

Machine Vision

**Introduction to computer vision
lecture 1**

Adam Szmigelski

aszmigie@pjwstk.edu.pl

materials: *ftp(public) : //aszmigie/WMA*

Semester lecture schedule

1. Computer vision - introduction,
2. Image formation,
3. Image processing - filtering,
4. Image processing - detection of geometric features,
5. Detection of image features,
6. Algorithms detection and comparison of image features,
7. Convolutional networks,
8. Deep learning,
9. Architectures of convolutional neural networks,
10. Frequency methods in image processing,
11. Stereovision,
12. Vision and movement,
13. Visual SLAM - Simultaneous localization and mapping,
14. Image segmentation,
15. Semantical image segmentation.

Semester laboratory schedule

1. Introductory classes, familiarization with the laboratory,
2. OpenCV classes, color models, filters,
3. Identification of colored objects - project 1,
4. Track color objects,
5. Identification of coins and tray - project 2,
6. Object description using descriptors (e.g. SIFT algorithm),
7. Object tracking using descriptors description - project 3,
8. Convolutional networks using Keras,
9. Learning convolutional networks - creating your own patterns,
10. Object identification using neural networks,
11. Implementation of the Optical Flow algorithm (OF in openCV) - project 4
12. Implementation of the Optical Flow algorithm (OF in openCV),
13. Implementation of the segmentation algorithm - project 5,
14. Colloquium,
15. Final classes - submitting projects, giving grades,

How to pass WMA

- There are two grades in the MWR subject - exercises grade and exam grade,
- The subject ends with an exam. A colloquium will take place at the end of the semester. The grade from the test is the grade from the zero exam. There are no exemptions from the exam.
- 100 points to be obtained = 70 p. laboratory + 30 p. test
 - *Laboratory 70 p.* - During the classes, students will get several mini-projects to be carried out during the classes. The project will be able to be submitted for the next class without losing points.
 - *test 30 p.* - test on the 14th laboratory.
- In the case of justified absence from the test (week 14th) it is possible to write a test during 15 exercises.

- Laboratory grade table:

grade	points = lab. + test
2	0- 50
3	50,5 - 60
3,5	60,5 - 70
4	70,5 - 80
4,5	80,5 - 90
5	90,5 - 100

Test - zero exam

- Test is treated as a zero exam,
- The grade from the test can be treated as grade of zero exam, based on the following table:

grade	test points
2	0- 15
3	15,5 - 18
3,5	18,5 - 21
4	21,5 - 24
4,5	24,5 - 27
5	27,5 - 30

- The condition of passing the grade from the test to the zero exam is passing the laboratory.

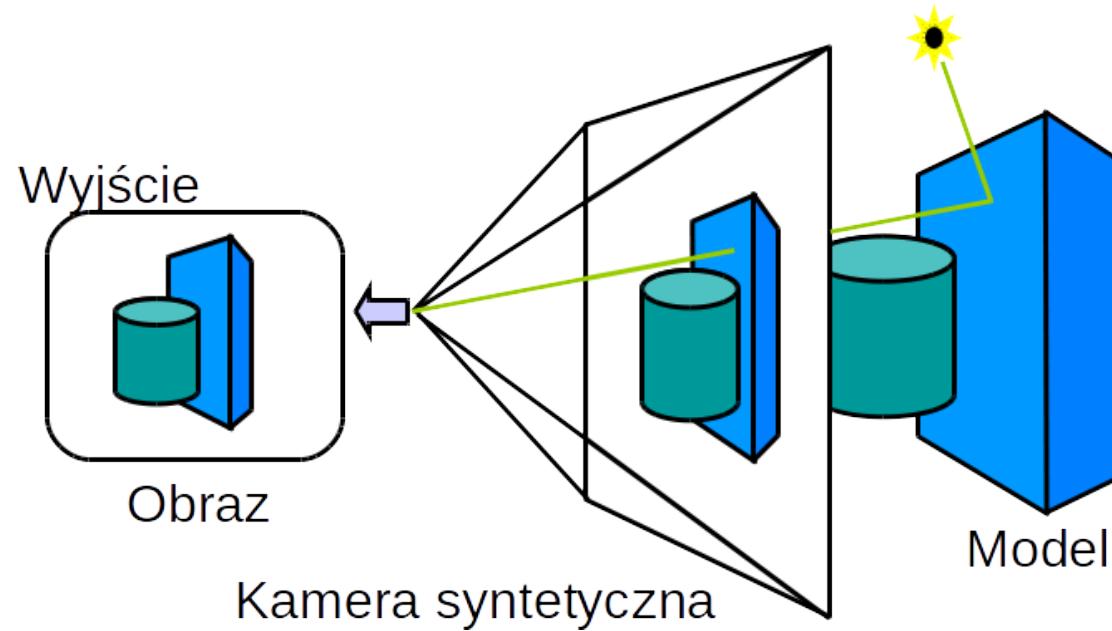
Software

Projects will be discussed using Python 3.

Software

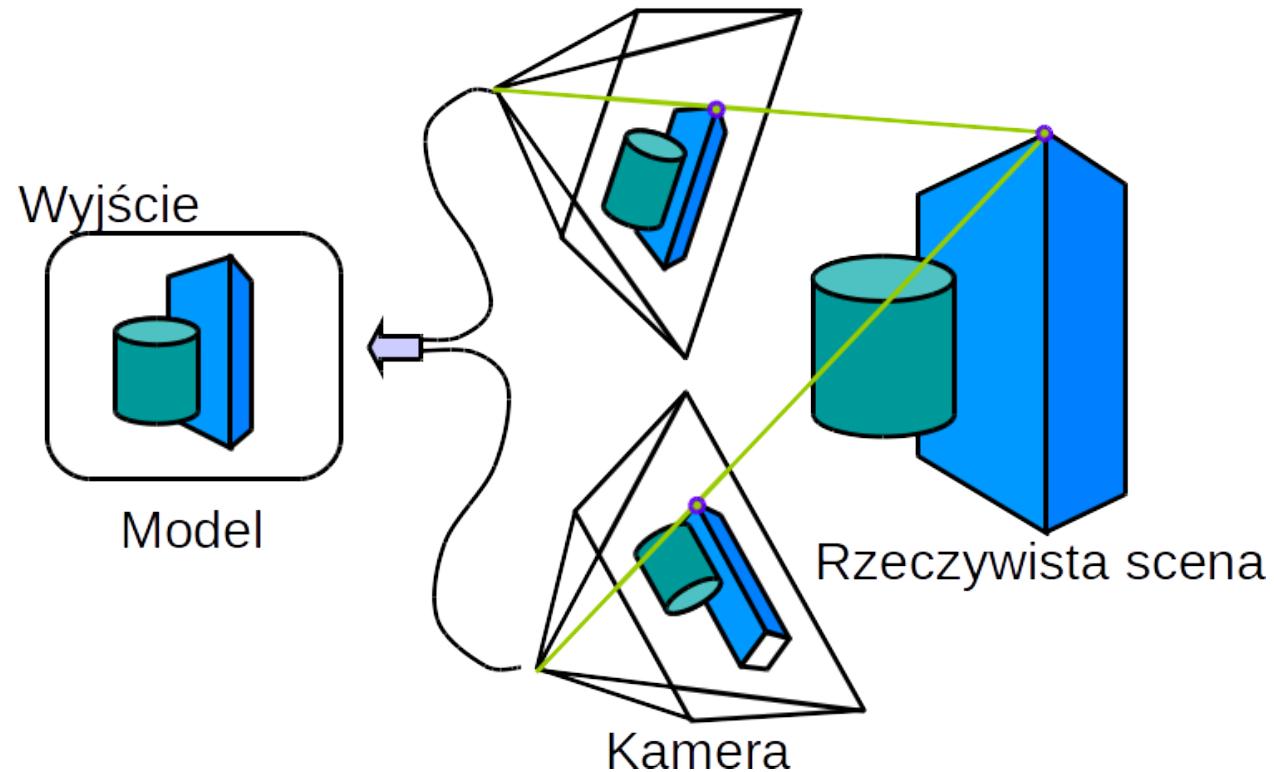
- Python 3
- OpenCV
- Biblioteki numpy, tensorflow, keras etc. - needed environment to manage modules, eg PIP

Computer Graphics



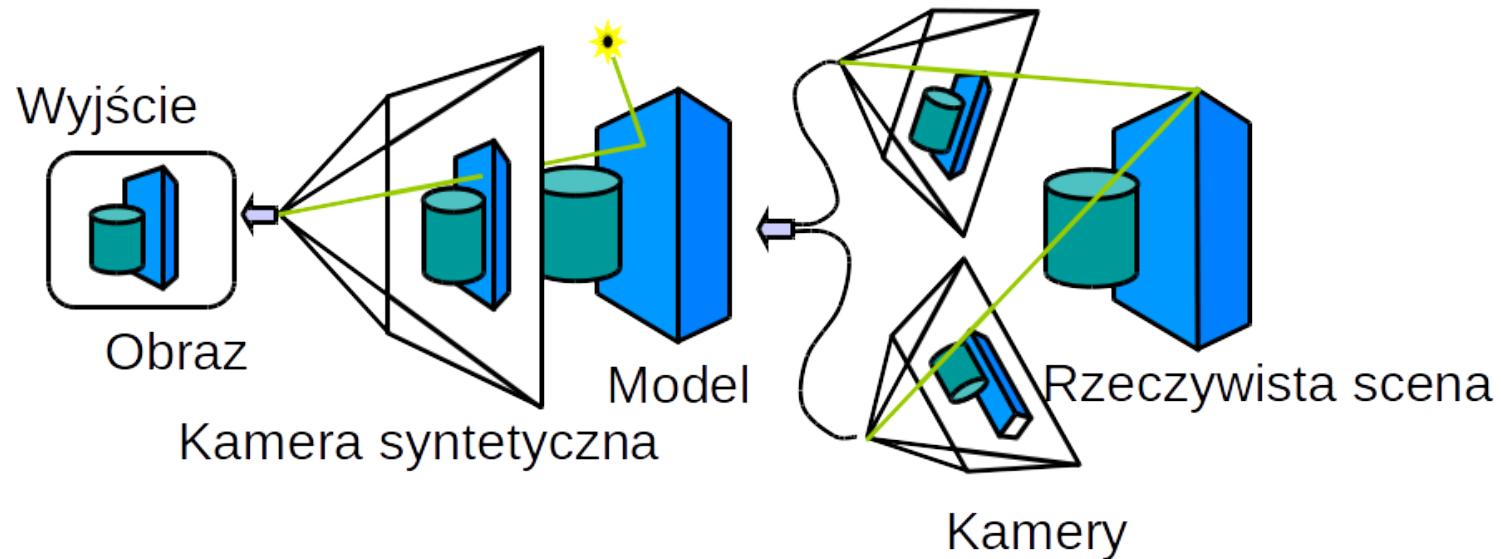
•

Computer Vision



•

Combined systems



- Augmented reality,

Computer vision

- Basics
 - Image creation and camera calibration,
 - Image features
- 3D reconstruction:
 - stereovision,
 - Mosaic picture,
- Object detection and recognition:
 - Grouping,
 - Detection
 - Segmentaiton,
 - Classification,
 - Tracking.

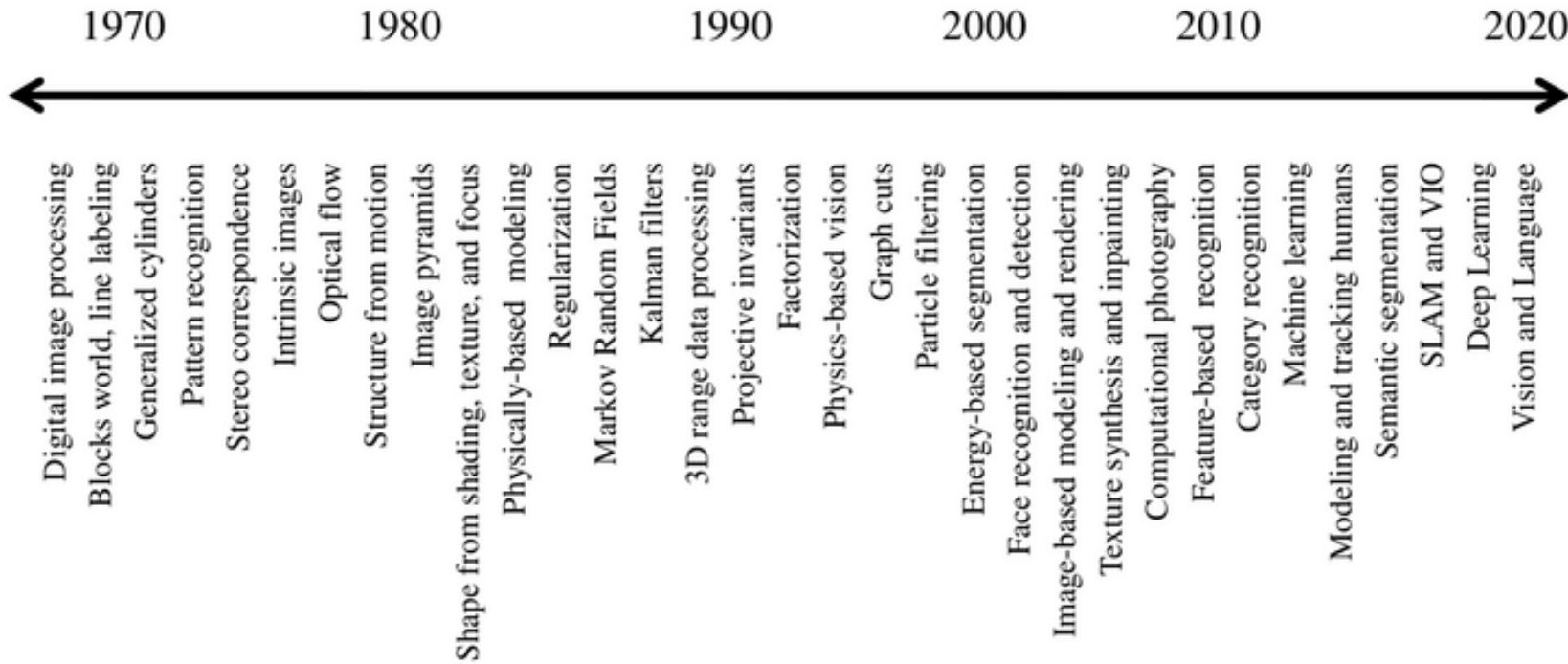
Application of Computer Vision

- Optical character recognition (OCR): reading handwritten letters,
- Machine inspection: rapid parts inspection for quality assurance using stereo vision,
- Retail: object recognition for automated checkout lanes and fully automated stores,
- Warehouse logistics: autonomous package delivery,
- Medical imaging: registering pre-operative and intra-operative imagery,
- Self-driving vehicles: capable of driving point-to-point between cities,
- 3D model building (photogrammetry): fully automated construction of 3D models from drone photographs
- Motion capture (mocap): using markers viewed from multiple cameras
- Surveillance: monitoring for intruders, analyzing highway traffic and

monitoring pools

- Fingerprint recognition and biometrics: for automatic access authentication as well as forensic applications,
- Face detection: for improved camera focusing as well as more relevant image searching,
- Visual authentication: automatically logging,

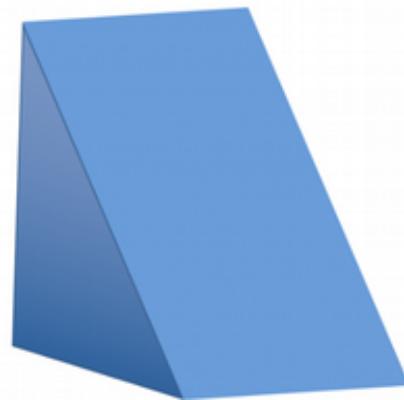
A brief history of Computer Vision



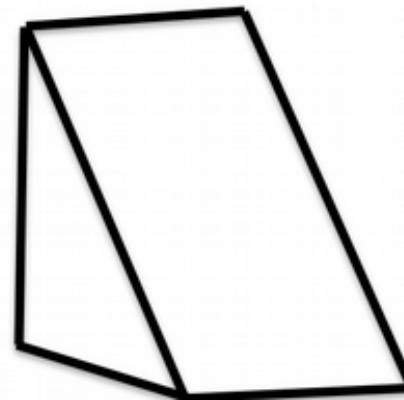
History of computer vision

- 1966: Minsky begins a summer bachelor's project in computer vision,
- 1960s (Larry Roberts, 1963) Interpretation of synthetic material,
- 1970s Attempts to interpret the image,
- 1980s trying to create a formal description - in the context of geometry,
- 90 years Face recognition - statistical analysis,
- years 2000 Large data sets with labels, video processing begins.

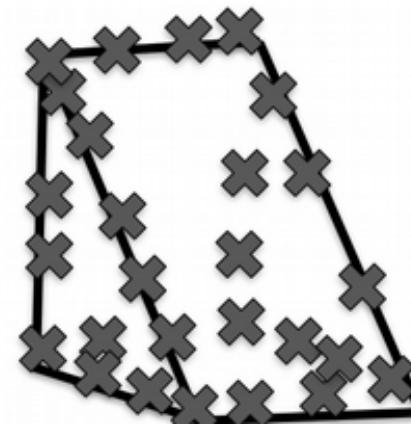
Block world Larry Roberts, 1963



original picture

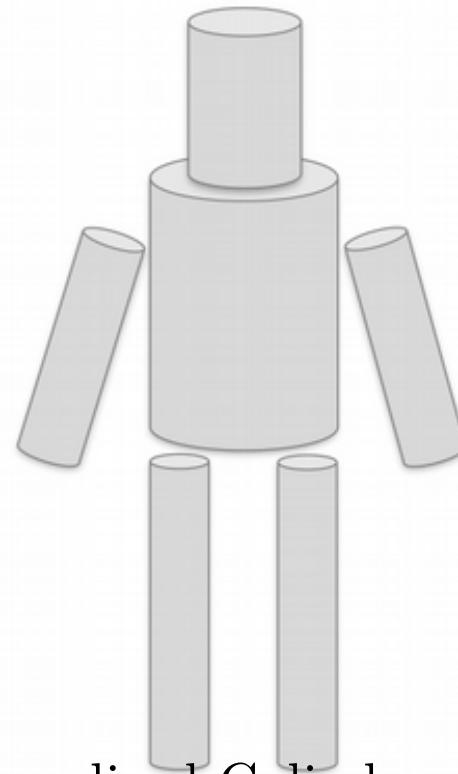


differential description

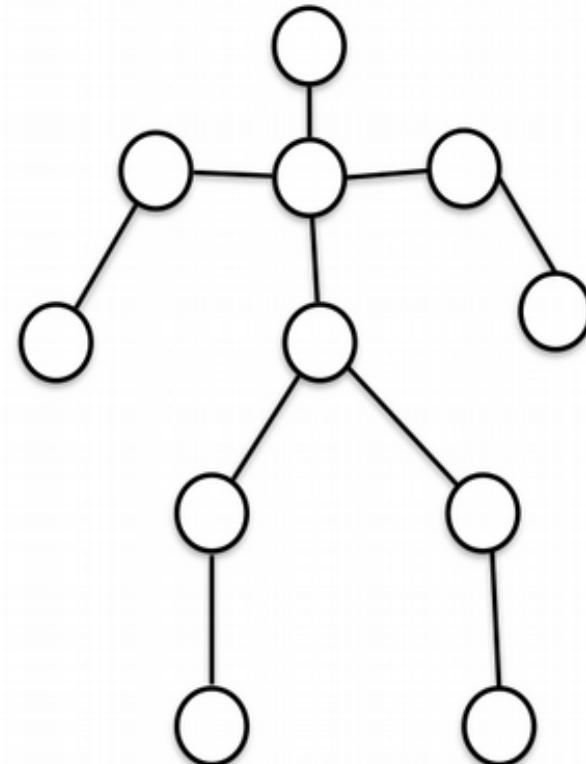


selected points

Image models - interpretation



Generalized Cylinder
Brooks Binford, 1979



Pictorial Structure
Fischler and Elschlager, 1973

David Lowe, 1987 - Filtration

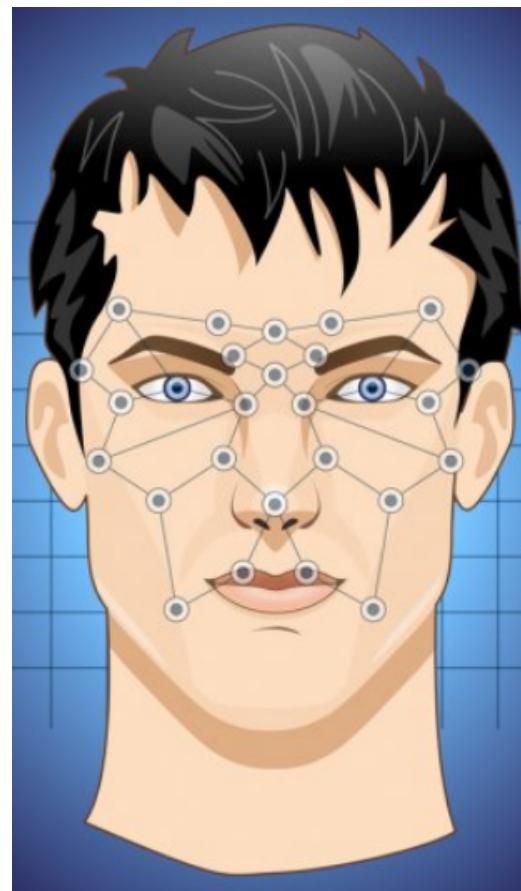


original image



after filtration

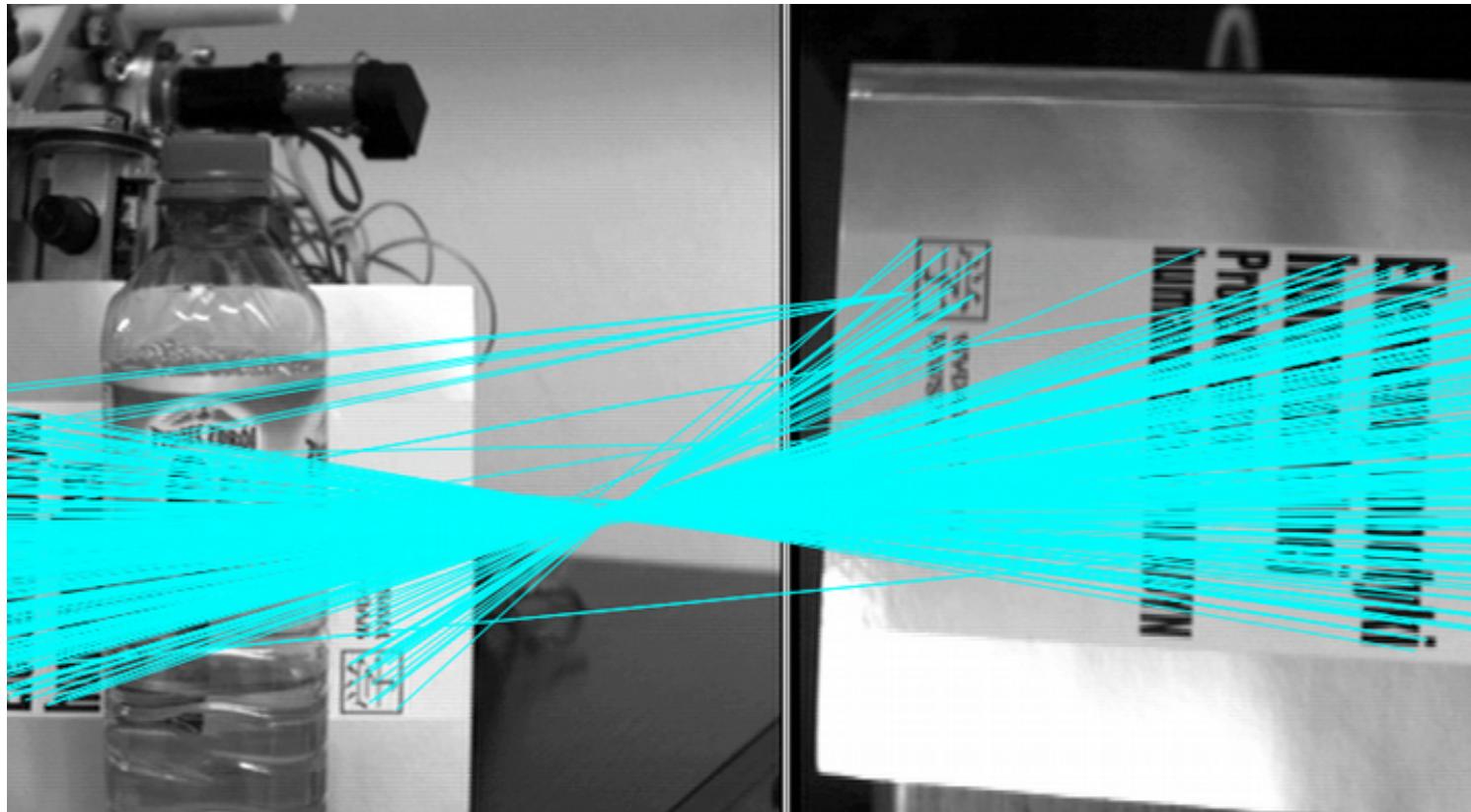
Face recognition - 90's



Decade 2000s

- Embraced data-driven and learning approaches as core components of vision,
- During this decade was the emergence of feature-based techniques (combined with learning) for object recognition,
- Most aspects of computer vision, was the application of sophisticated machine learning techniques to computer vision problems .

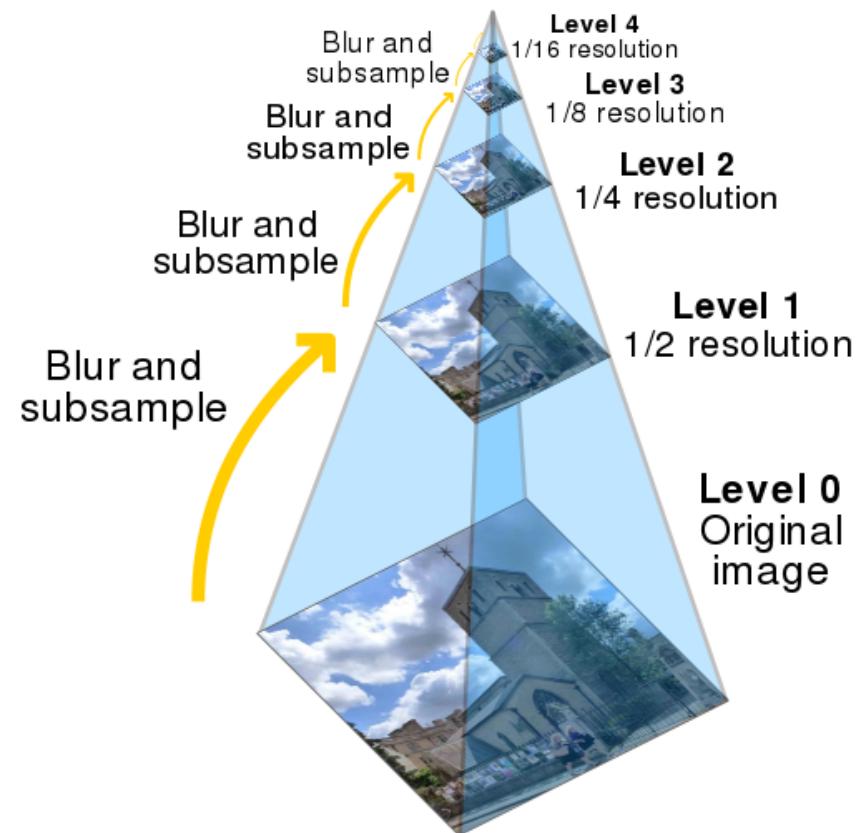
SIFT and pattern recognition, David Lowe, 1999



Decade 2010s

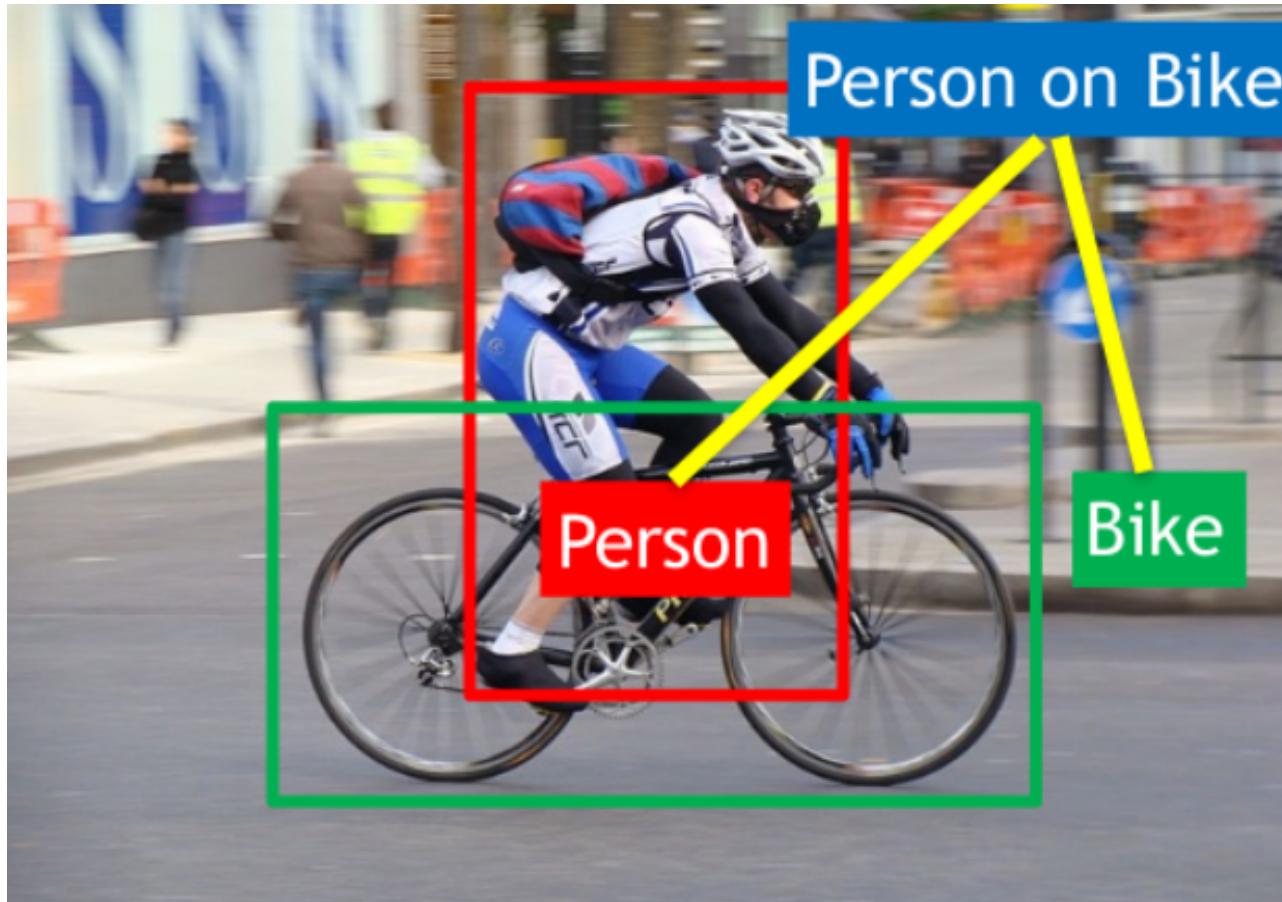
- The trend towards using large labeled datasets to develop machine learning algorithms,
- These datasets provided not only reliable metrics for tracking the progress of recognition and semantic segmentation algorithms,
- Specialized sensors and hardware for computer vision tasks also continued to advance - Kinect depth camera, released in 2010, quickly became an essential component of many 3D modeling,
- Real-time SLAM (simultaneous localization and mapping) and VIO (visual inertial odometry).

