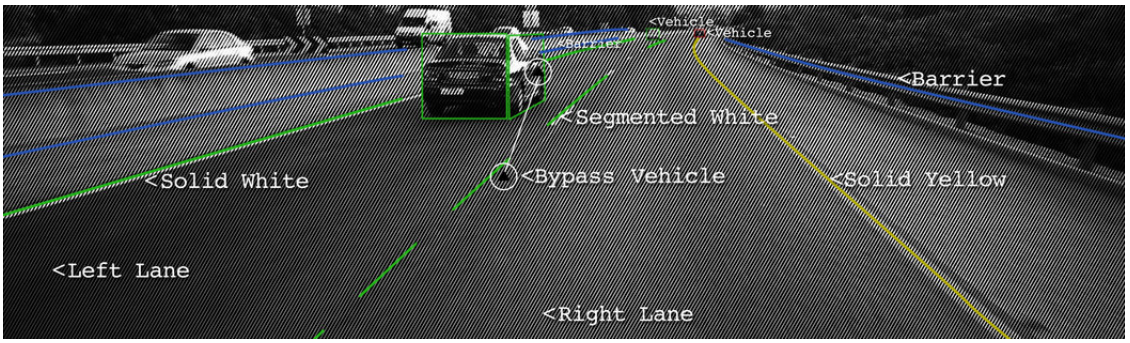
EE 046746 - Technion - Computer Vision

Tutorial 01 - Feature Detection



- Image source

Agenda

Reminder - Derivatives and Edges

- An edge is a place of rapid change in the image intensity function.

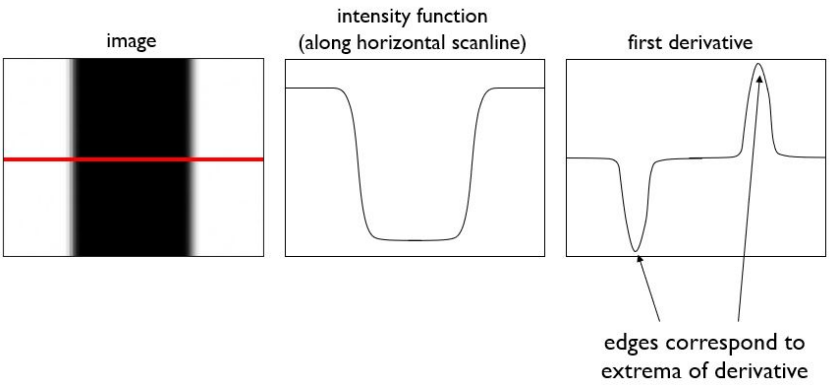
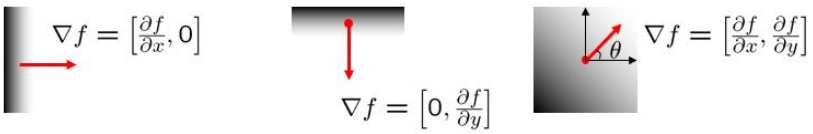


Image Gradient

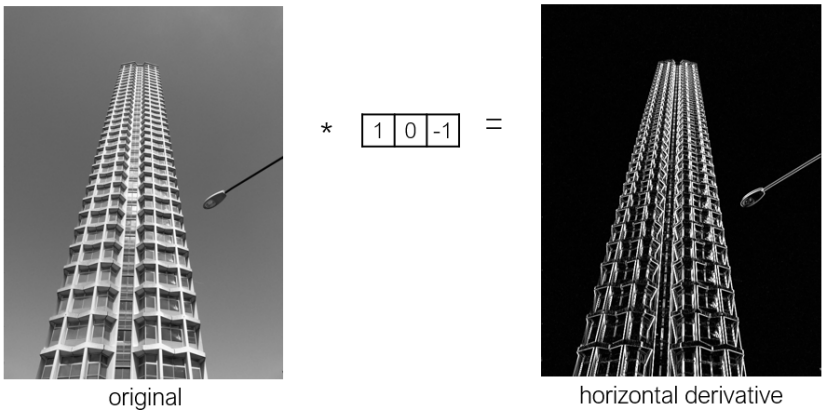
- The gradient of an image: $\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}]$
- The gradient points in the direction of most rapid change in intensity

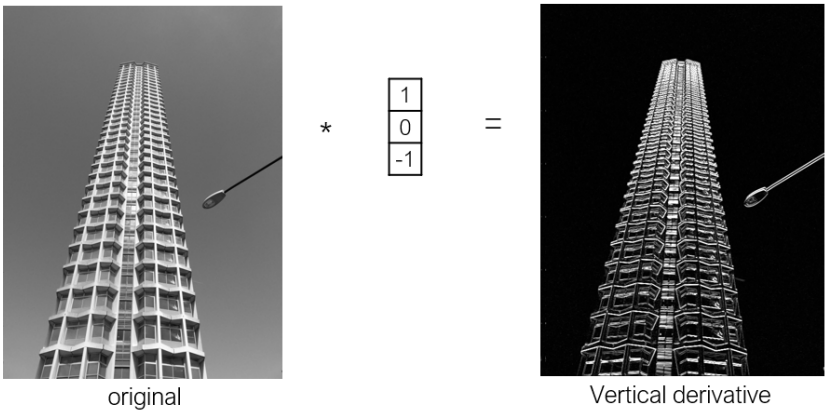


- The gradient direction (orientation of edge normal) is given by: $\theta = tan^{-1}(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x})$
- The edge strength is given by the gradient magnitude

$$||\nabla f|| = \sqrt{(\frac{\partial f}{\partial y})^2 + (\frac{\partial f}{\partial x})^2}$$

Image derivatives can be implemented as a convolution:

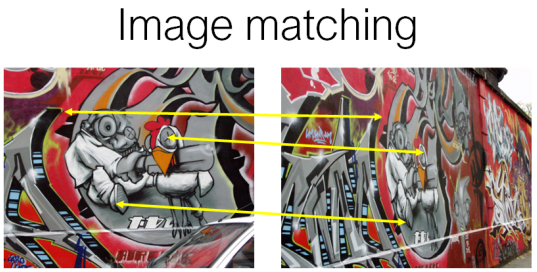


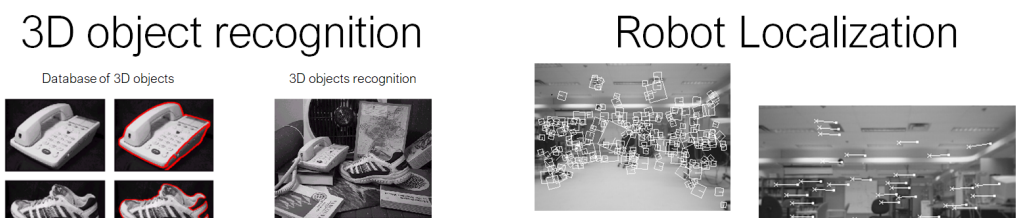


Corner Detection

Although edges are good for human perception, and contain most of the information, they are not allways good enough. Another, more useful building block for many computer vision applications is a corner detector. Corners are better features for tasks such as:

- Image alignment (homography, fundamental matrix)
- 3D reconstruction
- Motion tracking
- Object recognition
- Indexing and database retrieval
- Robot navigation





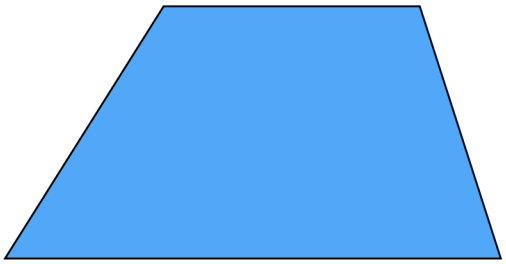
Why Corners?

suppose we select the yellow window, which contains an edge, as a feature. we want to find it in an image from a different angle:

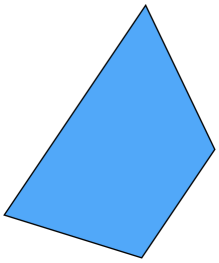
Where is the match?

There are multiple windows with similar content, just by looking at the local content we won't be able to tell which one matches.

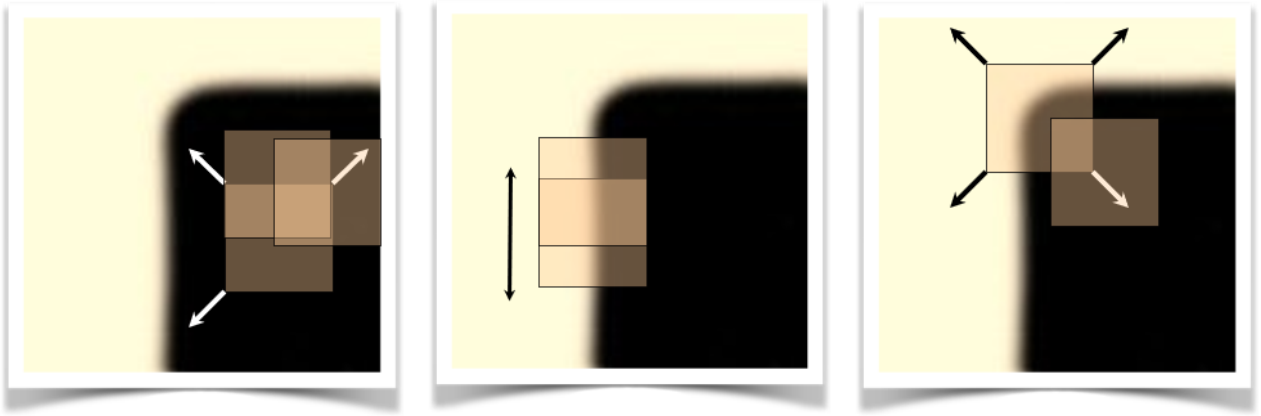
Let's see another example. Pick a point for a feature in the following image:



Now find it again in the next image:



What type of feature would be a good selection? Corners of course! Corners are easily recognised by looking at a small window. Shifting the window in any direction should give large change in intensity.



"flat" region: no change in all directions

"edge": no change along the edge direction

"corner": significant change in all directions

Harris Corner Detector

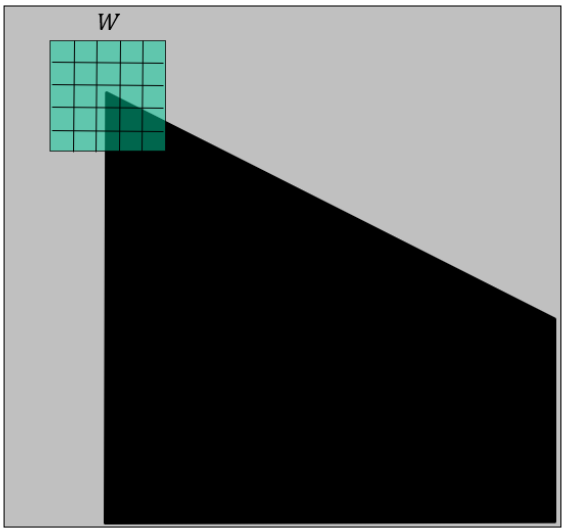
Given a window W of nxn pixels, we perform the following steps to determine whether it's a corner:

1. Compute image gradients over small region
2. Subtract the mean from each image gradient
3. Compute the covariance matrix
4. Compute eigenvectors and eigenvalues
5. Use threshold on eigenvalues to detect corners

We slide a window through the image and perform all of these steps for each window. We will discuss each step separately.

1. Compute image gradients over a small region

For our window W, we compute the derivative in each direction (x and y), which generates two nxn arrays W_x, W_y



