


浙江大学爱丁堡大学联合学院

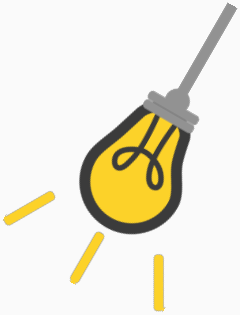
ZJU-UoE Institute

Image features

Nicola Romanò - nicola.romano@ed.ac.uk

Learning objectives

- Define and give examples of image features
- Describe and apply methods for detection of shapes in an image
- Describe and apply methods for particle detection
- Discuss advantages and issues with these methods



Features

Feature, n.: a distinctive or characteristic part of a thing; some part which arrests the attention by its conspicuousness or prominence.
(*Source: Oxford English Dictionary*)

Features

Feature, n.: a distinctive or characteristic part of a thing; some part which arrests the attention by its conspicuousness or prominence.
(*Source: Oxford English Dictionary*)

In image analysis, a **feature** is a characteristic of the image, or of a part of it, that defines some of its properties.

Examples of features include edges, corners, ridges, texture, colour, shape etc.

Features

Feature, n.: a distinctive or characteristic part of a thing; some part which arrests the attention by its conspicuousness or prominence.
(*Source: Oxford English Dictionary*)

In image analysis, a **feature** is a characteristic of the image, or of a part of it, that defines some of its properties.

Examples of features include edges, corners, ridges, texture, colour, shape etc.

Detecting features is a fundamental step to extract information from images.

Features in images

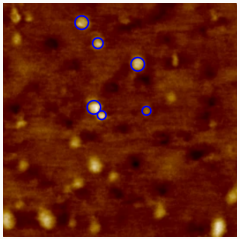
Figure: View in the Gran Sasso mountain range, Italy - Photo: Nicola Romanò

Which are the prominent features of this image?

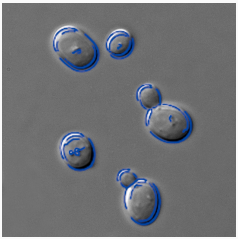
Outline

Geometric features

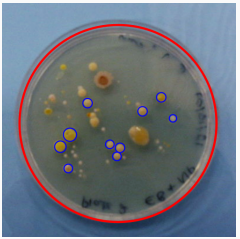
Examples of **geometric features** include:



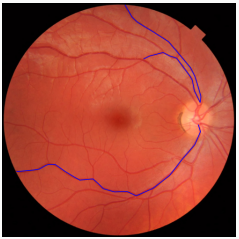
(a) Blobs - Marsh et al., Sci Rep 2018



(b) Corner and edges (see lecture 5) - Credits: Masur, Wikipedia, CC-0



(c) Lines, circles, etc. - Credits: Vivien Rolfe, Flickr, CC-BY-SA-2.0



(d) Ridges - Credits: Medical gallery of Mikael Häggström., CC-0

Outline

Blob detection

Blob detection is the task of detecting blobs in an image.

Blobs are defined as regions of pixels with uniform properties that are clearly distinguishable from the background.

There are many methods for detecting blobs in an image; today we will look at using the Laplacian of the Gaussian (**LoG**) method, and its approximation by the difference of Gaussians (**DoG**) method.

The Laplacian operator

In the last lecture we used the first-order derivatives of the image (the gradient) to detect edges. This time we will look at the second order derivatives.

The Laplacian operator

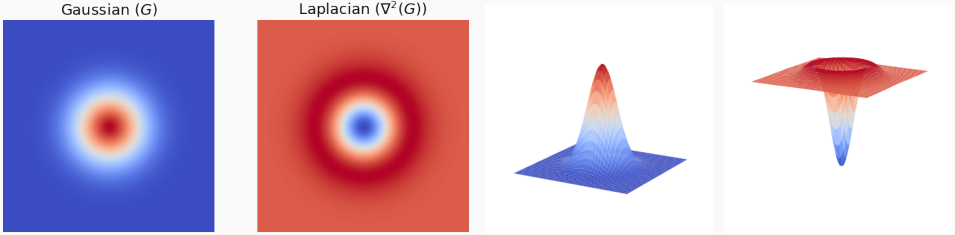
In the last lecture we used the first-order derivatives of the image (the gradient) to detect edges. This time we will look at the second order derivatives.