Pyramid algorithm

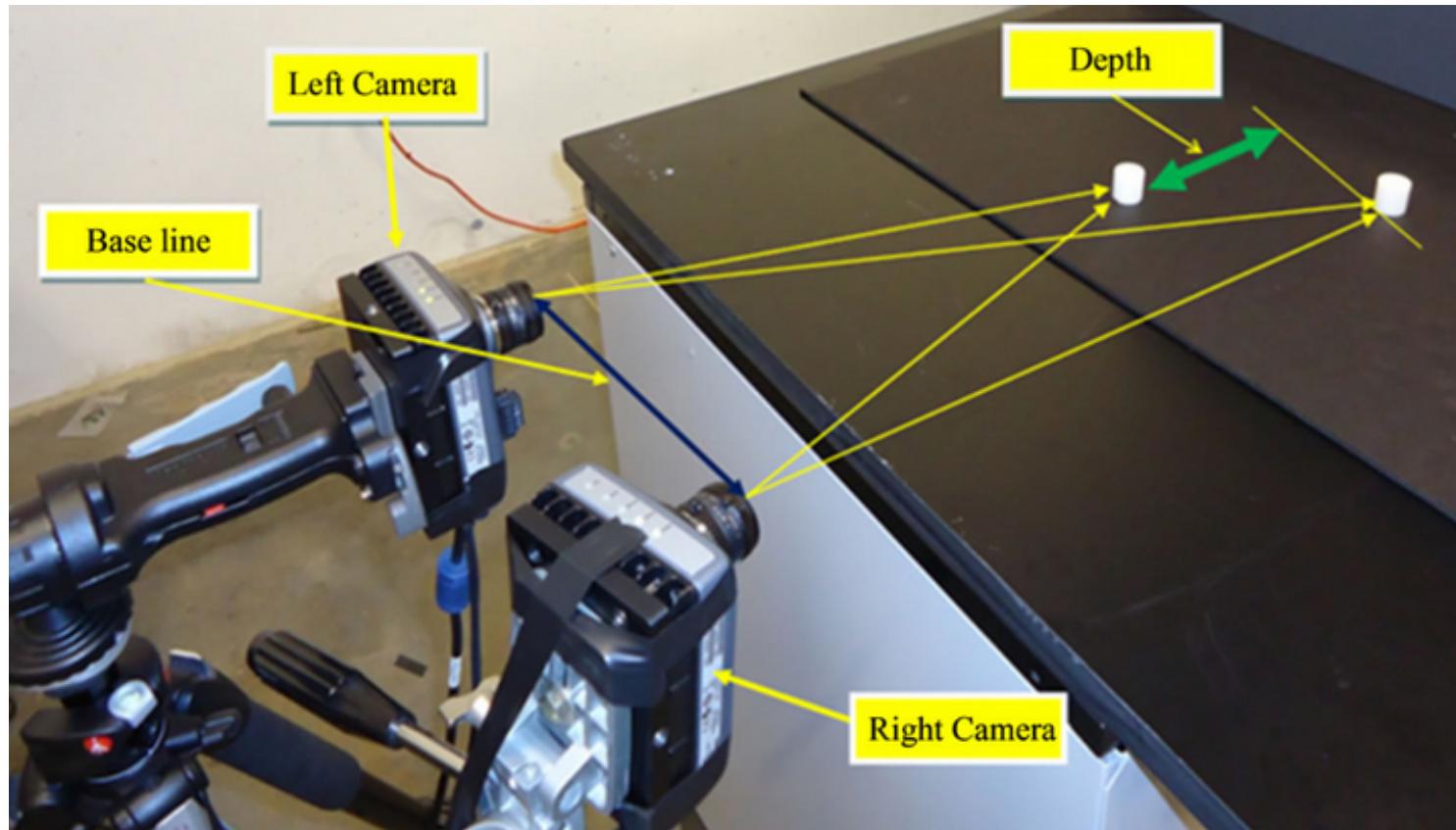


- This algorithm has the property of preserving invariants of linear transformations - translations, rotation, scaling.
- This property gives the opportunity to describe the pattern as a set of transformation invariants and is the reason for the emergence of a new type of network - CNN convolution networks

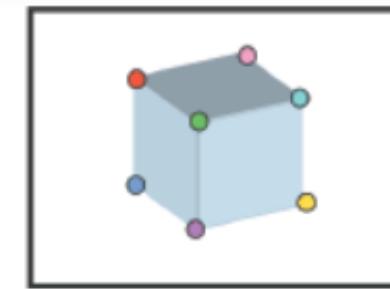
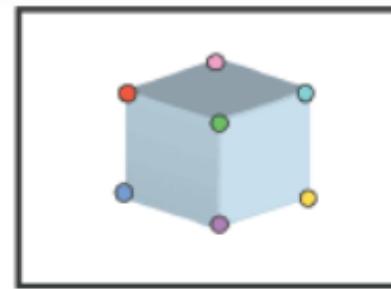
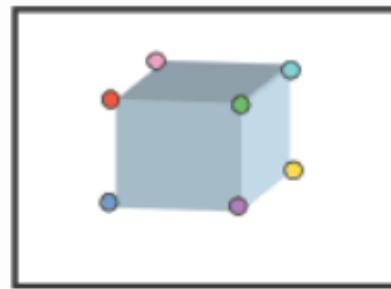
Visual Object Challenge - up to nowadays



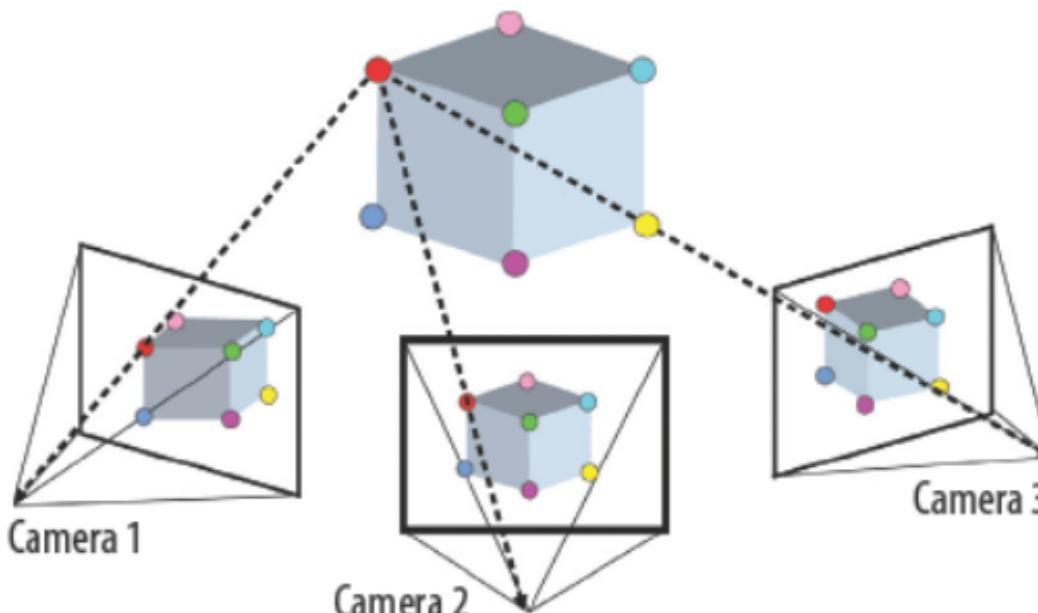
Stereovision - determination of image depth



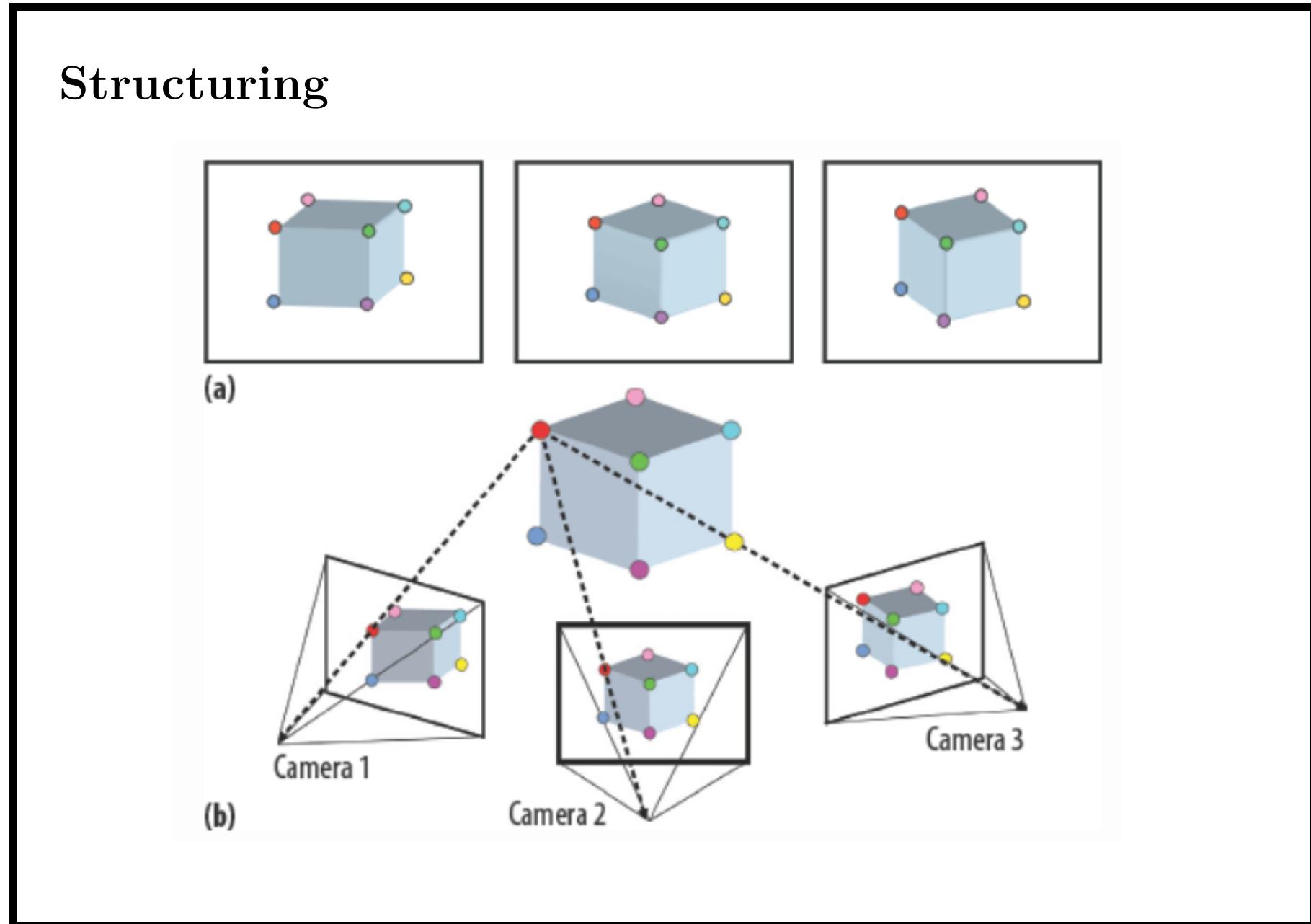
Structuring



(a)



(b)



Literature

1. Richard Szeliski: *Computer Vision: Algorithms and Applications*, Springer 2010
2. Richard Szeliski: *Computer Vision: Algorithms and Applications*, 2nd Edition
<http://szeliski.org/Book/>
3. Richard O. Duda, Peter E. Hart, David G. Stork: *Pattern Classification*, 2ed, Wiley, 2001
4. Christopher M. Bishop: *Pattern Recognition and Machine Learning*, Springer, 2006
5. Ian Goodfellow, Yoshua Bengio, Aaron Courville: *Deep Learning Systemy uczące się*, PWN, 2018
6. Sebastian Raschka: *Python Uczenie maszynowe*, Helion, 2018

7. Josh Patterson, Adam Gibson: *Deep Learning. Praktyczne Wprowadzenie*, Helion, 2018
8. Aurelien Geron: *Uczenie maszynowe z użyciem scikit learn i tensorflow*, Helion, 2018
9. Francois-Chollet: *Deep learning praca z językiem python i biblioteka keras*, Helion, 2018

Machine Vision

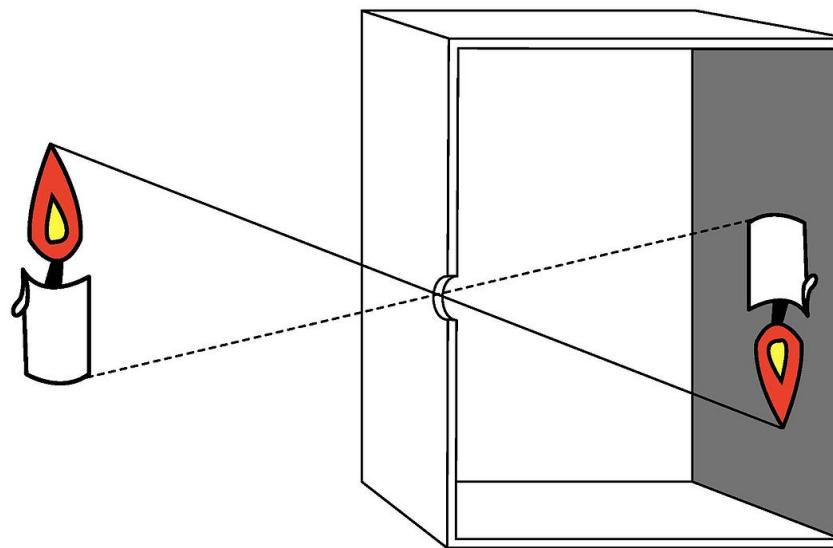
Image formation- lecture 2

Adam Szmigielski

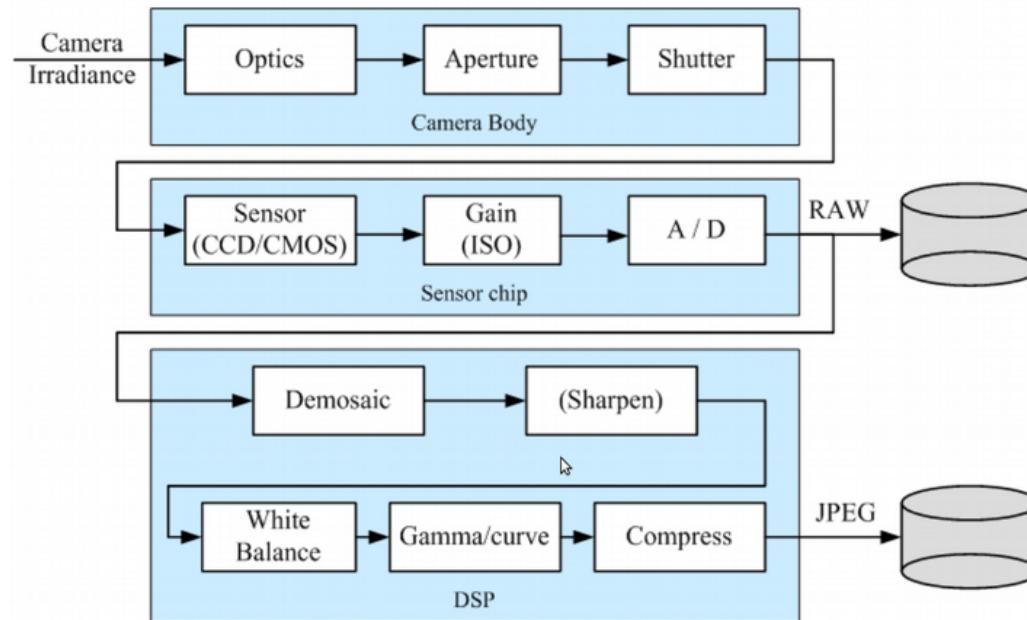
aszmigie@pjwstk.edu.pl

materials: *ftp(public) : //aszmigie/WMA*

Optics - pinhole camera



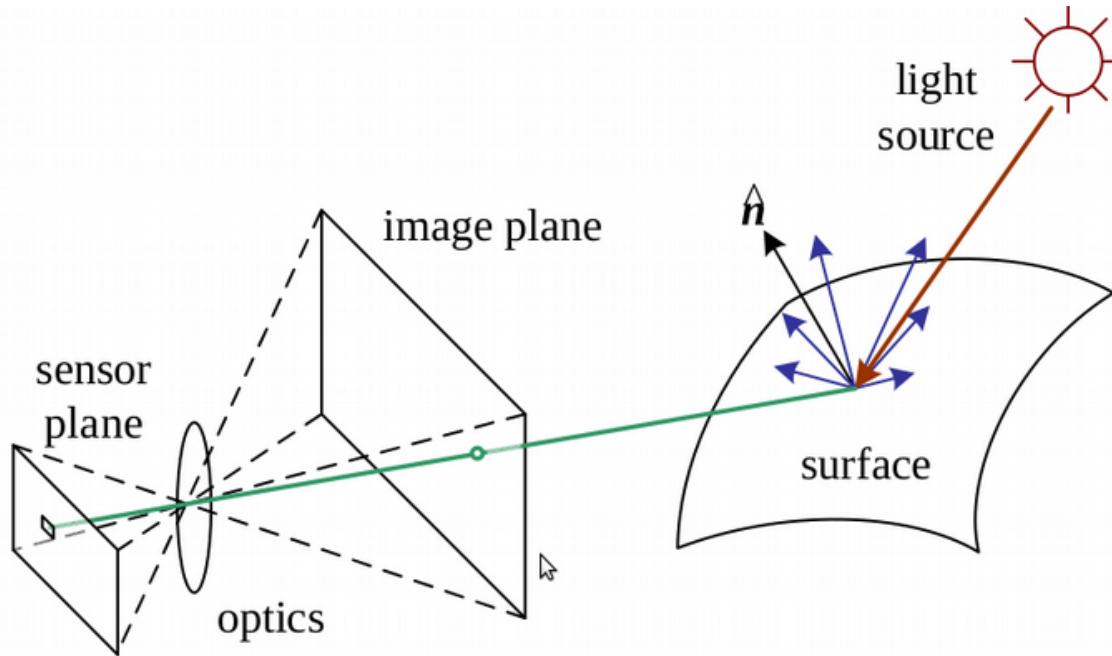
The digital camera



- *Shutter speed* directly controls the amount of light reaching the sensor
- *Sampling pitch* is the physical spacing between adjacent sensor cells on the imaging chip.
- *Fill factor* is the active sensing area size
- *Chip size*

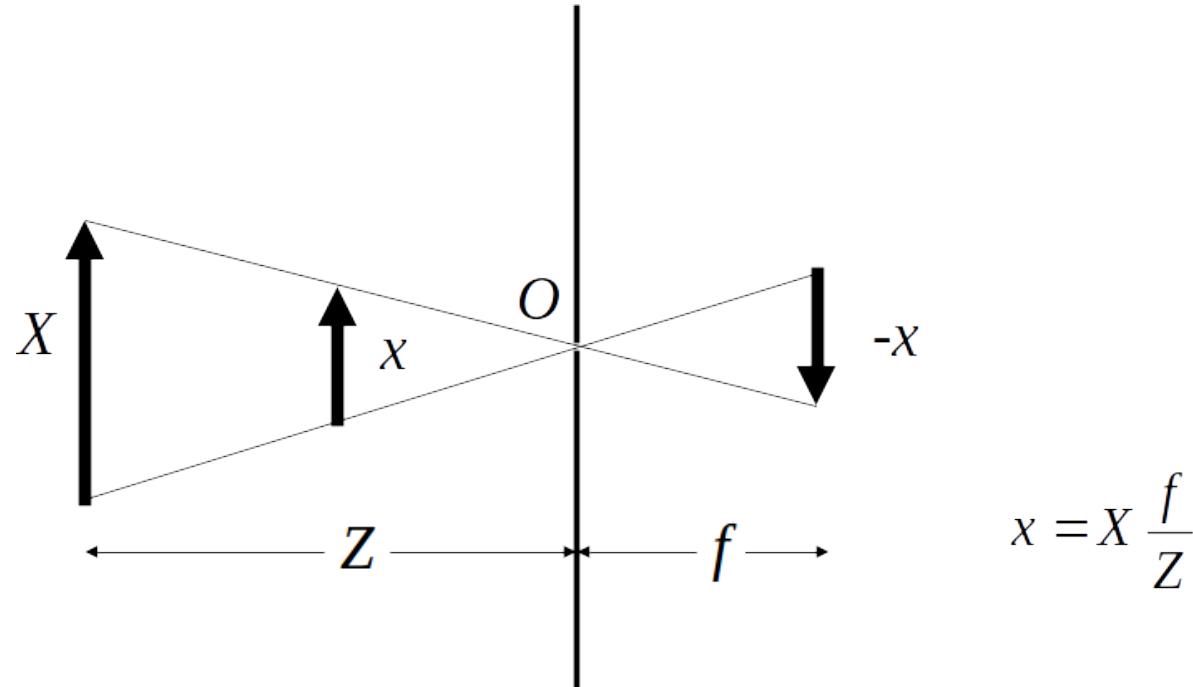
- *Analog gain* - before analog-to-digital conversion, the sensed signal is usually amplified,
- *Sensor noise* - throughout the whole sensing process, noise is added from various sources, which may include fixed pattern noise, dark current noise, shot noise, amplifier noise and quantization noise,
- *ADC resolution* - the final step in the analog processing chain occurring within an imaging sensor is the analog to digital conversion (ADC),
- *Digital post-processing* - once the irradiance values arriving at the sensor have been converted to digital bits, most cameras perform a variety of digital signal processing.

Photometric image formation



- **Lighting** to produce an image, the scene must be illuminated with one or more light sources.
- Light sources can generally be divided into point and area light sources. A point light source originates at a single location in space, potentially at infinity (e.g., the sun).

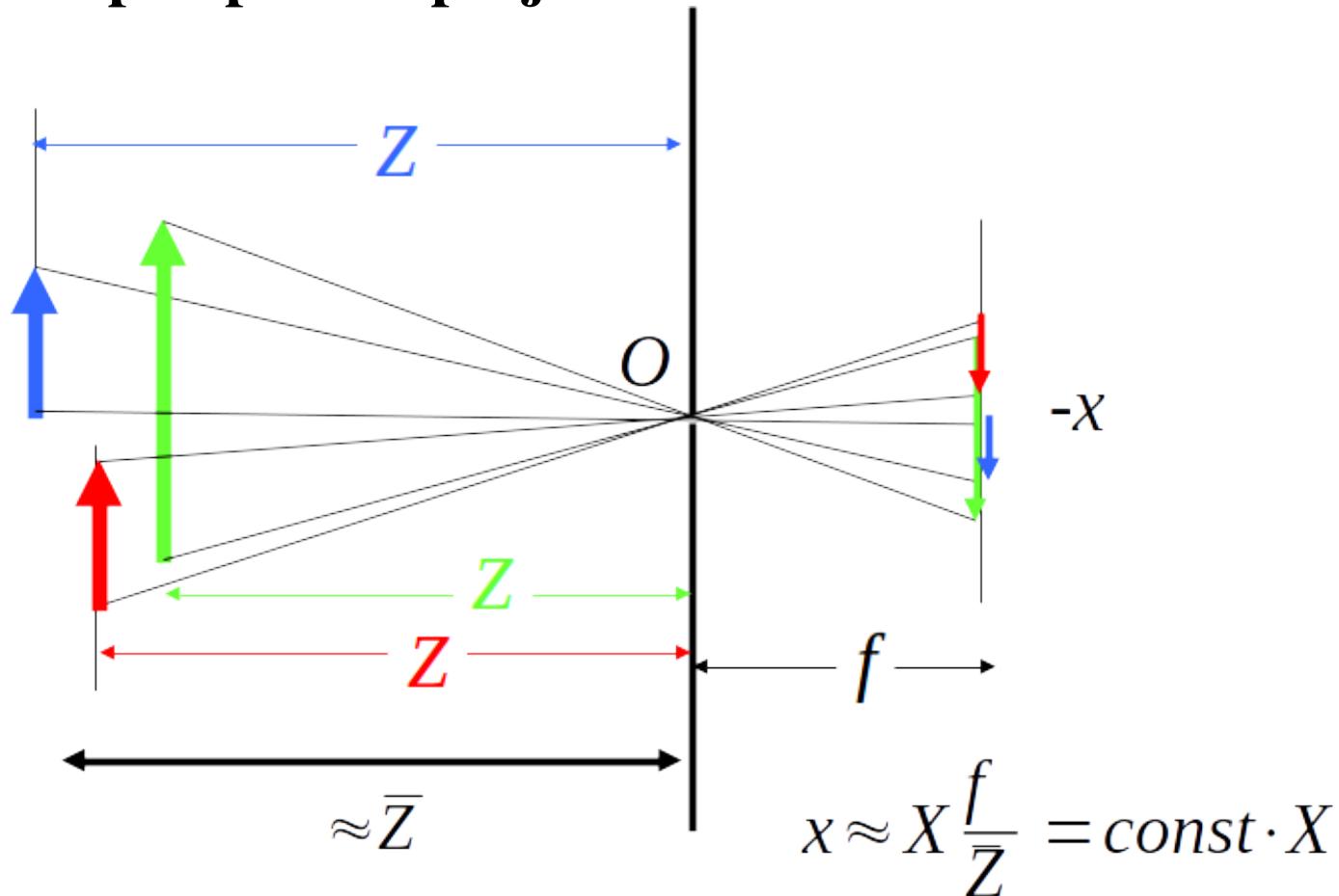
Perspective projection



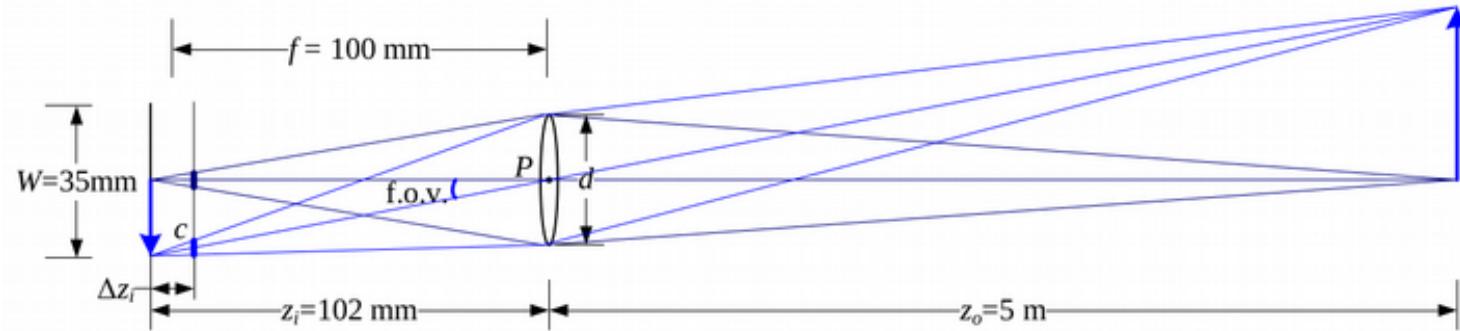
where:

- f - focal Length,
- Z - distance to camera,
- $X, -x$ - object and its image.

