for each feature.

Finally, as I mentioned, we are going to talk about how do you describe those key points in terms of local information about that key point. Once again, we are going to use some information from the gradients, but we want to develop a representation for that key point, which is invariant to a lot of transformations. Why do we need a representation? Remember that, we said that if you want to match two images and stitch them together, we need to compare key points in first image to key points in the second image, having a representation allows you to compare. Otherwise you would not be able to compare, just coordinate locations because they might be very different in the two images.

(Refer Slide Time: 05:33)



So the first step, as I just said, is scale space extrema detection. And the way SIFT implements this is to firstly construct what is known as a scale space. A scale space as shown in the diagram here is simply taking an image, convolving it with Gaussian, that is one image. Then convolving it with K times sigma. So you have another Gaussian with K times sigma, that is another image, but the size of the image stays the same. It might only be more blurry. Then, you have a convolution with a Gaussian with K square sigma. Once again, the size of the image remains the same. We are not sub-sampling at this time. Simply making it more blurred depending on what you choose to be done.

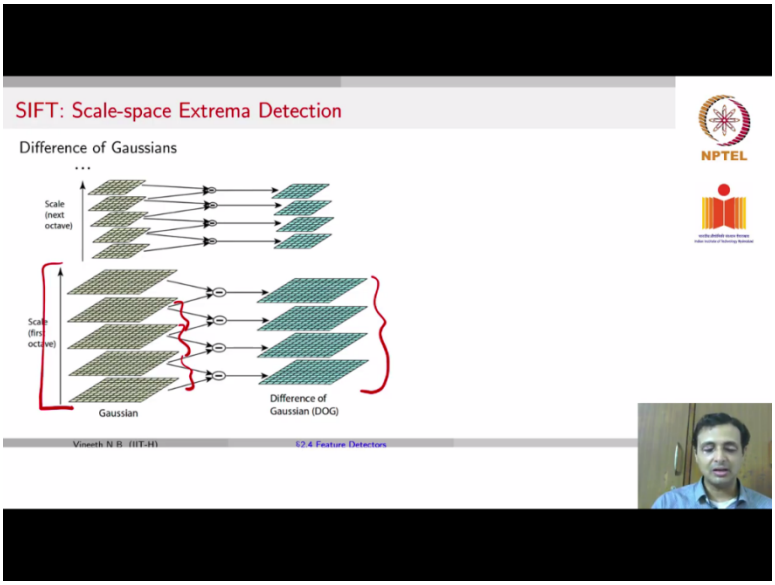
All these images from what is known as one octave. So you are going to have many such images in one octave, depending on the values of K and how many images you wanted. In the next step,

you construct your second octave, where the ideal goal would be to sub-sample the image and then repeat the same process that you had for your first octave.

Instead of doing a sub-sampling and repeating the same process, you can also do simply a Gaussian convolution with two sigma. Remember that will make it wider. So you can simply do a Gaussian convolution with two sigma and that gives you a similar effect as constructing a second octave of subsampled images.

Remember when you take Gaussian 2 sigma, your spread is going to be wider for the Gaussian. So you are going to blur out more pixels and probably consider more pixels that are further out, which is what would have happened if you had sub-sampled and then Gaussian with sigma as the standard deviation.

(Refer Slide Time: 07:39)



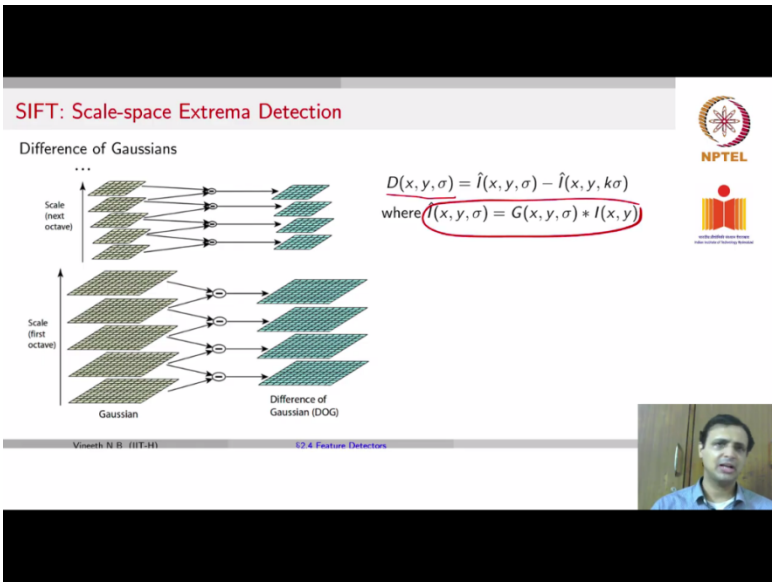
To be able to detect the extrema amongst these Gaussians, we ideally want to use the idea of Laplacians. Remember, again, if you recall our Laplacian as an edge detector, we said that unlike an edge detector like Sobel, where you get high intensities, wherever there is an edge, for a Laplacian, you get an edge whenever there is a zero crossing, please go back and refresh your lecture on Laplacian.

We said, whenever the Laplacian assumes a zero value and it is surrounded by some intensities, you can assume that there is edge information there. So we ideally want to use a Laplacian to be able to obtain the information of corners and edges in the image.

But remember we spoke about it in last lecture that Laplacian can be implemented as a difference of Gaussians. So, which means all these Gaussians that you have in the first octave, remember, one of those images would have been G of sigma, one of them, G of K sigma, G of K square sigma, so on and so forth, those are the filters. You would have ideally convolved your input image with each of those filters.

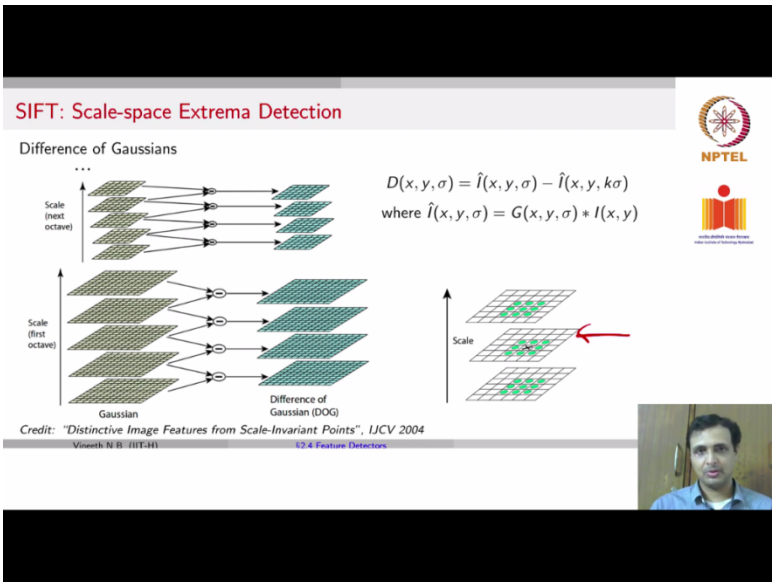
Now, you would simply take the difference between successive Gaussians on the images in the octave, and you will get a set of difference of Gaussian images, which, remember, is very similar to the Laplacian of those original images. So you construct these different Gaussian images in each octave separately.

(Refer Slide Time: 09:30)



We are going to write that as D(x, y, sigma) is equal to I-hat(x, y, sigma) - I-hat(x, y, k sigma), where I-hat is the convolution of a Gaussian with the appropriate Sigma with the original image. We are just writing mathematically, whatever we described so far.

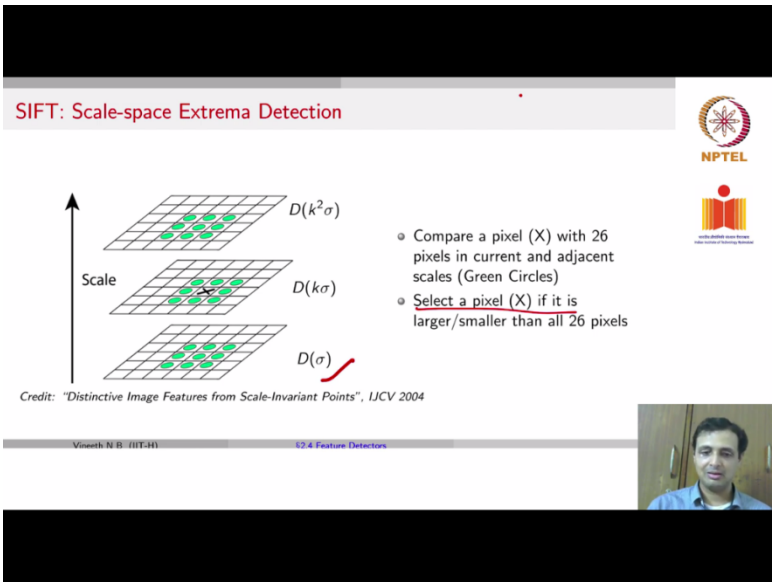
(Refer Slide Time: 09:50)



Now to find the scale space extrema detection, what we are going to do is for every pixel, so let us say you take that crossed pixel, in the image on the right. You take a three by three neighborhood around that point in its own image. And then you take a three cross three neighborhood in the image on the next higher scale and a three cross three neighborhood at the same location in the next lower scale.

With this, you will have a set of total 27 pixels and nine pixels on the top on the higher scale, 9 pixels on the bottom scale and 9 pixels in the appropriate scale, including the pixel at the center. So you have all of these pixels given to you.