array of x gradients

$$W_x^{i,j} = \frac{\partial W^{i,j}}{\partial x}$$

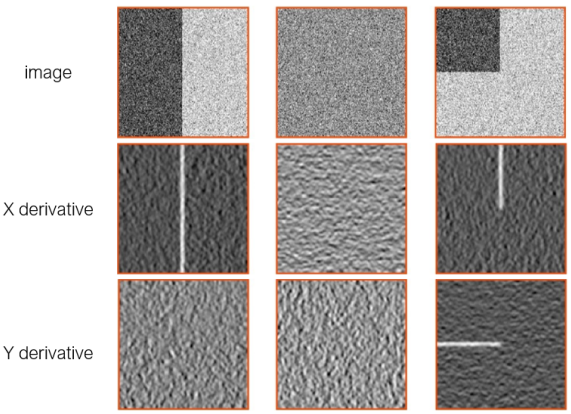
W_x

array of y gradients

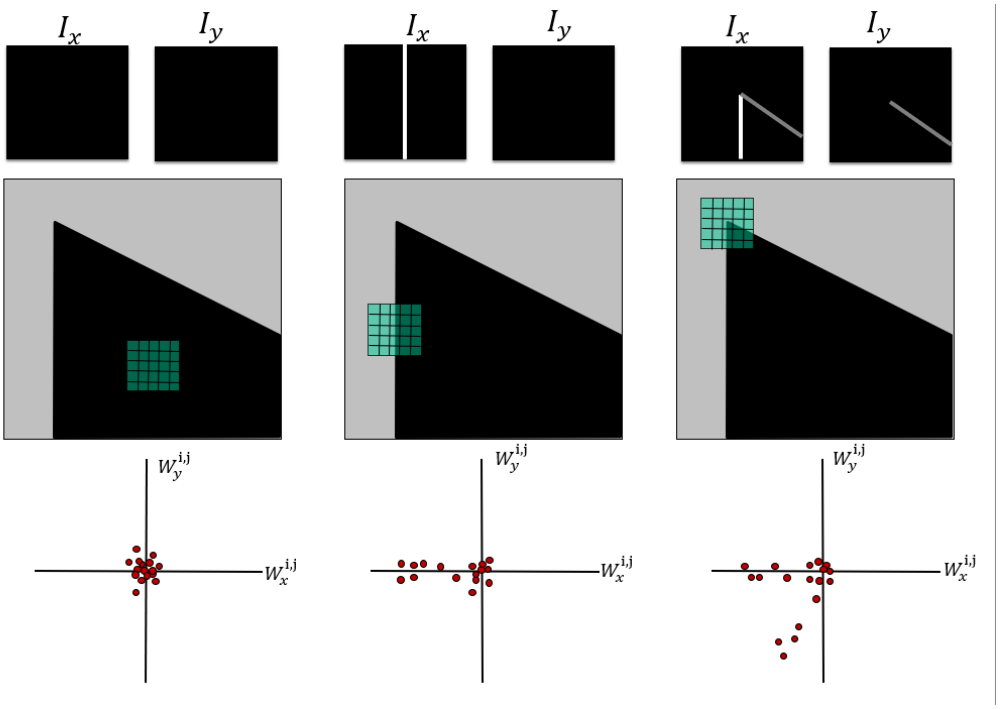
$$W_y^{i,j} = \frac{\partial W^{i,j}}{\partial y}$$

W_y

Recall that we get high response in W_x, W_y if there is an edge on the corresponding direction.



Note: Since we are going to perform this process on a sliding window, we are going to need to compute the gradient at all points any way.In practice it will be more efficient to compute the gradients for the entire image and extract the current window from the derivatives image.
For each pixel i, j in the window, we have a gradient vector $(W_x^{i,j}, W_y^{i,j})$. We would like to look at the distribution of these gradients for the window. We can plot the n^2 gradients on a 2d plane, let's see some examples:



on the left pach, we have a flat region, therefore the derivative in both directions are small, and most of the points will be around the origin (assume there is some noise, that's why gradients are not exactly zero)

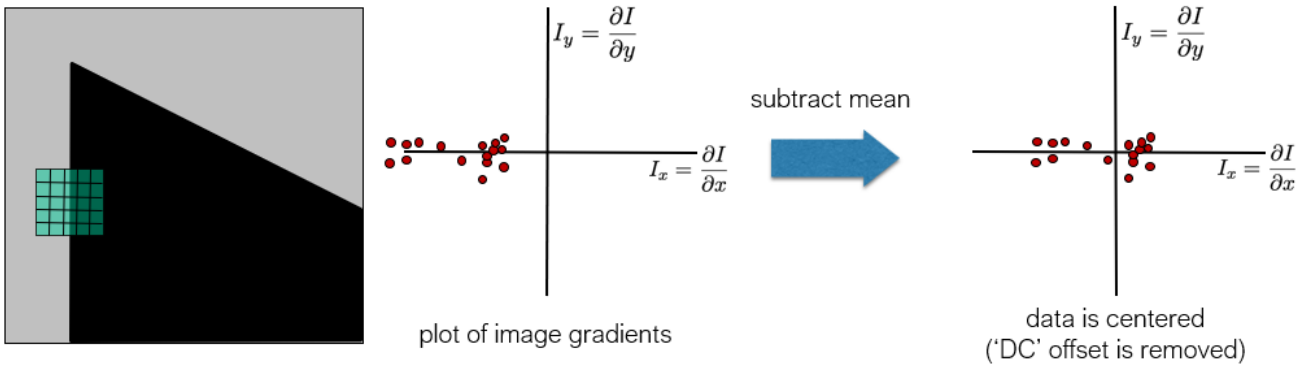
on the central patch, we have a vertical edge, therefore the x derivative is high on some points, we get a distribution around the x axis.

on the right patch, there are some pixels with vertical only gradient and some pixels with a diagonal gradient, so we get points spread along two lines.

2D plot are nice, but we want a way to quantify the results and determine if there's a corner in the patch. We are going to do that by looking at the covariance matrix of the gradients.

2. Subtract the mean from each image gradient

To define the covariance we first compute the mean of the data and subtract it, so that the data is centered around 0.



3. Compute the covariance matrix

For a general distribution, given a set of N 2D vectors $v^1, v^2, \ldots$ (after mean subtraction) the covariance is defined as

$$\Sigma = \frac{1}{N}\sum_{j=1}^{N} v^j v^{jT} = \frac{1}{N}\begin{bmatrix} \sum_{j=1}^{N} v_x^j v_x^j & \sum_{j=1}^{N} v_x^j v_y^j \\ \sum_{j=1}^{N} v_x^j v_y^j & \sum_{j=1}^{N} v_y^j v_y^j \end{bmatrix}$$

Note that there are 3 sums in the covariance matrix. In our case this covariance matrix is:

$$\Sigma = \frac{1}{N}\sum_{i,j\in 1\ldots n} g^j g^{jT} = \frac{1}{n^2}\begin{bmatrix} \sum_{i,j\in 1\ldots n} W_x^{i,j^2} & \sum_{i,j\in 1\ldots n} W_x^{i,j} W_y^{i,j} \\ \sum_{i,j\in 1\ldots n} W_x^{i,j} W_y^{i,j} & \sum_{i,j\in 1\ldots n} W_y^{i,j^2} \end{bmatrix}$$

The sums can be computed the following way:

$$\sum_{i,j \in 1...n} W_x^{i,j^2} = \text{sum}(\ \underbrace{\square}_{\text{array of x gradients}} .* \underbrace{\square}_{\text{array of x gradients}}\ )$$

$$\sum_{i,j \in 1...n} W_x^{i,j} W_y^{i,j} = \text{sum}(\ \overset{W_x}{\underbrace{\square}}_{\text{array of x gradients}} .* \overset{W_y}{\underbrace{\square}}_{\text{array of y gradients}}\ )$$

$$\sum_{i,j \in 1...n} W_y^{i,j^2} = \text{sum}(\ \overset{W_y}{\underbrace{\square}}_{\text{array of y gradients}} .* \overset{W_y}{\underbrace{\square}}_{\text{array of y gradients}}\ )$$

where .* means matrix elementwise multipication.