Weak perspective projection



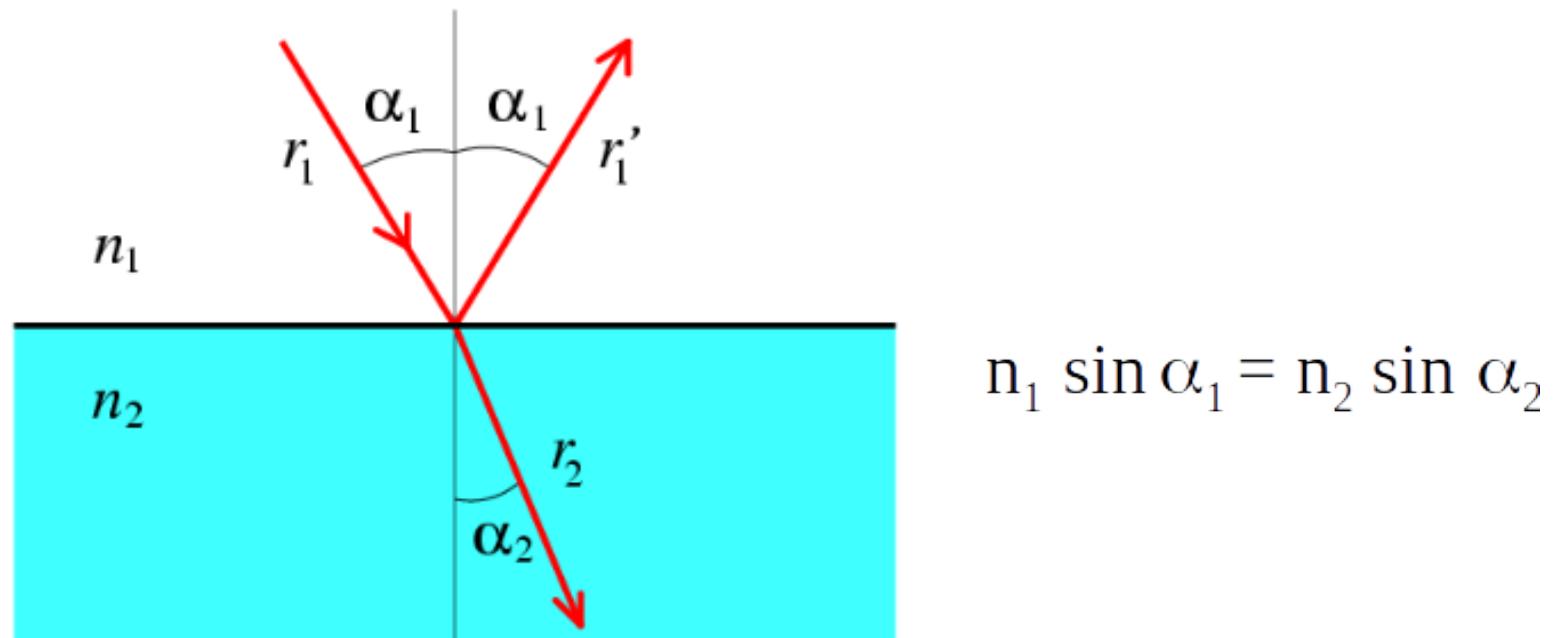
Optics - more complex model



$$\frac{1}{z_o} + \frac{1}{z_i} = \frac{1}{f}$$

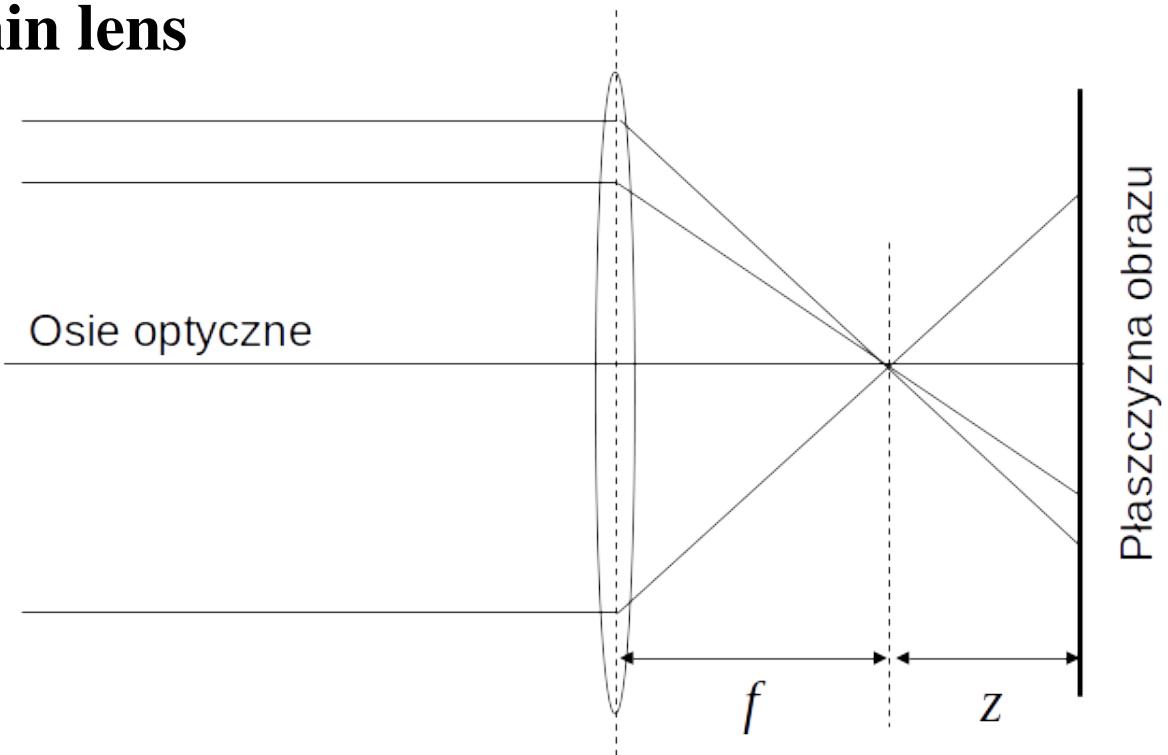
- Once the light from a scene reaches the camera, it must still pass through the lens before reaching the sensor
- It suffices to treat the lens as an ideal pinhole that simply projects all rays through a common center of projection.
- The circle of confusion c depends on the distance of the image plane motion Δz_i relative to the lens aperture diameter d . The field of view depends on sensor width W and the focal length f .

Refraction of light on the border region



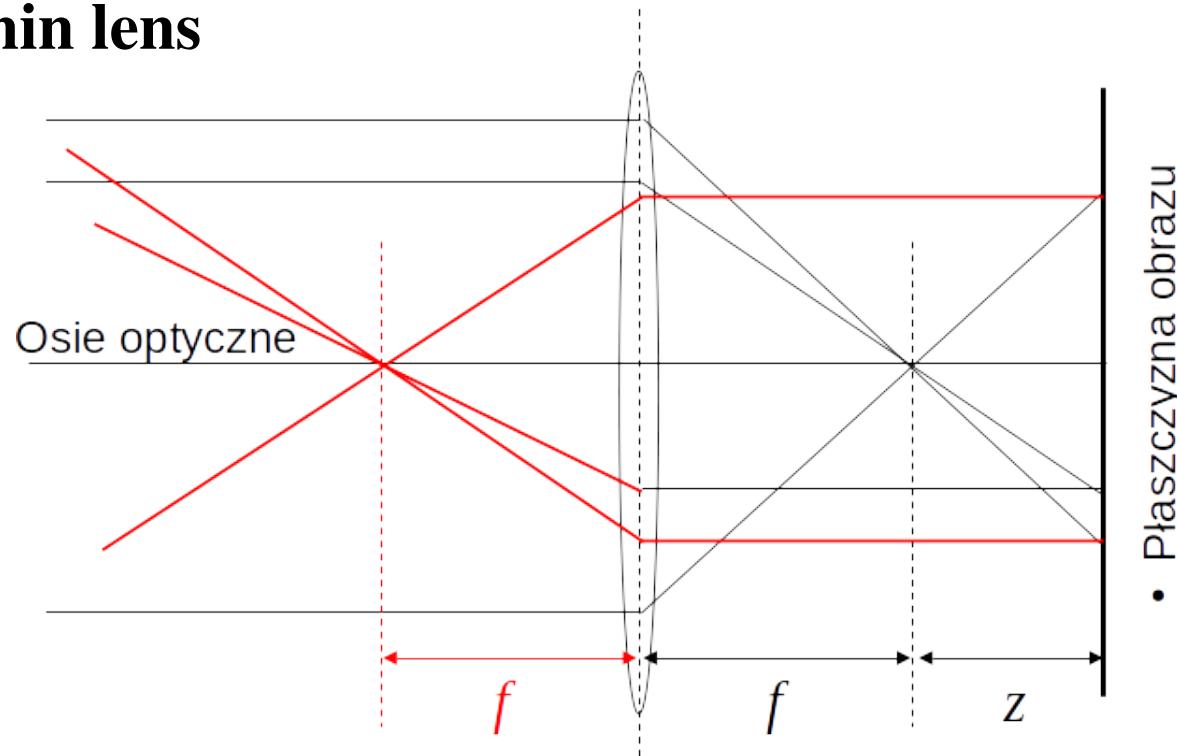
•

Thin lens



- Model spherical lens with negligible thickness,
- Parallel rays are focused on a single point.

Thin lens

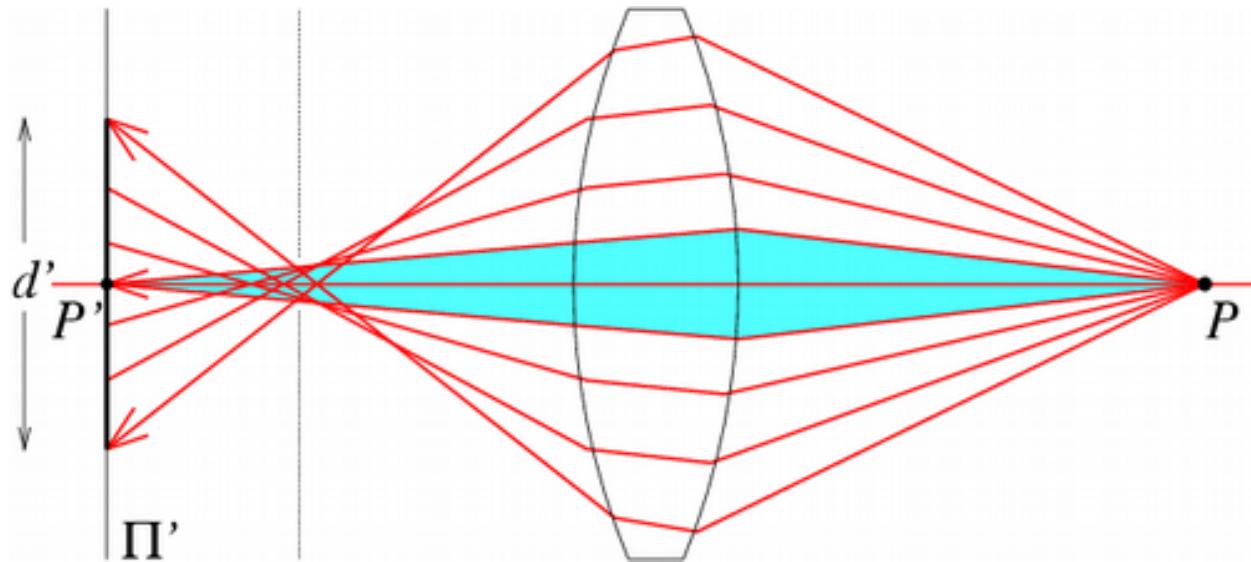


- The light from the fire spreads in parallel,

Image distortion

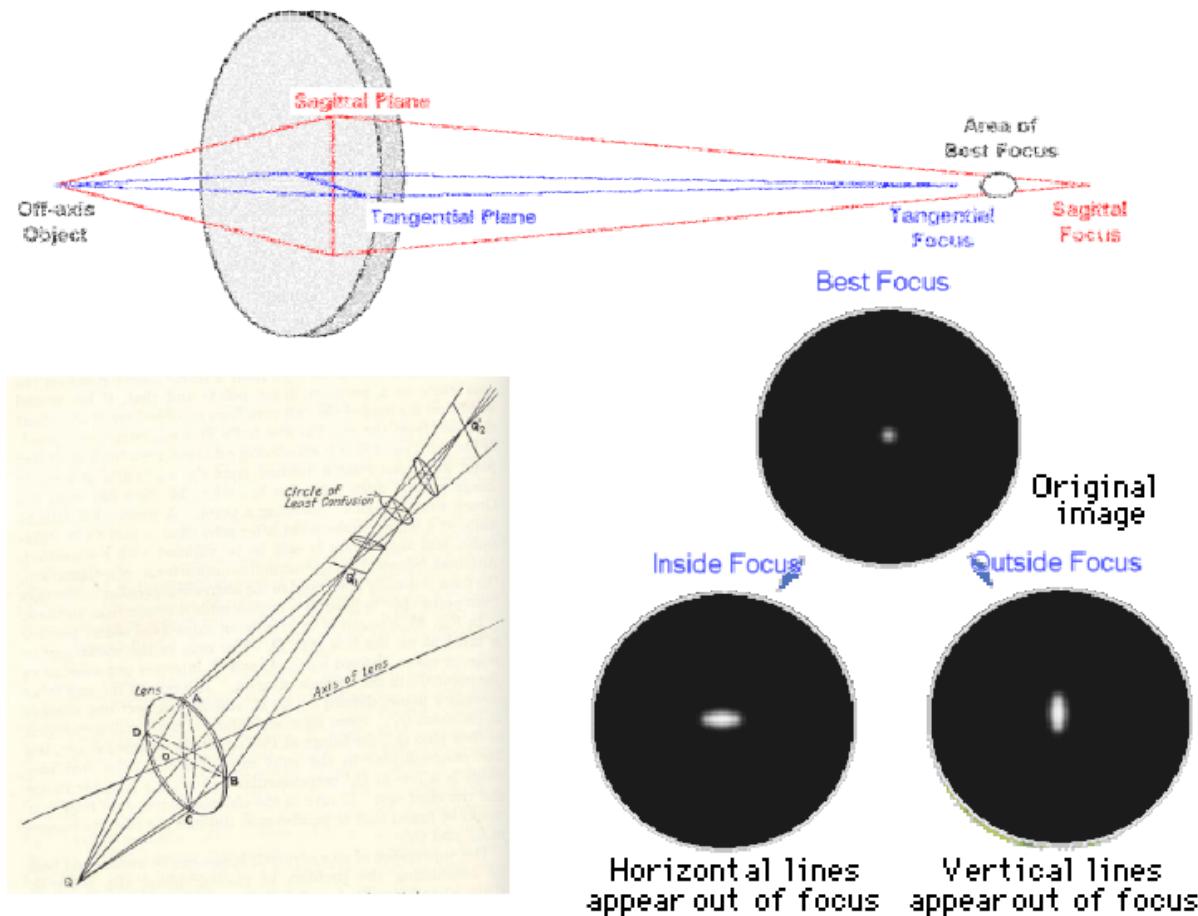
- Geometric: lens geometry
 - Spherical aberration
 - Astigmatism
 - Angular distortion
 - Comma distortion
- Chromatic: the refractive factor depends on the wavelength.

Spherical aberration



- The rays parallel to the axis do not focused in one point,
- The outer parts of the lens reduce the focal length.

Astigmatism



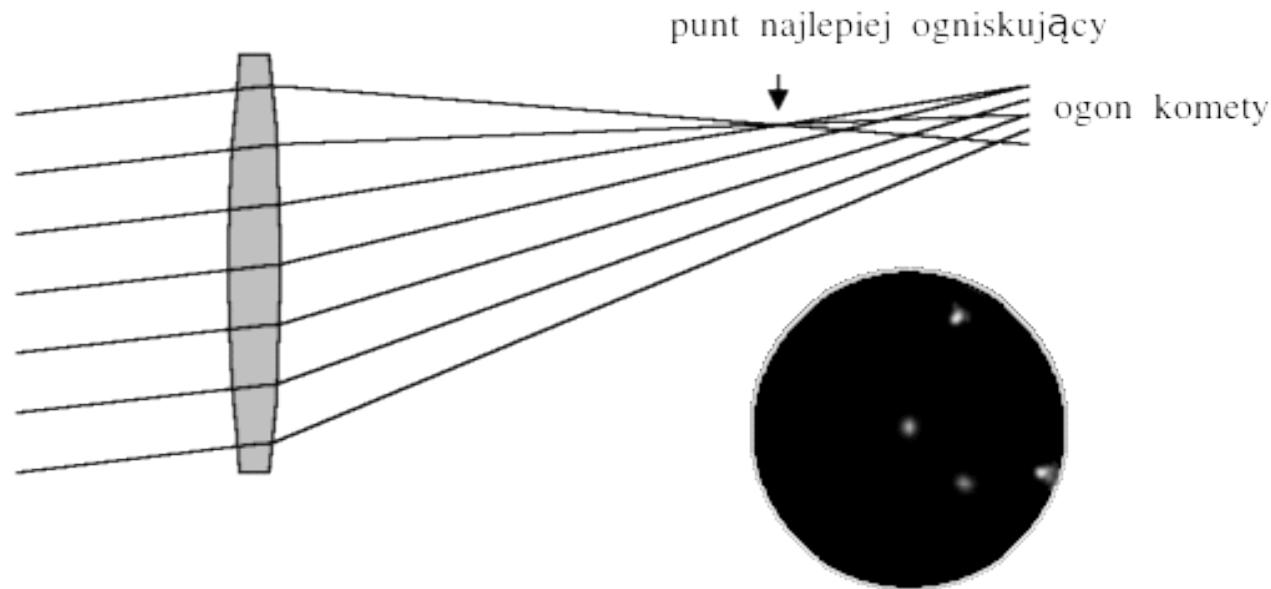
- Different focal lengths for oblique rays.

Angular distortion



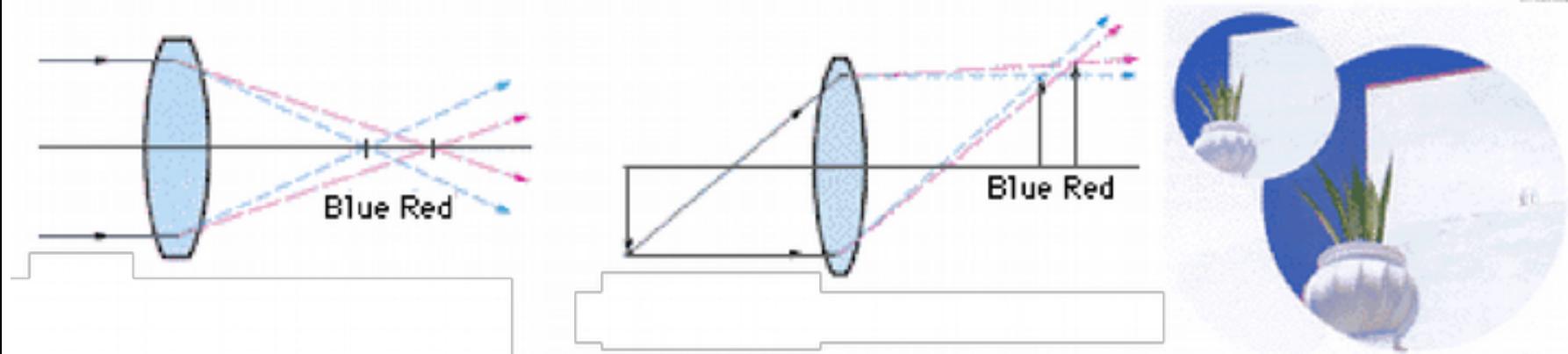
- Barrel (wide angle),
- Pillow type,
- Can be corrected.

Coma distortion



- The distortions are drop-shaped (comet's tail)

Chromatic distortion

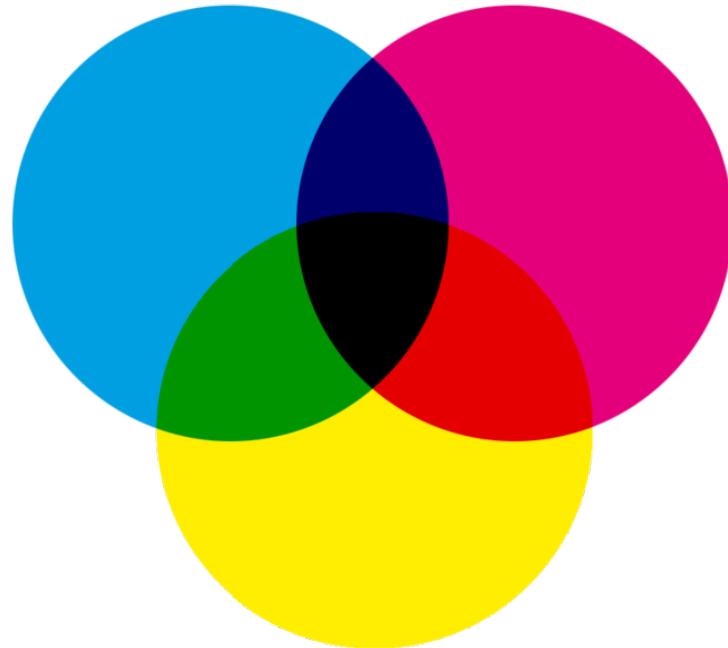


- Rays of different wavelengths are focused in different planes,
- Cannot be completely removed.

Popular Color Models

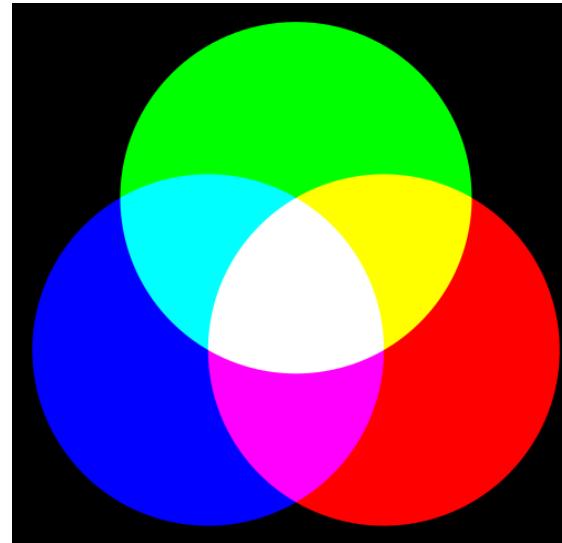
- additive RGB,
- subtractive CMYK,
- other, e.g. HSV.

CMYK Model



- Model components: Cyan, Magenta, Yellow and black (blacK),
- CMYK is a subtractive model in which colors, when mixed, generate a color similar to black.

RGB and XYZ Models



- Model components: Red, Green and Blue or X,Y,Z
- An additive model in which white color is created by combining all three colors.
- With the advent of HDTV and newer monitors, a new standard was created, which specifies the XYZ values of each of the color primaries:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

YIQ standards

- The earliest color representation developed for video transmission was the YIQ standard (NTSC in USA, PAL or SECAM in Europe).
- It has a luma channel Y (luminance) and chroma channels.
- Y signal, which is gamma compressed, is obtained from:

$$Y = 0.299R + 0.587G + 0.114B$$

- Newer color definitions for HDTV formula is:

$$Y = 0.2125R + 0.7154G + 0.0721B.$$

- The UV components are derived from scaled versions of $(B - Y)$ and $(R - Y)$, namely, $U = 0.492111(B - Y)$ and $V = 0.877283(R - Y)$

YUV colour model

The YUV model defines a color space in terms of one luma component (Y) and two chrominance components, called U (blue projection) and V (red projection) respectively.

The YUV color model is used in the PAL composite color video

```
import cv2

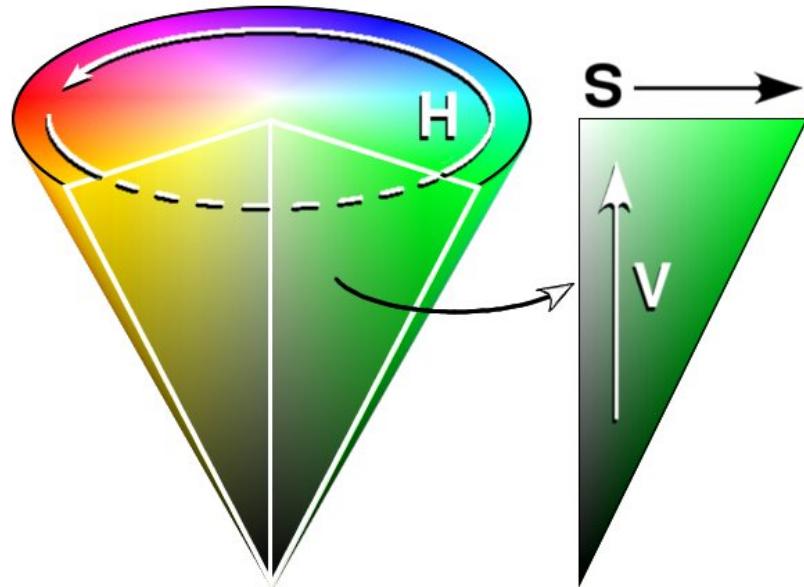
img = cv2.imread('./images/input.jpg', cv2.IMREAD_COLOR)

g,b,r = cv2.split(img)
gbr_img = cv2.merge((g,b,r))
rbr_img = cv2.merge((r,b,r))

cv2.imshow('Original', img)
cv2.imshow('GRB', gbr_img)
cv2.imshow('RBR', rbr_img)

cv2.waitKey()
```

HSV Model



- A color space model with values of light shade (Hue), color saturation (Saturation) and white light power (Value).

```
import cv2

img = cv2.imread('images/input.jpg')
gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
h
, s, v = cv2.split(hsv_img)
cv2.imshow('Grayscale image', gray_img)
cv2.imshow('H channel', h)
cv2.imshow('S channel', s)
cv2.imshow('V channel', v)

cv2.waitKey()
```

Pixel transforms

A general image processing operator is a function that takes one or more input images $f_0(x) \dots f_n(x)$ and produces an output image $g(x)$:

$$g(x) = h(f_0(x), \dots, f_n(x)),$$

- Commonly used point processes are multiplication and addition with a constant,

$$g(x) = af(x) + b$$

The parameters $a > 0$ and b are often called the **gain** and **bias** parameters (to control contrast and brightness),

- Bias and gain parameters can also be spatially varying

$g(x) = a(x)f(x) + b(x)$, e.g., when simulating the graded density filter to selectively darken the sky or when modeling vignetting in an optical system.

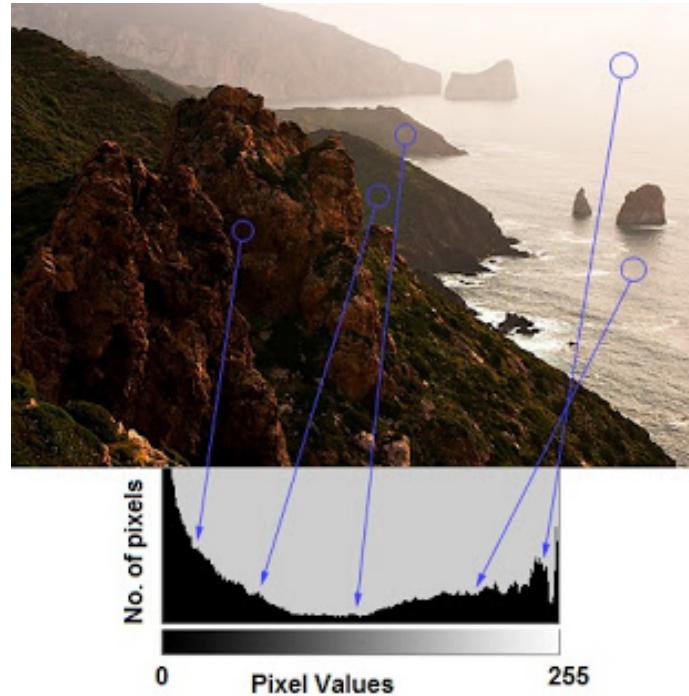
Compositing and matting

$$B \times (1 - \alpha) + \alpha F = C$$

Compositing equation $C = (1 - \alpha)B + \alpha F$.

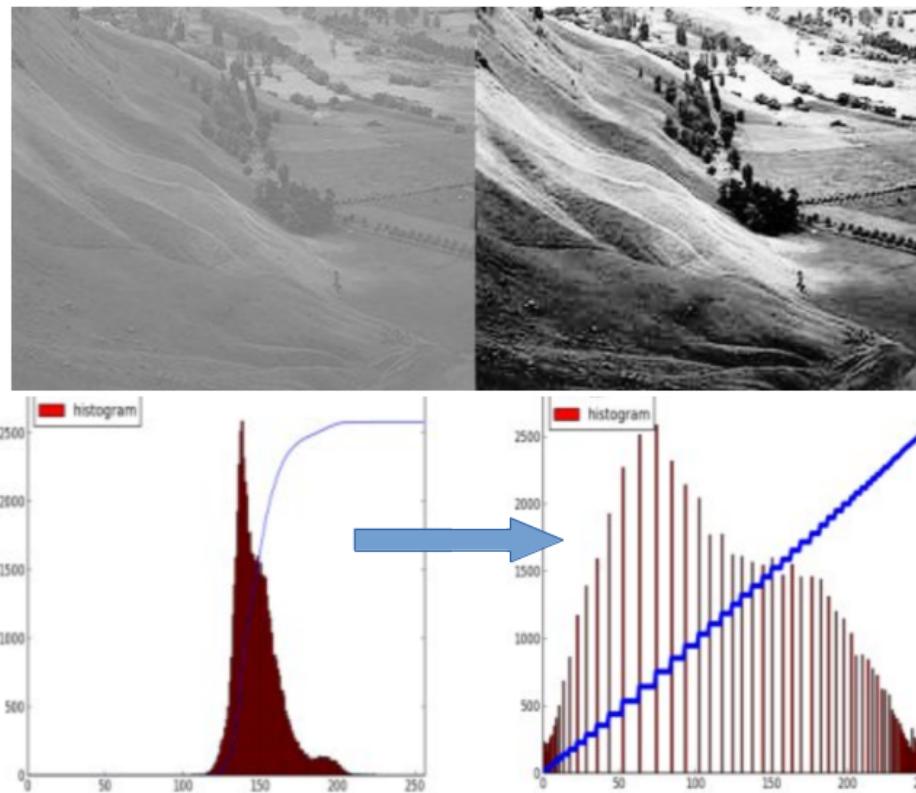
- In addition to the three color RGB channels, an alpha-matted image contains a fourth alpha channel α (or A) that describes the relative amount of opacity or fractional coverage at each pixel
- Pixels within the object are fully opaque ($\alpha = 1$), while pixels fully outside the object are transparent ($\alpha = 0$).
- Pixels on the boundary vary smoothly between these two extremes.