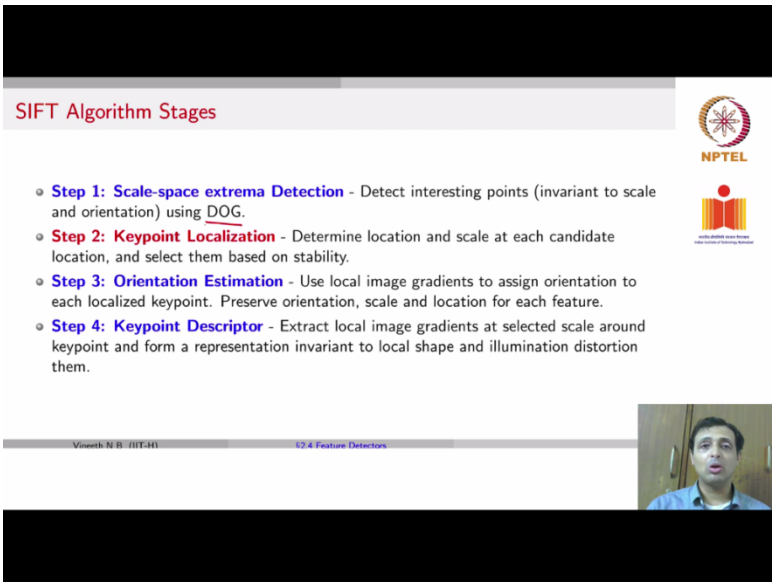
(Refer Slide Time: 10:41)



And what you do now is you compare a pixel with all of the 26 pixels in and around it. So that is going to be around it in terms of location, spatial as well as around it in terms of scale, the next higher scale and the next lower scale. And then, you select a pixel with, if it is larger or smaller than all of the 26 pixels around it. So you select both the minimum and the maximum of all of those 27 pixels in a spatial and scale neighborhood.

That is why we call this scale space extrema detection. So we are trying to detect extrema in both scale and space. That is your first step.

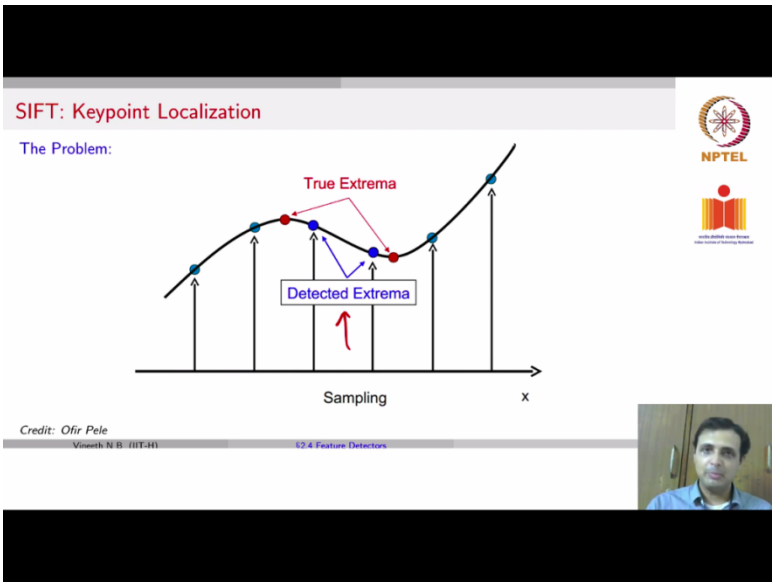
(Refer Slide Time: 11:30)



So we detect interesting points, invariant scale and orientation to a certain extent, using DOG or difference of Gaussians. Remember, again, the difference of Gaussians is an approximation of a Laplacian of Gaussian. Just one clarification here. Last time, we said the difference of Gaussian is an approximation of the Laplacian corrected to Laplacian of Gaussian. The difference of gaussian is an approximation to the Laplacian of Gaussian.

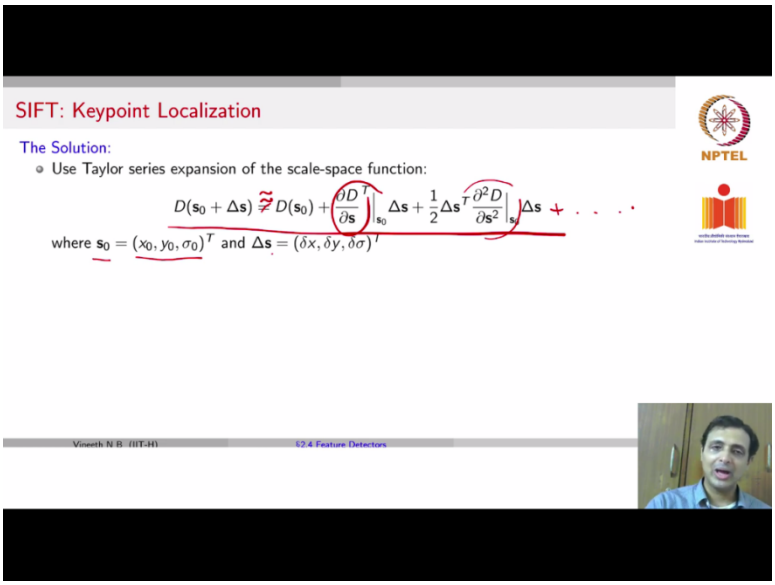
The next step is key point localization. We ideally want to see if we have found the exact point where we have the extrema, the maximum and the minimum.

(Refer Slide Time: 12:20)



To do this, we are going to look at this difference of Gaussian function D as any other function. And if we knew that our detector extrema were at these blue points, which are known as your detected extrema here. We ideally want to find these red points where the minimum and the maximum is actually achieved. So it means that minimum and maximum could be achieved in between two coordinate locations or in between two scales that we considered in, in the experiment so far. How do we find this?

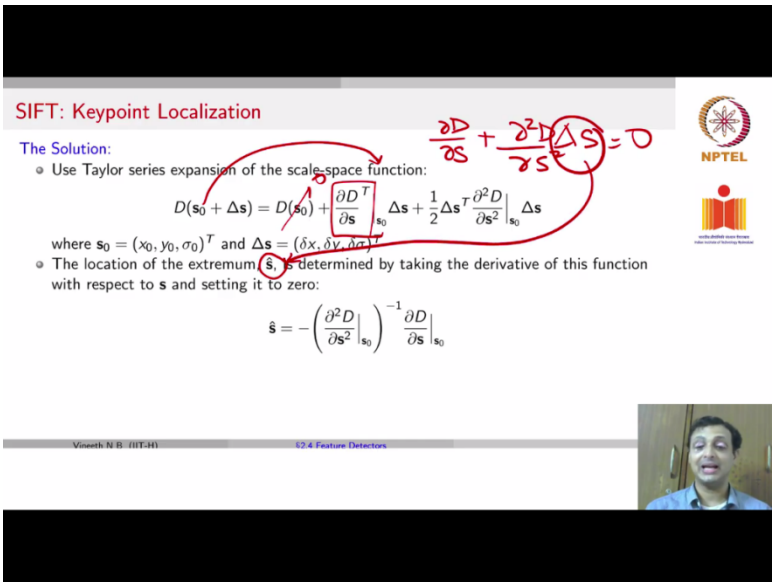
(Refer Slide Time: 12:55)



The way we find this is we are going to consider the Taylor series expansion. Given an s_0, which is given by (x_0 y_0 sigma_0). And at delta S which is given by (delta x, delta y, delta sigma), so s_0 is the minimum or maximum scale and coordinate location that we already have found in the earlier step, in step one and delta x, delta y and delta Sigma is what we want to find, to find out where the exact extrema is achieved.

So we write out your traditional Taylor series expansion, which is given by D in our case, that is the difference of Gaussian output. s not plus delta s is not equal to, it is just approximate, because we are not considering the higher order terms here, approximately D(s_0) + the first derivative x delta s + 1/2 delta transposed second derivative delta s and higher order terms.

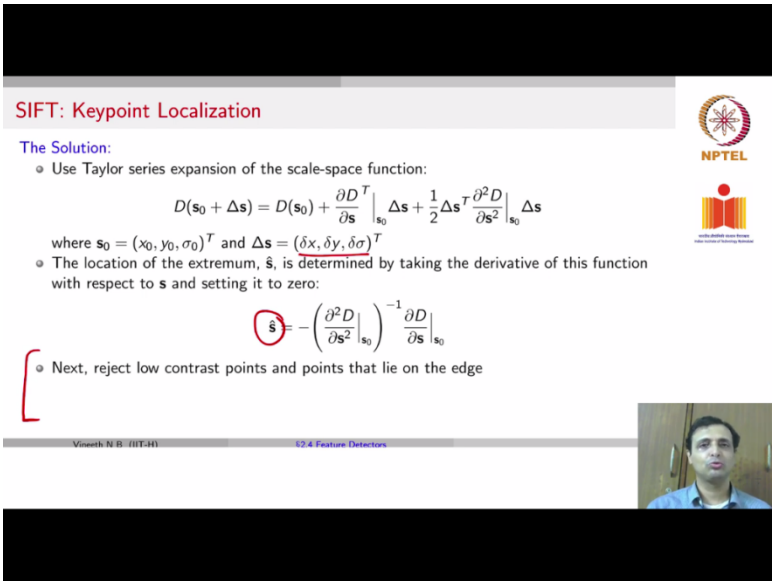
(Refer Slide Time: 14:00)



Now, to find out where this exactly attends the minima. We ideally have to differentiate this Taylor series expansion with respect to s or delta s in this particular case and so all for delta s. So we say that our solution for delta s, which we denote as s hat is going to be given by, this is a simple derivative, we just differentiate this with respect to s. The first term derivative will go to zero. Second derivative, remember the gradient is evaluated at s not. So that would be a constant with respect to delta s. So you would be left with, if you would differentiate this Taylor series expansion, you will end up having d(D)/ d(s) + d^2(D)/ d(s)^2 delta s = 0.

And when you solve for delta s, the solution is what we denote by S hat. It comes from simple differentiation of your Taylor series expansion.

