



Deep Learning for Computer Vision
Professor. Vineeth N Balasubramanian
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
Lecture No. 09
From Edges to Blobs and Corners

Last class, we spoke about edge detection and went into details of the canny edge detector and this lecture, we will move a little forward and talk about other kinds of artefacts from images that are practically useful, specifically blobs and corners.


(Refer Slide Time: 0:40)

Review: Using Canny Edges to get Straight Lines

- Compute Canny edges
 - Compute $\nabla_x f, \nabla_y f$ (gradients in x, y directions)
 - Compute $\theta = \tan^{-1} \frac{\nabla_y f}{\nabla_x f}$
- Assign each edge to one of 8 directions. For each direction d , obtain "edgelets":
 - Find connected components for edge pixels with directions in $\{d-1, d, d+1\}$

Vineeth N B. (IIT-H)
§2.2 Blob and Corner Detection



Before we go there, we did leave one question for you at the end of the last lecture, hope you spend some time working out the solution yourself, if not, we will briefly discuss the outline right now and wait for you to work out the details yourself. So, we talked about trying to use canny edges to get straight lines in images. How do you do this? You first compute canny edges in your image, which means you would compute your gradients in the X direction and the Y direction and you could also compute your thetas, which are your angles of each of your edges, which is given by $\tan^{-1} \frac{\nabla_y f}{\nabla_x f}$.

Now as the next step, what we do is assign each edge to one of 8 directions, 8 is just a number you could choose anything else, but you assign each edge to one of 8 directions. So, you probably combine several inches into a single bin. And for each of these 8 directions, let us call one of those directions to be d , we obtain what are known as edgelets and what are edgelets? Edgelets are connected components for edge pixels with directions in $(d-1, d, d+1)$.

So for a given, so you have 8 possible directions that you have bin all your edge orientations into and you take one of them and connect them with edges, with orientation $d-1$ and $d+1$. So, you will get a set of what are known as edgelets.

(Refer Slide Time: 2:28)

Review: Using Canny Edges to get Straight Lines

- Compute Canny edges
 - Compute $\nabla_x f, \nabla_y f$ (gradients in x, y directions)
 - Compute $\theta = \tan^{-1} \frac{\nabla_y f}{\nabla_x f}$
- Assign each edge to one of 8 directions. For each direction d , obtain "edgelets":
 - Find connected components for edge pixels with directions in $\{d-1, d, d+1\}$
- Compute straightness and orientation, θ of edgelets using eigenvectors and eigenvalues of second moment matrix, M , of edge pixels


$$M = \begin{bmatrix} \sum (x - \mu_x)^2 & \sum (x - \mu_x)(y - \mu_y) \\ \sum (x - \mu_x)(y - \mu_y) & \sum (y - \mu_y)^2 \end{bmatrix}$$

$[v, \lambda] = \text{eig}(M)$

$\theta = \tan^{-1}(v_1/v_0)$ where v_1 is the 'larger' eigenvector; straightness = λ_2/λ_1
- Threshold straightness appropriately and store line segments

Credit: Derek Hoiem

Vineeth N B (IIT-H)
§2.2 Blob and Corner Detection



Once you compute your edgelets, we then compute two quantities, straightness and orientation of the edgelets. And the way we go about doing this is using the eigenvectors and eigenvalues of the second moment matrix of the edge pixels. So what we do here is you have your edge pixels that belong to a particular edgelets, so you take all of those pixels and compute such a matrix. So if you look at the first entry here, the first entry here corresponds to $(X - \mu_x)$, μ_x is the mean X dimension of all of those pixels in the edgelets and each X corresponds to one of those X-coordinates of one of those edgelet pixels, so you are simply computing in some sense, a sense of variance with respect to the main edge pixel along the X direction. Similarly, you have a variance along the Y direction of those edge pixels with respect to their means and you also find the covariance with respect to X and Y directions.

So once you have this very similar to how principle component analysis happens, we take the eigenvectors and eigenvalues of this second moment matrix and remember, the eigenvector corresponding to the largest eigenvalue will give you the direction of the maximum variance among these pixels and that is what we are looking for at this time.

So we finally decide that the orientation of the edgelet is going to be given by $\tan^{-1} \frac{v_1}{v_0}$, where v_1 is the larger eigenvector. By larger we mean the eigenvector corresponding to the larger eigenvalue. Remember here, that M is a two cross to matrix, which means you will only have two eigenvalues at maximum, and you are going to take the eigenvector corresponding to the larger eigenvalue and we call that to be v_1 and v_0 is the other eigenvector.

So if you take a tan inverse of these two vectors, that theta will give you the direction of that overall edgelet. So in case, you are finding this difficult to understand, please go back and read principle component analysis and you will be able to connect that to this particular idea.

And we define straightness to be $\frac{\lambda_2}{\lambda_1}$, where λ_2 is the second largest eigenvalue and λ_1 is the largest eigenvalue. So the straightness here is going to be the highest value when both the λ_2 s and λ_1 s are close to equal. And once you get this quantity, you just threshold the straightness appropriately and store your line segments.

So it really does not matter whether you measure straightness as $\frac{\lambda_2}{\lambda_1}$ or $\frac{\lambda_1}{\lambda_2}$, you just have to ensure that you construct your threshold appropriately. If you inward the straightness ratio, you just have to ensure that you threshold it at an appropriate value and then obtain your line segments.