Histogram



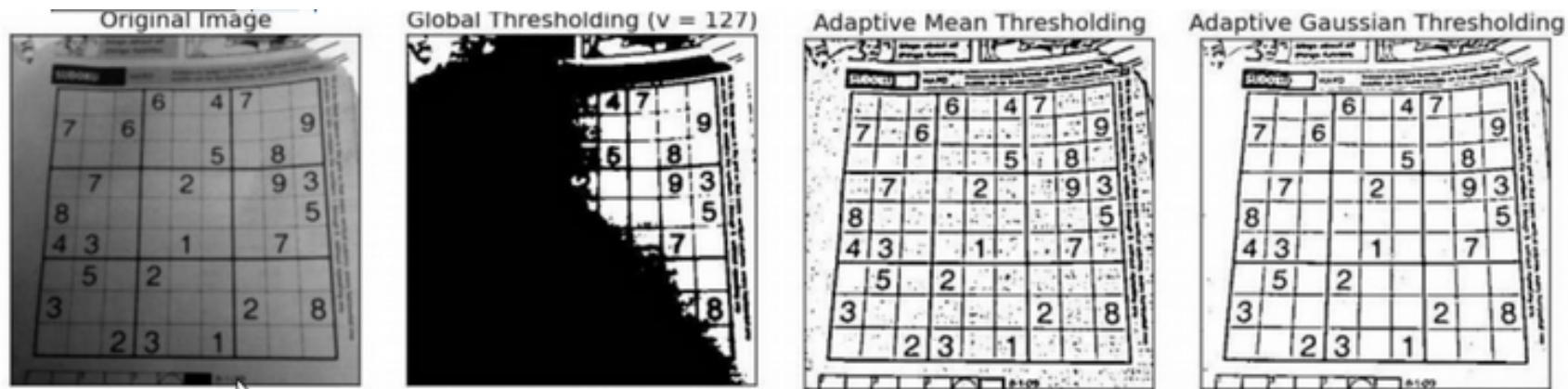
- We can consider histogram as a graph or plot, which gives an overall idea about the intensity distribution of an image.
- It is a plot with pixel values in X-axis and corresponding number of pixels in the image on Y-axis.
- By looking at the histogram of an image, we got information about contrast, brightness, intensity distribution etc.

Histogram equalization



- Original image pixel values are confined to some specific range,
- Better image will have pixels from all regions of the image (to stretch this histogram to either ends).

Histogram equalization - thresholding



How can we automatically determine their best values of brightness and gain controls?

- One approach might be to look at the darkest and brightest pixel values in an image and map them to pure black and pure white.
- Another approach might be to find the average value in the image, push it towards middle gray, and expand the range so that it more closely fills the displayable values

Geometric primitives - 2D points

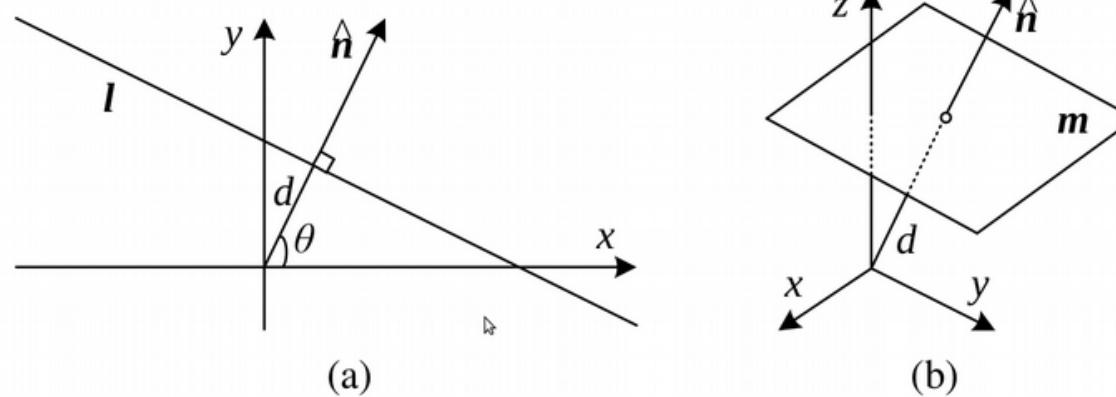
Geometric primitives form the basic building blocks used to describe three-dimensional shapes.

- **2D points** (pixel coordinates in an image) can be denoted using a pair of values, $x = (x, y) \in R^2$
 $2D$ points can also be represented using homogeneous coordinates,

$$\tilde{x} = (\tilde{x}, \tilde{y}, \tilde{w}) = \tilde{w}\bar{x} \in P^2$$

where vectors \tilde{x} that differ only by scale \tilde{w} are considered to be equivalent,
 $\bar{x} = (x, y, 1)$ is the *augmented vector* and P^2 is called the *2D projective space*.

Geometric primitives - 2D lines



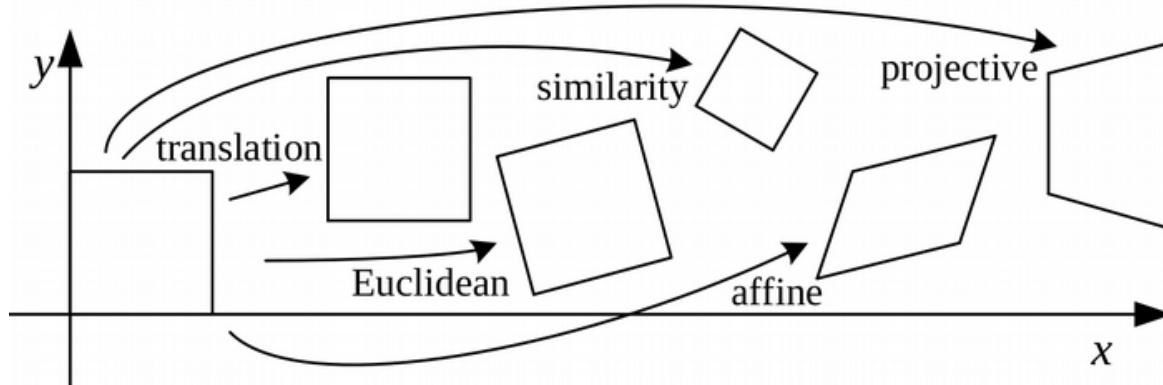
- **2D lines** can also be represented using homogeneous coordinates $\tilde{l} = (a, b, c)$. The corresponding line equation is

$$\tilde{x} \cdot \tilde{l} = ax + by + c = 0.$$

We can normalize the line equation vector so that $l = (\hat{n}_x, \hat{n}_y, d) = (\hat{n}, d)$ with $\|\hat{n}\| = 1$ this case, \hat{n} is the normal vector perpendicular to the line and d is its distance to the origin.

- The combination (Θ, d) is also known as *polar coordinates*.

2D transformations



- **2D translations** can be written as $x' = x + t$ or

$$x' = \begin{bmatrix} I & t \end{bmatrix} \cdot \bar{x},$$

where I is the 2×2 identity matrix.

- **Rotation + translation.** This transformation is also known as 2D rigid body motion or the 2D Euclidean transformation. It can be written as

$$x' = \begin{bmatrix} R & t \end{bmatrix} \bar{x}$$

where

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

is an orthonormal rotation matrix with $R \cdot R^T = I$ and $|R| = 1$.

- **Scaled rotation**, also known as the similarity transform, this transformation can be expressed as $x' = sRx + t$ where s is an arbitrary scale factor. It can also be written as

$$x' = \begin{bmatrix} sR & t \end{bmatrix} \bar{x} = \begin{bmatrix} a & -b & t_x \\ b & a & t_y \end{bmatrix} \bar{x}$$

where we no longer require that $a^2 + b^2 = 1$. The similarity transform preserves angles between lines.

- **Affine transformation** is written as $x' = A\bar{x}$ where A is an arbitrary

2×3 matrix, i.e.,

$$x' = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix} \bar{x}$$

Parallel lines remain parallel under affine transformations (does not keep angles between stight lines and distance between points)

- **Projective transformation**, also known as a perspective transform or homography, operates on homogeneous coordinates,

$$\tilde{x}' = \tilde{H}\tilde{x},$$

where \tilde{H} is an arbitrary 3×3 matrix. \tilde{H} is only defined up to a scale, and that two \tilde{H} matrices that differ only by scale are equivalent.

Hierarchy of 2D coordinate transformations

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \mathbf{I} & \mathbf{t} \end{bmatrix}_{2 \times 3}$	2	orientation	
rigid (Euclidean)	$\begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}_{2 \times 3}$	3	lengths	
similarity	$\begin{bmatrix} s\mathbf{R} & \mathbf{t} \end{bmatrix}_{2 \times 3}$	4	angles	
affine	$\begin{bmatrix} \mathbf{A} \end{bmatrix}_{2 \times 3}$	6	parallelism	
projective	$\begin{bmatrix} \tilde{\mathbf{H}} \end{bmatrix}_{3 \times 3}$	8	straight lines	

- Each transformation also preserves the properties (similarity preserves not only angles but also parallelism and straight lines).
- The 2×3 matrices are extended with a third $[0^T 1]$ row to form a full 3×3 matrix for homogeneous coordinate transformations.

Image translation Python



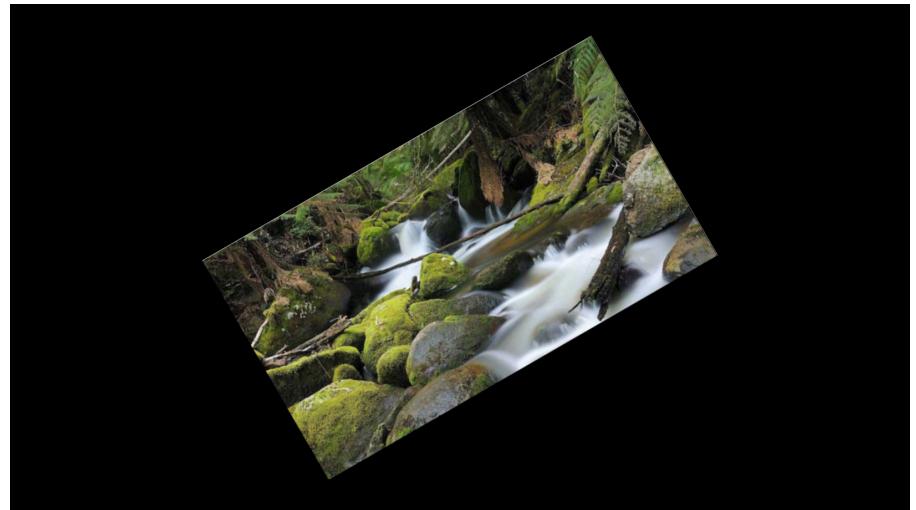
```
import cv2
import numpy as np

img = cv2.imread('images/input.jpg')
num_rows, num_cols = img.shape[:2]

translation_matrix = np.float32([ [1,0,70], [0,1,110] ])
img_translation = cv2.warpAffine(img, translation_matrix, (num_cols + 70, num_rows + 110))
translation_matrix = np.float32([ [1,0,-30], [0,1,-50] ])
img_translation = cv2.warpAffine(img_translation, translation_matrix, (num_cols + 70 + 30, num_rows + 110 + 50))

cv2.imshow('Translation', img_translation)
cv2.waitKey()
```

Image rotation Python



```
import cv2
import numpy as np
img = cv2.imread('images/input.jpg')
num_rows, num_cols = img.shape[:2]

translation_matrix = np.float32([ [1,0,int(0.5*num_cols)], [0,1,int(0.5*num_rows)] ])
rotation_matrix = cv2.getRotationMatrix2D((num_cols, num_rows), 30, 1)

img_translation = cv2.warpAffine(img, translation_matrix, (2*num_cols, 2*num_rows))
img_rotation = cv2.warpAffine(img_translation, rotation_matrix, (num_cols*2, num_rows*2))

cv2.imshow('Rotation', img_rotation)
cv2.imwrite('rotation.png', img_rotation)
```

Image scaling Python



```
import cv2

img = cv2.imread('images/input.jpg')

img_scaled = cv2.resize(img, None, fx=1.2, fy=1.2, interpolation = cv2.INTER_LINEAR)
cv2.imshow('Scaling - Linear Interpolation', img_scaled)

img_scaled = cv2.resize(img, None, fx=1.2, fy=1.2, interpolation = cv2.INTER_CUBIC)
cv2.imshow('Scaling - Cubic Interpolation', img_scaled)

img_scaled = cv2.resize(img,(450, 400), interpolation = cv2.INTER_AREA)
cv2.imshow('Scaling - Skewed Size', img_scaled)
cv2.imwrite('scaling.png', img_scaled)
```

Image affine transformation - Python



```
import cv2
import numpy as np

img = cv2.imread('images/input.jpg')
rows, cols = img.shape[:2]
src_points = np.float32([[0,0], [cols-1,0], [0,rows-1]])
dst_points = np.float32([[0,0], [int(0.6*(cols-1)),0], [int(0.4*(cols-1)),rows-1]])

affine_matrix = cv2.getAffineTransform(src_points, dst_points)
img_output = cv2.warpAffine(img, affine_matrix, (cols,rows))

cv2.imshow('Input', img)
cv2.imshow('Output', img_output)
cv2.imwrite('affine.png', img_output)
```

Image projective Python



```
import cv2
import numpy as np

img = cv2.imread('images/input.jpg')
rows, cols = img.shape[:2]
src_points = np.float32([[0,0], [cols-1,0], [0,rows-1], [cols-1,rows-1]])
dst_points = np.float32([[0,0], [cols-1,0], [int(0.33*cols),rows-1], [int(0.66*cols),rows-1]])

projective_matrix = cv2.getPerspectiveTransform(src_points, dst_points)
img_output = cv2.warpPerspective(img, projective_matrix, (cols,rows))

cv2.imshow('Input', img)
cv2.imshow('Output', img_output)
cv2.imwrite('projective.png', img_output)
```

Machine Vision

Image Processing - Filtering
lecture 3

Adam Szmigiełski

aszmigie@pjwstk.edu.pl

materials: *ftp(public) : //aszmigie/WMA*

Image filter

Modifies image pixels based on neighborhood pixels

10	5	3
4	5	1
1	1	7

Funkcja
→

	7	

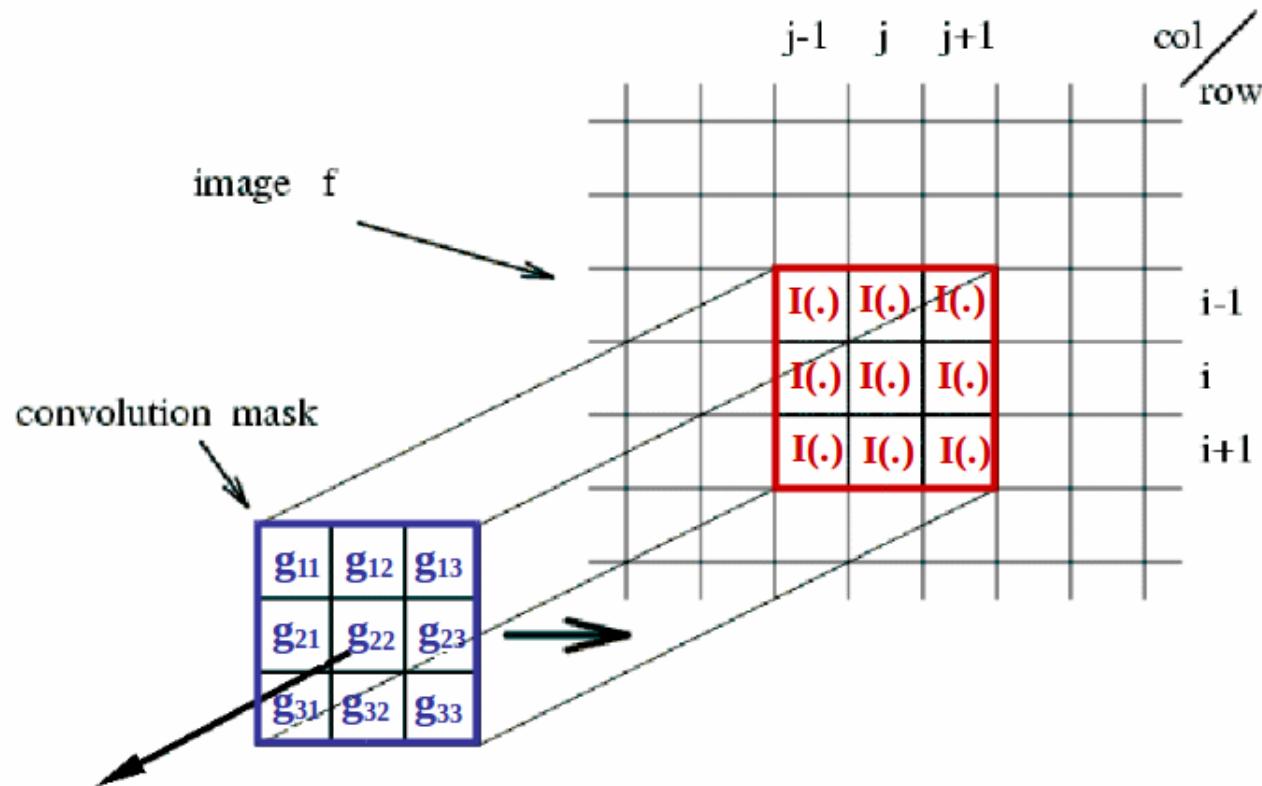
Linear filtration

$$\begin{array}{|c|c|c|} \hline 10 & 5 & 3 \\ \hline 4 & 5 & 1 \\ \hline 1 & 1 & 7 \\ \hline \end{array} \otimes \begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 0 & 0.5 & 0 \\ \hline 0 & 1.0 & 0.5 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline & 7 & \\ \hline & & \\ \hline \end{array}$$

Jądro splotu

- linear is the simplest and most useful,
- Replaces each pixel with a linear combination of neighbors,
- The function (way) for a linear combination is called a *convolution kernel*.

Line filter - convolution



$$f(i,j) = g_{11} I(i-1,j-1) + g_{12} I(i-1,j) + g_{13} I(i-1,j+1) + \\ g_{21} I(i,j-1) + g_{22} I(i,j) + g_{23} I(i,j+1) + \\ g_{31} I(i+1,j-1) + g_{32} I(i+1,j) + g_{33} I(i+1,j+1)$$

Neighborhood operator as linear filter

$$\begin{array}{|c|c|c|c|c|c|c|c|c|} \hline
 45 & 60 & 98 & 127 & 132 & 133 & 137 & 133 \\ \hline
 46 & 65 & 98 & 123 & 126 & 128 & 131 & 133 \\ \hline
 47 & 65 & 96 & 115 & 119 & 123 & 135 & 137 \\ \hline
 47 & 63 & 91 & 107 & 113 & 122 & 138 & 134 \\ \hline
 50 & 59 & 80 & 97 & 110 & 123 & 133 & 134 \\ \hline
 49 & 53 & 68 & 83 & 97 & 113 & 128 & 133 \\ \hline
 50 & 50 & 58 & 70 & 84 & 102 & 116 & 126 \\ \hline
 50 & 50 & 52 & 58 & 69 & 86 & 101 & 120 \\ \hline
 \end{array}
 \quad *
 \quad
 \begin{array}{|c|c|c|} \hline
 0.1 & 0.1 & 0.1 \\ \hline
 0.1 & 0.2 & 0.1 \\ \hline
 0.1 & 0.1 & 0.1 \\ \hline
 \end{array}
 \quad =
 \quad
 \begin{array}{|c|c|c|c|c|c|c|c|} \hline
 69 & 95 & 116 & 125 & 129 & 132 \\ \hline
 68 & 92 & 110 & 120 & 126 & 132 \\ \hline
 66 & 86 & 104 & 114 & 124 & 132 \\ \hline
 62 & 78 & 94 & 108 & 120 & 129 \\ \hline
 57 & 69 & 83 & 98 & 112 & 124 \\ \hline
 53 & 60 & 71 & 85 & 100 & 114 \\ \hline
 \end{array}$$

- The image on the left is convolved with the filter in the middle to yield the image on the right.
 - The light blue pixels indicate the source neighborhood for the light green destination pixel.
 - Output pixel's value is determined as a weighted sum of input pixel

values within a small neighborhood N :

$$g(i, j) = \sum_{k,l} f(i + k, j + l)h(k, l).$$

- The entries in the weight *kernel* or *mask* $h(k, l)$ are often called the *filter coefficients*. The above **correlation operator** can be notated as

$$g = f \otimes h.$$

A common variant on this formula is:

$$g(i, j) = \sum_{k,l} f(i - k, j - l)h(k, l) = \sum_{k,l} f(k, l)h(i - k, j - l),$$

where the sign of the offsets in f has been reversed, This is called the **convolution operator**,

$$g = f * h$$

and h is then called the *impulse response function* - The continuous version of convolution can be written as $g(x) = \int f(x - u)h(u)du$.

Correlation and convolution as matrix-vector multiplying

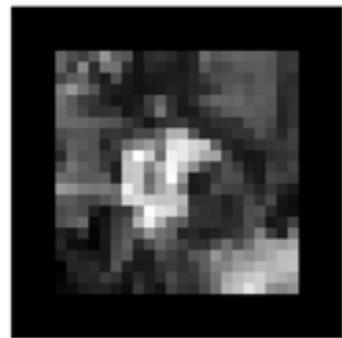
$$\begin{bmatrix} 72 & 88 & 62 & 52 & 37 \end{bmatrix} * \begin{bmatrix} 1/4 & 1/2 & 1/4 \end{bmatrix} \Leftrightarrow \frac{1}{4} \begin{bmatrix} 2 & 1 & . & . & . \\ 1 & 2 & 1 & . & . \\ . & 1 & 2 & 1 & . \\ . & . & 1 & 2 & 1 \\ . & . & . & 1 & 2 \end{bmatrix} \begin{bmatrix} 72 \\ 88 \\ 62 \\ 52 \\ 37 \end{bmatrix}$$

- One-dimensional convolution can be represented in matrix-vector form.
- *Correlation* and *convolution* can both be written as a matrix-vector multiply, if we first convert the two-dimensional images $f(i, j)$ and $g(i, j)$ into raster-ordered vectors f and g :

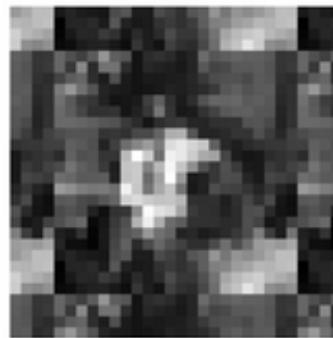
$$g = Hf,$$

where the (sparse) H matrix contains the convolution kernels.

Padding (border effects)



zero



wrap



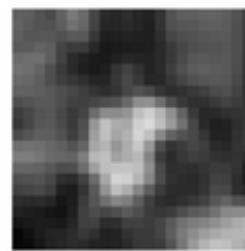
clamp



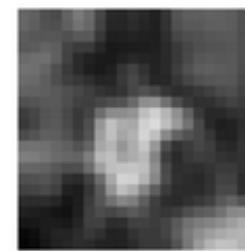
mirror



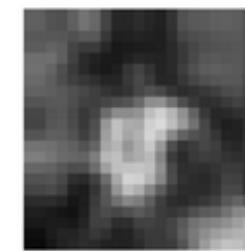
blurred zero



normalized zero



blurred clamp



blurred mirror

- Correlation result is smaller than the original image, which may not be desirable in many applications.
- This is because the neighborhoods of typical correlation and convolution operations extend beyond the image boundaries

- A number of different padding or extension modes have been developed for neighborhood operations:
 - *zero*: set all pixels outside the source image to 0 (a good choice for alpha-matted cutout images);
 - *constant (border color)*: set all pixels outside the source image to a specified border value;
 - *clamp (replicate or clamp to edge)*: repeat edge pixels indefinitely;
 - *(cyclic) wrap (repeat or tile)*: loop “around” the image in a “toroidal” configuration;
 - *mirror*: reflect pixels across the image edge;
 - *extend*: extend the signal by subtracting the mirrored version of the signal from the edge pixel value.

Separable filtering

$$\frac{1}{K^2} \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix}$$

$$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$$\frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

$$\frac{1}{8} \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\frac{1}{4} \begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

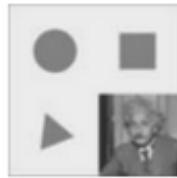
$$\frac{1}{K} \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}$$

$$\frac{1}{4} \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

$$\frac{1}{16} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

$$\frac{1}{2} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

$$\frac{1}{2} \begin{bmatrix} 1 & -2 & 1 \end{bmatrix}$$

(a) box, $K = 5$

(b) bilinear

(c) "Gaussian"

(d) Sobel

(e) corner

- The process of performing a convolution requires K^2 (multiply-add) operations per pixel, where K is the size
- In many cases, this operation can be significantly sped up by first performing a one-dimensional horizontal convolution followed by a one-dimensional vertical convolution, which requires a total of $2K$ operations per pixel.

- A convolution kernel for which this is possible is said to be separable.
- Two-dimensional kernel K corresponding to successive convolution with a horizontal kernel h and a vertical kernel v is the outer product of the two kernels,

$$K = vh^T$$

- Design of convolution kernels for computer vision applications is often influenced by their separability.
- This can often be done by inspection or by looking at the analytic form of the kernel
- more direct method is to treat the 2D kernel as a 2D matrix K and to take its singular value decomposition (SVD),

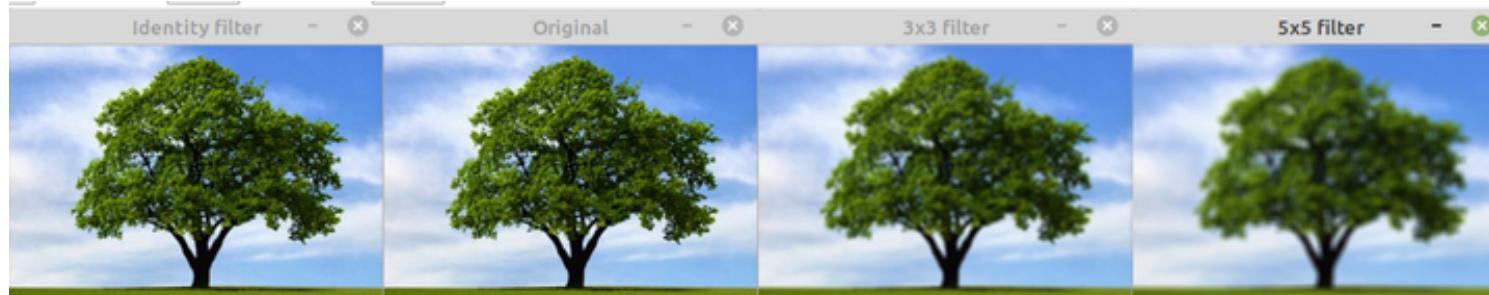
Median filtering

- Median filter selects the median value from each pixel's neighborhood,
- Median values can be computed in expected linear time,
- Since the noise value usually lies well outside the true values in the neighborhood, the median filter is able to filter away such bad pixels.
- Another possibility is to compute a weighted median - this is equivalent to minimizing the weighted objective function:

$$\sum_{k,l} w(k, l) |f(i + k, j + l) - g(i, j)|^p,$$

where $g(i, j)$ is the desired output value and $p = 1$ for the weighted median. The value $p = 2$ is the usual weighted mean, which is equivalent to correlation.

2D Convolution (Image Filtering) - Python



```
import cv2
import numpy as np

img = cv2.imread('images/tree_input.png', cv2.IMREAD_COLOR)

kernel_identity = np.array([[0,0,0], [0,1,0], [0,0,0]])
kernel_3x3 = np.ones((3,3), np.float32) / 9.0
kernel_5x5 = np.ones((5,5), np.float32) / 25.0

cv2.imshow('Original', img)
output = cv2.filter2D(img, -1, kernel_identity) # value -1 is to maintain source image
cv2.imshow('Identity filter', output)
output = cv2.filter2D(img, -1, kernel_3x3)
cv2.imshow('3x3 filter', output)
output = cv2.filter2D(img, -1, kernel_5x5)
cv2.imshow('5x5 filter', output)
cv2.waitKey(0)
```

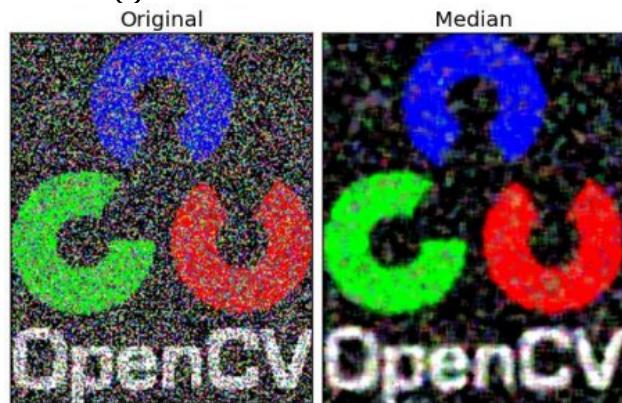
Image Blurring (Image Smoothing)

- **Averaging** - This is done by convolving an image with a normalized box filter e.g $K = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$
- **Gaussian Blurring** - Instead of a box filter, a Gaussian kernel is used.



```
blur = cv2.GaussianBlur(img,(5,5),0)
```

- **Median Blurring** - takes the median of all the pixels under the kernel area and the central element is replaced with this median value. This is highly effective against salt-and-pepper noise in an image.



```
median = cv2.medianBlur(img, 5)
```

Non-linear filtering

1	2	1	2	4
2	1	3	5	8
1	3	7	6	9
3	4	8	6	7
4	5	7	8	9

(a) median = 4

1	2	1	2	4
2	1	3	5	8
1	3	7	6	9
3	4	8	6	7
4	5	7	8	9

(b) α -mean= 4.6

2	1	0	1	2
2	0.1	0.3	0.4	0.3
1	0.3	0.6	0.8	0.6
0	0.4	0.8	1.0	0.8
1	0.3	0.6	0.8	0.6
2	0.1	0.3	0.4	0.3

(c) domain filter

0.0	0.0	0.0	0.0	0.2
0.0	0.0	0.0	0.4	0.8
0.0	0.0	1.0	0.8	0.4
0.0	0.2	0.8	0.8	1.0
0.2	0.4	1.0	0.8	0.4

(d) range filter

- In linear filter output pixel is a weighted summation of some number of input pixels.
- In many cases, however, better performance can be obtained by using a non-linear combination of neighboring pixels.