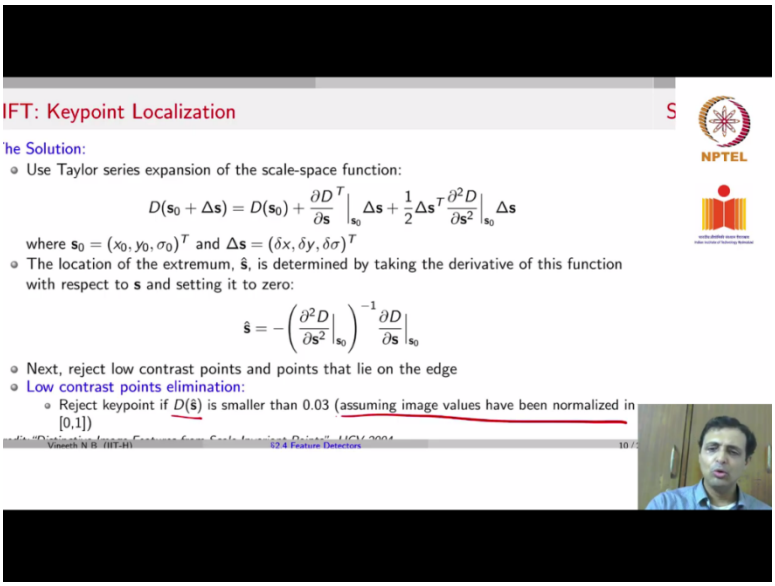
(Refer Slide Time: 15:20)



So, and how do we solve this? We can compute your second derivative and first derivative, simply by finite differences. Remember, you can take to complete the first derivative in both space and scale. You can simply do finite differences with the next locations on the left, next locations on the right or the higher locations on the scale and so on and so forth. Simple finite differences, the way we talked about computing gradients with respect to edges exactly the same way this can be computed. And we can solve for delta s, which gives us delta x, delta y, delta sigma, where the actual extrema is achieved.

So now we exactly know where the extrema is achieved. So this could help us localize better. Once we do this, we also do one more step. We also want to now remove all points, which have low contrast and all points, which are edge points because of their edge points, they are not useful corners as we already talked about in last lecture.

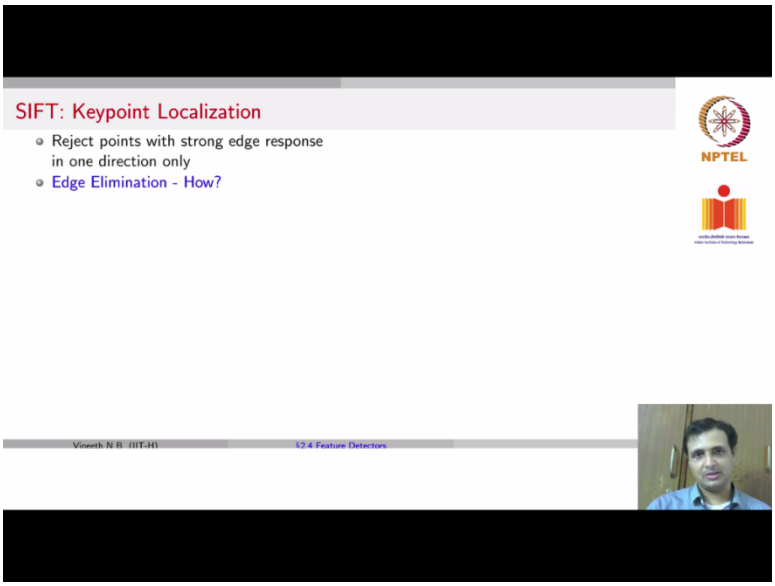
(Refer Slide Time: 16:26)



How do we remove low contrast points? We removed low contrast points by simply saying that if D at S hat, which is that new location is smaller than a certain value 0.03. If that value is small, we are going to say that it has the difference of Gaussian output there is very small, assuming of course, that your image values have been normalized in a certain range.

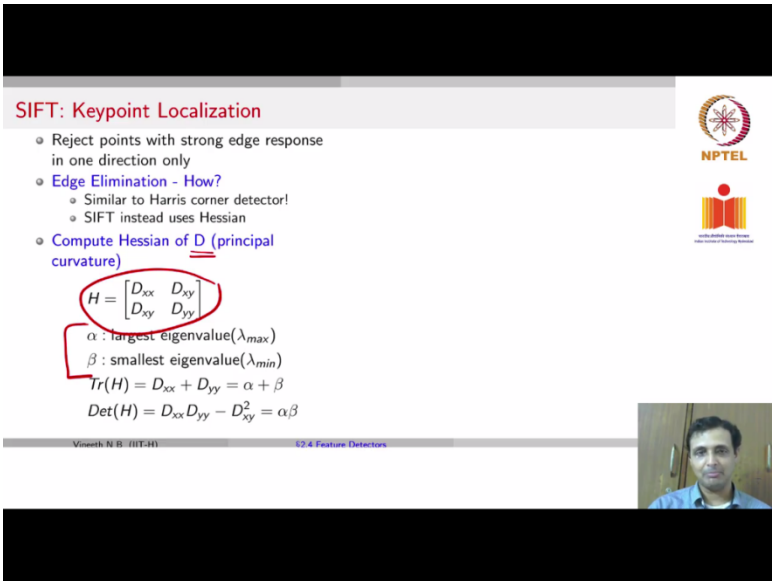
We are going to say that the value is too small, and we can actually reject it and not consider those points for future processing. How do you remove edge points though?

(Refer Slide Time: 17:00)



That should ring a bell, using a very similar approach, like the Harris corner detector. However, David Lowe and SIFT takes a slightly different approach and uses a Gaussian of your D, which is your difference of Gaussian output again, instead of using an auto correlation.

(Refer Slide Time: 17:20)

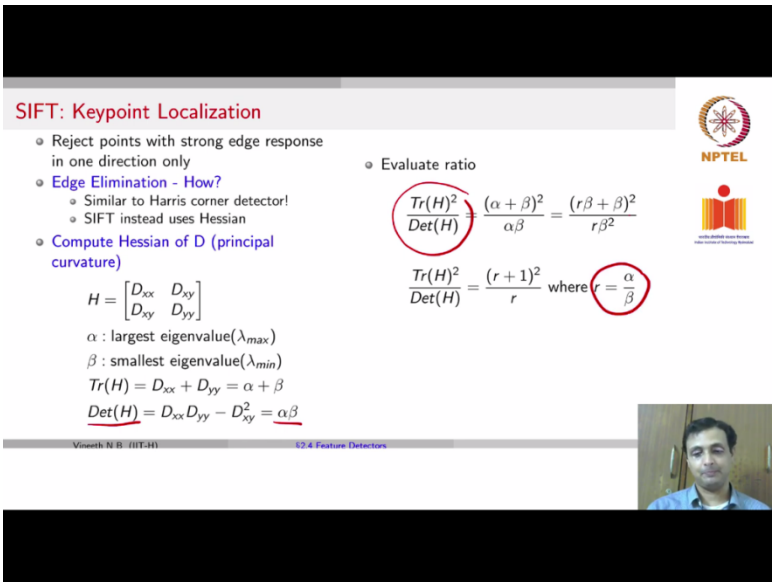


So there have been other people who also extended the Harris corner detector to use corners based on the Hessian, SIFT uses that kind of an approach where the Hessian of D which can also

be looked at as the curvatures. The Hessian effectively, the second derivative effectively captures the curvatures so the Hessian, which is your pairwise second derivative matrix, gives you an idea of curvatures in different directions or the changes in different, sharp changes in different directions.

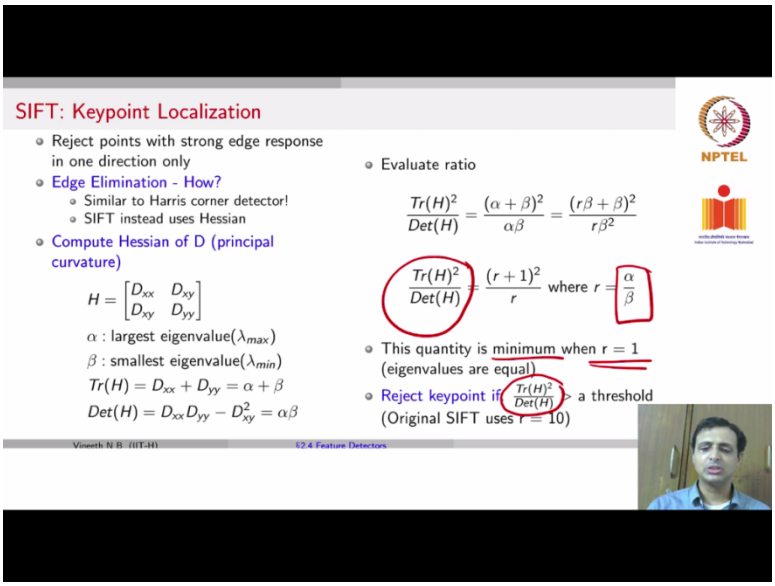
So the eigenvalues of the Hessian are also good estimates to understand corners. So very similar to the Harris corner detector, SIFT proposes using the Hessian, computing its largest and smallest eigenvalue, which are denoted by alpha and beta. Then you, once again, compute your trace and determinant of your hessian matrix, very similar to how we detect for the Harris corner detector, and then we evaluate different ratios with respect to trace and determinant.

(Refer Slide Time: 18:20)



So you could write your trace is going to be alpha plus beta. Your determinant is going to be, alpha x beta, so your trace square by determinant is going to be (alpha + beta)^2 /alpha x beta, which can be written as (r beta + beta) / r beta^2, where r is written as alpha / beta..