(Refer Slide Time: 6:02)

Review: Using Canny Edges to get Straight Lines

Credit: Derek Hoiem

Vineeth N B (IIT-B) §2.2 Blob and Corner Detection

For accessing this content for free (no charge), visit : npTEL.ac.in

So here is a visual example. So you can see that after applying canny, you get all these edges on the left. So you do get lots of edges, which are fairly well connected, not too noisy, but not all of them correspond to straight lines, many of them correspond to edges that represent that say the texture of the floor and so on and so forth. But we really do not want that when the model, let us say, there is an application where we really do not want it. So then you use this kind of an approach to obtain which is the straight lines from your canny edges. Think more about this and you can, I think knowing more about PCA could help you understand this a bit better.

(Refer Slide Time: 6:44)

Going Further to a Second Derivative

What if we took the Laplacian of Gaussian?

Gaussian $h_{\sigma}(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$

Derivative of Gaussian $\frac{\partial}{\partial x} h_{\sigma}(u, v)$

Laplacian of Gaussian $\nabla^2 h_{\sigma}(u, v)$

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

Credit: S Seitz, K Grauman

Vineeth N B (IIT-H) §2.2 Blob and Corner Detection

For accessing this content for free (no charge), visit : npTEL.ac.in

(1)

Moving forward. As we said, the focus is going to be going from edges to newer artifacts such as blobs and corners. So remember, last lecture, we talked about taking the Laplacian of Gaussian and we said that taking the zero crossings of the Laplacian of Gaussian could give you a measure of edges in an image. That was another way you could obtain edges in an image. We are going to talk about it slightly differently now. So once again, just to recap, remember this is your Gaussian, this is your derivative of the Gaussian and this is your Laplacian of Gaussian where the Laplacian we define as $\nabla^2 f$ is equal to $\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$.

(Refer Slide Time: 7:35)

Laplacian of Gaussian

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$




Example of a 3×3
Laplacian of Gaussian
filter:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

How did we obtain this
filter?

Vineeth N B (IIT-H) §2.2 Blob and Corner Detection

For accessing this content for free (no charge), visit : nptel.ac.in



Now, let us say an example of a three cross three Laplacian of Gaussian filter is going to look somewhat like this, where you have a minus four in the center and one, one, one, one in its nearest neighbors and 0, 0, 0, 0 in its next level of nearest neighbors. But can you try to guess why this is a relevant Laplacian of Gaussian filter? One point to notice here is an equivalent Laplacian of Gaussian filter could also have been 0, 0, 0, 0, -1, -1, -1, -1 and 4.

So if you actually visualize this filter, you would see that at the center, there is a peak at its immediate nearest neighbors, there is a -1, there is a value that goes below underneath your 0 and then there are 0s in other places. So if you try to visualize this filter would be very, very similar to the shape of the Laplacian of Gaussian that we just saw on the earlier slide, but that is from a geometric or a conceptual perspective.

Why did we say 4? Why not 8, why not any other number? Once again it goes back to approximating the gradients in some manner and we can work this out to show you how this works out.

(Refer Slide Time: 8:59)



Laplacian of Gaussian

- Discrete approximation of the second derivative:

Example of a 3 × 3 Laplacian of Gaussian filter:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

How did we obtain this filter?

$$\frac{\partial^2 f}{\partial x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y)$$

$$\frac{\partial^2 f}{\partial y^2} = f(x, y+1) + f(x, y-1) - 2f(x, y)$$


Substituting in the LoG equation: $\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$

$$\nabla^2 f = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y)$$

- Converting this equation to a filter results in the given LoG matrix.

Vineeth N B (IIT-H) §2.2 Blob and Corner Detection

For accessing this content for free (no charge), visit : npTEL.ac.in



So remember we are computing second derivatives here. So $\frac{\partial^2 f}{\partial x^2}$ can be written as approximating using first principles, you can say $f(x + 1, y) + f(x - 1, y) - 2f(x, y)$. Similarly, you can write it out for $\frac{\partial^2 f}{\partial y^2}$. Now, if you try to put both of these into your Laplacian of Gaussian equation or your Laplacian equation in this particular case, you would then have $\nabla^2 f$ is equal to $f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y)$. This should straight away ring a bell for you as to why you have got this kind of a filter.

Remember, at $f(x, y)$ the coefficient is -4 at its neighbors $x + 1, x - 1$ and similarly $y + 1, y - 1$, the coefficient is 1. So this simply came from taking an approximation of the gradient. Remember, you could have other kinds of approximations of the gradient with respect to the local neighborhood. You could consider larger windows one and so forth. If you do that, obviously the definition of the Laplacian of Gaussian filter would have to be appropriately modified.

(Refer Slide Time: 10:26)

Laplacian of Gaussian



Original Image Laplacian Laplacian of Gaussian

Vineeth N B (IIT-H) §2.2 Blob and Corner Detection

For accessing this content for free (no charge), visit : npTEL.ac.in



Here is another visual illustration. Here is the original image, here is simply taking the Laplacian and here is the result if you take the Laplacian of Gaussian. Remember again, that Laplacian of Gaussian gives you the smoothing effect, which smoothens out the noise and then takes the Laplacian of your original image. Once again, remember that we are just taking a filter and convolving the filter by moving it around at every point of the image and getting your output.