Bilateral filtering

- In the bilateral filter, the output pixel value depends on a weighted combination of neighboring pixel values

$$g(i, j) = \frac{\sum_{k,l} f(k, l)w(i, j, k, l)}{\sum_{k,l} w(i, j, k, l)}$$

The weighting coefficient $w(i, j, k, l)$ depends on the product of a domain kernel ,

$$d(i, j, k, l) = \exp\left(-\frac{(i - k)^2 + (j - l)^2}{2\sigma_d^2}\right)$$

and a data-dependent range kernel:

$$d(i, j, k, l) = \exp\left(-\frac{||f(i, j) - f(k, l)||^2}{2\sigma_r^2}\right)$$

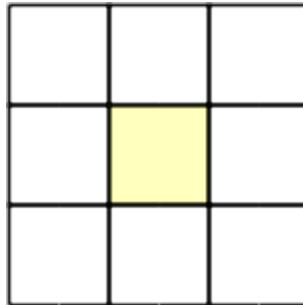
When multiplied together, these yield the data-dependent bilateral weight function

$$d(i, j, k, l) = \exp\left(-\frac{(i - k)^2 + (j - l)^2}{2\sigma_d^2} - \frac{\|f(i, j) - f(k, l)\|^2}{2\sigma_r^2}\right)$$

Bilateral Filtering `cv.bilateralFilter()` is highly effective in noise removal while keeping edges sharp. But the operation is slower compared to other filters.



Morphological operations



- The basic concept of morphological transformation is the so-called structural element of the image,
- This is a certain section of the image (a subset of elements) with one point - the central point,
- Structural element (circle with unit radius) on a square grid.
- The structuring element can be any shape, from a simple 3×3 box filter, to more complicated disc structures.

Structuring Element - Python

We may create a structuring elements with help of Numpy. In some cases, you may need elliptical or circular shaped kernels. For this purpose, OpenCV has a function, *cv.getStructuringElement()*.

```
# Rectangular Kernel
>>> cv.getStructuringElement(cv.MORPH_RECT,(5,5))
array([[1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1]], dtype=uint8)

# Elliptical Kernel
>>> cv.getStructuringElement(cv.MORPH_ELLIPSE,(5,5))
array([[0, 0, 1, 0, 0],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [0, 0, 1, 0, 0]], dtype=uint8)

# Cross-shaped Kernel
>>> cv.getStructuringElement(cv.MORPH_CROSS,(5,5))
array([[0, 0, 1, 0, 0],
       [0, 0, 1, 0, 0],
       [1, 1, 1, 1, 1],
       [0, 0, 1, 0, 0],
       [0, 0, 1, 0, 0]], dtype=uint8)
```

General algorithm of morphological transformation

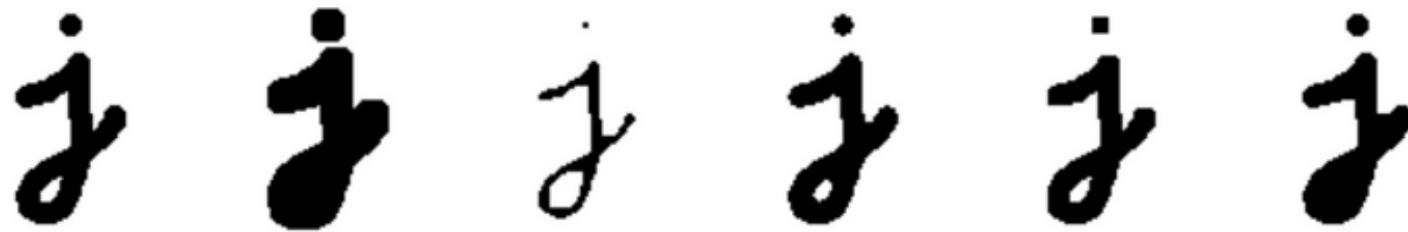
- The structural element is moved around the whole image and for each point of the image a comparison of image points and the structural element is made,
- At each point of the image, it is checked if the actual configuration of the pixels of the image in the vicinity of this point is consistent with the model structural element,
- If a match between the image pixel pattern and the structural element template is detected, some operation is performed on the tested point.

Morphological operations

- **Erosion** - an eroded figure is a set of all centers of circles with a radius r that are entirely contained within the area of X
- **Dylation** - the dylation figure is a set of centers of all B circles for which at least one point coincides with any point of the initial figure
- **Opening** - involves rolling wheel B on the inside of the figure's edge and rejecting all those points that cannot be reached by the circle.
- **Closing** - involves rolling wheel B on the outside of the figure's edge and adding to it all those points that cannot be reached by the circle.

Morphology - examples

The most common binary image operations are called morphological operations, since they change the shape of the underlying binary objects.



(a)

(b)

(c)

(d)

(e)

(f)

- Binary image morphology: (a) original image; (b) dilation; (c) erosion; (d) majority; (e) opening; (f) closing. The structuring element for all examples is a 5×5 square. The effects of majority are a subtle rounding of sharp corners. Opening fails to eliminate the dot, since it is not wide enough.

Erosion

The basic idea of erosion is just like soil erosion only, it erodes away the boundaries of foreground object .

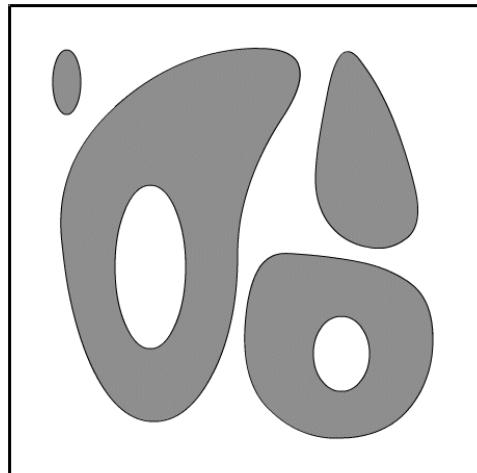


figura przed erozją

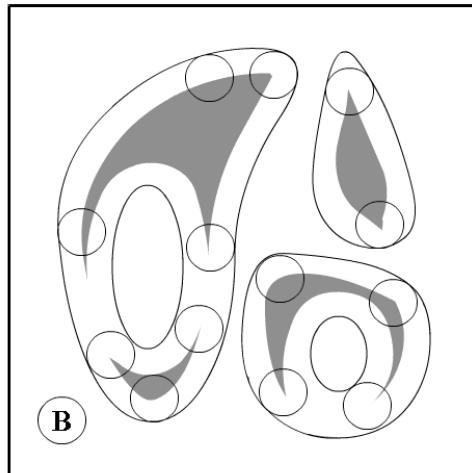


figura po erozji

```
import cv2 as cv
import numpy as np
img = cv.imread('j.png', 0)
kernel = np.ones((5, 5), np.uint8)
erosion = cv.erode(img, kernel, iterations = 1))
```

Dilation

It is just opposite of erosion. It increases the region in the image or size of foreground object increases

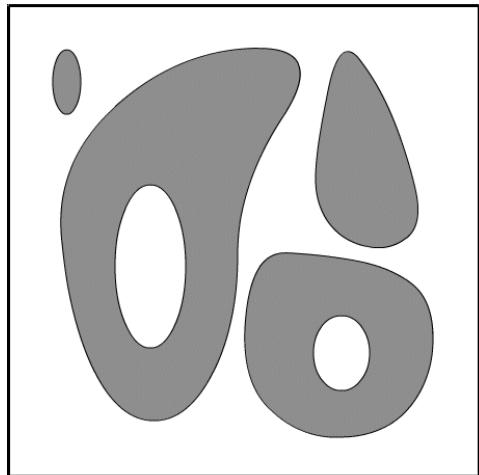


Figura przed dylatacją

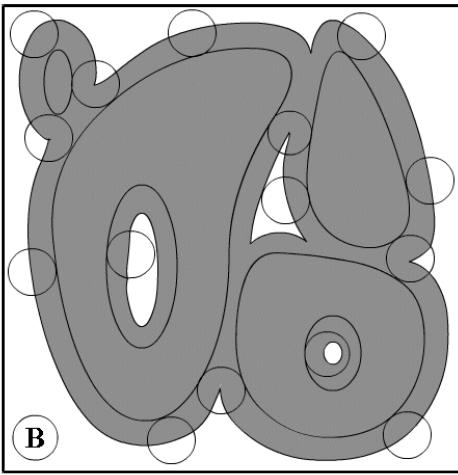


figura po dylatacji

```
dilation = cv.dilate(img, kernel, iterations = 1)
```

Opening

Opening is just another name of erosion followed by dilation. It is useful in removing noise, as we explained above.

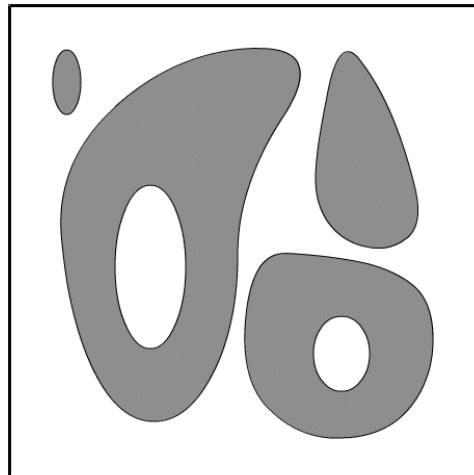


figura przed otwarciem

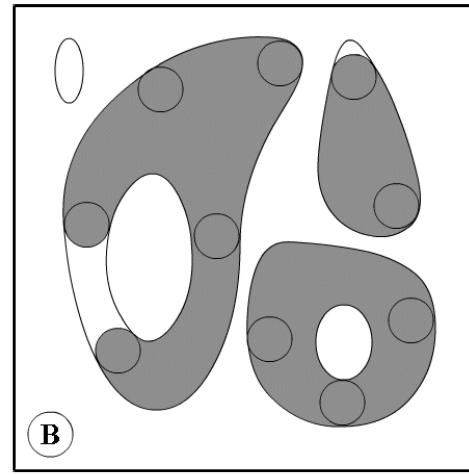


figura po otwarcia

- Opening = erosion + dilation: $O(x) = D(E(x))$

```
opening = cv.morphologyEx(img, cv.MORPH_OPEN, kernel)
```

Closing

Closing is reverse of Opening, Dilation followed by Erosion. It is useful in closing small holes inside the foreground objects, or small points on the object.

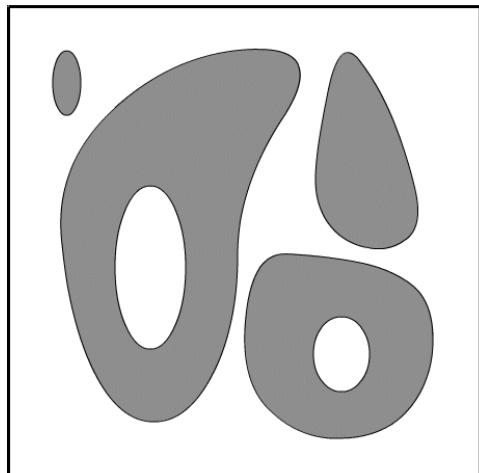


figura przed zamknięciem

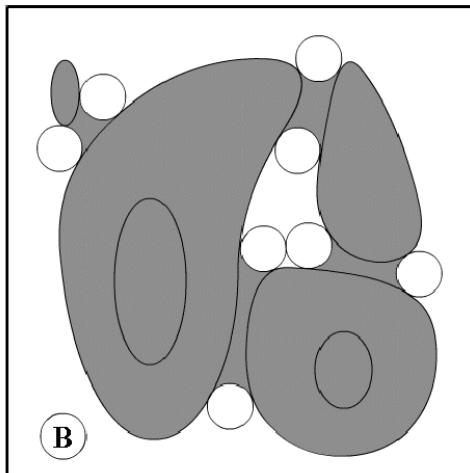


figura po zamknięciu

- Closing = dilation + erosion: $C(x) = E(D(x))$

```
closing = cv.morphologyEx(img, cv.MORPH_CLOSE, kernel)
```

Morphological Gradient

It is the difference between dilation and erosion of an image.

The result will look like the outline of the object



- Gradient = dilation - erosion: $G(x) = D(x) - E(x)$

```
gradient = cv.morphologyEx(img, cv.MORPH_GRADIENT, kernel)
```

Top Hat

It is the difference between input image and Opening of the image.
Below example is done for a 9×9 kernel.



- TopHat = Image - opening: $TH(x) = x - O(x)$

```
tophat = cv.morphologyEx(img, cv.MORPH_TOPHAT, kernel)
```

Black Hat

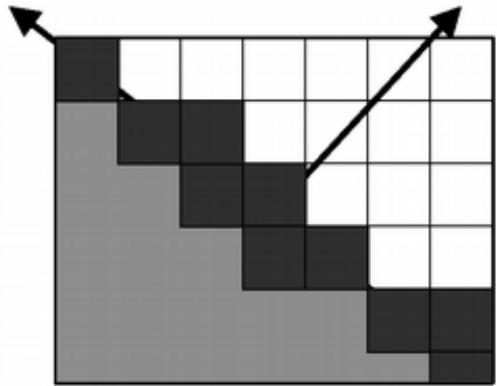
It is the difference between the closing of the input image and input image.



- $\text{BlackHat} = \text{Image} - \text{opening}$: $BH(x) = C(x) - x$

```
blackhat = cv.morphologyEx(img, cv.MORPH_BLACKHAT, kernel)
```

Normal to the edge



- Edge Size: $S = \sqrt{dx^2 + dy^2}$
- Edge direction: $\alpha = \arctan\left(\frac{dy}{dx}\right)$

Sobel's operator

$$S_1 = \begin{array}{|c|c|c|} \hline -1 & -2 & -1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$

$$S_2 = \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array}$$

- Edge Size: $\sqrt{S_1^2 + S_2^2}$
- Edge direction: $\arctan\left(\frac{S_1}{S_2}\right)$

Different directions of edge filtration

-1	0	1
-2	0	2
-1	0	1

0	1	2
-1	0	1
-2	-1	0

1	2	1
0	0	0
-1	-2	-1

2	1	0
1	0	-1
0	-1	-2

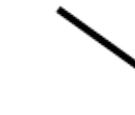


1	0	-1
2	0	-2
1	1	-1

0	-1	-2
-1	0	-1
2	1	0

-1	-2	-1
0	0	0
1	2	1

-2	-1	0
-1	0	1
0	1	2



Other linear filters

1	1	1
1	-2	1
-1	-1	-1

Prewitt 1

5	5	5
-3	0	-3
-3	-3	-3

Kirsch

-1	$-\sqrt{2}$	-1
0	0	0
1	$\sqrt{2}$	1

Frei & Chen

1	1	1
0	0	0
-1	-1	-1

Prewitt 2

1	2	1
0	0	0
-1	-2	-1

Sobel

- You can design your own linear filter.

Canny edge detection

Filtration stages:

1. stage 1 - smoothing,
2. stage 2 - calculating the gradient,
3. stage 3 - remove the maximum pixels.

Corner detection

Corner in the concept of “ sharply curving edge ”, intersection of 2 edges, lines

- Type of points of interest, points of interest (other: line endings, light or dark points),
- The first algorithms that detect corners:
 - edge detection,
 - tracing the extracted edges and detecting a sudden change in their direction,
- Newer generation of algorithms - high gradient curvature,
- It is recommended to smooth the image earlier.

Distance transforms

The distance transform is useful in quickly precomputing the distance to a curve or set of points.

Two commonly used metrics:

- city block

$$d_1(k, l) = |k| + |l|$$

- Manhattan distance

$$d_2(k, l) = \sqrt{(k^2 + l^2)}$$

Connected components

Another useful semi-global image operation is finding connected components, which are defined as regions of adjacent pixels that have the same input value or label.

Such statistics include for each individual region R :

- the area (number of pixels),
- the perimeter (number of boundary pixels),
- the centroid (average x and y values),
- the second moments,

$$M = \sum_{(x,y) \in R} \begin{bmatrix} x - \bar{x} \\ y - \bar{y} \end{bmatrix} \cdot \begin{bmatrix} x - \bar{x} & y - \bar{y} \end{bmatrix}$$

from which the major and minor axis orientation and lengths can be computed using eigenvalue analysis.

Machine Vision

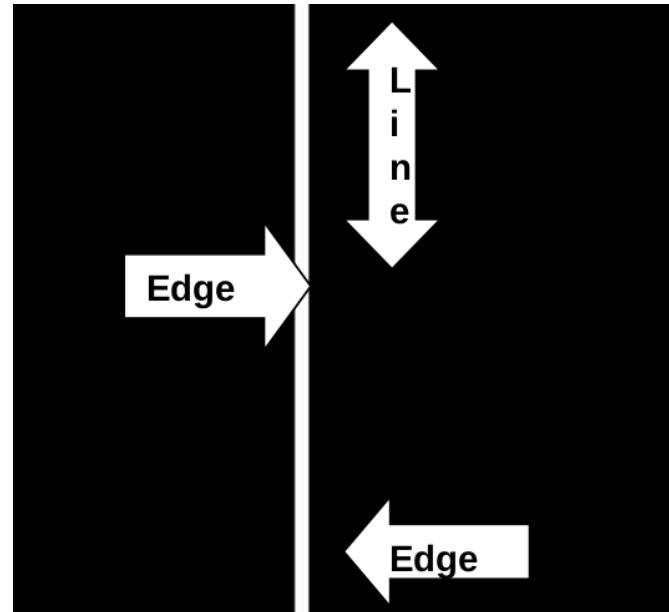
**Image Processing - Geometric features
detection
lecture 4**

Adam Szmigielski

aszmigie@pjwstk.edu.pl

materials: *ftp(public) : //aszmigie/WMAEnglish*

Edges and Lines Relationship



- Edges and lines are perpendicular,
- The line shown here is vertical and the edge direction is horizontal.
- In this case the transition from black to white occurs along a row, this is the edge direction, but the line is vertical along a column.

Goals of Edge Detection

- Produce a line drawing of a scene from an image of that scene.
- Important features can be extracted from the edges of an image (e.g., corners, lines, curves).
- These features are used by higher-level computer vision algorithms (e.g., segmentation, recognition).

Goals of an Edge Detector

Goal to construct edge detection operators that extracts

- the orientation information (information about the direction of the edge) and
- the strength of the edge.

Some methods can return information about the existence of an edge at each point for faster processing

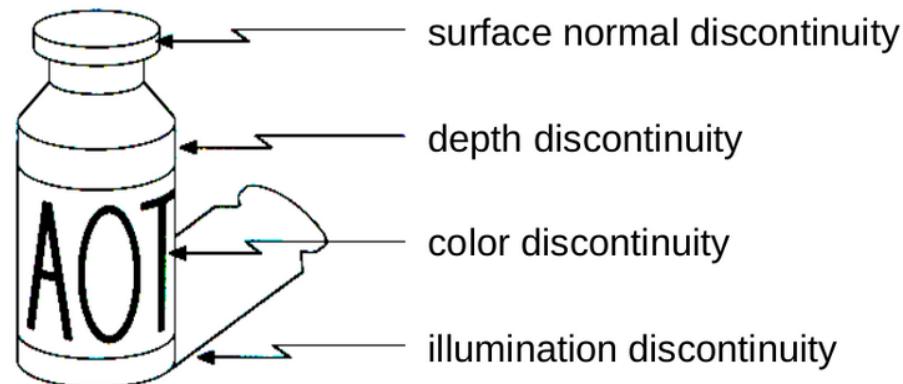
What Causes Intensity Changes?

Geometric events

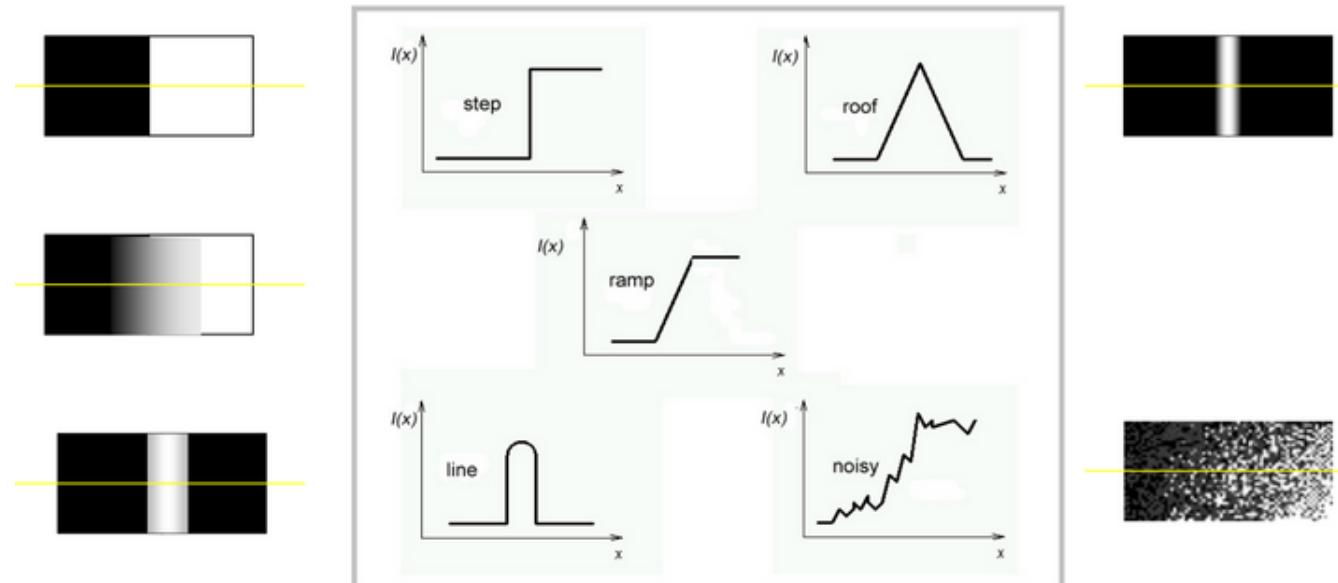
- surface orientation (boundary) discontinuities
- depth discontinuities
- color and texture discontinuities

Non-geometric events

- illumination changes,
- specularities,
- shadows
- inter-reflections

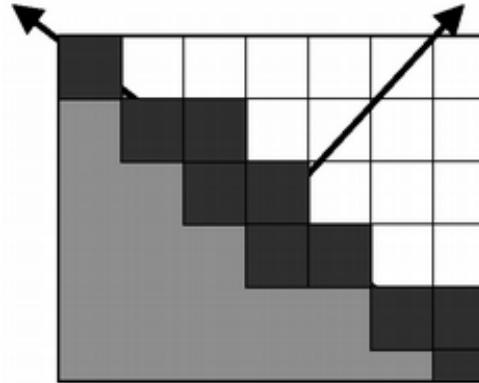


Edge Definition



- Edge is a boundary between two regions with relatively distinct gray level properties.
- Edges are pixels where the brightness function changes abruptly.
- Edge detectors are a collection of very important local image preprocessing methods used to locate (sharp) changes in the intensity function.

Edge Descriptors



- Edge Size: $S = \sqrt{dx^2 + dy^2}$
- Edge direction: unit vector to perpendicular to the edge normal.
 $\alpha = \arctan\left(\frac{dy}{dx}\right)$
- Edge normal: unit vector in the direction of maximum intensity change.
- Edge position or center: the image position at which the edge is located.
- Edge strength: related to the local image contrast along the normal.

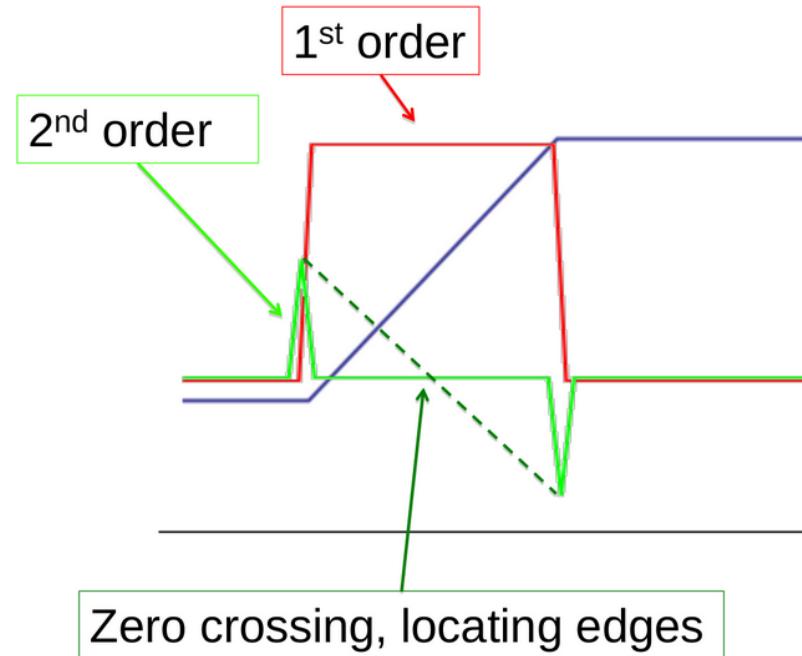
Main Steps in Edge Detection

1. **Smoothing:** suppress as much noise as possible, without destroying true edges,
2. **Enhancement:** apply differentiation to enhance the quality of edges (i.e. sharpening),
3. **Thresholding:** determine which edge pixels should be discarded as noise and which should be retained,
4. **Localization:** determine the exact edge location.

Edge Detection using Derivatives

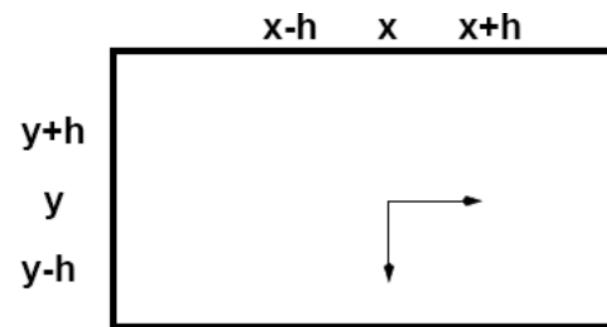
- Calculus describes changes of continuous functions using derivatives.
- An image is a 2D function, so operators describing edges are expressed using partial derivatives.
- Points which lie on an edge can be detected by either:
 - detecting local maxima or minima of the first derivative,
 - detecting the zero-crossing of the second derivative

1st and 2nd order derivatives



- 1st order derivative gives thick edges,
- 2nd order derivative gives double edge,
- 2nd order derivatives enhance fine detail much better.

Edge Detection Using First Derivative



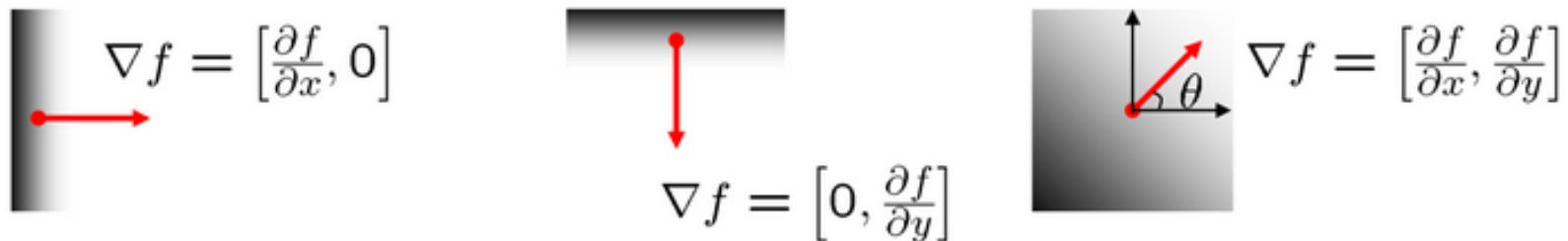
- Backward difference: $f'(x) \approx f(x) - f(x - 1)$
- Forward difference: $f'(x) = f(x + 1) - f(x)$
- Central difference: $f'(x) = f(x + 1) - f(x - 1)$

Edge detection

- Gradient-based edge operators
 - Prewitt
 - Sobel
 - Roberts
- Laplacian zero-crossings
- Canny edge detector
- Hough transform for detection of straight lines
- Circle Hough Transform

Edge Descriptors Using Gradient

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



- The gradient direction (orientation of edge normal) is given by:

$$\alpha[x, y] \approx \tan^{-1}\left(\frac{\partial f(x, y)}{\partial y} / \frac{\partial f(x, y)}{\partial x}\right)$$

- The edge strength is given by the gradient magnitude:

$$M[x, y] \approx \sqrt{\left(\frac{\partial f(x, y)}{\partial x}\right)^2 + \left(\frac{\partial f(x, y)}{\partial y}\right)^2}$$

Edge Detection Using Second Derivative

- Approximate finding maxima/minima of gradient magnitude by finding places where:

$$\frac{\partial f^2(x, y)}{\partial x^2} = 0$$

- Can't always find discrete pixels where the second derivative is zero - look for zero-crossing instead.