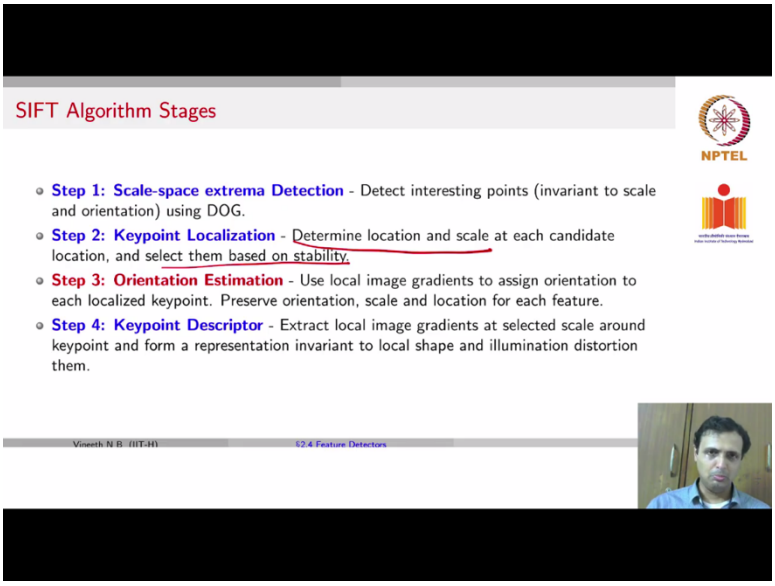
(Refer Slide Time: 19:00)



Using this approach, the trace square by determinant can be written as (r + 1)^2 / r and the final way we removed edges is by noting that this quantity now is going to be minimum when r is equal to 1, because when r is equal to 1, we know that alpha and beta are close to each other, are equally high, which is what we want to find your corner points.

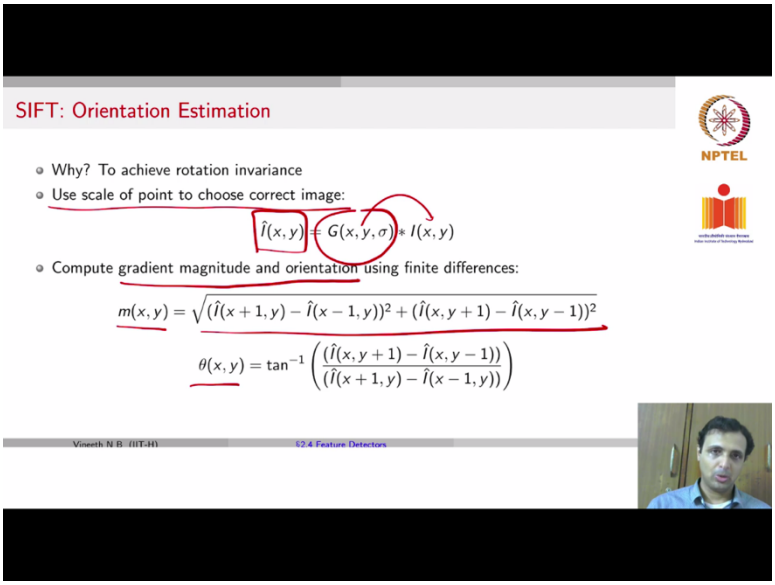
So SIFT proposes that you reject the key point if the trace of the Hessian square by determinant of H is greater than the threshold, because you want it to be close to 1. So if it is greater than the threshold, you simply reject the key point. The original SIFT paper uses r is equal to 10, in this particular case, the threshold to be 10 in this particular case.

(Refer Slide Time: 19:50)



So at the end of the step two, we have determined the exact location and scale at every extrema point. And we have also selected some of these key points based on stability by eliminating low contrast key points as well as edge points.

(Refer Slide Time: 20:15)



Now coming to the next step, which is on estimating the orientation of these key points. Let us first start by asking, why do we really need the orientation? The answer is simple. We want to

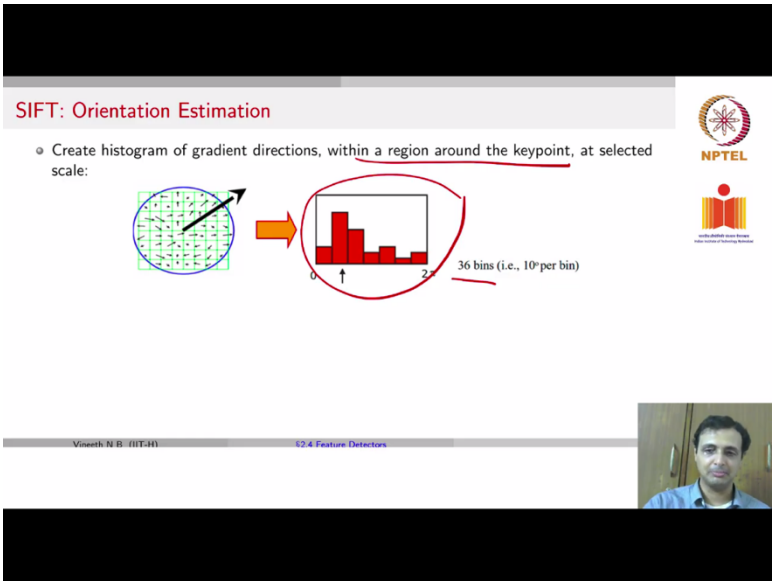
help it get rotation invariance by getting a sense of orientation and that would become clear when we describe how we are going to do it.

So we use the scale of the point to choose the correct image. Remember now that every key point is denoted as (x, y, sigma), where x and y are the coordinate locations and sigma is the scale at which it was found to be a key point, very similar to the scale invariant Harris detector, where the final corner is measured is highest at a particular scale.

Here again, a corner point or a key point is defined by (x, y, and sigma). So you use the scale of the point to choose an appropriate Gaussian and you convolve that Gaussian with your input image to get an image called I-hat. Now you compute the gradient magnitude and orientation, using a simple finite differences method for the I-hat image. So you would have m of every point, which is your gradient magnitude at every point, which you would be given by simply your root of gradient in x direction square + gradient in y direction square. That is going to be your magnitude of the gradient. Similarly, your orientation of the gradient at that location is going to be tan inverse of the y gradient by the x gradient.

At the end of this step, we are going to have a magnitude and an orientation for every point in your image, including the points that you have identified so far. Now, what do we do?

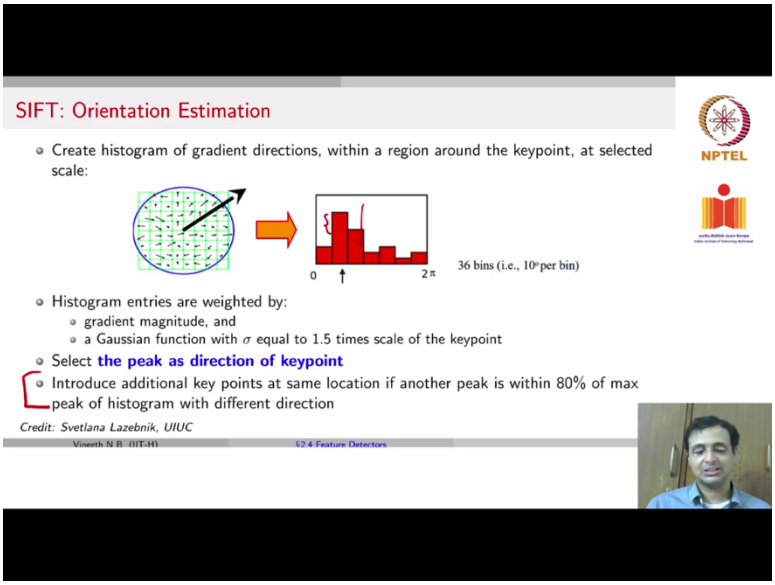
(Refer Slide Time: 22:05)



Now, what we are going to do is you take a certain region around the key point. We are not going to rely only on the magnitude and orientation of the key point alone, because we want to capture the local characteristics, not just the characteristics at the point. You take a region around the key point, a certain window, say 5 x 5 window or whatever that might be. And you consider the orientations of all the points, the magnitudes and the orientations of all the points in the neighborhood around the key point.

And you construct a histogram out of the orientations of all of these points. This histogram in the SIFT paper has 36 bins, with 10 degrees per bin. So we have an orientation angle depending on what the orientation angle is. So if you had an orientation angle of 185 degrees, you would put it in the 180 to 90 bin, so every bin is 10 degrees. When you have 360 degrees, so you have 36 bins.

(Refer Slide Time: 23:10)



The SIFT paper also recommends a few heuristics to improve performance here to say that the histogram entries are weighted also with a gradient magnitude. If the gradient magnitude is high, that weight in the histogram in a particular bin is increased. And if the gradient magnitude is low, that weight in the bin for that point is reduced. And it also has a Gaussian function with sigma equal to 1.5 times the scale of the key point.

So if you added a key point, you would place a Gaussian on top of the key point, and then with a certain sigma (sigma is given as 1.5 times the scale of the key point itself). And then you look if a particular point in the neighborhood that you considering is farther away, then it contributes lesser to the histogram. And if a certain point is closer to the key point, it contributes more of the histogram.

These are some heuristics that were used to improve performance. And once you construct this histogram, after these heuristics, the peak, whichever bin had the highest number of votes in the orientation is considered as the peak of that particular key point. Once again, keep in mind here, although we call it the peak of that key point, the peak is decided by the local characteristics of the key point rather than the point alone.

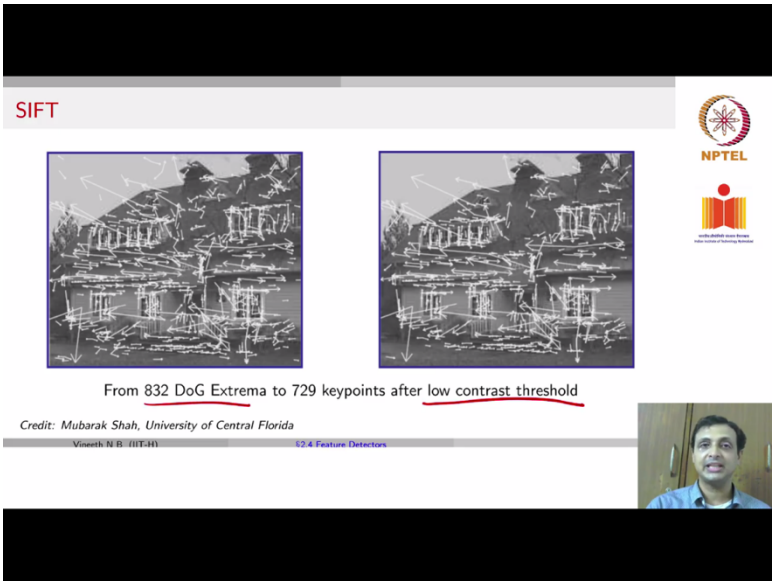
The SIFT work also suggests to introduce additional key points at the same location, if there is another peak in the histogram, which is at the 80% value of the original peak. If there were other such peaks, then you introduce another key point in the same location with a different orientation. So it is possible. You have two key points at the same location with different orientation, which gets more robust results.