(Refer Slide Time: 10:56)

Laplacian of Gaussian



Original Image Laplacian Laplacian of Gaussian

What else can LoG do?

Credit: K Grauman

Vineeth N B (IIT-H) §2.2 Blob and Corner Detection

For accessing this content for free (no charge), visit : npTEL.ac.in

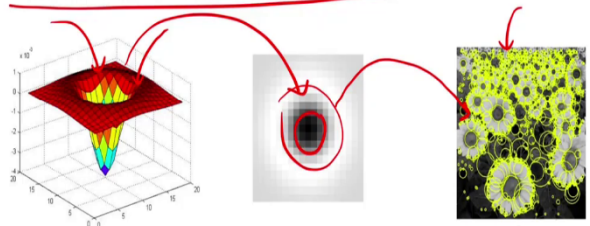


Now let us ask the question. So this is something that you partially saw the last thing. So what else can Laplacian of Gaussian do? Do you have ideas?

(Refer Slide Time: 11:12)

LoG as a Blob Detector


- Recall that convolution with a filter can be viewed as comparing a little "picture" of what you want to find against all local regions in the image.



- Observing the LoG filter matrix reveals that it is circularly symmetric. Thus it can be used for blob detection!

Credit: S Lazebnik

Vineeth N B (IIT-H) 2.2 Blob and Corner Detection



The other thing Laplacian of Gaussian can do is detect blobs. Why? Remember that a Laplacian of Gaussian filter looks somewhat like this. Remember, you could write the Laplacian this way or the other way, it really does not matter. So at the end of the day for edge detection and similar detection of other artifacts, they only take the absolute value, whether the output is negative or positive, we just take the absolute value.

So that would change if you had a white circle with a black hole or a black circle with the white hole. We ideally do not care, both of them are blobs to us and we ideally want to recognize blobs in both these cases. So you could write your Laplacian of Gaussian this way or another way where your central peak goes up on top and the other part comes below. This is similar to writing the Laplacian of Gaussian with a -4 in the center, and 1, 1, 1 around or a four in the center and -1, -1, -1 around.

So now if you try to visualize this as an image, so if you took a Laplacian of Gaussian filter, 3 by 3 maybe too small, let us say you expand it, take a larger neighborhood. Let us say you take a 7 by 7 Laplacian of Gaussian filter or 11 by 11 Laplacian of Gaussian filter, you would find that the Laplacian of Gaussian filter, just the filter itself remember that is also a matrix that also can be visualized as an image would look something like this, a black blob in the middle or white ring, which is where you have a peak and gray all the way in other places.

You could also have an invert of this, where you have white in the middle, a black ring, and then gray all over. Both of these are similar. And by looking at the filter, you can say that it is likely to detect blobs. In some sense, convolution of the filter can be viewed as comparing a


little picture of what you want to find against all local regions in the image. So there is a slight nuance which is different. This does look a bit similar to template matching for which you use cross correlation, but in convolution, you would double flip the filter and search for that across the image. But when your filter is symmetric, it really does not matter, both cross correlation and convolution will be looking for the same filter in the image.

So once you have a Laplacian of Gaussian like this, you can probably count sunflowers in a field or you can probably detect red blood cells in your blood test, any structure with blobs across an image. Clearly in this image with sunflowers, there are blobs of different sizes. So you would have to run a Laplacian of Gaussian with different blob sizes to be able to capture all of those blobs in the image.

(Refer Slide Time: 14:06)




From Blobs to Corners

◦ In the following image, what are some interesting features to choose?



Credit: K Grauman, R Urtasun

Vineeth N B (IIT-H) §2.2 Blob and Corner Detection



Let us now move forward to the next artifact, which is very useful to extract from images. For a large part extracting corners from an image was a very very important area of research in computer vision in the late 90s and early 2000s, a lot probably entire 90s and early 2000s.

So we will try to describe one popular method today and let us start by asking the question. Suppose we had an image such as this, what would be interesting features that set apart this image from any other image? Remember that if you want to do any processing, we want to be able to extract some unique elements of the image. So what are those unique elements in this image?




(Refer Slide Time: 14:53)

From Blobs to Corners

- Look for image regions that are unusual. How to define "unusual"?
- Textureless patches are nearly impossible to localize.
- Patches with **large contrast changes** (gradients) are easier to localize.
- But straight line segments at a single orientation suffer from the **aperture problem** (we'll see next slide), i.e., it is only possible to align the patches along the direction normal to the edge direction.
- Gradients in at least two (significantly) different orientations are the easiest, e.g., corners.

Credit: R Urtaşun

Vineeth N B (IIT-H) §2.2 Blob and Corner Detection



And you are going to say that those are going to be your corners, but let us try to go there. We ideally want to look for images, image regions that are unusual, something that sets apart that image, that region in the image from other images. If you have a region that is textureless, for example, the blue sky, then that could be common across several images and it may not really be unique to that particular image, you may not be able to localize that region in any other image with the similar content.