4. Compute Eigenvalues

Reminder

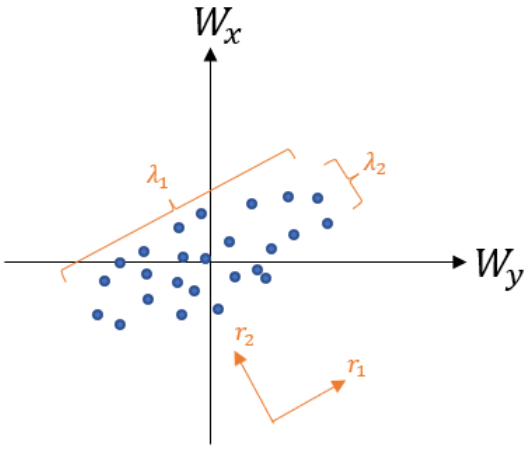
Since the covariance matrix is symetric (and PSD = Positive Semi Defined), it can be decomposed:

$$\Sigma = R^T \Lambda R = \begin{bmatrix} -\bar{r_1}^T- \\ -\bar{r_2}^T- \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \begin{bmatrix} \bar{r_1} & \bar{r_2} \\ | & | \end{bmatrix}$$

For a symetric PSD matrix this is called SVD - Singular Value Decomposition. Where λ_1 and λ_2 are the eigenvalues, and R is an orthogonal matrix (can say it's a rotation matrix), built from orthogonal eigenvectors.

Note that in this decomposition $R^{-1} = R^T$

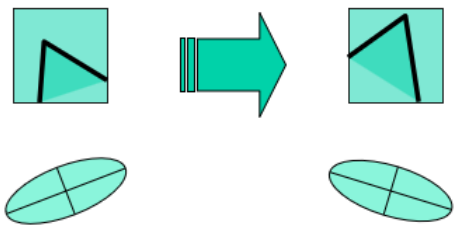
In a 2D covariance matrix, the eigenvectors point to the directions with the highest and lowest variance. The eigenvalues are the corresponding variances.



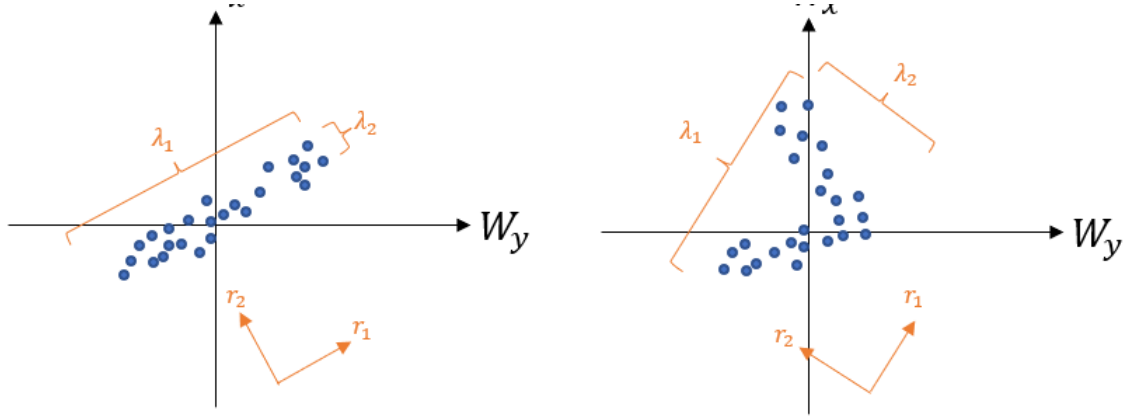
Back to corners

In the decomposition $\Sigma = R^T \Lambda R$ the eigenvectors (columns of R) are the directions where there is maximal and minimal variance in the gradients of the window, and the corresponding eigenvalues are the variance. That means that if we rotate the patch, only R will change, but the eigenvalues will remain the same.

We want to use the eigenvalues for corner detection, this will also promise us that our detector will be invariant to rotations.

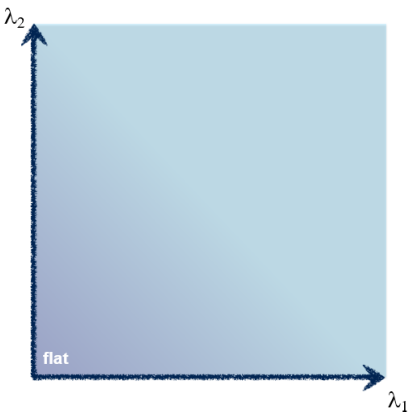


In a corner, both eigenvalues should be high

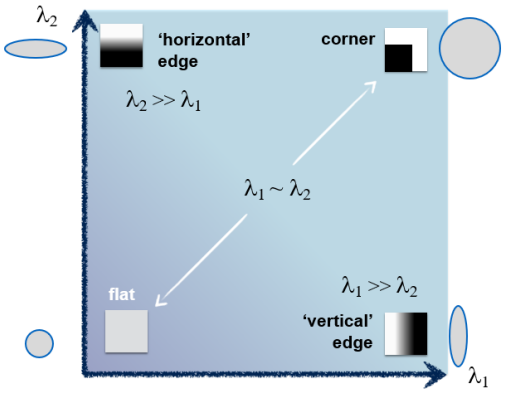


5. Use threshold on eigenvalues to detect corners

We have the eigenvalues, and we want to decide whether our patch is a corner. Where do we want (λ_1, λ_2) to be?

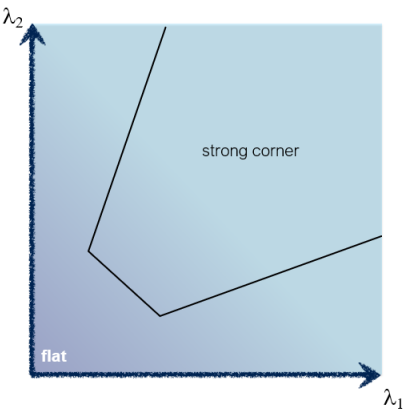


- In a flat window, we have low variance in any direction, which means both lambda's will be close to zero.
- In an edge, we have high variance on one direction but low variance on the orthogonal direction, resulting one lambda high and one low.
- In a corner, we have high variance in two directions (our ellipse will become a large circle), both lambda will be high.