(Refer Slide Time: 25:10)

Here are some visual illustrations of how this works. Here is an input image. The first step is about capturing the extrema of the difference of Gaussians. So you can see here that each point

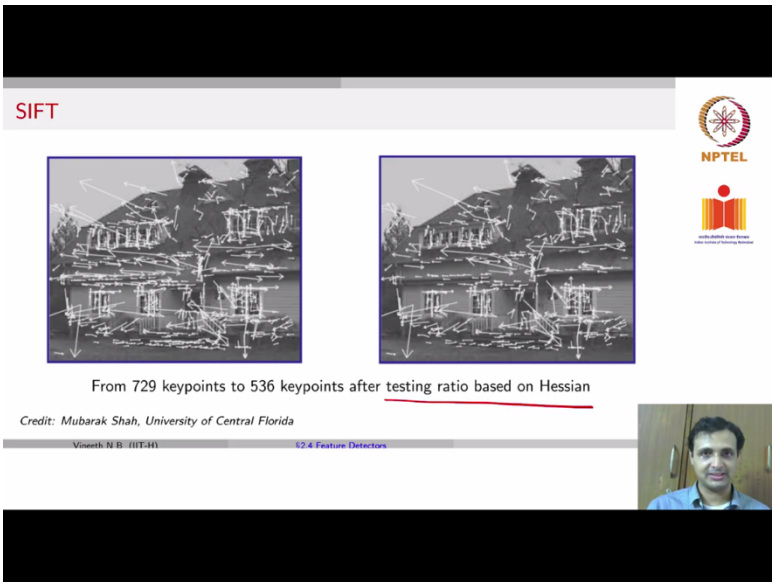
also has an arrow. The arrow denotes the orientation of the key point and the length of the arrow denotes the magnitude of the gradient at that key point. So the base of those arrows are the actual points. So this is your first extrema detection step.

(Refer Slide Time: 25:50)



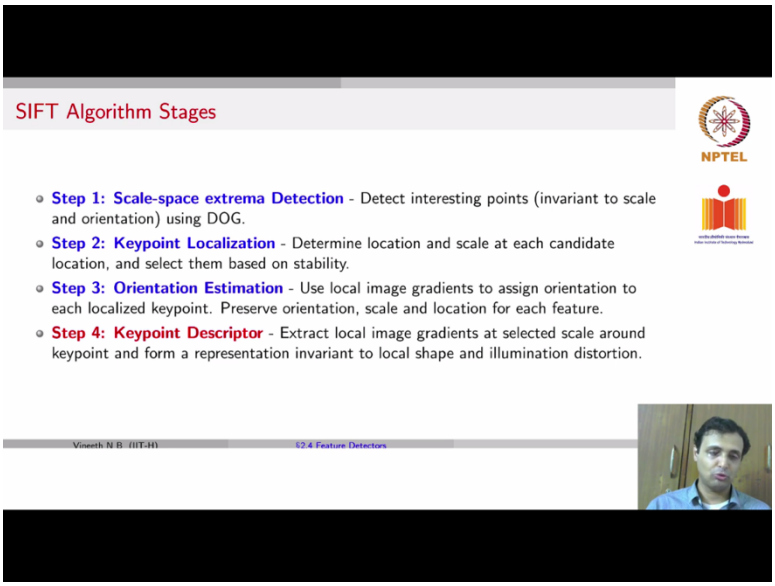
So then after doing a low contrast threshold these 832 DoG extrema lower down to about 729. You eliminate key points which have lower than a threshold in terms of contrast.

(Refer Slide Time: 26:00)



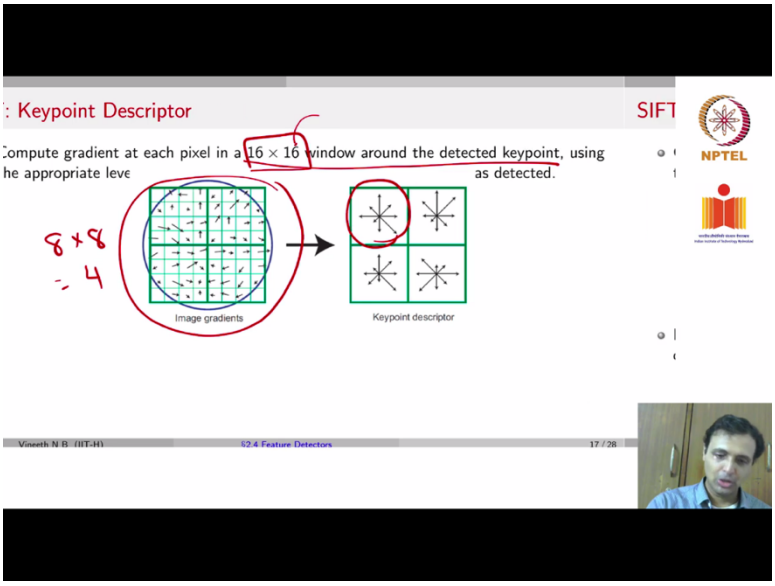
Then we also do the testing ratio based on hessians to eliminate edge-like artifacts. And then your key points come down from 729 to 536.

(Refer Slide Time: 26:18)



So you are pruning out noisy key points that may not be really of value. And as we already said, the arrows that you saw were the orientations that you obtained using your step three. Now we go to the last step, which is about finding a way to describe that key point. So far, we have found the key point, we have found the scale at which the key point exists, we also have an orientation for the key point. Now, the question is how do we describe the key point?

(Refer Slide Time: 26:50)



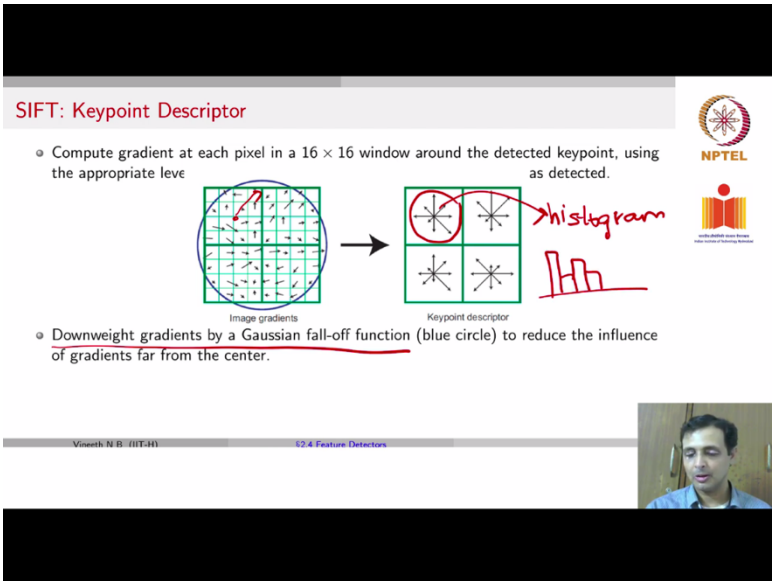
So to do this, we are once again going to use the gradient information in the local neighborhood around the key point. So we take a 16 cross 16 window around the detected key point. So bear with this image. This image shows an eight cross eight window, but for just for illustrative purposes, but the actual method suggests to take a 16 cross 16 window around the key point.

So you take your 8 cross 8 or 16 cross 16 window. And then you divide that 16 cross 16 window into quadrants of 4 cross 4 each. So for a 8 cross 8 window you will have 4 such quadrants, for a 16 cross 16 window, you would have 16 such quadrants, where each quarter was 4 cross 4.

Inside each of those quadrants, you construct a histogram of the orientations of points inside the patch and align it along 8 bins. So instead of aligning along 36 bins, this time you are going to do more closer alignment and align it only along 8 bins. This is what SIFT proposes. So you would have once again, just to remind in a 8 cross 8 window you would have four such histograms. But in a 16 cross 16 window, you would have 16 such histograms.

So each histogram has 8 orientations because you have 8 bins there. And every point in the neighborhood in that 4x4 patch contributes to one of the bins in the histogram.

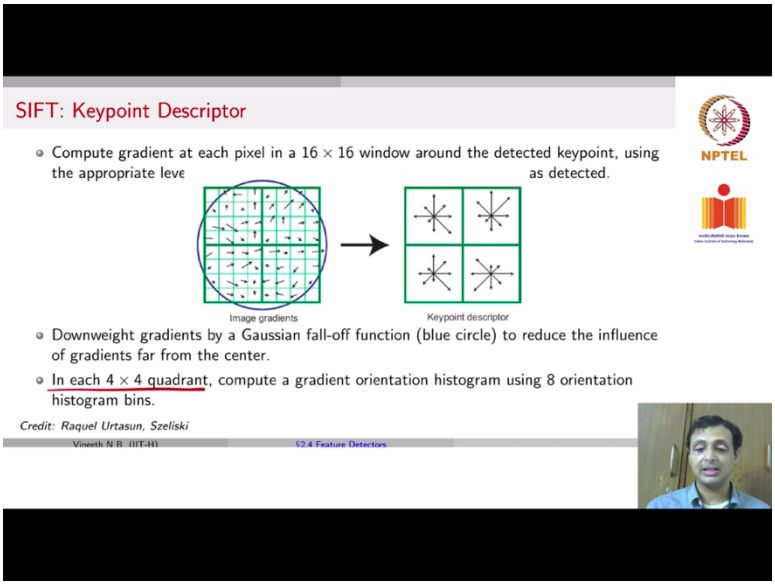
(Refer Slide Time: 28:40)



Similar to how we found the orientation. Here also, we use heuristic such as down weighting the gradients by a Gaussian fall off function. So within a particular neighborhood, if there was a point further out that contributes a little lesser to the histogram and a point closer contributes more to the histogram.