We define the **laplacian** of an image as the sum of the second order partial derivatives of the image.

$$\text{Laplacian(I)} = \nabla^2(I) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

The Laplacian of the Gaussian

What does the Laplacian of a Gaussian function look like?



So... how do we use this to detect blobs?

The LoG method for blob detection

The idea is to:

1. Apply a Gaussian filter $G(\sigma)$ to the image
2. Calculate the Laplacian of the Gaussian-filtered image

The LoG method for blob detection

The idea is to:

1. Apply a Gaussian filter $G(\sigma)$ to the image
2. Calculate the Laplacian of the Gaussian-filtered image
3. Dark blobs on light background will appear as bright spots in the Laplacian of the Gaussian-filtered image; bright blobs on dark background will appear as dark spots.

The LoG method for blob detection

The idea is to:

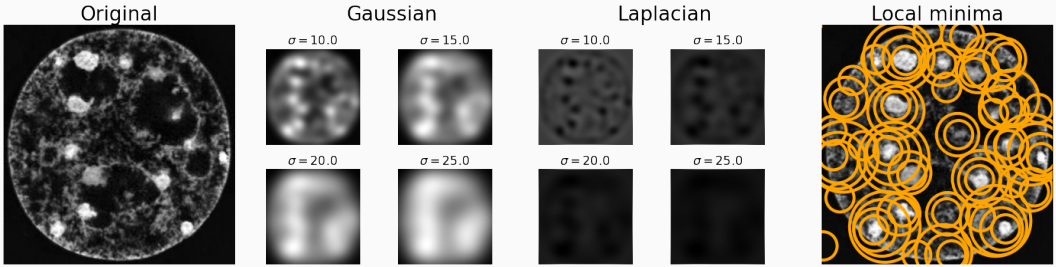
1. Apply a Gaussian filter $G(\sigma)$ to the image
2. Calculate the Laplacian of the Gaussian-filtered image
3. Dark blobs on light background will appear as bright spots in the Laplacian of the Gaussian-filtered image; bright blobs on dark background will appear as dark spots.
4. We find local maxima (or minima) in the Laplacian of the Gaussian-filtered image to determine the position of the blobs
5. The blob size is approximately $\sqrt{2}\sigma$

The DoG method for blob detection

To detect blobs of different size, we can use a Gaussian filter with different standard deviations.

The DoG method for blob detection

To detect blobs of different size, we can use a Gaussian filter with different standard deviations.

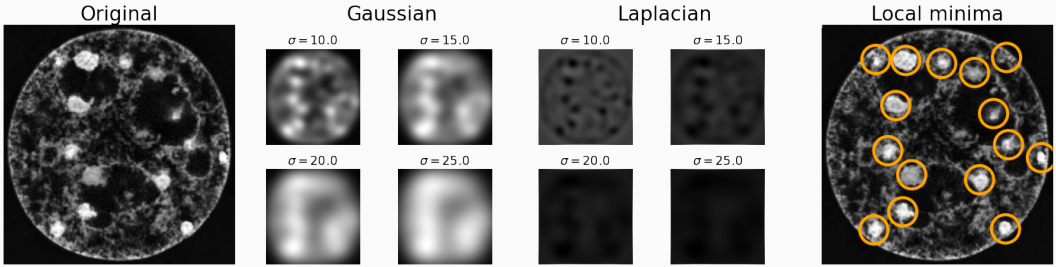


We can apply a threshold to the DoG image to remove spurious blobs and can also remove blobs overlapping over a certain %.

.

The DoG method for blob detection

To detect blobs of different size, we can use a Gaussian filter with different standard deviations.

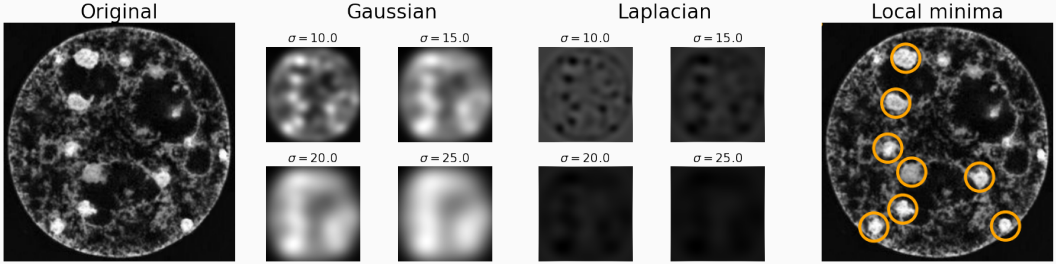


We can apply a threshold to the DoG image to remove spurious blobs and can also remove blobs overlapping over a certain %.