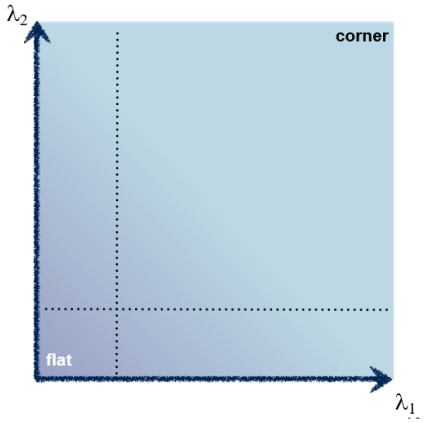
Clarification: we can get $\lambda_1 << \lambda_2$ or $\lambda_2 << \lambda_1$ in a non vertical or horizontal edge as well. If we have a diagonal edge, one eigenvalue will represent the variance on it's direction and one on the orthogonal direction. This is due to how the SVD works.

Can you think of a good score function $R(\lambda_1, \lambda_2)$ for a window?



One such a function we can use is the min eigenvalue. If it is high then we have a corner.

$$R = min(\lambda_1, \lambda_2)$$



Note: In some formulations the SVD decompoziation is defined such that the eigenvalues are in descending order, this way we can have threshold only on λ_2

The problem with the minimum function, is that we need to know the eigenvalues. performing the computation is quite expensive. Recall that we have to do it many times on a sliding window. We can use some tricks to find a cheaper score function.

Recall that the sum of eignevalues of matrix equals to it's trace, and their multipications to it's determinant. for 2x2 matrix:

$$\Sigma = \begin{bmatrix} \sigma_1 & \sigma_2 \\ \sigma_3 & \sigma_4 \end{bmatrix}$$

$$det(\Sigma) = \lambda_1\lambda_2 = \sigma_1\sigma_4 - \sigma_2\sigma_3$$
$$trace(\Sigma) = \lambda_1 + \lambda_2 = \sigma_1 + \sigma_4$$

We don't have the eigenvalues, but we have their multipication and sum. we can define another, less exprensive to compute, score function:

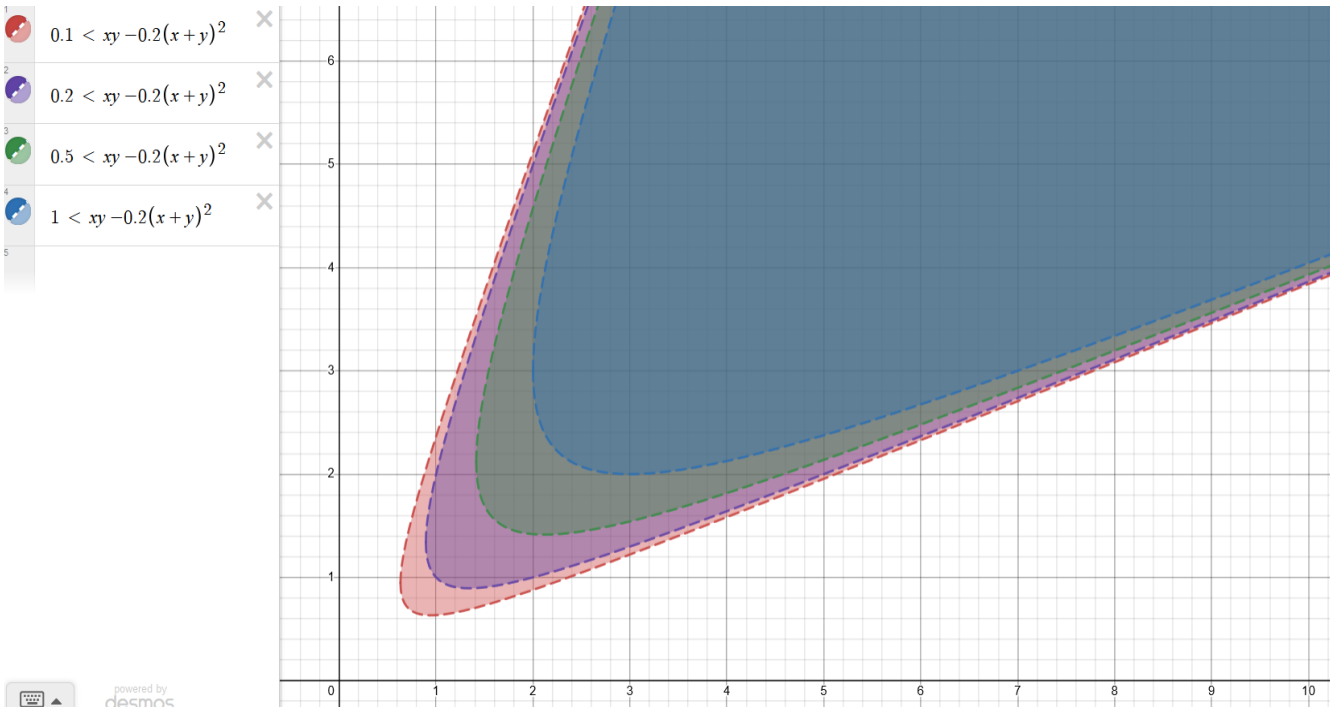
$$R = \lambda_1\lambda_2 - \kappa(\lambda_1 + \lambda_2)^2$$
$$\kappa < 0.25$$

Let's look at the function, intuitively

- If $\lambda_1 = \lambda_2 = 0$ we get R = 0
- If $\lambda_1 > 0, \lambda_2 = 0$, then $\lambda_1\lambda_2 = 0, (\lambda_1 + \lambda_2)^2 > 0$ we get R<0
- If $\lambda_1 = \lambda_2 > 0$ we get $R = \lambda_1^2(1 - 4\kappa) > 0$

So this score is high when both eigenvalues are high

Here is an example with $\kappa = 0.2$ and different thresholds for the score:



https://www.desmos.com/calculator/oi9q42yvt4

Harris Conclusion With Vectorized Operations

When working with images we want as match as possible to define our operations with matrices. This is a more efficient way to implement with high level programming languages, and this is the only way to make use of computation accelerators.

Let's conclude the Harris Detector, with more efficient vector operations. Given an image I:

1

Compute x and y derivatives

$$I_x = G^x * I$$

$$I_y = G^y * I$$

G^x and G^y are gaussian derivatives filters, which perform the blur and derivation simultaneously.