Gradient-based edge detection

- dea (continuous-space): local gradient magnitude indicates edge strength

$$|grad(f(x, y))| = \sqrt{\left(\frac{\partial f(x, y)}{\partial x}\right)^2 + \left(\frac{\partial f(x, y)}{\partial y}\right)^2}$$

- Digital image: use finite differences to approximate derivatives:

difference $\begin{pmatrix} -1 & 1 \end{pmatrix}$

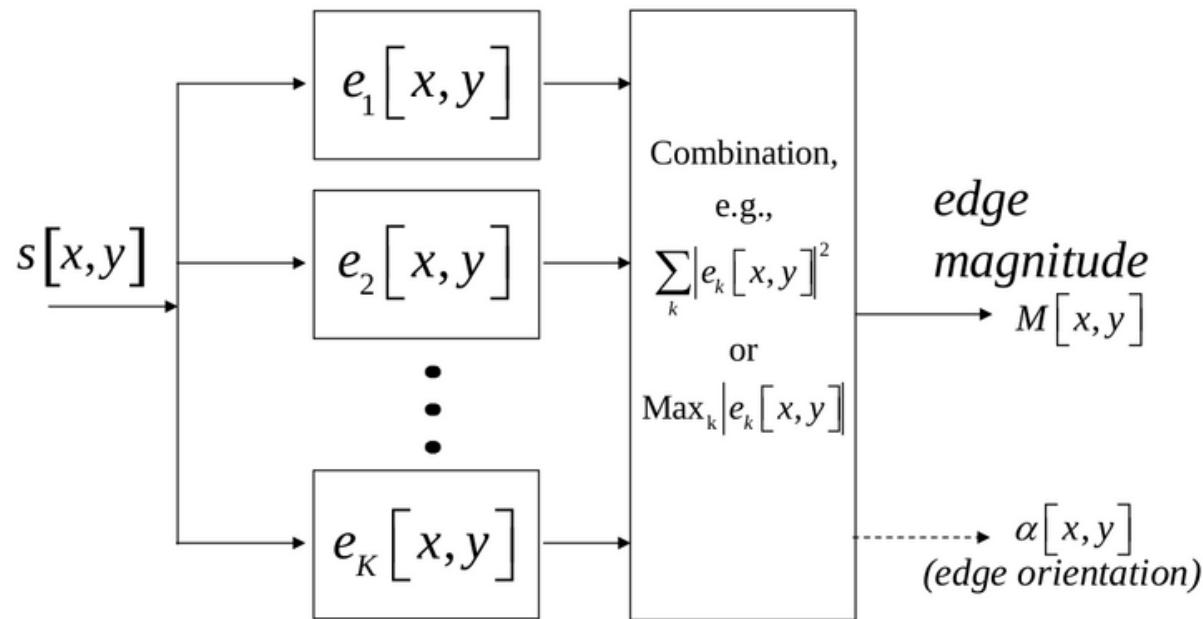
central difference $\begin{pmatrix} -1 & [0] & 1 \end{pmatrix}$

Prewitt $\begin{pmatrix} -1 & 0 & 1 \\ -1 & [0] & 1 \\ -1 & 0 & 1 \end{pmatrix}$

- Edge templates

Sobel $\begin{pmatrix} -1 & 0 & 1 \\ -2 & [0] & 2 \\ -1 & 0 & 1 \end{pmatrix}$

Practical edge detectors



- Edges can have any orientation,
- Typical edge detection scheme uses $K = 2$ edge templates,
- Some use $K > 2$.

Gradient filters (K=2)

Central Difference $\begin{pmatrix} 0 & 0 & 0 \\ -1 & [0] & 1 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & -1 & 0 \\ 0 & [0] & 0 \\ 0 & 1 & 0 \end{pmatrix}$

Roberts $\begin{pmatrix} [0] & 1 \\ -1 & 0 \end{pmatrix} \begin{pmatrix} [1] & 0 \\ 0 & -1 \end{pmatrix}$

Prewitt $\begin{pmatrix} -1 & 0 & 1 \\ -1 & [0] & 1 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} -1 & -1 & -1 \\ 0 & [0] & 0 \\ 1 & 1 & 1 \end{pmatrix}$

Sobel $\begin{pmatrix} -1 & 0 & 1 \\ -2 & [0] & 2 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} -1 & -2 & -1 \\ 0 & [0] & 0 \\ 1 & 2 & 1 \end{pmatrix}$

Gradient filters (K=8)

Kirsch $\begin{pmatrix} +5 & +5 & +5 \\ -3 & [0] & -3 \\ -3 & -3 & -3 \end{pmatrix} \begin{pmatrix} -3 & +5 & +5 \\ -3 & [0] & +5 \\ -3 & -3 & -3 \end{pmatrix} \begin{pmatrix} -3 & -3 & +5 \\ -3 & [0] & +5 \\ -3 & -3 & +5 \end{pmatrix} \begin{pmatrix} -3 & -3 & -3 \\ -3 & [0] & +5 \\ -3 & +5 & +5 \end{pmatrix}$

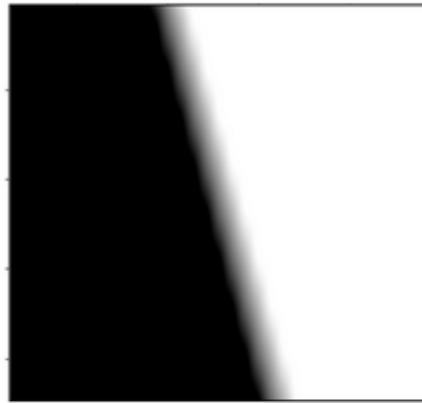
$$\begin{pmatrix} -3 & -3 & -3 \\ -3 & [0] & -3 \\ +5 & +5 & +5 \end{pmatrix} \begin{pmatrix} -3 & -3 & -3 \\ +5 & [0] & -3 \\ +5 & +5 & -3 \end{pmatrix} \begin{pmatrix} +5 & -3 & -3 \\ +5 & [0] & -3 \\ +5 & -3 & -3 \end{pmatrix} \begin{pmatrix} +5 & +5 & -3 \\ +5 & [0] & -3 \\ -3 & -3 & -3 \end{pmatrix}$$

Edge orientation

Central
Difference

$$\begin{pmatrix} 0 & 0 & 0 \\ -1 & [0] & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

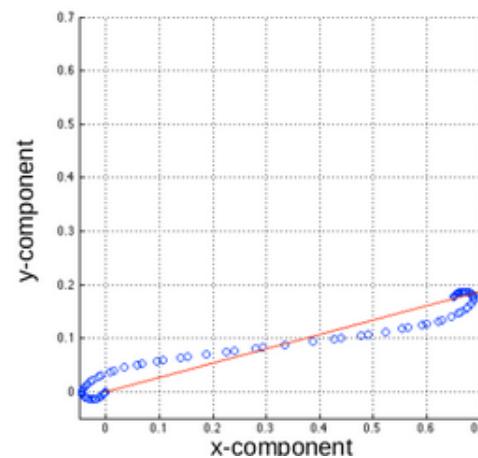
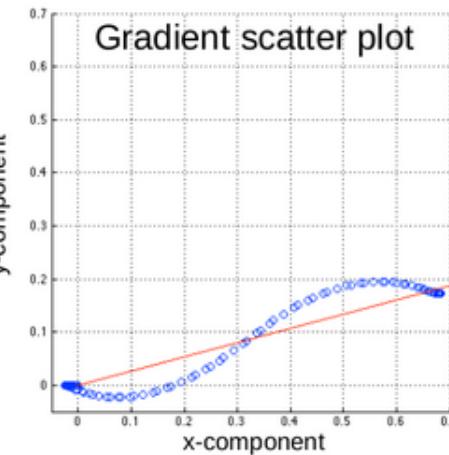
$$\begin{pmatrix} 0 & -1 & 0 \\ 0 & [0] & 0 \\ 0 & 1 & 0 \end{pmatrix}$$



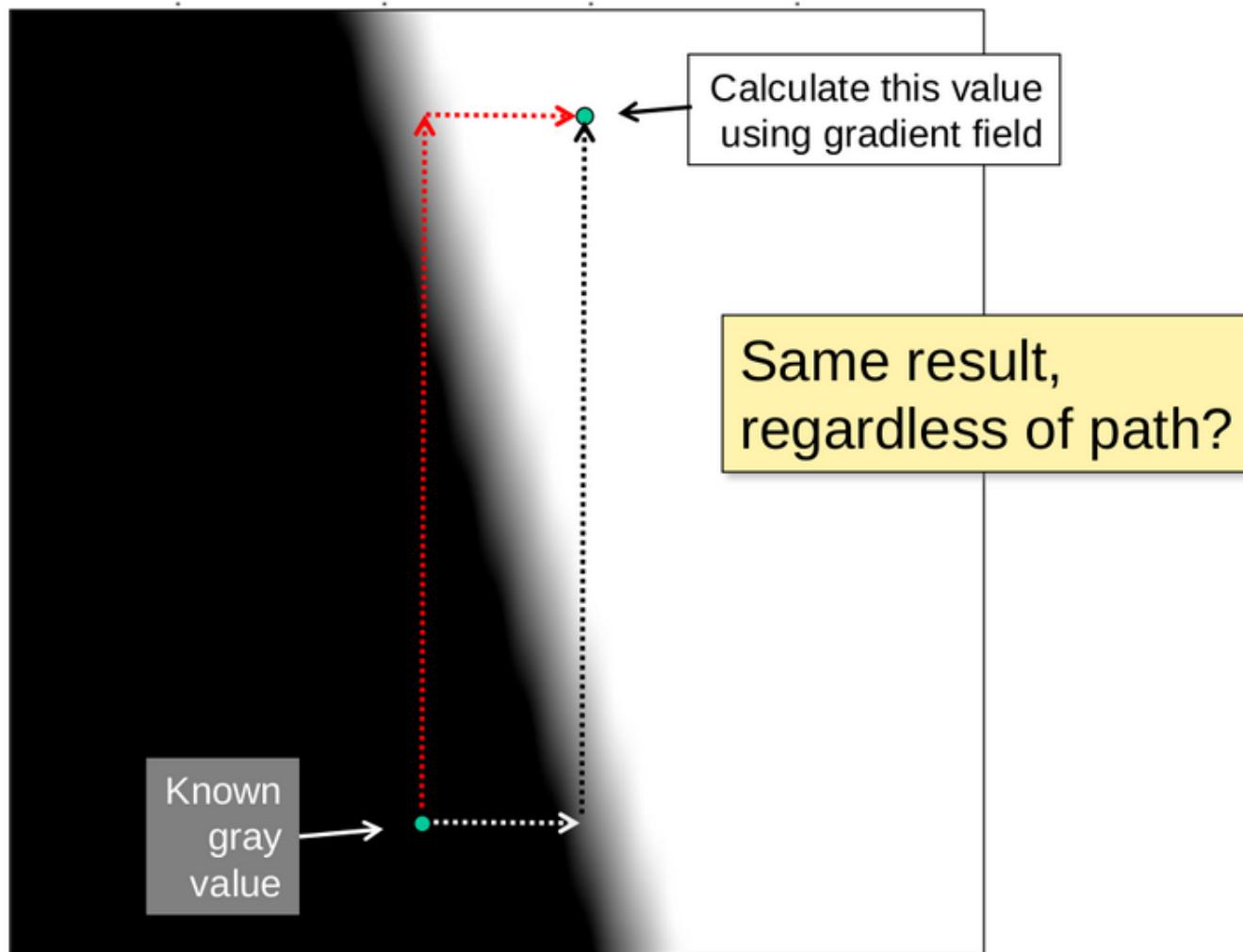
Roberts

$$\begin{pmatrix} [0] & 1 \\ -1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} [1] & 0 \\ 0 & -1 \end{pmatrix}$$



Gradient consistency problem

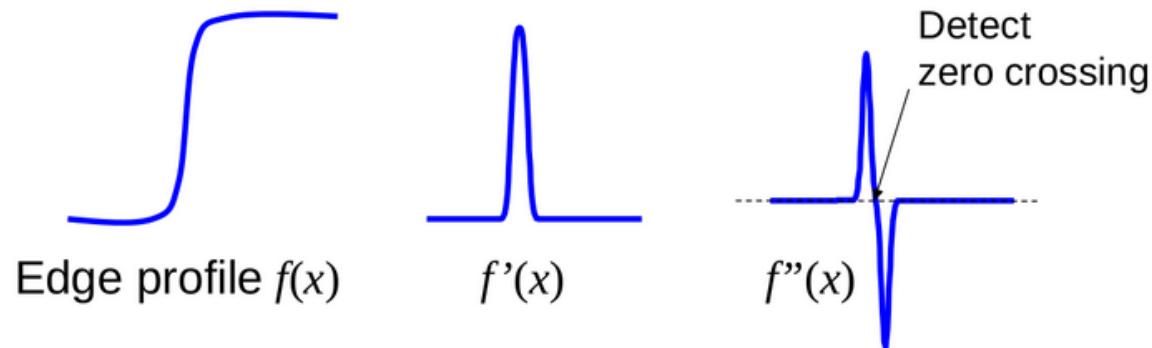


Laplacian operator

Detect edges by considering second derivative:

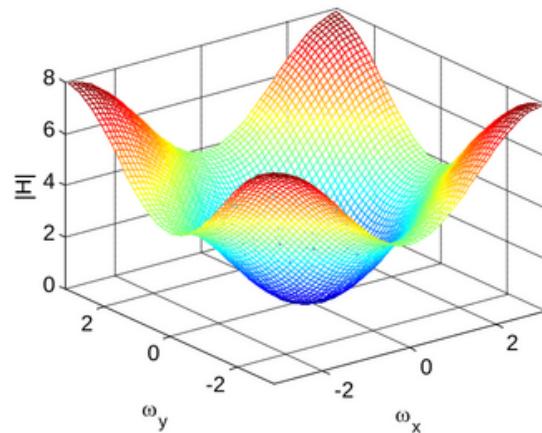
$$\nabla^2 f(x, y) = \frac{\partial^2 f(x, y)}{\partial x^2} + \frac{\partial^2 f(x, y)}{\partial y^2}$$

- Isotropic (rotationally invariant) operator
- Zero-crossings mark edge location

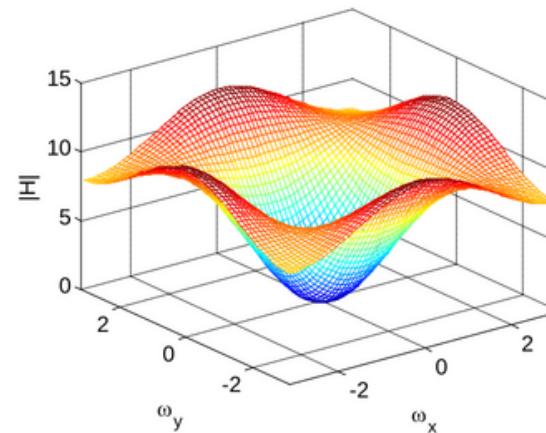


Approximations of Laplacian operator by 3×3 filter

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & [-4] & 1 \\ 0 & 1 & 0 \end{pmatrix}$$



$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & [-8] & 1 \\ 1 & 1 & 1 \end{pmatrix}$$



- Sensitive to very fine detail and noise -> blur image first,
- Responds equally to strong and weak edges -> suppress zero-crossings with low gradient magnitude.

Second Derivative in 2D: Laplacian

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

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

$$\nabla^2 f = -4f(i, j) + f(i, j + 1) + f(i, j - 1) + f(i + 1, j) + f(i - 1, j)$$

$$\begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 1 & -2 & 1 \\ \hline 0 & 0 & 0 \\ \hline \end{array} + \begin{array}{|c|c|c|} \hline 0 & 1 & 0 \\ \hline 0 & -2 & 0 \\ \hline 0 & 1 & 0 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 0 & 1 & 0 \\ \hline 1 & -4 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array}$$

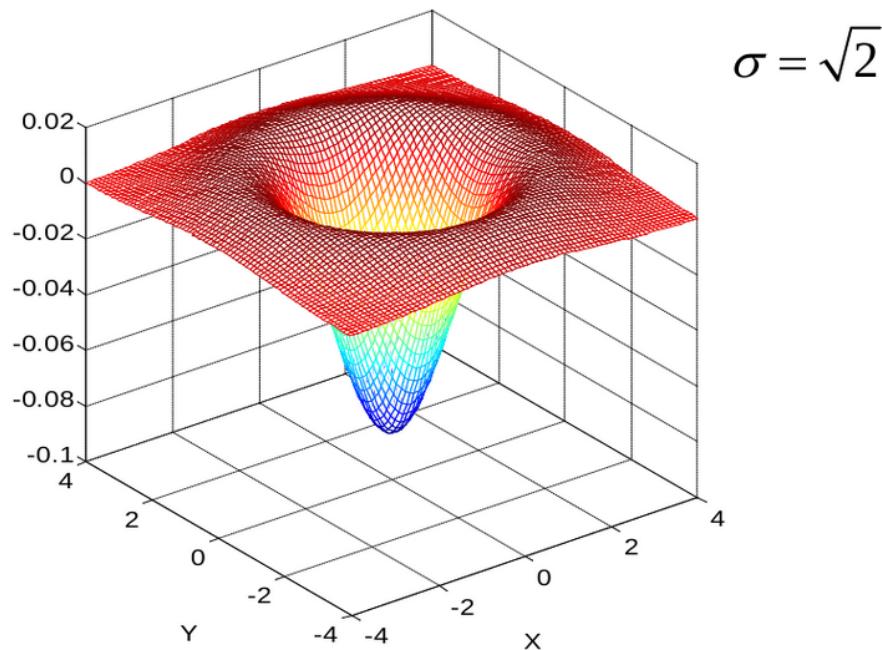
Properties of Laplacian

- It is an isotropic operator.
- It is cheaper to implement than the gradient (i.e., one mask only).
- It does not provide information about edge direction.
- It is more sensitive to noise (i.e. differentiates twice).

Laplacian of Gaussian (LoG)

Filtering of image with Gaussian and Laplacian operators can be combined into convolution with Laplacian of Gaussian (LoG) operator

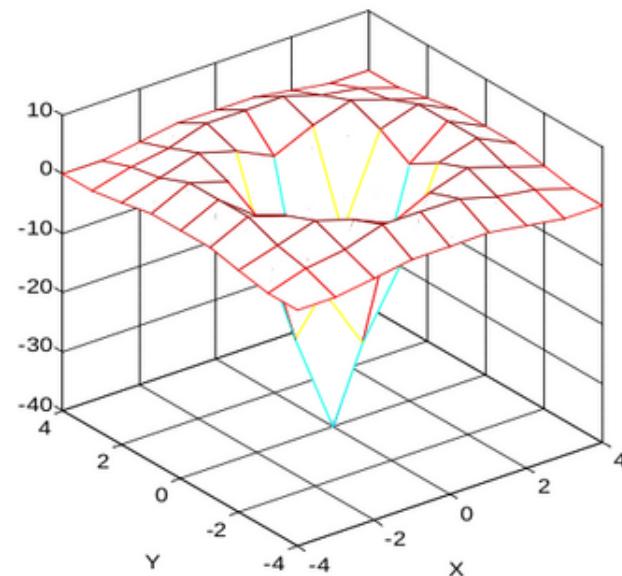
$$LoG(x, y) = -\frac{1}{\pi\sigma^4} \left(1 - \frac{x^2 + y^2}{2\sigma^2}\right) e^{-\frac{x^2+y^2}{2\sigma^2}}$$



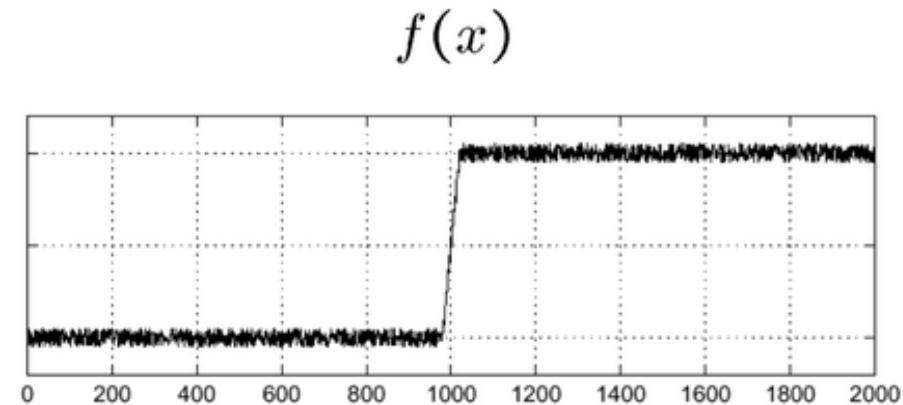
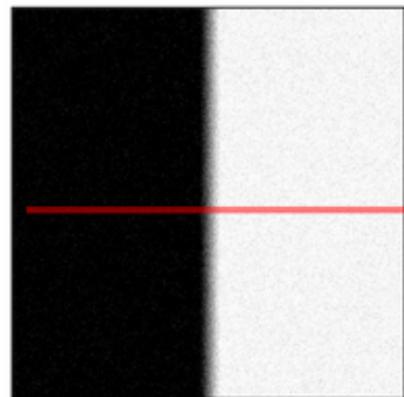
Discrete approximation of Laplacian of Gaussian

$$\sigma = \sqrt{2}$$

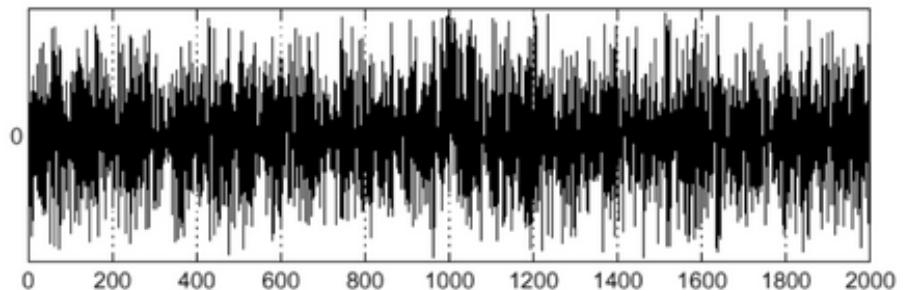
0	0	1	2	2	2	1	0	0
0	2	3	5	5	5	3	2	0
1	3	5	3	0	3	5	3	1
2	5	3	-12	-23	-12	3	5	2
2	5	0	-23	-40	-23	0	5	2
2	5	3	-12	-23	-12	3	5	2
1	3	5	3	0	3	5	3	1
0	2	3	5	5	5	3	2	0
0	0	1	2	2	2	1	0	0



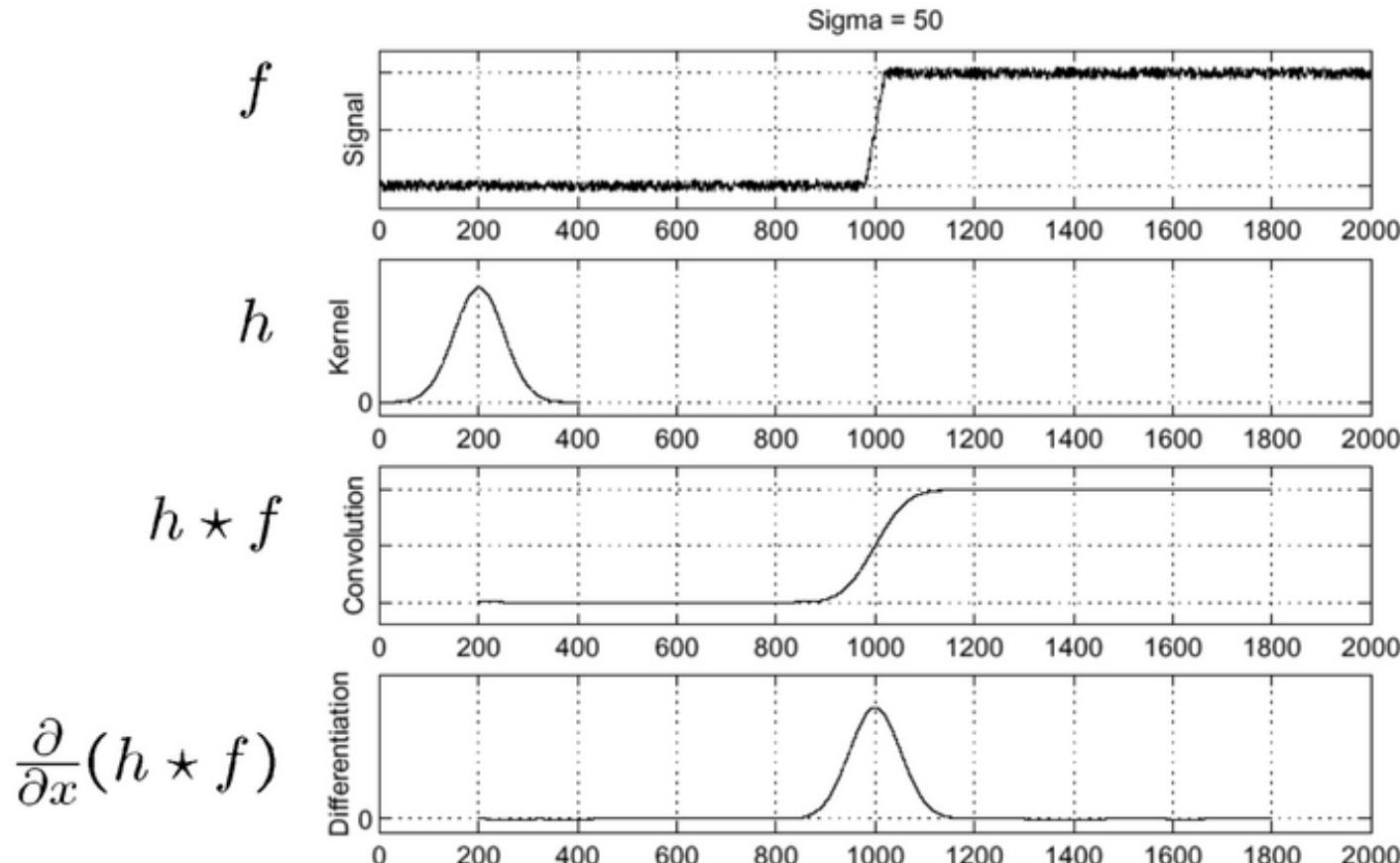
Noise Effect on Derivates



$$\frac{d}{dx}f(x)$$



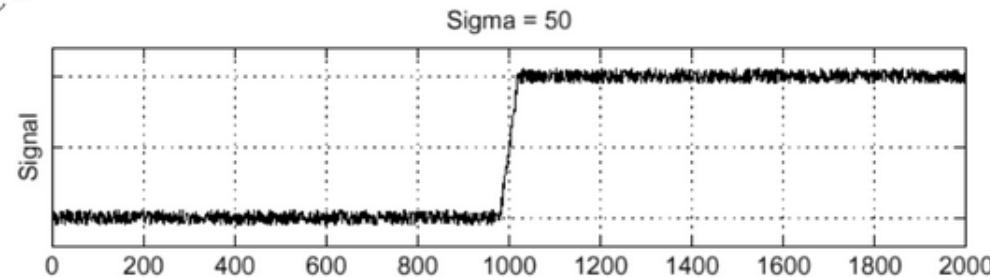
Effect of Smoothing on Derivatives



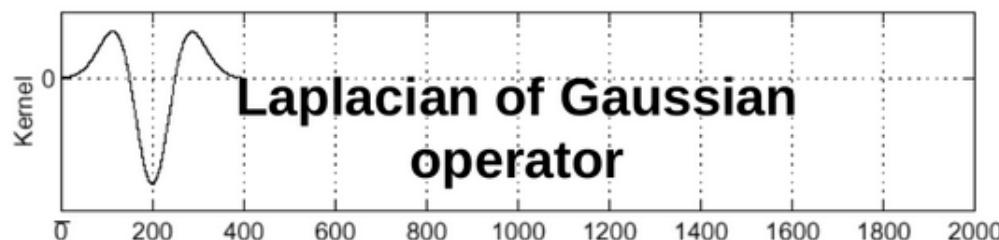
Effect of Smoothing on Laplacian of Gaussian

$$\cdot \frac{\partial^2}{\partial x^2}(h * f)$$

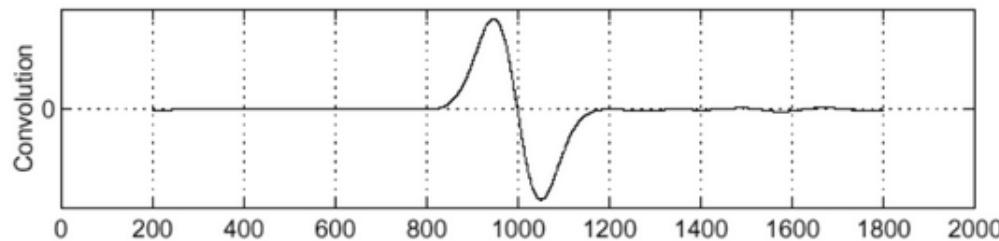
f



$$\frac{\partial^2}{\partial x^2} h$$



$$(\frac{\partial^2}{\partial x^2} h) * f$$



Canny edge detector

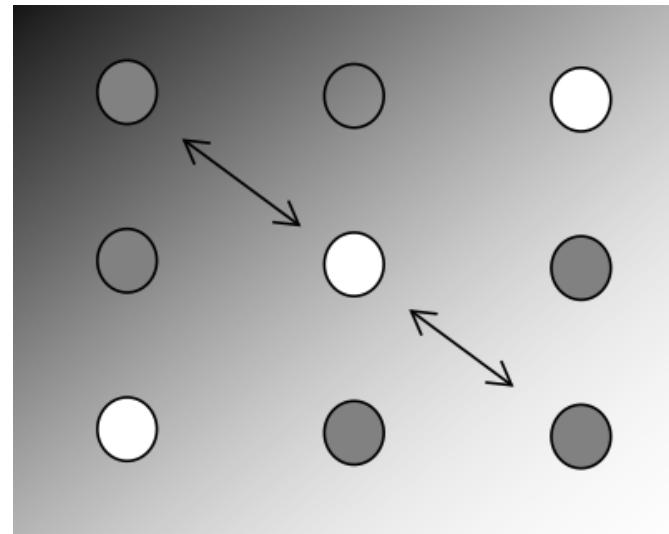
1. Smooth image with a Gaussian filter,
2. Approximate gradient magnitude and angle (use Sobel, Prewitt . . .)

$$M[x, y] \approx \sqrt{\left(\frac{\partial f(x, y)}{\partial x}\right)^2 + \left(\frac{\partial f(x, y)}{\partial y}\right)^2} \quad \alpha[x, y] \approx \tan^{-1}\left(\frac{\partial f(x, y)}{\partial y} / \frac{\partial f(x, y)}{\partial x}\right)$$

3. Apply nonmaxima suppression to gradient magnitude,
4. Double thresholding to detect strong and weak edge pixels,
5. Reject weak edge pixels not connected with strong edge pixels.