.

The DoG method for blob detection

To detect blobs of different size, we can use a Gaussian filter with different standard deviations.

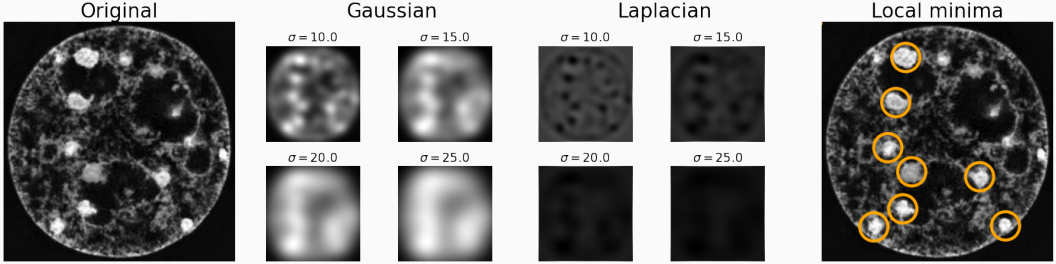


We can apply a threshold to the DoG image to remove spurious blobs and can also remove blobs overlapping over a certain %.

.

The DoG method for blob detection

To detect blobs of different size, we can use a Gaussian filter with different standard deviations.



We can apply a threshold to the DoG image to remove spurious blobs and can also remove blobs overlapping over a certain %.

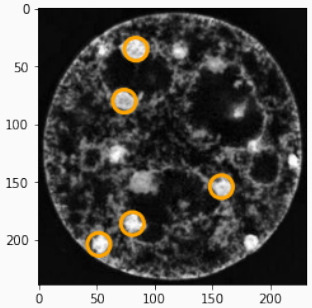
Scikit image implements these steps in the `skimage.feature.blob_log` function.

Example of blob detection

```
from skimage.feature import blob_log
from skimage.io import imread

nucleus = imread("nucleus_blob.png")
blobs = blob_log(nucleus, min_sigma=10, max_sigma=15,
                 num_sigma=5, threshold=0.3, overlap=.4)
```

Example of blob detection

```
from skimage.feature import blob_log
from skimage.io import imread

nucleus = imread("nucleus_blob.png")
blobs = blob_log(nucleus, min_sigma=10, max_sigma=15,
                 num_sigma=5, threshold=0.3, overlap=.4)
plt.imshow(nucleus, cmap="gray")

for b in blobs:
   y, x, radius = b
   # Create a circle at the blob center
   c = plt.Circle((x, y), radius, color='orange',
                  linewidth=3, fill=False)
   # Draw the circle (gca = "get current axis")
   plt.gca().add_artist(c)
plt.show()
```



Other methods for blob detection

Scikit image impements two other methods for blob detection: the DoG (Difference of Gaussians) and the DoH (Determinant of the Hessian) methods.

Other methods for blob detection

Scikit image impements two other methods for blob detection: the DoG (Difference of Gaussians) and the DoH (Determinant of the Hessian) methods.

The **DoG method** is a fast approximation of the LoG method. It works by calculating the difference between two version of the image blurred by Gaussians with different σ.

The **DoH method** is the fastest method. It works by calculating the determinant of the Hessian of the image (the Hessian is a matrix of second partial derivatives). Contrary to DoG and LoG, the detection speed is independent of the size of the blobs. However, DoH is less accurate at detecting small blobs.

Other methods for blob detection

Scikit image impements two other methods for blob detection: the DoG (Difference of Gaussians) and the DoH (Determinant of the Hessian) methods.

The **DoG method** is a fast approximation of the LoG method. It works by calculating the difference between two version of the image blurred by Gaussians with different σ.

The **DoH method** is the fastest method. It works by calculating the determinant of the Hessian of the image (the Hessian is a matrix of second partial derivatives). Contrary to DoG and LoG, the detection speed is independent of the size of the blobs. However, DoH is less accurate at detecting small blobs.

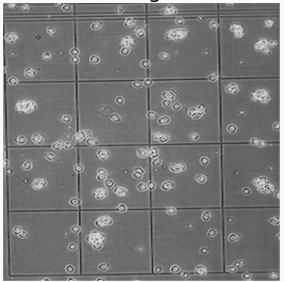
These are implemented in `skimage.feature.blob_dog` and `skimage.feature.blob_doh` respectively.