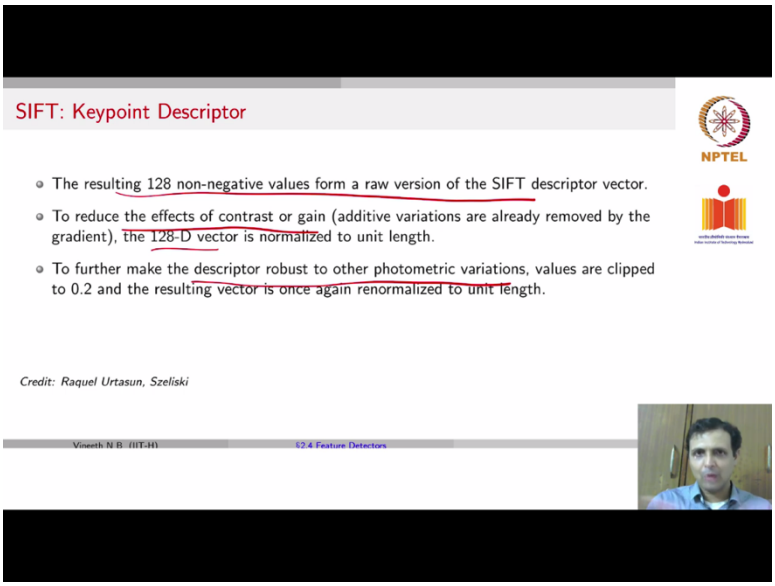
Just to clarify on the image on the right, this is just another way of representing a histogram, just keep in mind that the same diagram could also have been drawn something like this, with 8 bins as I said.

(Refer Slide Time: 29:22)



So the length here denotes the strength of the bin in the histogram. So in each 4x4 quadrant, you compute your gradient orientation histogram using 8 orientation bins.

(Refer Slide Time: 29:30)

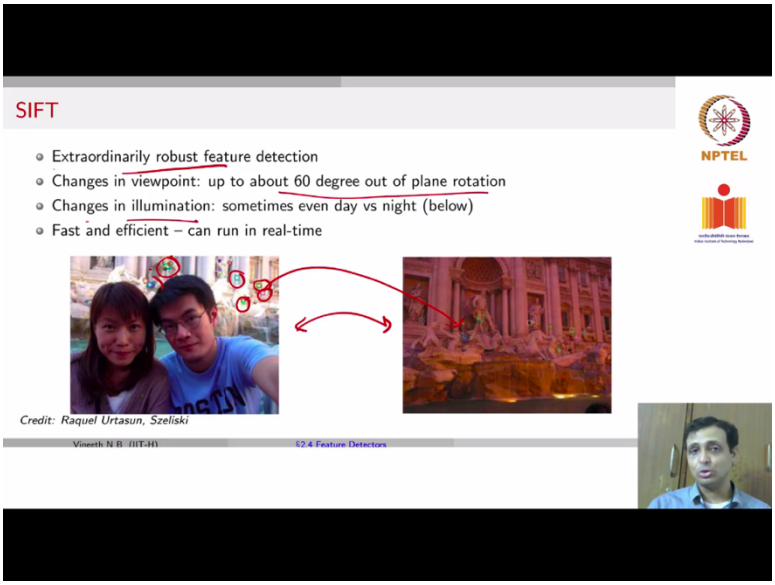


Once you do this, you are going to have totally 16 histograms, each with 8 values. This set of 16x8, which is 128 becomes your raw version of a SIFT descriptive. So this is how we are going to describe that key point, just by the gradient orientations of the neighborhood around that key

point. To reduce effects of any contrast or local changes, this 128 dimensional vector is normalized to unit length.

Further, to ensure that the descriptor can be robust to other kinds of illumination variations. The values are also clipped to 0.2. And the resulting vector is once again renormalized to unit length. So you do not consider very small values. You clip it to 0.2 and once again normalize it. This is, you could consider this as a step of removing outliers, something very low, we just want to clip it and then renormalize it to the original length. These are heuristics that the paper recommends to get better performance.

(Refer Slide Time: 30:40)

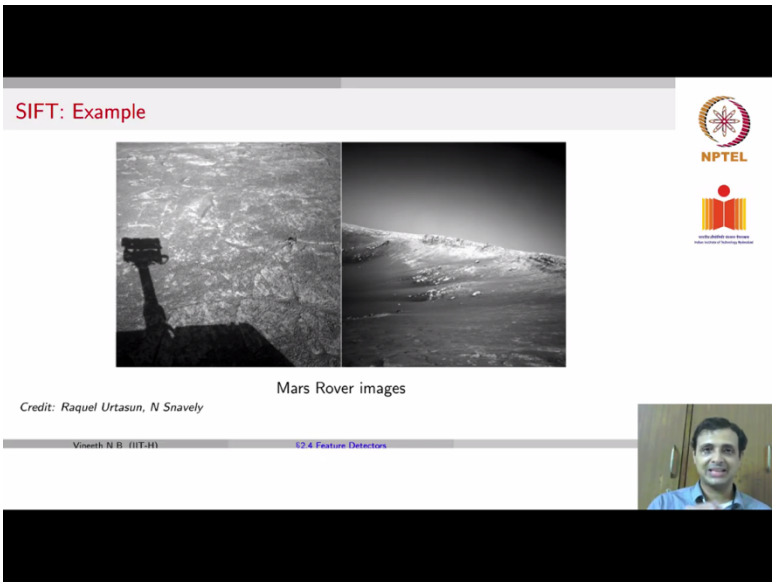


Let us see a few examples. So here you can see a couple of tourist images of a popular monument. And you can see these background values here, which are the key points detected by SIFT in this image. And if you actually compare even on the second image where the significant rotation, perhaps almost a morningtel evening change we detect similar key points in both these settings.

So it is an extraordinarily robust feature detector. It handles up to about 60 degrees of changes in rotation. It can also handle a good amount of changes in illumination. And it is also pretty fast and efficient can run in real time. There is a lot of code available for people willing, wanting to

use SIFT. And SIFT, as I already mentioned has been used in many vision applications since it was initially proposed in the early 2000s.

(Refer Slide Time: 31:43)



Here is another challenging example of images from the Mars Rover. And we ideally want to see this is a practical application, where, when the Mars Rover takes images of different scenes since there is no human there, it would be good to understand which scenes actually match with each other so that we know which part of Mars is the rover currently going on.

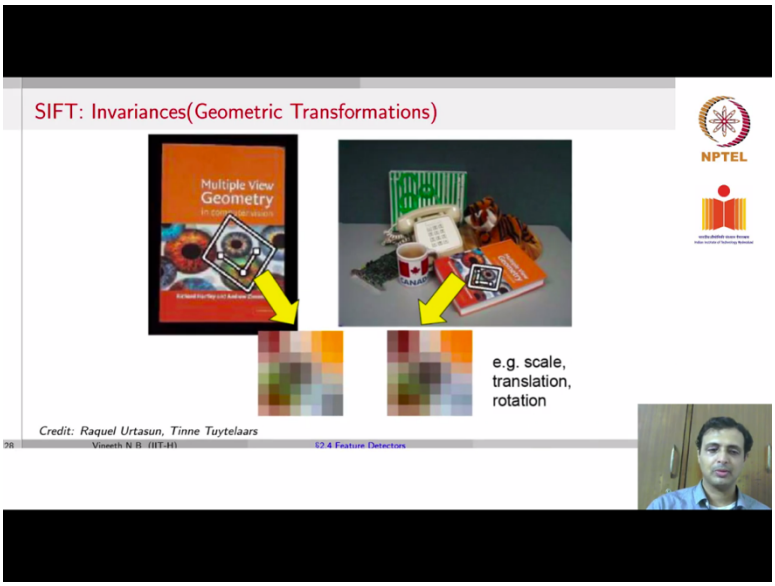
But then because the scales and the angles may be different in these images matching becomes a little hard.

(Refer Slide Time: 32:23)

So this particular example, although these two images look very different, a SIFT matching between these two images finds some features, which are actually have very similar descriptors in these two images.

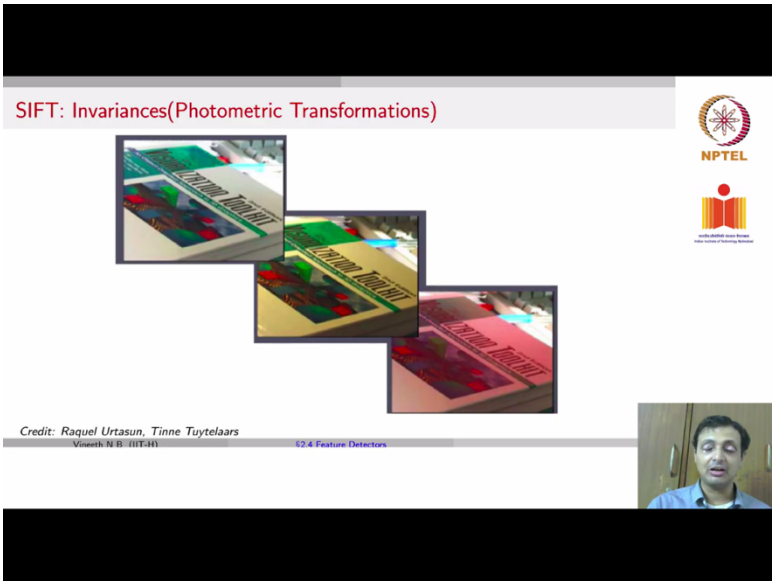
Now you could match these two images to say where the rover was with respect to a broader scene on the landscape.