Canny nonmaxima suppression

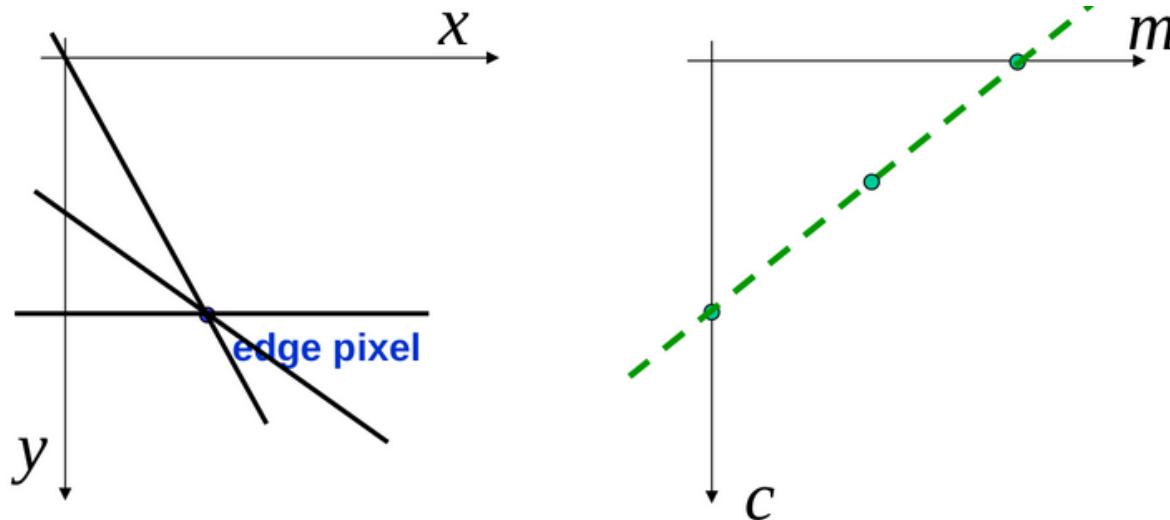
- Quantize edge normal to one of four directions:
horizontal, -45° , vertical, $+45^\circ$
- If $M[x, y]$ is smaller than either of its neighbors in edge normal direction -> suppress; else keep.



Canny thresholding and suppression of weak edges - hysteresis thresholding

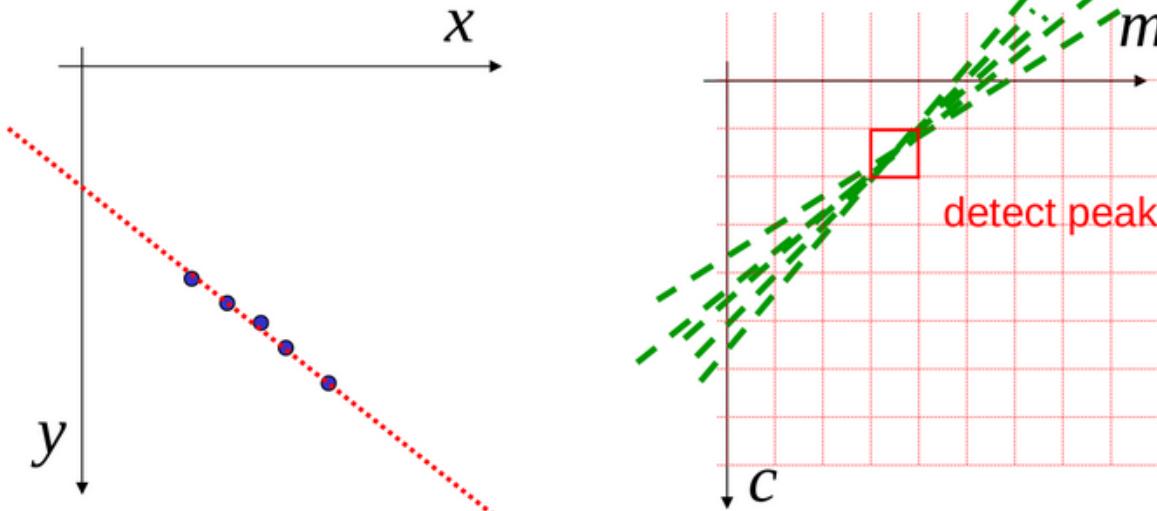
- Double-thresholding of gradient magnitude:
 - Strong edge: $M[x, y] \geq \Theta_{high}$ definitely an edge,
 - Weak edge: $\Theta_{high} > M[x, y] \geq \Theta_{low}$ maybe an edge,
 - No edge: $M[x, y] < \Theta_{low}$ definitely not an edge,
- Typical setting: $\Theta_{high}/\Theta_{low} = 2 \dots 3$
- Region labeling of edge pixels
- Reject regions without strong edge pixels
- For “maybe” edges, decide on the edge if neighboring pixel is a strong edge.

Hough transform



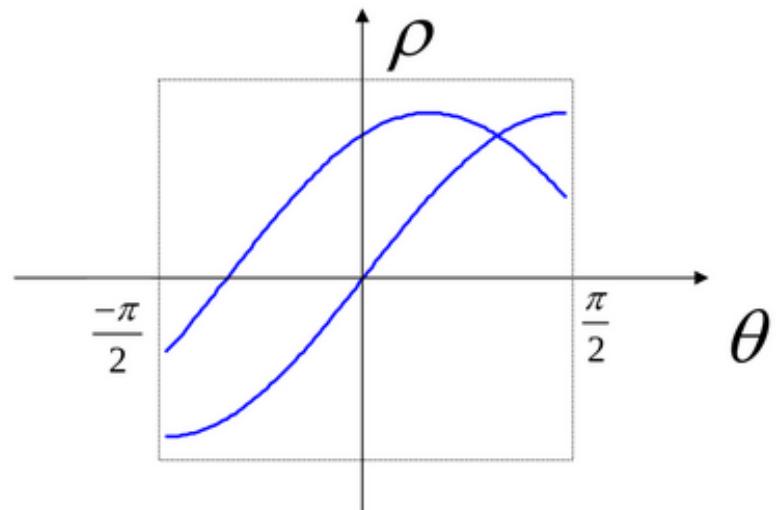
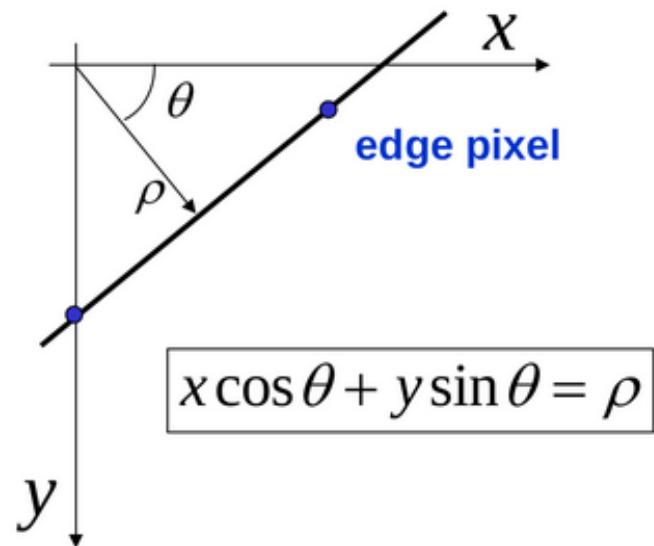
- Problem: fit a straight line (or curve) to a set of edge pixels
- Hough transform (1962): generalized template matching technique
- Consider detection of straight lines $y = mx + c$

Hough transform



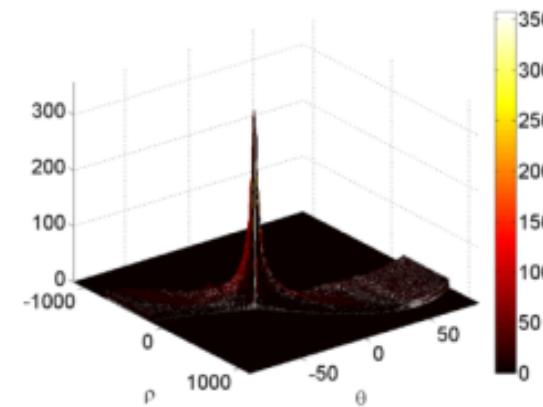
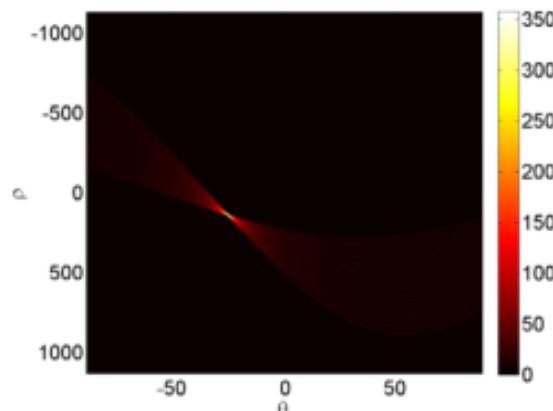
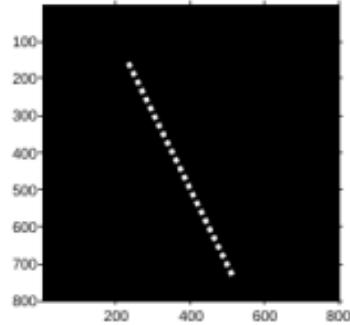
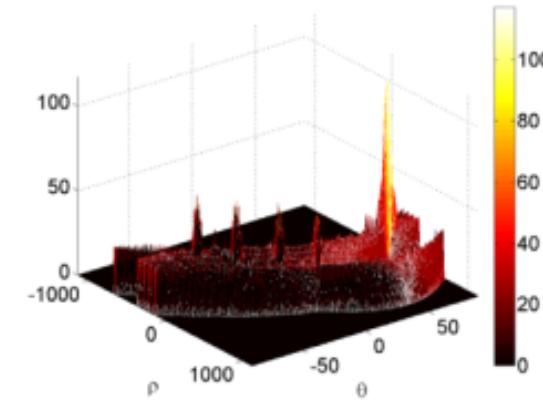
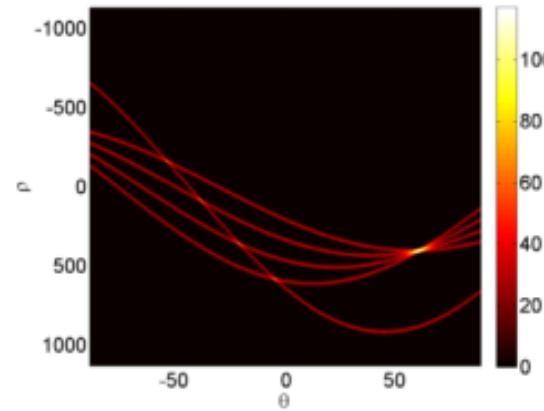
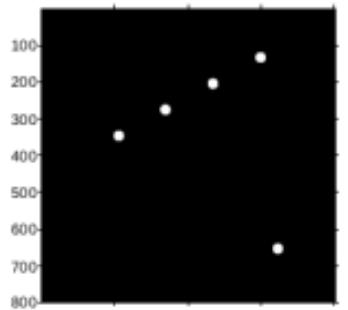
- Subdivide (m, c) plane into discrete “bins” initialize all bin counts by 0,
- Draw a line in the parameter space $[m, c]$ for each edge pixel $[x, y]$ and increment bin counts along line.
- Detect peak(s) in $[m, c]$ plane.

Hough transform

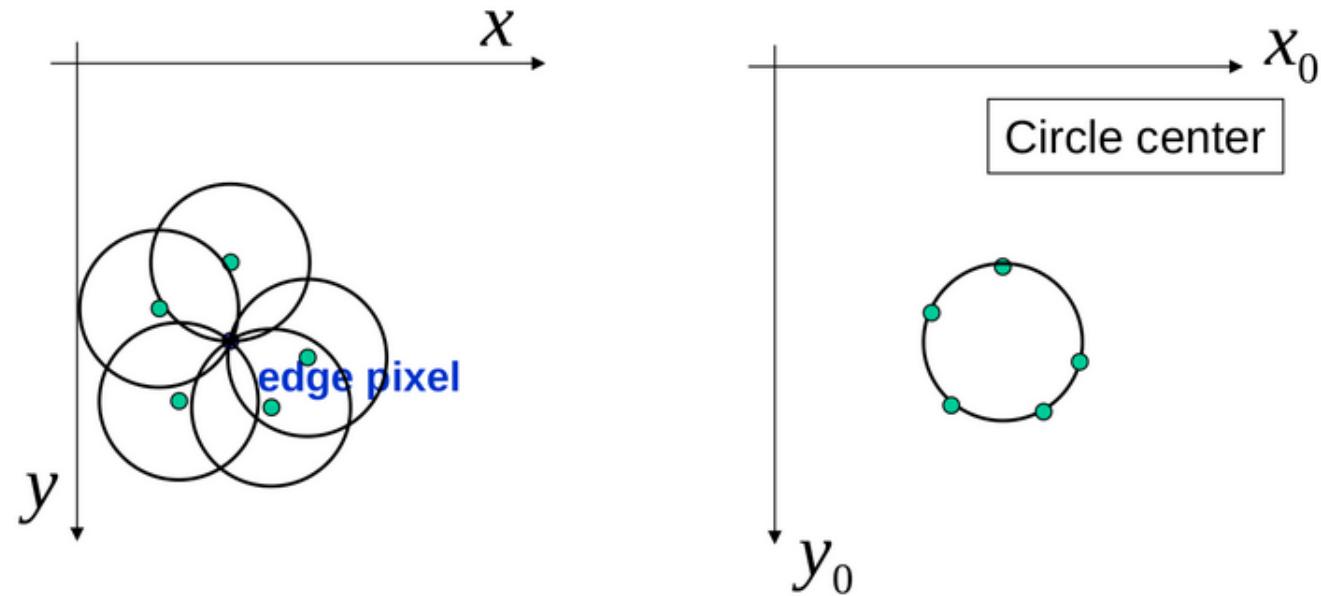


- Alternative parameterization avoids infinite-slope problem
- Similar to Radon transform

Hough transform example

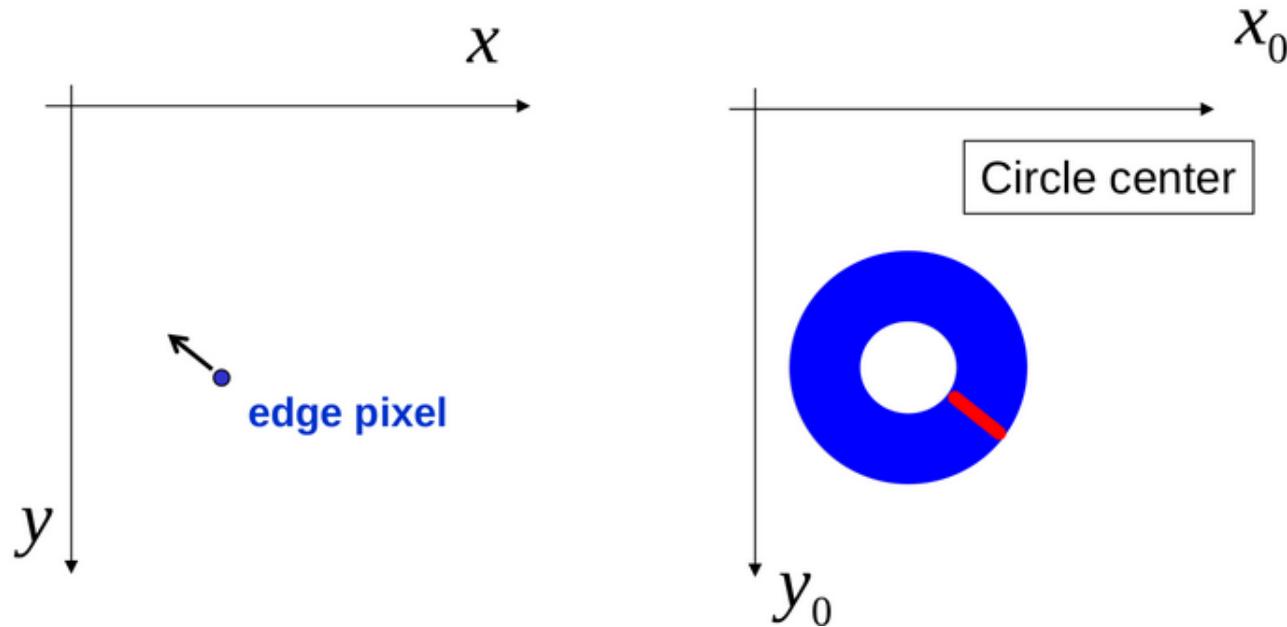


Circle Hough Transform



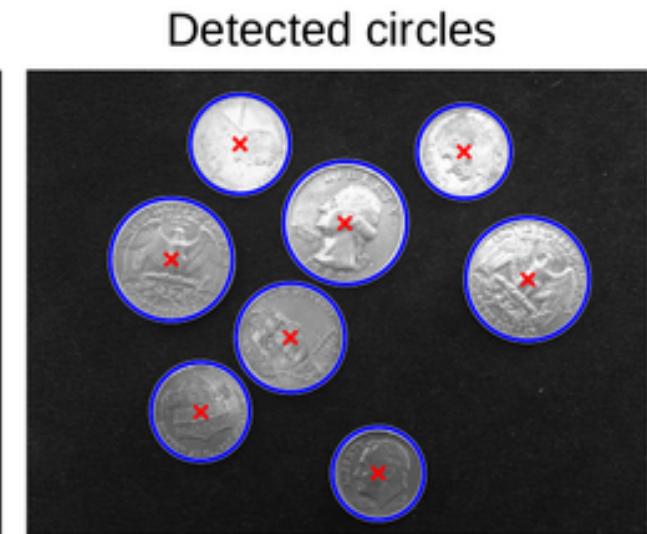
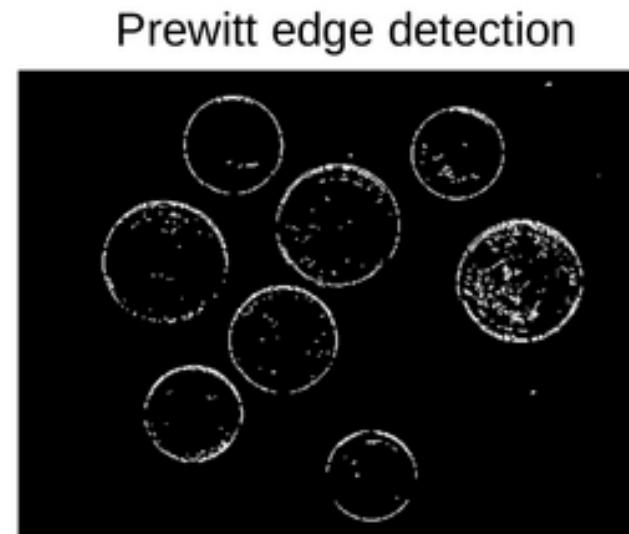
- Find circles of fixed radius r
- Equivalent to convolution (template matching) with a circle

Circle Hough Transform for unknown radius



- 3 – d Hough transform for parameters (x_0, y_0, r) ,
- 2 – d Hough transform aided by edge orientation -> “spokes” in parameter space.

Circle detection by Hough transform - example



Corner detection

Corner in the concept of “ sharply curving edge ”, intersection of 2 edges, lines

- Type of points of interest, points of interest (other: line endings, light or dark points),
- The first algorithms that detect corners:
 - edge detection,
 - tracing the extracted edges and detecting a sudden change in their direction,
- Newer generation of algorithms - high gradient curvature,
- It is recommended to smooth the image earlier.

Task for laboratory

A certain number of 5 zł coins and a certain 5 gr. coins are given. The coins can be located in or out of the tray. Write a program that will determine the number and type of coins in and out of the tray. Test the program on pictures. Tune the system to get the best possible efficiency. Consider pre image analysis (such as blurring, thresholding, edge detection, etc.). Use both Hough transformations. The effectiveness of the system will be assessed.

Computer Vision

Feature detection - lecture 5

Adam Szmigielski

aszmigie@pjwstk.edu.pl

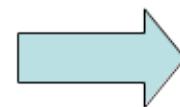
materials: *ftp(public) : //aszmigie/WMAEnglish*

Feature detection

- Feature detection and matching are an essential component of many computer vision applications,
- Kind of feature that we may notice are specific locations in the images,
- These kinds of localized feature are often called *keypoint features* or *interest points*
- Another class of important features are edges,
- Edges can be grouped into longer curves and straight line segments.

Features in computer vision

Features of computer vision - local, significant and detectable parts of the image.



cecha 1
cecha 2
:
cecha N



Algorytmy wizji
komputerowej

- The feature is an information carrier,
- The feature does not depend on the point of view, lighting,
- The feature reduces computational complexity.

Features of computer vision

The use of computer vision features

- Calibration,
- Image segmentation,
- Correspondence in many images (stereo, motion structure),
- Detection, classification of objects.

Properties of computer vision features

- Independence from the point of view (scale, orientation, translation)
- Independence from lighting
- Uniqueness
- Matched to the task

Properties of good features

- **Local:** features are local, robust to occlusion and clutter (no prior segmentation),
- **Accurate:** precise localization.
- **Invariant (or covariant)**
- **Robust:** noise, blur, compression, etc. do not have a big impact on the feature.
- **Distinctive:** individual features can be matched to a large database of objects.
- **Efficient:** close to real-time performance.

Invariant / Covariant

- A function $f()$ is **invariant** under some transformation $T()$ if its value does not change when the transformation is applied to its argument:

$$f(x) = y \Rightarrow f(T(x)) = y$$

- A function $f()$ is **covariant** when it commutes with the transformation $T()$:

$$f(x) = y \Rightarrow f(T(x)) = T(f(x)) = T(y)$$

Cross-Correlation and Auto-correlation

- **Cross Correlation** to find the template in an image,

$$f \otimes h = \sum_k \sum_l f(k, l)h(i + k, j + l)$$

- **Auto-correlation** maximum indicate high similarity

$$f \otimes f = \sum_k \sum_l f(k, l)f(i + k, j + l)$$

Points and patches

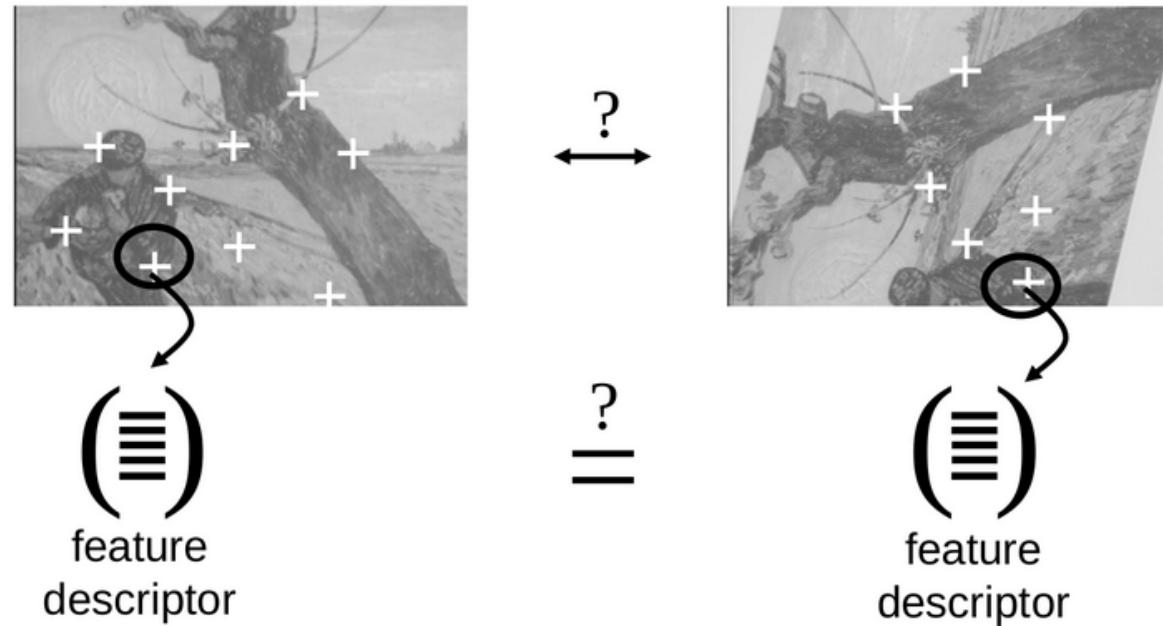
- Point features can be used to find a sparse set of corresponding locations in different images, often as a reference to compute the camera pose,
- There are two main approaches to finding feature points and their correspondences.
 - to find features in one image that can be accurately tracked using a local search technique, such as correlation or least squares,
 - to independently detect features in all the images under consideration and then match features based on their local appearance

Interest Points



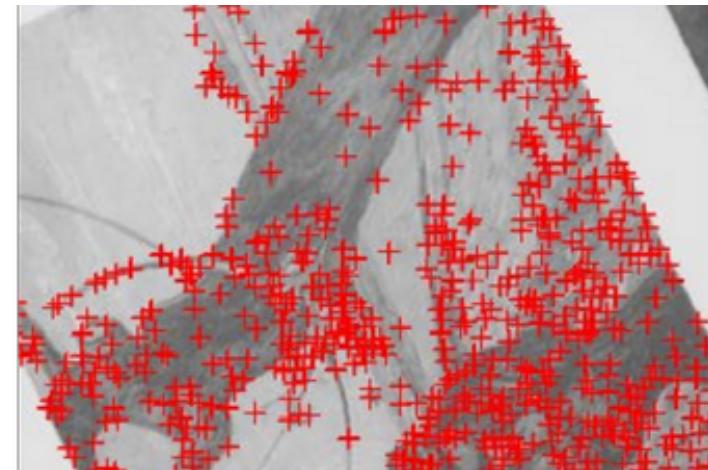
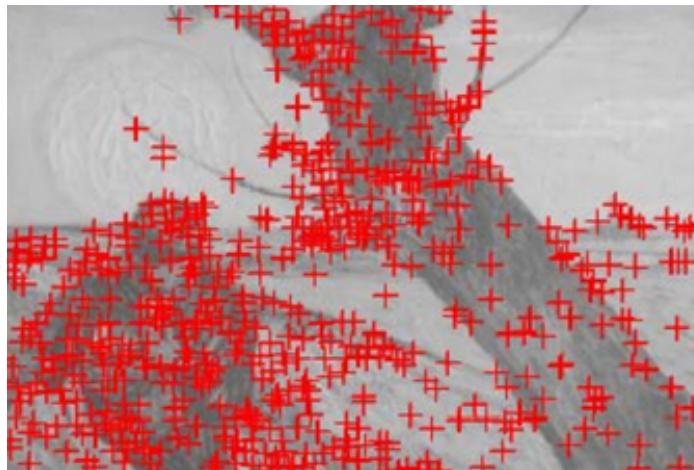
- Corresponding points (or features) between images enable the estimation of parameters describing geometric transforms between the images.

What if we don't know the correspondences?