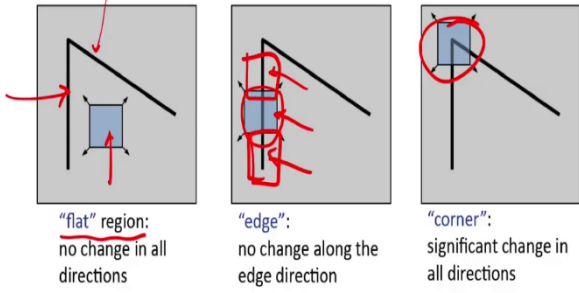
You are ideally looking for patches with large contrast changes, large gradients. Edges are a good example, but we will talk about why edges may not be the right artifact in a moment. But we are ideally looking for some patches or some regions in the image which have large contrast changes, such as gradients. But the problem with edges is edges suffer from what is known as the aperture problem. By aperture problem we will just see a visual illustration on the next slide, but while edges are good, unique aspects of a particular image, they do suffer from a problem, which we will talk about in the next slide.

So what we ideally are looking for are regions of the image or what we are going to call corners, where gradients in at least two different directions are significant. So remember an edge is an artifact where you have a significant gradient in one direction, which is going to be normal to that edge, it could be whatever direction the edge maybe, the normal direction to that edge is going to be the orientation of that edge. But we are saying, we want those points where there could be significant gradient in two directions. What does that mean? We will try to answer.

(Refer Slide Time: 16:50)

From Blobs to Corners

Consider a small window of pixels. How does the window change when you shift it?






"flat" region:
no change in all directions

"edge":
no change along the edge direction

"corner":
significant change in all directions

Credit: S Seitz, D Frolova, D Simakova, R Urtasun
Vineeth N B (IIT-H) 3.2 Blob and Corner Detection



Let us take a tangible example. Let us assume that there is an artifact such as this, this looks like some inverted V let us say. And we have such an artifact and we want to find out, which part of this image, let us assume this is the full image or ignore the blue box, the blue box is for explaining it to you. Just consider the image with just this inverted V. We want to find out now which aspect of the image is unique to it, which can help us recognize it, say other times, or if you view the image from other angles.

So if you consider this blue box placed here, you see that that is a flat textureless region. There is no change in intensity in that region. So it is not going to be very useful. It is like the blue sky with absolutely no change. It is not going to be very useful when you try to compare this image with other images.

So if you now place the blue box on the edge part of the image, this is good. There is some artifact that is useful, but the problem is if you move this patch here or here, it would all have the same response. You would never know whether you placed your box here or here or here, because all of them would have exactly the same response. And there is no difference in the local characteristics in all of these places.

So which means while edges are useful, there is something that they are lacking. So which means if you try to match or let us say you take a panoramic photo in your phone and you try to align two images, you may not know which part of the edge to align at. We are ideally looking for placing the box at that kind of a point where there is change in two directions and

that point could be unique to this particular image. And ideally, we are looking for many such points in an image, but such points are the kind of points that we want to detect in an image.

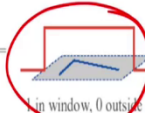
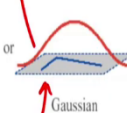
(Refer Slide Time: 18:59)

Autocorrelation

- In the previous slide, how to quantify the "significant" change of the window?
- Answer: **Autocorrelation function**. Compute the sum of squared differences between pixel intensities with respect to small variations in the image patch position.


$$E_{AC}(\Delta u) = \sum_{x,y} w(\mathbf{p}_i) |I(\mathbf{p}_i + \delta u) - I(\mathbf{p}_i)|^2$$

where $\mathbf{p}_i = (x, y)$, a particular position of the image.

Window function $W(x, y) =$  or 
1 in window, 0 outside Gaussian

Credit: R Urtasun

Vineeth N B (IIT-H) §2.2 Blob and Corner Detection



How do we find such points in an image? We know how to do edges now, but let us try to go one level further to try to see how do you find such corners in an image. To do that, we are going to define a quantity called auto correlation. As the name states it is autocorrelation, it is correlation with itself. So you are not going to use any external filter, we are going to take a patching image and see how it correlates with itself. What does that mean? Let us try to quantify that.

So we are going to define the auto correlation function as: you take a patch in the image and you compute the sum of squared differences between pixel intensities with small variations in the image patch position. So if you had a point p_i in the image, let us say you place it at the center, that is what is going to be p_i . And if you now have a small δu , which is going to be the difference, you are going to move that patch by a certain δu , remember p_i and δu are both 2D coordinate, so even Δu will probably have a δu and δv , for instance, they are going to be two dimensions there, it is a vector, which is going to be, so you take $I(p_i)$, which is the image intensity at that point p_i , then you move that point, $p_i + \Delta u$, you move it a little bit and you see what is the image intensity at that new location and you take the sum of squared differences for all points in that particular patch.

So if you take a square patch, take every point, move it a bit, so move that entire box a bit and then you compute pairwise distances between the same locations in the original patch and the new patch and sum them all up. This is what we define as autocorrelation.



About this function $w(p_i)$. $w(p_i)$ is a function that tells you how much you want to give weight for each particular point in that patch. Maybe for the central pixel, you want to give more weight. For a pixel at the periphery of that square of that blue square that you have, you may want to weight it a little lesser. So you could have a fixed weight for all points in that square or you could have a Gaussian which defines $w(p_i)$, where you weight the center pixel more and the pixels at the periphery a little less. We define this as auto correlation.