(Refer Slide Time: 32:40)



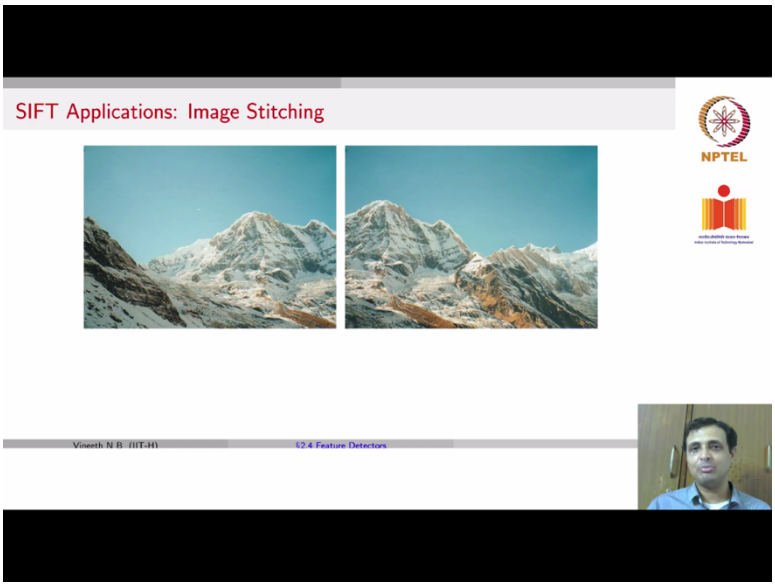
Here are more examples. So SIFT is fairly invariant to geometric transformations, such as scale translation and rotation.

(Refer Slide Time: 32:55)



It is also reasonably invariant to photometric transformations such as changing in color hues because we largely rely on gradients more than the pixel intensity itself.

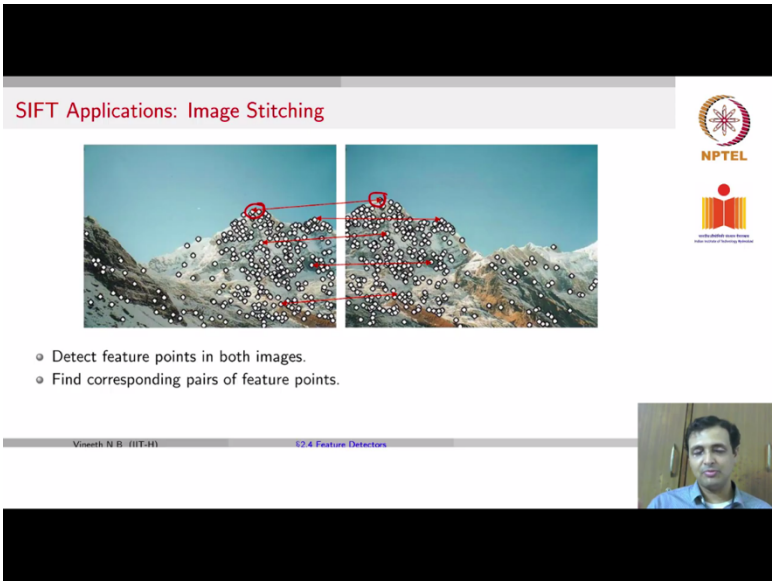
(Refer Slide Time: 33:07)



One of the popular applications that SIFT can be used for as we started the last lecture with is image stitching, where you find SIFT key points in both of these images and then match descriptors and find out which key point has gone to from image one has gone to which key point in image two.

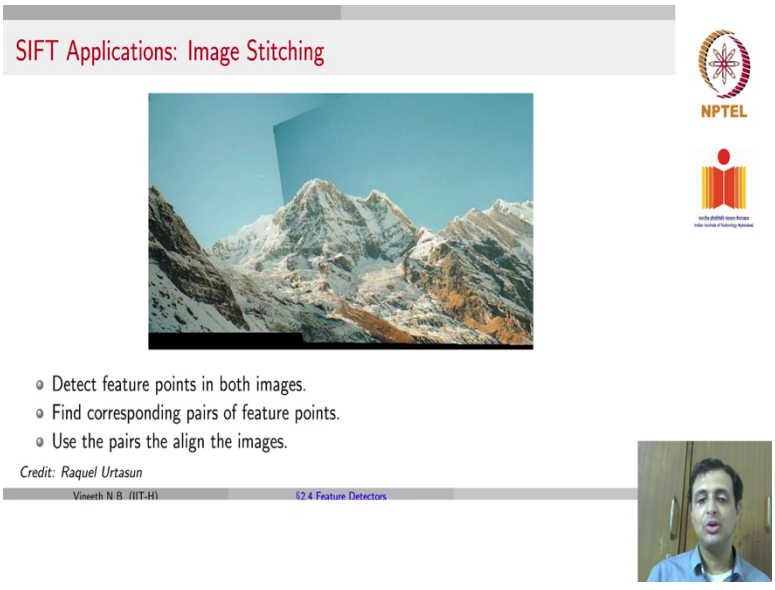
And once you find these matching key points, you can solve for a system of equations to find the transformation between two images. And once you find the transformation between two images, you can stitch them together to make a panorama. This part of it, we will talk about a little later when we talk about image matching.

(Refer Slide Time: 34:16)



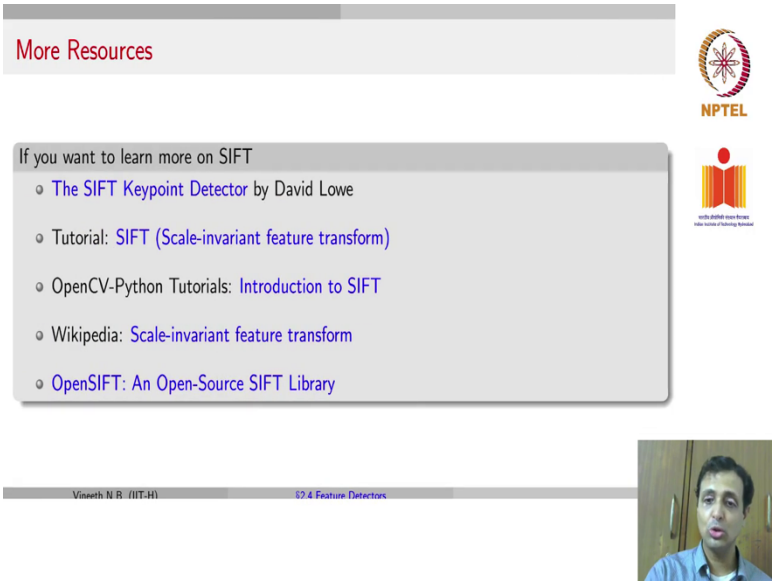
So here is an example. You detect feature points in both images. Then you find out using the descriptors that you get from SIFT. Remember every key point is going to be represented by a 128 dimensional vector in case of SIFT. And you try to see, match the 128 dimensional vector here with the 128 dimensional vector here. If the vectors match, you know, that that key point matches with that key point in the second image.

(Refer Slide Time: 32:40)



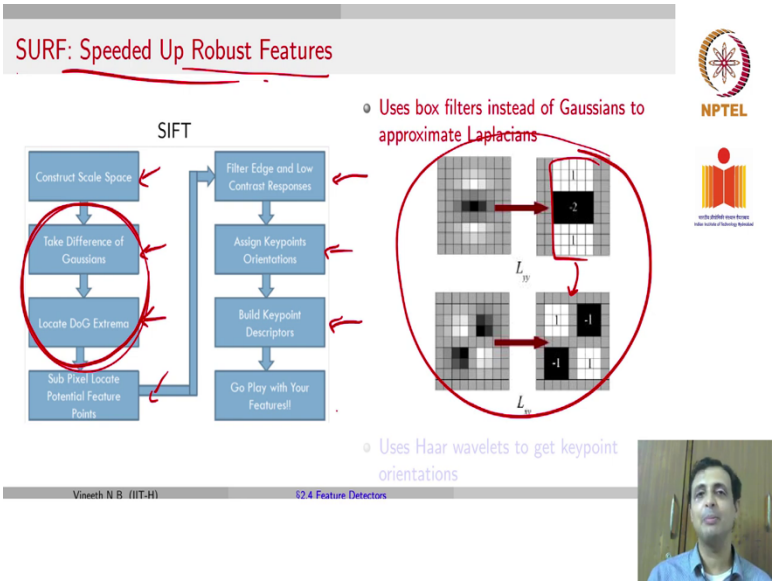
Once you do that, you can align the images and place them one or the other.

(Refer Slide Time: 32:23)



If you want to know more about SIFT, there are plenty of resources online. David Lowe has an excellent paper, as I already mentioned, please do read the paper. There are also python tutorials, the Wikipedia entry is also fairly informative as well as an open SIFT library.

(Refer Slide Time: 34:43)

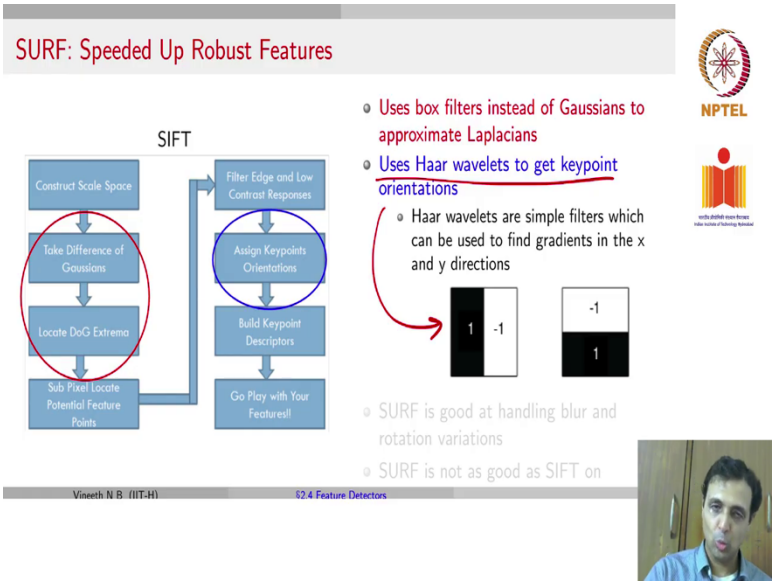


Before we conclude this lecture, we are also going to look at variants of SIFT that have been developed. At least a few variants that have been developed since SIFT was developed. One of

the most popular improvements on SIFT is known as SURF, stands for speeded up robust features. And what SURF does, it takes the pipeline of SIFT.