2

compute the product of the derivatives for every pixel

$$I_{x^2} = I_x \cdot I_x$$

$$I_{y^2} = I_y \cdot I_y$$

$$I_{xy} = I_x \cdot I_y$$

(we use elementwise product here)

3

compute the sum of the products for a window around each pixel. That can also be done with convolution with a gaussian blur filter to remove noise.

$$S_{x^2} = G_\sigma * I_{x^2}$$

$$S_{y^2} = G_\sigma * I_{y^2}$$

$$S_{xy} = G_\sigma * I_{xy}$$

4

Compute the score for each window. The score is defined by the eigenvalues of the covariance matrix:

$$\Sigma(x,y) = \begin{bmatrix} S_{x^2}(x,y) & S_{xy}(x,y) \\ S_{xy}(x,y) & S_{y^2}(x,y) \end{bmatrix}$$

but we don't need to compute it for the score, just the determinant and trace for each x,y. We can do it with one vectorized operation for all pixels. $\overline{\overline{R}}$ will be a 2d image with the score for each pixel (and the window around it).

$$\overline{\overline{R}} = det(\Sigma) - \kappa(trace(\Sigma))^2 = S_{x^2}S_{y^2} - S_{xy}^2 - \kappa(S_{x^2} + S_{y^2})^2$$

5

threshold on value of R to determine where are the corners.

Invariance to Image Deformation

Recall that we want to detect the same corner features in different images of the same object. These images may undergo various deformations:

- Change in view point
- Zoom (scale)
- Illumination
- Rotation
- ...

We already know that Harris Detector is invariant to rotation, since eigenvalues of the covariance matrix dont change under rotation, just the rotation matrix (the eigenvectors).

What about illumination? Illumination usually is descrived as affine intensity change.

for an image $I(x,y)$ we could define such transformation with parameters a, b as $I'(x,y) = aI(x,y) + b$.

Since Harris uses derivatives of the image, it won't be affected by the constant intensity change b, but it's not invariant to the change in scale caused by a.

What about scale? Harris is **not** invariant to scale!

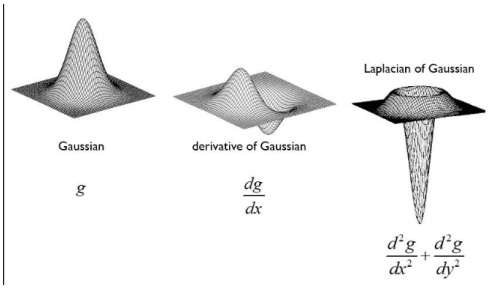
We will soon see what can we do about it.

Laplacian Feature Detector

Ther laplacian of an image is a version of a "scalar multidimensional second derivative". Given an image (or a window) $I(x, y)$ the laplacian is defined as:

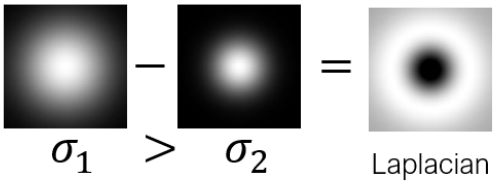
$$L(I) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

Laplacian filter is often combined with gaussian filter to blur the image. Applying gaussian blur and then laplacian on an image, is equivalent to applying the laplacian on the gaussian, and then convolve the resulted filter on the image. This way we get the LoG - Laplacian of Gaussian filter.

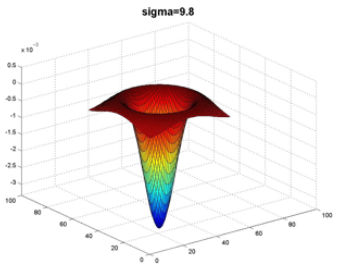


Instead of computing the second derivative of the gaussian, it is simpler and more efficient (we will see why soon) to just subtract two gaussians with different variances to approximate it.

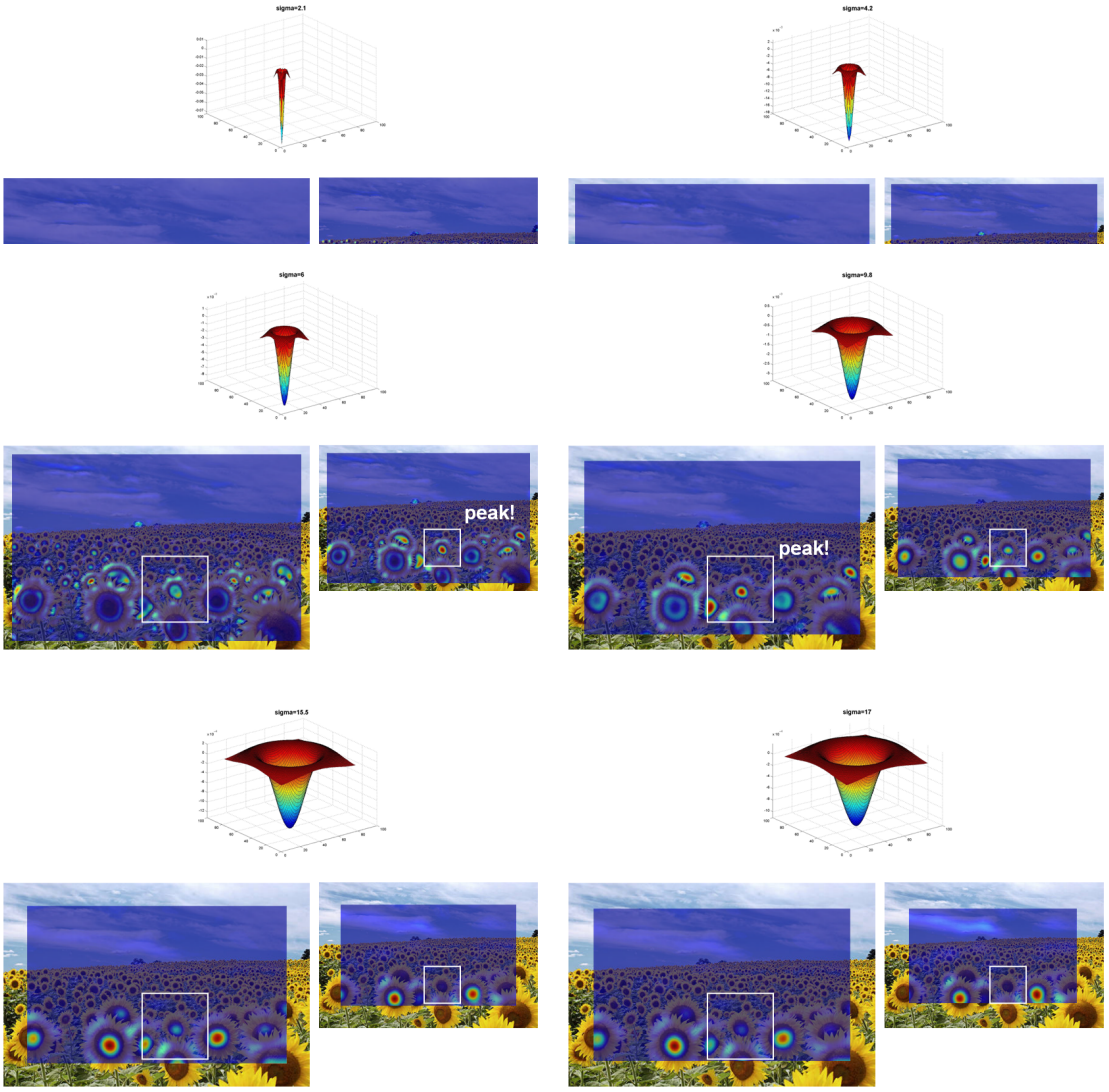
$$G_{\sigma_1} * \mathrm{I} - G_{\sigma_2} * \mathrm{I} = \left(G_{\sigma_1} - G_{\sigma_2}\right) * \mathrm{I}$$