(Refer Slide Time: 21:39)

Computing Autocorrelation

- Using a Taylor Series expansion $I(p_i + \Delta u) = I(p_i) + \nabla I(p_i) \Delta u$ with the image gradient

$$\nabla I(p_i) = \left(\frac{\delta I(p_i)}{\delta x}, \frac{\delta I(p_i)}{\delta y} \right)$$



Vineeth N B (IIT-H)
§2.2 Blob and Corner Detection

Now let us try to see how do you compute autocorrelation and then come back to how do you compute corners using this autocorrelation. So let us look at this more deeply. Let us consider the Taylor Series expansion of your $I(p_i + \Delta u)$, remember $(p_i + \Delta u)$ is about taking the patch which was centered at p_i and moving it by a Δu and placing it at a slightly offset location in the same image. From Taylor Series expansion, you can write this as $I(p_i) + \nabla I(p_i) \Delta u$, where $\nabla I(p_i)$ is the image gradient at that particular location. $\left(\frac{\delta I(p_i)}{\delta x}, \frac{\delta I(p_i)}{\delta y} \right)$. We know how to compute the gradient, we have already seen that so far.

(Refer Slide Time: 22:35)

Computing Autocorrelation

- Using a Taylor Series expansion $I(\mathbf{p}_i + \Delta\mathbf{u}) = I(\mathbf{p}_i) + \nabla I(\mathbf{p}_i) \Delta\mathbf{u}$ with the image gradient

$$\nabla I(\mathbf{p}_i) = \left(\frac{\delta I(\mathbf{p}_i)}{\delta x}, \frac{\delta I(\mathbf{p}_i)}{\delta y} \right) \rightarrow (\delta u, \delta v)$$

- Autocorrelation can be approximated as:




$$E_{AC}(\Delta\mathbf{u}) = \sum_{x,y} w(\mathbf{p}_i) [I(\mathbf{p}_i + \delta\mathbf{u}) - I(\mathbf{p}_i)]^2$$

$$\approx \sum_{x,y} w(\mathbf{p}_i) [I(\mathbf{p}_i) + \delta I(\mathbf{p}_i) \delta\mathbf{u} - I(\mathbf{p}_i)]^2$$

$$= \sum_{x,y} w(\mathbf{p}_i) [\delta I(\mathbf{p}_i) \delta\mathbf{u}]^2$$

$$= \Delta\mathbf{u}^T \mathbf{A} \Delta\mathbf{u}$$

Credit: R Urtasun
Vineeth N B (IIT-H) §2.2 Blob and Corner Detection

Now let us write out autocorrelation. Remember the definition of autocorrelation that we wrote on the previous slide is this one, where we said, autocorrelation is $w(\mathbf{p}_i) [I(\mathbf{p}_i + \delta\mathbf{u}) - I(\mathbf{p}_i)]^2$. Now we are going to replace $I(\mathbf{p}_i + \delta\mathbf{u})$ with this Taylor Series expansion. So plugging that in here, you are going to have $w(\mathbf{p}_i)$, this first term here gets written as $I(\mathbf{p}_i) + \delta I(\mathbf{p}_i) \delta\mathbf{u} - I(\mathbf{p}_i)$.

So just to explain the notations here. By $\delta(u,v)$ mean a vector, $\delta\mathbf{u}$ and say $\delta\mathbf{v}$, because of this, we have taken out a summation here, we are simply writing it as one of those $\delta\mathbf{u}$'s inside. So anyway, there is going to be a summation for $\delta\mathbf{v}$, which will take care of the other component of $\delta\mathbf{u}$.

Once you have this, you can see that $I(\mathbf{p}_i)$ and $I(\mathbf{p}_i)$ gets canceled and you are left with summation over x, y , $w(\mathbf{p}_i) [\delta I(\mathbf{p}_i) \delta\mathbf{u}]^2$. And we are going to write that as a quantity, $\Delta\mathbf{u}^T \mathbf{A} \Delta\mathbf{u}$.

So remember that you have this $[\delta\mathbf{u}]^2$, you just split it into two parts where you have one part that comes here and one part that goes here and the rest of what you have $w(\mathbf{p}_i) \delta I(\mathbf{p}_i)$. You combine that into a matrix called \mathbf{A} .

How would \mathbf{A} look?

(Refer Slide Time: 24:23)

Computing Autocorrelation




- The autocorrelation is $E_{AC}(\Delta u) = \Delta u^T \mathbf{A} \Delta u$ with

$$\mathbf{A} = \sum_u \sum_v w(u, v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = w * \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$
- The weighted summations have been replaced with discrete convolutions with the weighting kernel w .
- The eigenvalues of \mathbf{A} reveal the amount of intensity change in the two principal orthogonal gradient directions in the window.

$$\mathbf{A} = \mathbf{U} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \mathbf{U}^T \quad \text{with} \quad \mathbf{A} \mathbf{u}_i = \lambda \mathbf{u}_i$$

Credit: R Urtasun

Vineeth N B (IIT-H)
§2.2 Blob and Corner Detection

How would \mathbf{A} look? \mathbf{A} would look like something like this. Remember \mathbf{A} is going to be a combination of w into your δu s, δI s, so it means \mathbf{A} is going to look something like this, $w(u, v)$, you are going to have remember two components of the summation. $I_x^2, I_x I_y, I_x I_y, I_y^2$, those are going to be your gradients and that is going to be your \mathbf{A} matrix.