So you could now write the pipeline of SIFT as construct the scale space, take the difference of Gaussians, locate the DoG extrema, find the sub-pixel localizations of those key points. Then filter the edges and low contrast responses, assign key point orientations, build key point descriptors, and then use those features for whatever you want. That is the pipeline, but that we just finished discussing. So one of the improvements that SURF comes up with on top of SIFT is instead of using difference of Gaussians and finding the extrema, they use what are known as box filters.

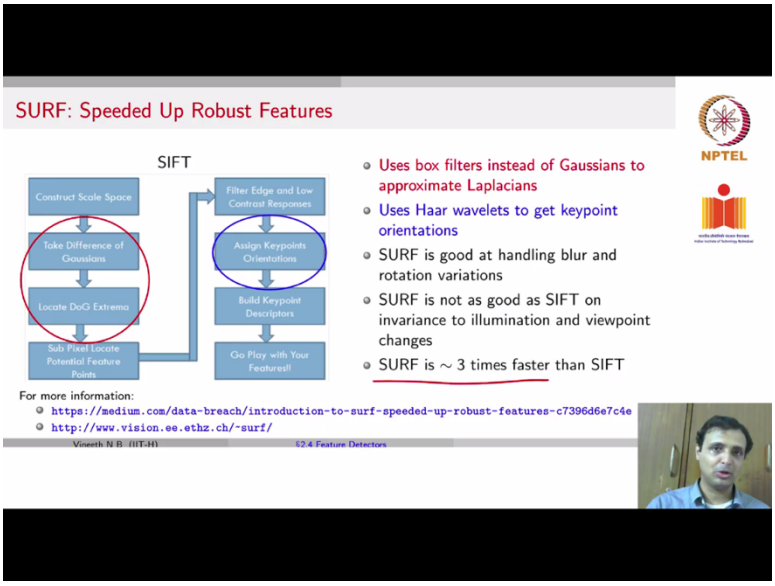
(Refer Slide Time: 36:26)



These are also known as Harr filters, we will talk about them in the next slide, but these are also known as box filters. Box filters are filters such as these, where you can have a filter, where one part of it is white, another part black, and another part white. You could do this in various combinations. You could do a checkerboard kind of an effect. You could do it with, you could increase the number of blacks and whites. You could have the blacks two times, whites three times, or you could flip it. All of those combinations are typically called box filters and they together form what are known as Haar wavelets. We are not going to go into wavelets here. But you could look at wavelets as a generalization of a Fourier basis.

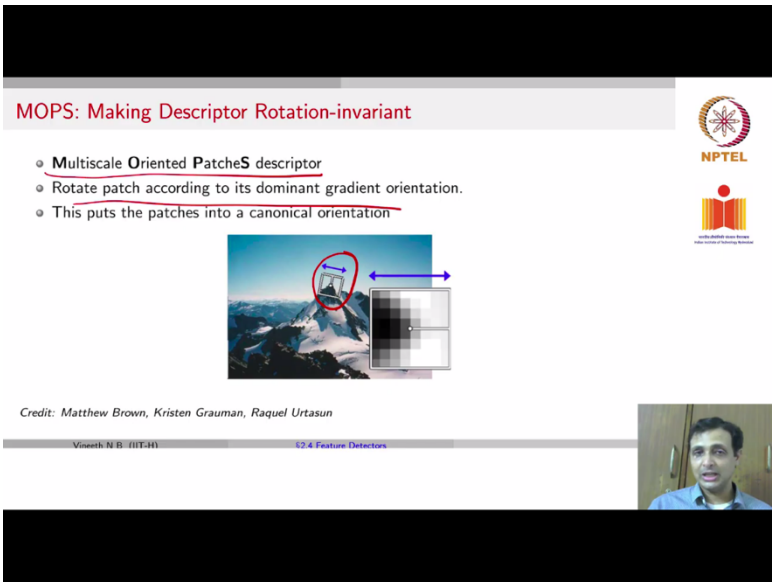
So we use the Haar wavelets, which are your Haar filters such as these to get key point orientations. So that is one difference of SURF and that allows SURF to be very good at handling blur and rotation variations.

(Refer Slide Time: 36:51)



So SURF is not as good as SIFT on invariants to illumination and viewpoint changes though, but it handles blur and rotations fairly well. Importantly, SURF is almost three times faster than SIFT because of these changes. There is a particular reason for that using Haar filters can make computations significantly faster. I am going to leave that to you as a homework question, please do read up why using Haar wavelets can make computations faster? A hint for you is to read up something called integrated images.

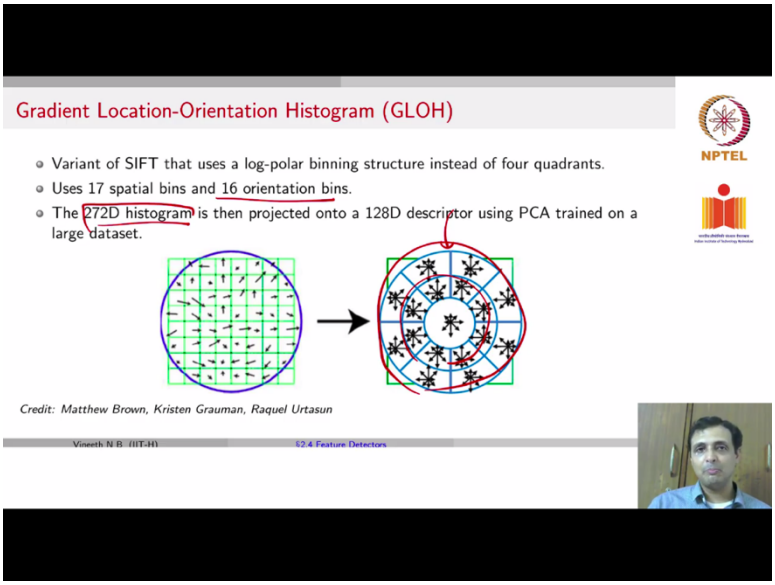
(Refer Slide Time: 37:40)



There is also another variant of SIFT called MOPS. MOPS stands for multi scale oriented patches descriptor. In this case, the patch that you have around the key point is rotated according to its dominant gradient orientation. And then you compute your histogram and the descriptive.

Why is this useful by rotating the patch to its dominant gradient orientation, you are ensuring that all key points have a canonical orientation, which is the same. So if you rotated a particular artifact in image one and image two, because the orientations would be different when you canonicalize them, they both would end up getting the same descriptor in both of these cases, that is the idea with a multi scale oriented patches descriptor.

(Refer Slide Time: 38:40)



Finally, there is also an approach called gradient location-orientation histogram or GLOH, which is a variant of SIFT that uses a log polar winning structure instead of using quadrants. This particular approach, so as you can see here has about 17 spatial bins and has about 16 orientation bins, where in each concentric circle, there are 8 of them. So you have 16 orientation bins instead of SIFT, where you had 8 orientation bins in the final step.

So because of this, the representation here becomes 272 dimensional. There are 17 spatial bins and 16 orientation bins, 17 into 16, 272 dimensional. And to make it computationally efficient, this method then uses principle component analysis to reduce the 272 dimensional descriptor to a

128 dimensional descriptor, which is actually used for as the descriptor using this particular method.

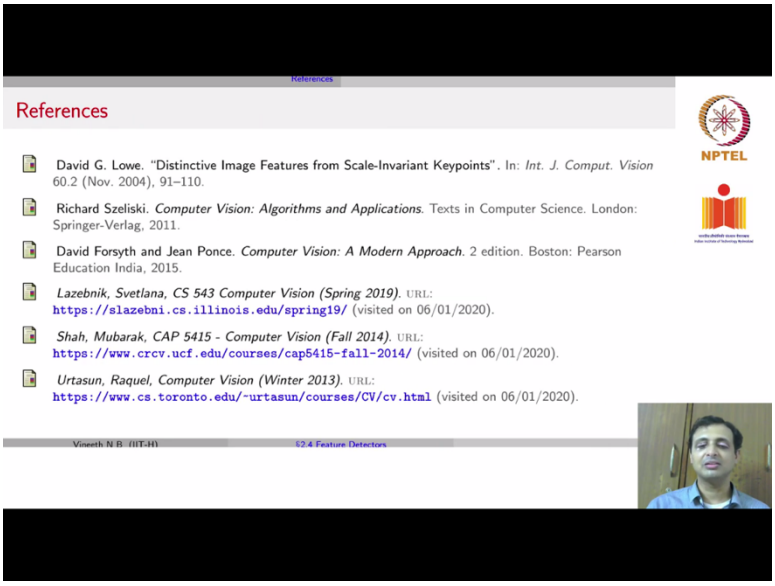
(Refer Slide Time: 39:40)

That concludes this lecture. Please read section 3.5 of Szeliski's book. For more information on SURF, there are open CV python tutorials as well as an entry on Wikipedia, which is informative again.

Please also look at other links on respective slides to be able to understand further. A couple of questions to take away other than why are wavelets are fast to compute is, which descriptor performs better SIFT or MOPS, think about it. Why is SIFT descriptor better than the Harris corner detector?

The answers to these questions are in the lectures itself. That is going to be your hint.

(Refer Slide Time: 40:25)



And here are other references to takeaway.