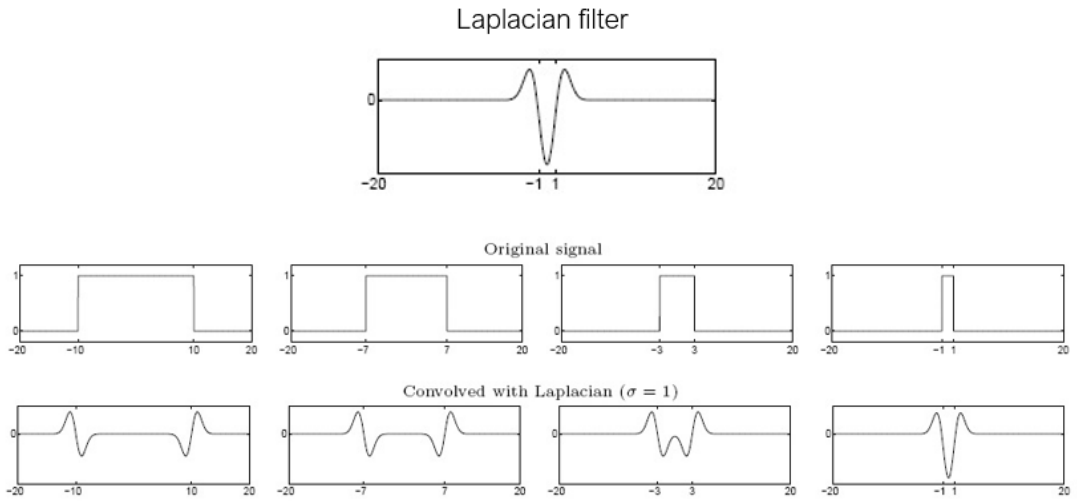
This filter have high response for blobby circular image areas.



The laplacian, like the Harris Detector, is affected by scale. a laplacian filter with different sigmas will have high response for different patches depending on their scale. Note the difference for several variances, on the same image with different scale:



Response is highest when signal has the same characteristic scale as the filter.



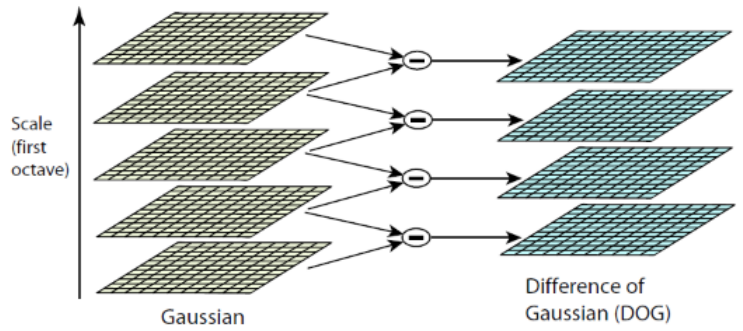
Multi-scale Detection

We are going to see multi-scale detection on Laplacian (DoG) feature detector. Same ideas can be applied on the Harris Detector, but the laplacian is a bit easier to visualize.

Multi-scale Laplacian

How can we make a feature detector scale-invariant? We can look for features in multiple scales, and find local maxima in both position and scale.

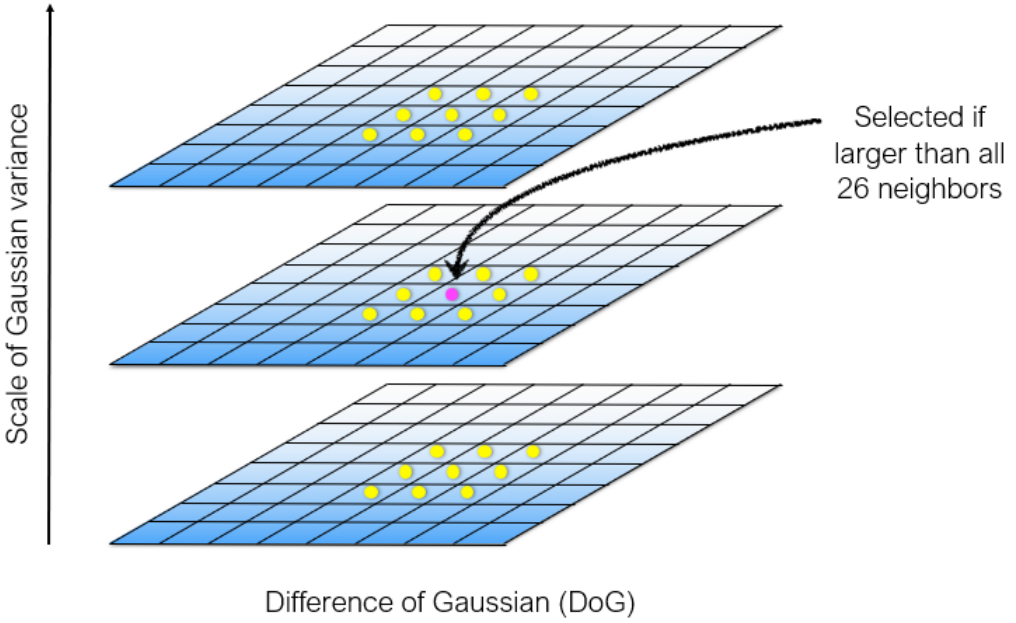
To compute multiple laplacians for an image, we first compute gaussians with different scales (variances), and then subtract each gaussian image from it's neighbor. We get multiple laplacians images with different scales. This is called DoG pyramid.