So we took the definition. So we defined autocorrelation in a particular manner where we said, we will take a patch, move it a bit and see how the image properties change in that local region. So we then took the definition of autocorrelation, played with it a little bit and then came up with an expansion which looks somewhat like this and we are not focusing on this matrix \mathbf{A} .

So this matrix \mathbf{A} , so δu is simply the change that you imposed in the patch location, that is what you imposed. \mathbf{A} is what is giving you how the gradients changed between those two patches and a w weighting factor that tells you how much you should weight the central part of that patch versus the peripheral parts.



So since \mathbf{A} is what defines the intensity change, we are going to consider an eigen decomposition of \mathbf{A} , which is given by $\mathbf{u} \boldsymbol{\lambda} \mathbf{u}^T$, where $\boldsymbol{\lambda}$ is going to be a diagonal matrix with λ_1 and λ_2 . Two eigenvalues, remember \mathbf{A} is a two dimensional matrix, which means the maximum number of eigenvalues you can have is two, where $\mathbf{A} \mathbf{u}_i = \lambda \mathbf{u}_i$, your standard eigen decomposition, which is wonderful.

So once again started with autocorrelation, wrote it in a slightly different manner and then now we are done in eigen decomposition of A. Where do we go from here? How do we go from here to finding a column?


(Refer Slide Time: 26:29)

Computing Autocorrelation

- How do the eigenvalues determine if an image point is a corner?

Credit: N Snavely, R Urtasun
Vineeth N B (IIT-H) §2.2 Blob and Corner Detection

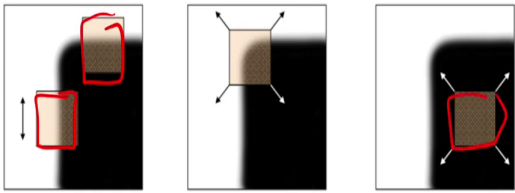


That is the question we are asking. Think about it for a moment. How do you think you can go from the eigenvalues of A to a position of a column?

Very similar to the discussion that we had at the early part of the lecture, when both λ_1 and λ_2 are large. You know that the intensity changes in both directions at those points, when either λ_2 is much greater than λ_1 or λ_1 is much greater than λ_2 , it is going to be an edge because there is going to be change only along one direction. And if both λ_1 and λ_2 are small, you are going to say it is a flat or textureless region. So which means we know that from the eigen decomposition of A, all what we are looking for is both the eigenvalues to be height and we know we have probably hit a corner.

(Refer Slide Time: 27:35)

Computing Autocorrelation



"edge":

$$\lambda_1 \gg \lambda_2$$

$$\lambda_2 \gg \lambda_1$$

"corner":

$$\lambda_1 \text{ and } \lambda_2 \text{ are large}$$

$$\lambda_1 \sim \lambda_2$$

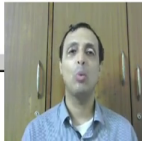
"flat" region

$$\lambda_1 \text{ and } \lambda_2 \text{ are small;}$$

Credit: K Grauman, R Urtasun

Vineeth N B (IIT-H)

§2.2 Blob and Corner Detection



So let us see how you actually do this inference. So it is another way of looking at it is, so wherever there is a vertical edge or a horizontal edge, you are going to have either λ_1 greater than λ_2 or λ_2 greater than λ_1 . At a corner, we are going to have both λ_1 and λ_2 to be large and in a flat region, you are going to have both λ_1 and λ_2 to be very small.

(Refer Slide Time: 28:02)

Harris Corner Detector

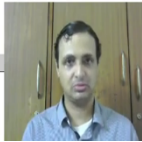


- Compute gradients at each point in the image.
 - Compute **A** for each image window to get its cornerness scores.
 - Compute the eigenvalues/compute the following function M_c
- $$M_c = \lambda_1 \lambda_2 - \kappa(\lambda_1 + \lambda_2)^2 = \det(A) - \kappa \text{trace}^2(A)$$
- Find points whose surrounding window gave larger cornerness response ($M_c > \text{threshold}$)
 - Take points of local maxima, perform non-maximum suppression.

Credit: R Urtasun

Vineeth N B (IIT-H)

§2.2 Blob and Corner Detection



So what do we do with this? So the way we are going to compute your corner, this was a method given by a person called Harris and that is why it is known as the Harris Corner Detector, it is a very popular detector. It was used for several years. Of course, there have been lots of improvements and better methods that people have developed but this was one of the earliest corner detectors that was developed and was used for many years. So the entire procedure follows something like this.

You compute gradients at each point in the image. Using that, you compute your A matrix. You can use a weighting function, or if you do not use a weighting function, you just assume that you are going to consider all of them to be equal, all of the patch position, positions of that patch to be equal. Then we ideally want to compute your eigenvalues and decide your cornerness based to the eigenvalue, but because eigen decomposition by itself can be a costly process, we try to do a slight deviation to be able to compute our cornerness measure.

So we are going to define a cornerness measure as: $\lambda_1\lambda_2 - \kappa(\lambda_1+\lambda_2)^2$. I will let you work this out to show that when this entire quantity is high, you will know that both those λ_1 s and λ_2 s are high. Work this out for yourself. Try out different λ_1 s and λ_2 s and you will see what I am saying. But what is interesting here is, $\lambda_1\lambda_2$ is nothing but the determinant of A and $(\lambda_1 + \lambda_2)$ is nothing but the trace of A, which means we can define our cornerness measure as $\det(A) - \kappa \cdot \text{trace}^2(A)$. κ is just a constant that you have to define to get what you want to get. For different images, you may have to set kappa differently.