Outline

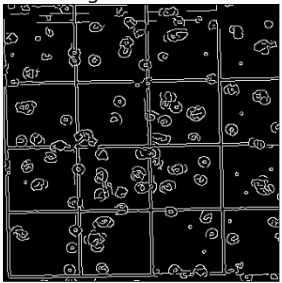
The Hough transform

The Hough transform is a technique originally developed by Paul Hough in 1962 for finding straight lines in an image. Extensions of this technique allow to find circles and quadrilaterals.

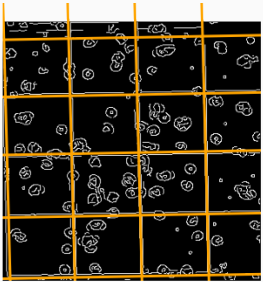
Image



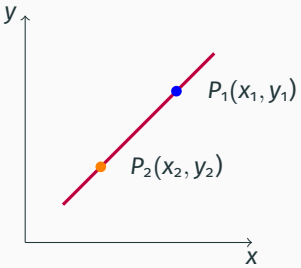
Edge detection



Detected lines

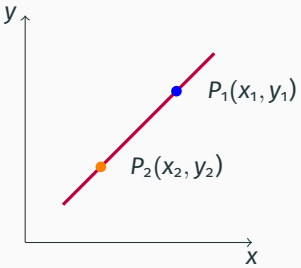


Representing a line

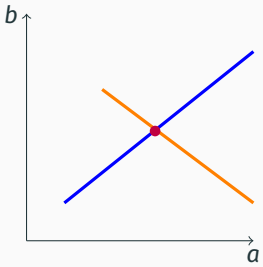


Consider a line with equation: $y = a \cdot x + b$ and two points P_1 and P_2 on it.

Representing a line

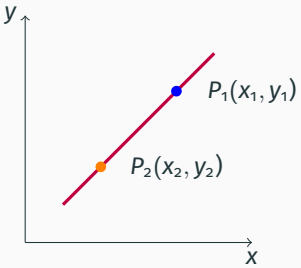


Consider a line with equation: $y = a \cdot x + b$ and two points P_1 and P_2 on it.

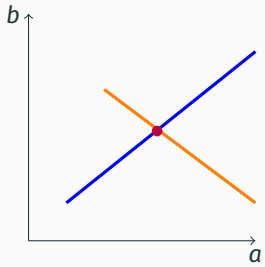


The families of lines passing through P_1 (or P_2) have equation: $b = -x_1 \cdot a + y_1$ (or $b = -x_2 \cdot a + y_2$).

Representing a line



Consider a line with equation: $y = a \cdot x + b$ and two points P_1 and P_2 on it.



The families of lines passing through P_1 (or P_2) have equation: $b = -x_1 \cdot a + y_1$ (or $b = -x_2 \cdot a + y_2$).

The intersection is the slope and intercept of the original line! Any point on the red line on the left will correspond to a line passing through the red point in the a/b space.

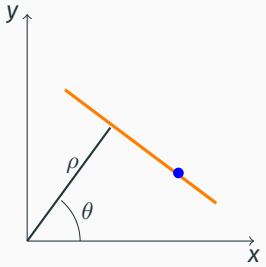
This is the principle of the Hough transform.

Representing a line - another way

We can also represent a line in terms of an angle and a distance from the origin. This is a better alternative, because we can also use it to represent a vertical line.

$$x \cdot \cos(\theta) + y \cdot \sin(\theta) = \rho$$

Where θ is the angle of the line and *rho* is the distance from the origin.

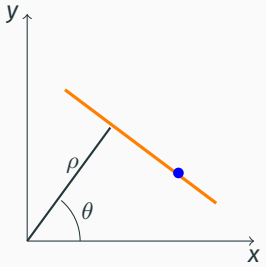


Representing a line - another way

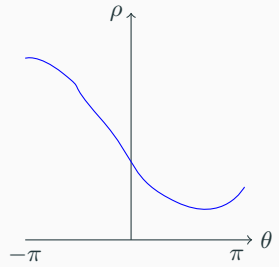
We can also represent a line in terms of an angle and a distance from the origin. This is a better alternative, because we can also use it to represent a vertical line.

$$x \cdot \cos(\theta) + y \cdot \sin(\theta) = \rho$$

Where θ is the angle of the line and *rho* is the distance from the origin.