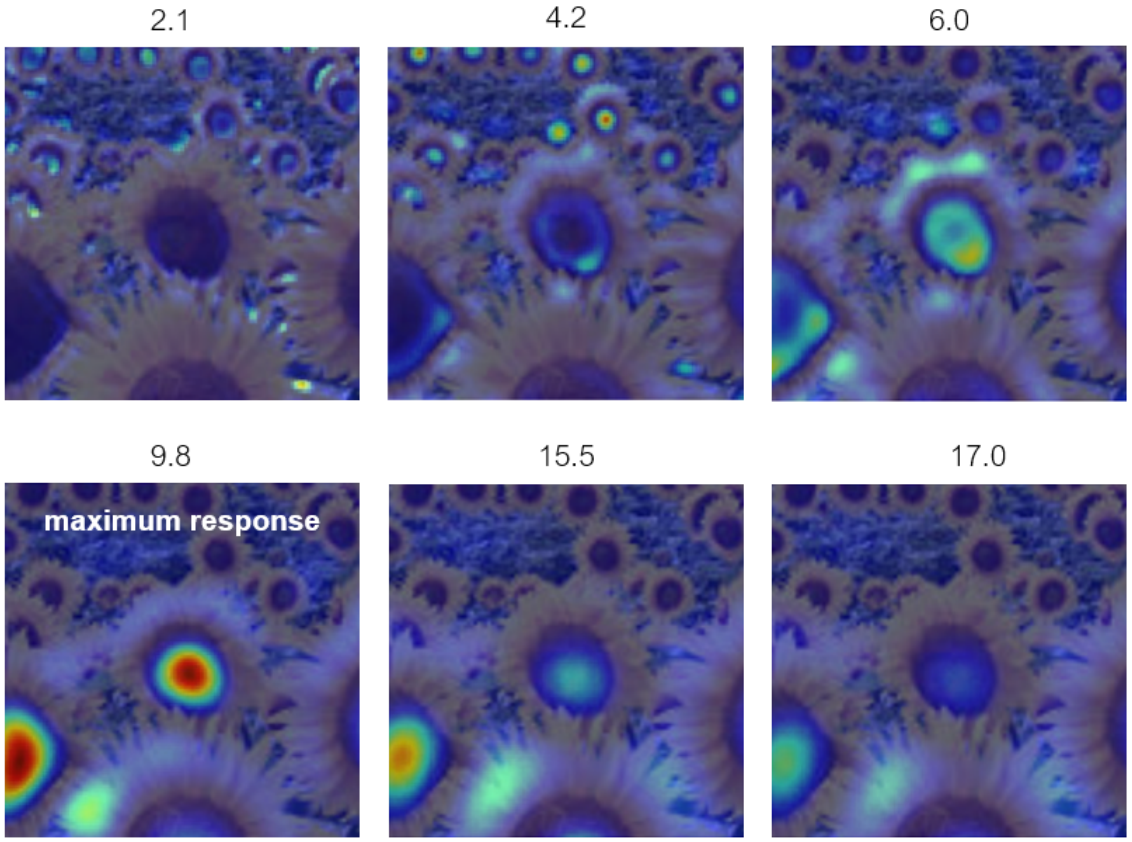


(a)

In normal Laplacian Detector, we would have chosen features at points with local maximal response to the filter in space (x and y).

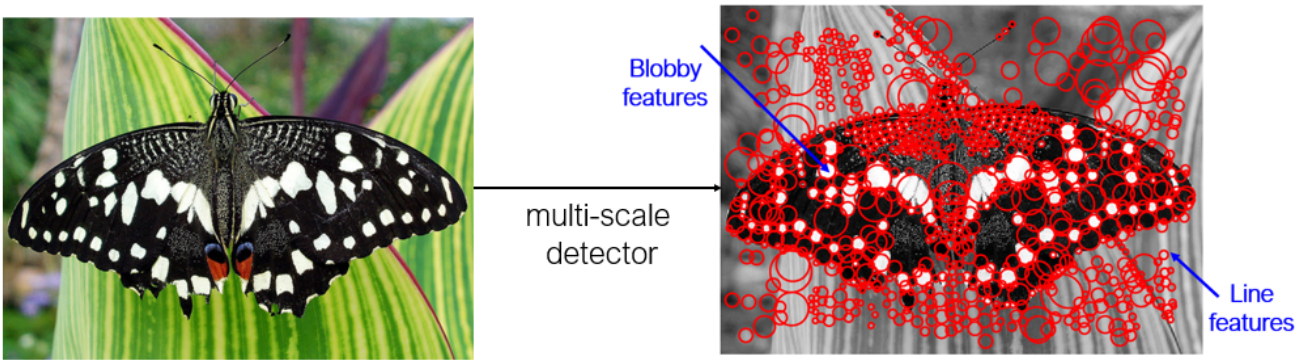
In multi-scale laplacian, we will choose maximal response in space and scale (x,y and scale):





Filtering Edge Features

Laplacian filter has high response to circles, but it also has quite a high response for straight lines and edges, which we don't want as features, although they are local maximum points.



Detection is usually followed by cleaning stage to maintain only informative features. Usually using techniques similar to those we saw in Harris Detector: look at eigenvalues of the covariance matrix of local derivatives.

Multi-scale With Harris

We have seen multi-scale detection of Laplacian (difference of Gaussian) features. One can also define a Harris corner detector in a multi-scale way: change the scale of the derivative filter and change the size of the window over which we average gradients. We can then look for local extrema in the scale space. For various historical reasons, the popular SIFT descriptor you will see in the lecture detects keypoints using difference of Gaussians rather than Harris.

Recommended Videos

Warning!

- These videos do not replace the lectures and tutorials.
- Please use these to get a better understanding of the material, and not as an alternative to the written material.

Video By Subject

- Edge Detection (Sobel + Canny) - CSCI 512 - Lecture 09-1 Edge Detection | CSCI 512 - Lecture 09-2 Edge Detection
  - Edge Detection (Sobel) - Finding the Edges (Sobel Operator) - Computerphile
  - Edge Detection (Canny) - Canny Edge Detector - Computerphile -

Credits

- Slides - Anat Levin
- EE 046746 Spring 2020 - Dahlia Urbach
- Tutorial: Build a lane detector - Chuan-en Lin
- Icons from Icon8.com - https://icons8.com