Why is this useful? You no more need to compute the eigen decomposition of A matrix, you only need to compute your determinant and trace, which is a bit easier than computing your eigen decomposition.

Finally, you then take all points in the image whose cornerness measure is greater than the threshold. So you can take lots of different points, probably all points, compute their cornerness using autocorrelation, and then whichever is greater than a particular threshold you actually call them corners. You can finally also perform non-maximum suppression, where if you find that there are many corners in a very small local neighborhood, you pick the one with the highest cornerness measure. That is your non-maximum suppression that we also talked about in the canny edge detector. So this non-maximum suppression will keep coming back to us at various stages in this course in various use cases.

(Refer Slide Time: 31:05)

Harris Corner Detector: Example



Credit: K Grauman, R Urtasun

Vineeth N B (IIT-H)

§2.2 Blob and Corner Detection

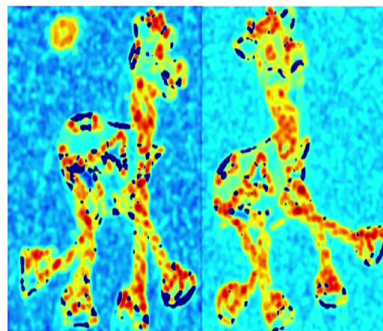


So, here is a visual illustration of the Harris corner detector. So, let us consider 2 images of the same object taken from different angles. The objects are at different poses and the illumination is also different. Let us try to run the Harris corner detector. Ideally what we want is that in both these images the same parts or the same corners in the doll get picked.

Why is that relevant? Once again if I take the example of say stitching different images in your cell phone a panoramic mode or something like that, we want the same points in both those objects to be located, so that they can be matched and probably stitched together to get a panoramic image. Just as an example of an application.

(Refer Slide Time: 31:53)

Computing Cornerness



Credit: K Grauman, R Urtasun

Vineeth N B (IIT-H)

§2.2 Blob and Corner Detection



Let us see it now. So here is the step of computing the cornerness where you do auto correlation, get your cornerness values.




(Refer Slide Time: 32:02)

Finding High Response



Credit: K Grauman, R Urtasun

Vineeth N B (IIT-H) §2.2 Blob and Corner Detection



And then you take all the high responses while using the threshold.

(Refer Slide Time: 32:06)

Non-max Suppression



Credit: K Grauman, R Urtasun

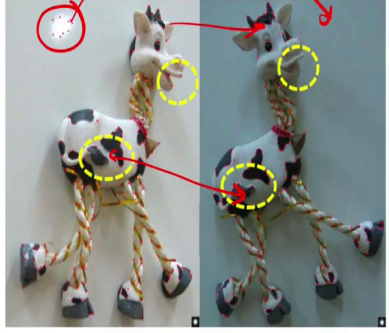
Vineeth N B (IIT-H) §2.2 Blob and Corner Detection



You do non-maximum suppression, that is why you see many of those points in regions you just take the highest cornerness value, you do a non max suppression and you get a bunch of different points.




(Refer Slide Time: 32:21)

Results



Credit: K Grauman, R Urtasun

Vineeth N B (IIT-H) §2.2 Blob and Corner Detection



And you let us try to visualize them on the image and you can actually see that if you look at say these highlighted examples, you can see that, although those are fairly differently tilted, you can get similar corners in both of these regions. You can see many other regions too. Obviously, you get a few more corners which are not there in the second image but those can be overcome at the matching face as you see a bit later on this course. The focus of this

particular lecture is on cornerness, just detecting the cornerness. How do you do the matching between the two images will come back to that a bit later.

(Refer Slide Time: 33:03)

Harris Corner Detector: Variants

- Harris and Stephens '88 is rotationally invariant and downweights edge-like features where $\lambda_1 \gg \lambda_0$.

$$\det(\mathbf{A}) - \alpha \text{trace}(\mathbf{A})^2 = \lambda_0 \lambda_1 - \alpha (\lambda_0 + \lambda_1)^2$$




- Triggs '04 suggested $\lambda_0 - \alpha \lambda_1$.
- Brown et al. '05 use harmonic mean:

$$\frac{\det(\mathbf{A})}{\text{trace}(\mathbf{A})} = \frac{\lambda_0 \lambda_1}{\lambda_0 + \lambda_1}$$

which is smoother when $\lambda_0 \approx \lambda_1$

Credit: R Urtasun

Vineeth N B (IIT-H)
§2.2 Blob and Corner Detection

While we saw one variant of the Harris corner detector here, which was developed by Harris and Stephens in 88 where we use the determinant minus κ or α times your trace that is what we used here, there have been other improvisations of the same method where a researcher, Triggs, suggested that you can use $\lambda_0 - \alpha \lambda_1$ where λ_0 is the first eigen value of the larger eigen value and λ_1 is the second eigen value. A researcher, Brown and a team, proposed $\frac{\det(A)}{\text{trace}(A)}$ instead of doing the $\det(A) - \text{trace}^2(A)$.