We can map each point in the θ/ρ space (called the Hough space) as we did before.



The Hough transform

There are three steps in the Hough transform:

1. Edge detection - e.g. using the Canny edge detector. This gives us a binary image.

The Hough transform

There are three steps in the Hough transform:

1. Edge detection - e.g. using the Canny edge detector. This gives us a binary image.
2. We create a *MxN* matrix of 0s, representing the Hough space. This will correspond to *M* different values of ρ and *N* different values of θ.

The Hough transform

There are three steps in the Hough transform:

1. Edge detection - e.g. using the Canny edge detector. This gives us a binary image.
2. We create a *MxN* matrix of 0s, representing the Hough space. This will correspond to *M* different values of ρ and *N* different values of θ.
3. Iterate through all the pixels of the image and if they are an edge map they get to "cast a vote" onto the Hough space.

The Hough transform

There are three steps in the Hough transform:

1. Edge detection - e.g. using the Canny edge detector. This gives us a binary image.
2. We create a *MxN* matrix of 0s, representing the Hough space. This will correspond to *M* different values of ρ and *N* different values of θ.
3. Iterate through all the pixels of the image and if they are an edge map they get to "cast a vote" onto the Hough space.
4. Each value in the Hough space matrix is use as an "accumulator". We add 1 to the value of the Hough space at the corresponding θ and ρ coordinates for each possible line passing through the edge point.

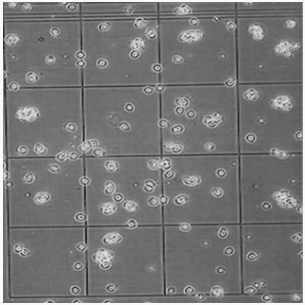
The Hough transform

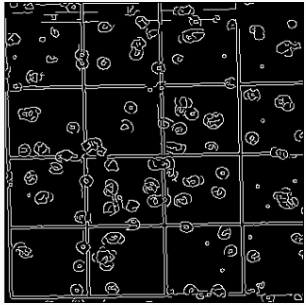
There are three steps in the Hough transform:

1. Edge detection - e.g. using the Canny edge detector. This gives us a binary image.
2. We create a *MxN* matrix of 0s, representing the Hough space. This will correspond to *M* different values of ρ and *N* different values of θ.
3. Iterate through all the pixels of the image and if they are an edge map they get to "cast a vote" onto the Hough space.
4. Each value in the Hough space matrix is use as an "accumulator". We add 1 to the value of the Hough space at the corresponding θ and ρ coordinates for each possible line passing through the edge point.
5. The Hough space matrix is then thresholded and non-maximum-suppression applied to find the strongest lines.

Hough transform - an example

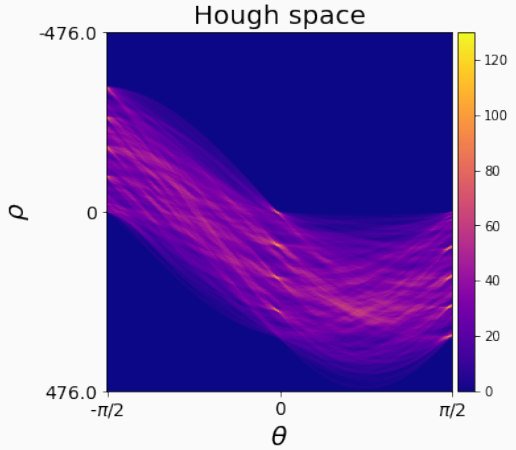
```
from skimage.feature import canny
from skimage.transform import hough_line, hough_line_peaks

chamber = imread("cell_counting_chamber.jpg")
chamber_canny = canny(chamber, sigma=1.5)
```





Hough transform - an example

```
hough_space, angles, d = hough_line(chamber_canny)
accum, theta, rho = hough_line_peaks(hough_space, angles, d)
for t, r in zip(theta, rho):
   print(f"np.rad2deg(t):0.2f, r:0.2f")
```




```
88.99, 253.77
88.99, 97.60
-1.51, 258.77
-1.51, 171.68
88.99, 332.85
-1.51, 84.59
88.99, 172.68
-1.51, 0.50
88.99, 23.52
```

Basic texture features

Range, variance

GLCM

HOG features