We said κ on two slides ago, that just does not matter just a constant. So you have $\frac{\det(A)}{\text{trace}(A)}$ all of these could be different base of playing around with the same quantities to get what you want. All of them effectively try to measure the cornerness using the eigen values of your A matrix which comes from autocorrelation.




(Refer Slide Time: 34:14)

Harris Corner Detector: Properties

- Scale-invariant?

Credit: R Urtasun

Vineeth N B (IIT-H) §2.2 Blob and Corner Detection



Let us ask a few questions about properties of the Harris corner detector before concluding this lecture. The first question we are going to ask is, Is the Harris corner detector scale invariant? What do we mean by scale? Remember, scale is complementary to resolution. So if you have an object as which is very small in image, we call that smaller scale or if it is large, we call that larger scale. So, you could have an artifact such as this curved line here in a particular image or you could also have this curved line on a larger canvas, maybe you just took one image from close up and another image further out. But it is the same image, it is the same object that is being taken.

So, in this particular case if you observe, in the second image this point would be considered a corner by the Harris corner detector. In the first image if you took the same size patch to do the autocorrelation, you would find that all of these would get categorized as edges and none of them would get categorized as corners, which means the Harris corner detector need not necessarily be scale invariant. How do you make it scale invariant will talk about that later.

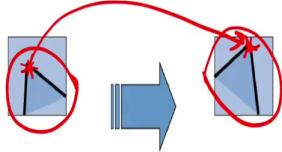
(Refer Slide Time: 35:36)

Harris Corner Detector: Properties

- Rotation-invariant?


$$\mathbf{A} = w * \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = \mathbf{U} \begin{bmatrix} \lambda_0 & 0 \\ 0 & \lambda_1 \end{bmatrix} \mathbf{U}^T \quad \text{with} \quad \mathbf{A} \mathbf{u}_i = \lambda_i \mathbf{u}_i$$

- Relative cornerness remains the same!



Credit: N Snavely, R Urtasun

Vineeth N B (IIT-H) §2.2 Blob and Corner Detection

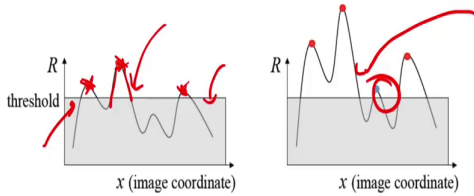


What about rotation invariant? Is the Harris corner detector rotation invariant? It happens that it would be rotational invariant. So, whether you have this particular inverted V like this or whether you rotate it and have the inverted V like this in an image, this particular corner will have high change in both directions in both cases and you would detect this corner both in this image as well as this image, as long as there is no change in scale, you would detect the same corner on both images and Harris corner detector in this particular case would be rotation invariant.

(Refer Slide Time: 36:19)

Harris Corner Detector: Properties


- Photometric change: Affine intensity change $I = aI' + b$
- Only derivatives are used, so it's invariant to shift $I = I' + b$.
- What about intensity scale?



Partially invariant to affine intensity change

Credit: K Grauman, R Urtasun

Vineeth N B (IIT-H) §2.2 Blob and Corner Detection



What about other kinds of invariances? What about photometric change, what does a photometric change mean? A photometric change is an affine intensity change. An affine

intensity change means that you take your intensity at every pixel of the image, scale the intensity by value a and may be translated by a value b . Remember that intensity of value lying between 0 on to 255 or if you normalized it, it comes to between 0 and 1 . You just multiply all of those values by a scalar say a , it could be less than 1 if you like, and also add a quantity b if you like. We call that as a photometric change or an affine intensity change.

So, in this particular case if you notice, for only the translation, it really does not matter, you would find the corner whether you increase the intensity in all places by an amount or otherwise it would still find the corner. But about the scaling part of the intensity change, by multiplying it by an a , it depends on the threshold that you choose. If you had a particular curve, any function for that matter, and you had a certain threshold, all these points which have a certain value, remember that the points closer to that would get suppressed due to non max suppression you will only have the peak in all of these points.

So, those peaks would get detected. But if you scale the curve, there could be newer points that get added. If you scale it up in a different way, there could be newer points that get added. So, which means it perhaps invariant to translation but not necessarily invariant to scaling the intensity. A couple of slide back we spoke about invariant to scale with respect to size of the artefacts itself, now we are talking about scaling with respect to the intensity value. Hope you see that these are two different things. In this case, you could still have a curve which is small but now you brighten it or darken it in a different way, and that is what we mean as an photometric change in this particular context.

(Refer Slide Time: 38:32)

The screenshot shows a presentation slide with the following content:

- Slide title: Homework Readings
- Section: Homework
- Section: Readings
 - Chapter 2, Szeliski, *Computer Vision: Algorithms and Applications*
- Section: Questions: Linear Algebra review
 - Show that the trace of a matrix is the sum of its eigenvalues.
 - Show that the determinant of a matrix is the product of its eigenvalues.

Logos for NPTEL and an open book icon are visible on the right side of the slide. A video inset in the bottom right corner shows the presenter, Vineeth N B, speaking.

So, the homework that you are going to have is to go ahead and continue to read your chapter 2 of the Szeliski's book and since a lot of the concepts that we used in this lecture were based on Eigen values, Eigen vectors, just go ahead and rush up your linear algebra, try to show that the trace of a matrix is the sum of its eigen values and the determinant of a matrix is a product of its eigen values.