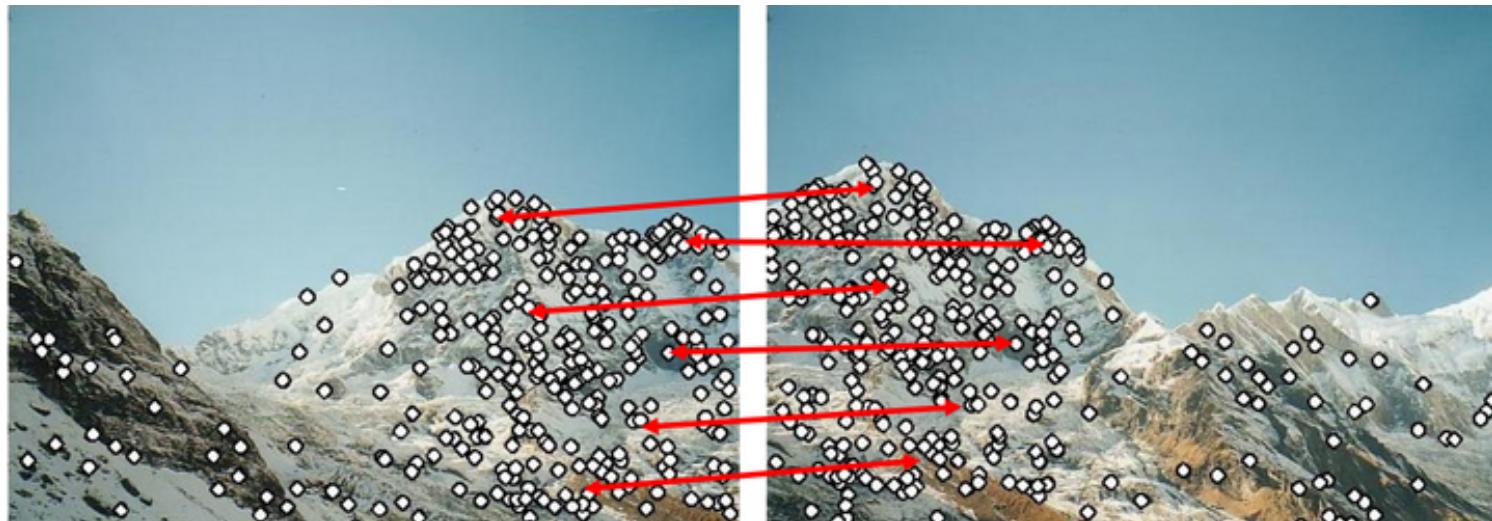
- Need to compare feature descriptors of local patches surrounding interest points.

Invariance



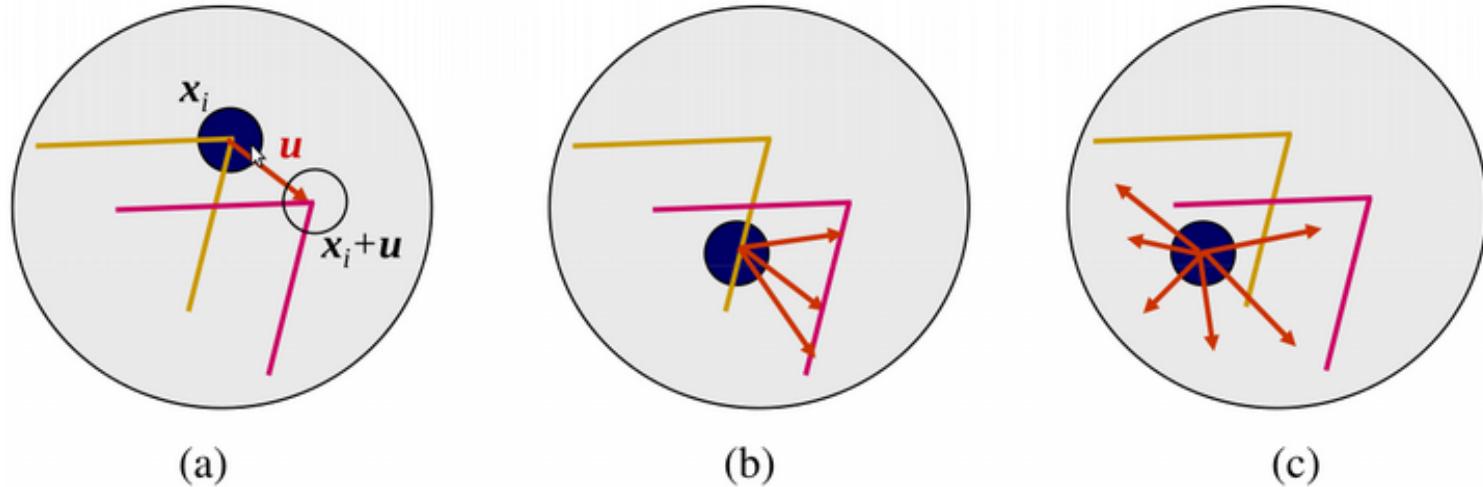
- Features should be detected despite **geometric** or **photometric** changes in the image.
- Given two transformed versions of the same image, features should be detected in corresponding locations.

Invariance - example



- Step 1: extract features
- Step 2: match features

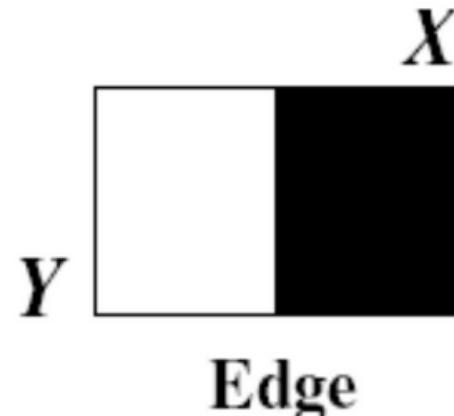
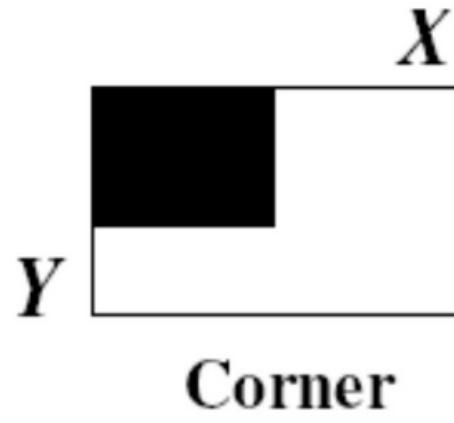
Aperture problems for different image patches



The red vector \mathbf{u} indicates the displacement between the patch centers and the $w(x_i)$ weighting function (patch window) is shown as a dark circle.

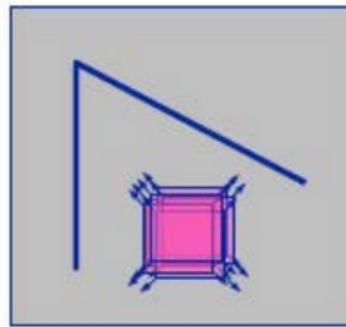
- (a) Stable ("corner-like") flow,
- (b) Classic aperture problem (barber-pole illusion);
- (c) Textureless region. The two images I_0 (yellow) and I_1 (red) are overlaid.

Corners vs Edges

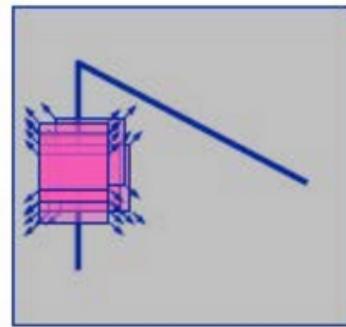


- **Corners** are locations where variations of intensity function $f(x, y)$ in both X and Y are high -
both partial derivatives f_x and f_y are large
- **Edges** are locations where variation of $f(x, y)$ in certain direction is high, while variation in orthogonal direction is low -
when edges is oriented along X f_x is large f_y small

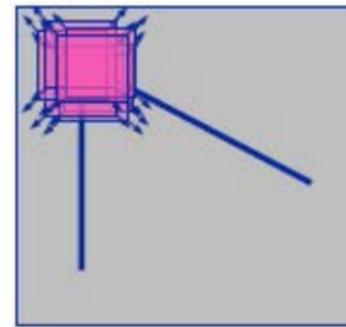
Corner Detector: Basic Idea



“flat” region:
no change in
all directions



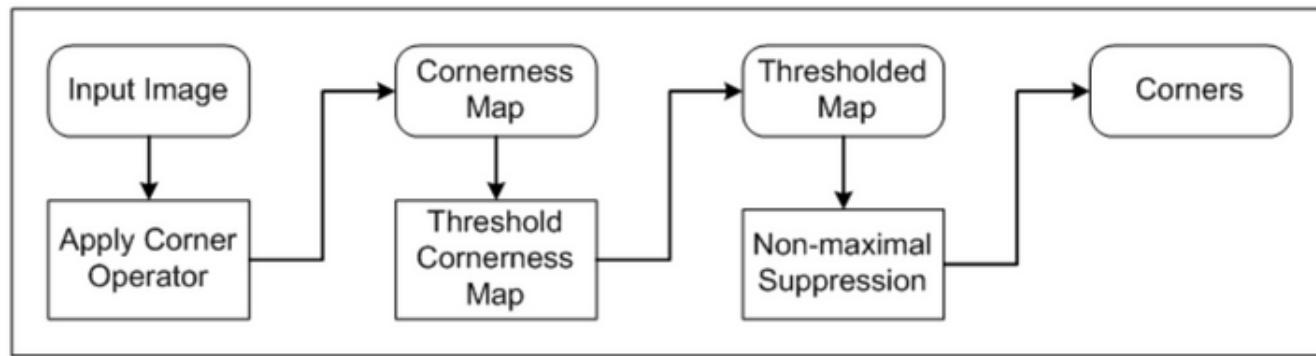
“edge”:
no change along
the edge direction



“corner”:
significant change
in all directions

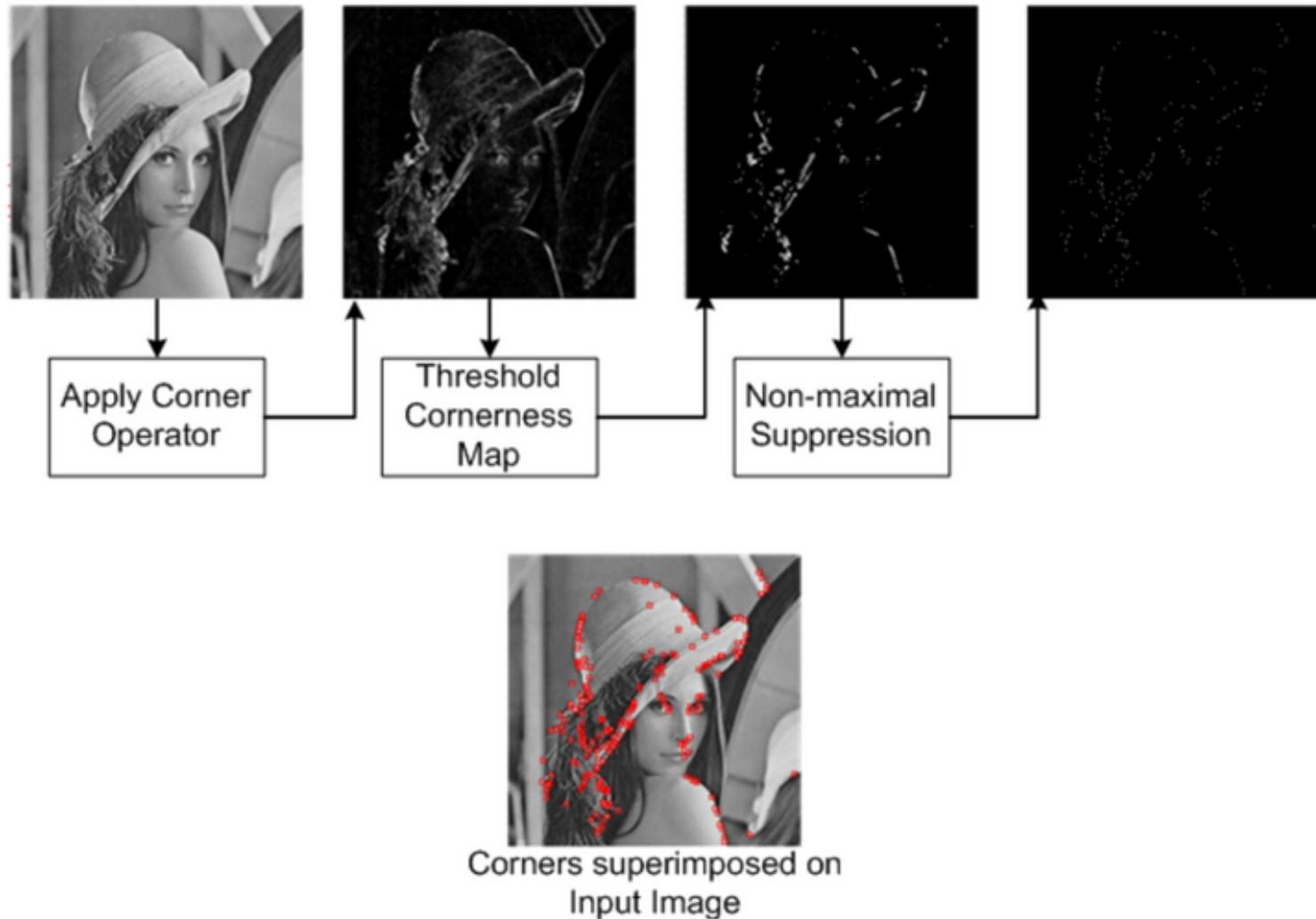
- We should easily recognize the point by looking through a small window,
- Shifting a window in *any direction* should give *a large change* in response

Mains Steps in Corner Detection

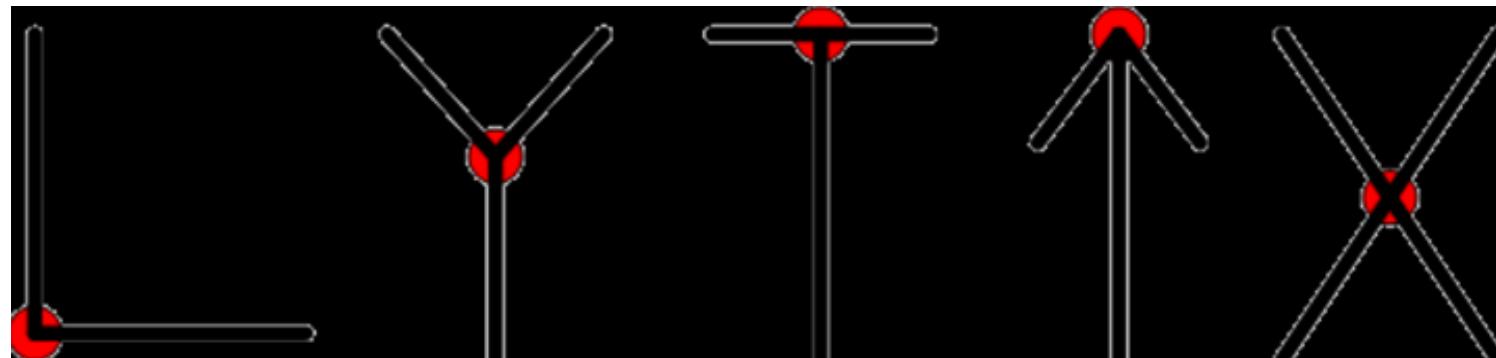


1. For each pixel in the input image, the corner operator is applied to obtain a cornerness measure for this pixel.
2. Threshold cornerness map to eliminate weak corners.
3. Apply non-maximal suppression to eliminate points whose cornerness measure is not larger than the cornerness values of all points within a certain distance.

Mains Steps in Corner Detection - example

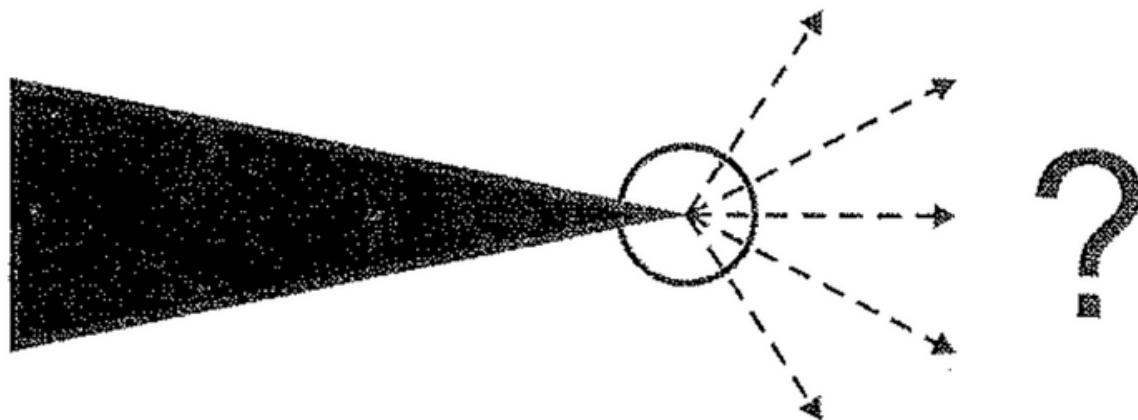


Corner Types



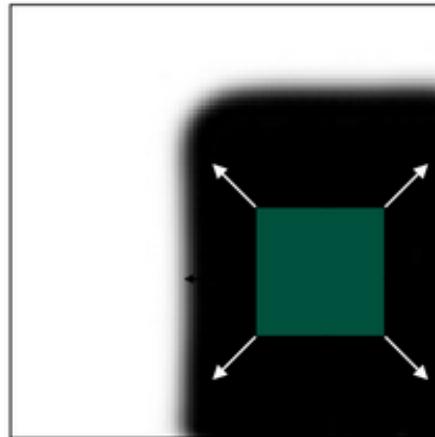
1. Example of L-junction, Y-junction, T-junction, Arrow-junction, and X-junction corner types

Corner Detection Using Edge Detection?

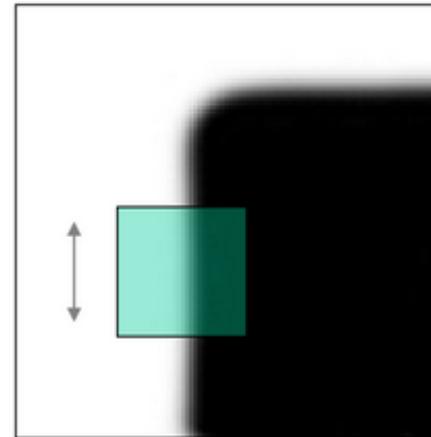


1. Edge detectors are not stable at corners,
2. Gradient is ambiguous at corner tip,
3. Discontinuity of gradient direction near. corner.

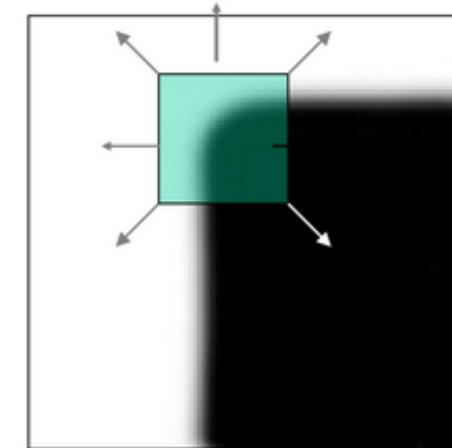
Corner Detection Using Intensity: Basic Idea



“flat” region:
no change in all
directions



“edge”: no change
along the edge
direction



“corner”: significant
change in all
directions

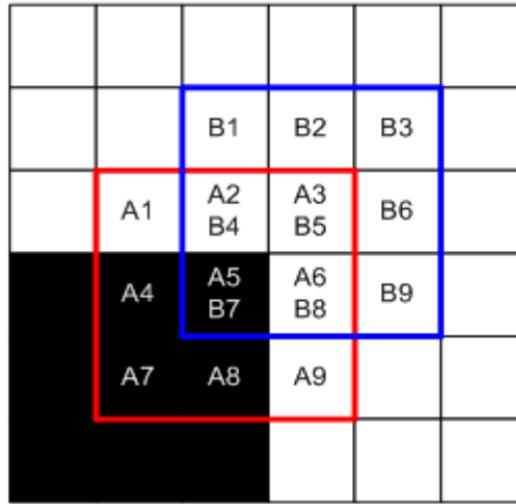
- Image gradient has two or more dominant directions near a corner.
- Shifting a window in any direction should give a large change in intensity.

Moravec Detector (1977)



- Measure intensity variation at (x,y) by shifting a small window (3×3 or 5×5) by one pixel in each of the eight principle directions (horizontally, vertically, and four diagonals). corner.

Moravec Detector (1977)



$$S_W(\Delta x, \Delta y) = \sum_{x_i \in W} \sum_{y_i \in W} (f(x_i, y_i) - f(x_i - \Delta x, y_i - \Delta y))^2$$

- 8 directions $\Delta x, \Delta y \in \{-1, 0, 1\}$
- $S_W(-1, -1), S_W(-1, 0), \dots, S_W(1, 1)$
- Calculate intensity variation by taking the sum of squares of intensity differences of corresponding pixels in these two windows.

Moravec Detector - Cornerness Map (normalized)

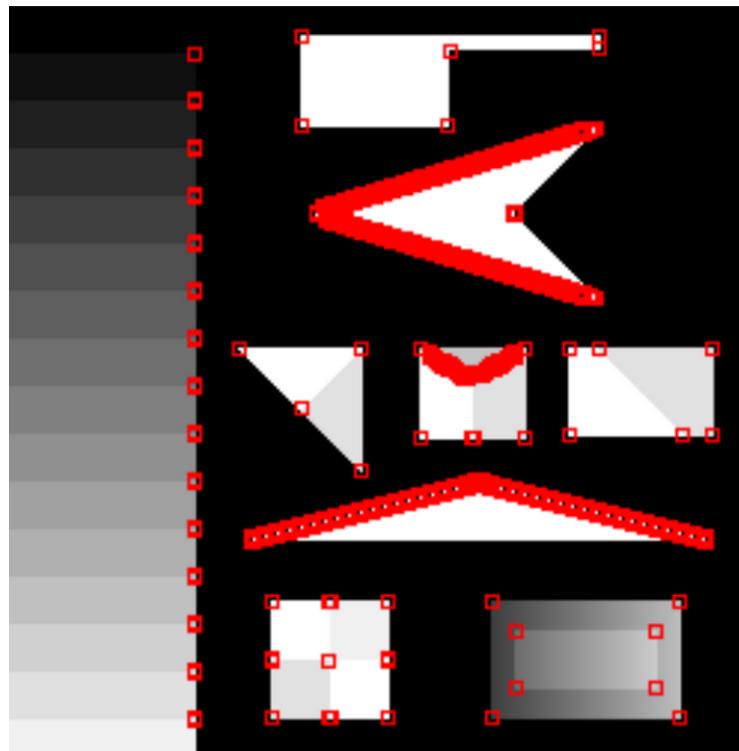
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
x	x	0	0	0	0	0	0	0	0	0	0	1	1	1	x	x	
x	x	0	0	0	0	0	0	1	1	0	0	1	2	1	x	x	
x	x	0	0	0	0	0	0	2	1	0	0	1	1	1	x	x	
x	x	0	0	0	0	0	0	0	0	0	0	0	0	0	x	x	
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	

- The “cornerness” of a pixel is the minimum intensity variation found over the eight shift directions:

$$\text{Cornerness}(x, y) = \min\{S_W(-1, -1), S_W(-1, 0), \dots, S_W(1, 1)\}$$

- Non-maximal suppression will yield the final corners.

Moravec Detector - effecting



- Does a reasonable job in finding the majority of true corners.
- Edge points not in one of the eight principle directions will be assigned a relatively large cornerness value.

Moravec Detector - effecting



Original Image



Image Rotated 30°

- The response is anisotropic as the intensity variation is only calculated at a discrete set of shifts (i.e., not rotationally invariant)

Harris Detector

Change of intensity for the shift $[u, v]$:

$$E(u, v) = \sum_{x, y} w(x, y) [I(x + u, y + v) - I(x, y)]^2$$

↑ ↑ ↑
Window function Shifted intensity Intensity

Window function $w(x, y) =$



- Improves the Moravec operator by avoiding the use of discrete directions and discrete shifts.
- Uses a Gaussian window instead of a square window.

Harris Detector

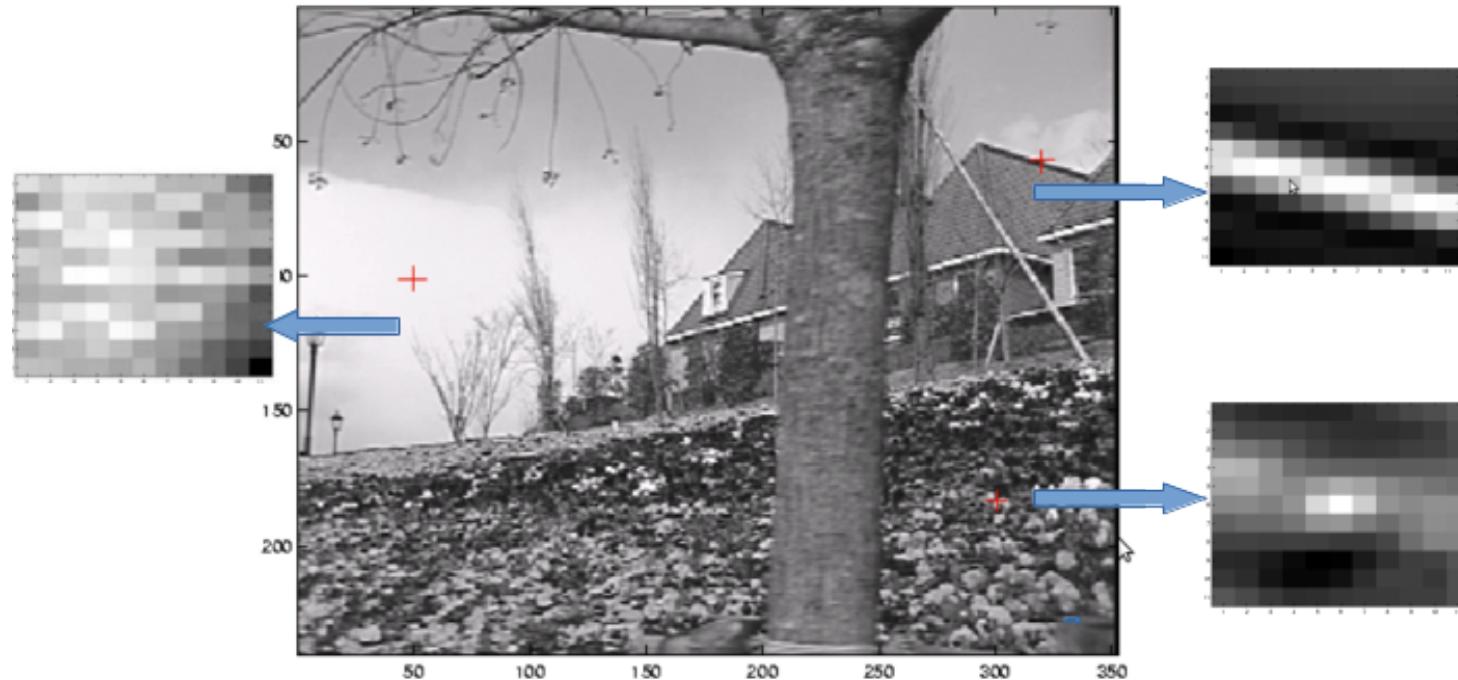
For small shifts $[u, v]$ we have a *bilinear approximation*:

$$E(u, v) \approx [u, v] M \begin{bmatrix} u \\ v \end{bmatrix}$$

where M is a 2×2 matrix computed from image derivatives:

$$M = \sum_{x,y} w(x, y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix}$$

Auto-correlation surfaces



- The correlation surface corresponding to the roof edge has a strong ambiguity along one direction,
- Correlation surface corresponding to the cloud region has no stable minimum.

Description of auto-correlation surface

Using expansion of the image $I_0(x_i + \Delta u) \approx I_0(x_i) + \nabla I_0(x_i) \cdot \Delta u$ we can approximate the auto-correlation surface as

$$E_{AC}(\Delta u) = \sum_i w(x_i)[I_0(x_i + \Delta u) - I_0(x_i)]^2 \approx \sum_i w(x_i)[I_0(x_i) + \nabla I_0(x_i) \cdot \Delta u - I_0(x_i)]^2 = \sum_i w(x_i)[\nabla I_0(x_i) \cdot \Delta u]^2 = \Delta u^T A \Delta u,$$

where

$\nabla I_0(x_i) = (\frac{\partial I_0}{\partial x}, \frac{\partial I_0}{\partial y})(x_i)$ is the image gradient at x_i .

- The auto-correlation matrix A can be written as

$$A = w \star \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix}$$

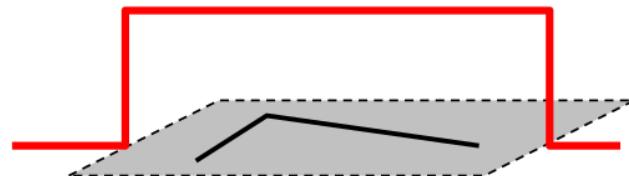
where we have replaced the weighted summations with discrete convolutions with the weighting kernel w

Harris Detector

General case – use window function:

$$S_W(\Delta x, \Delta y) = \sum_{x_i, y_i} w(x_i, y_i) [f(x_i, y_i) - f(x_i - \Delta x, y_i - \Delta y)]^2$$

default window function $w(x, y)$: 1 in window, 0 outside:



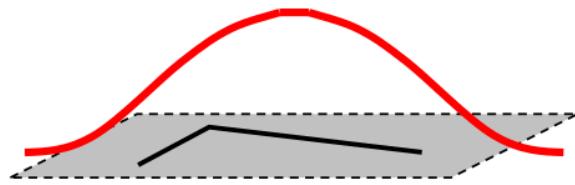
where M is a 2×2 matrix computed from image derivatives:

$$A_W = \begin{bmatrix} \sum_{x,y} w(x, y) f_x^2 & \sum_{x,y} w(x, y) f_x f_y \\ \sum_{x,y} w(x, y) f_y f_x & \sum_{x,y} w(x, y) f_y^2 \end{bmatrix} = \sum_{x,y} w(x, y) \begin{bmatrix} f_x^2 & f_x f_y \\ f_y f_x & f_y^2 \end{bmatrix}$$

Harris Detector

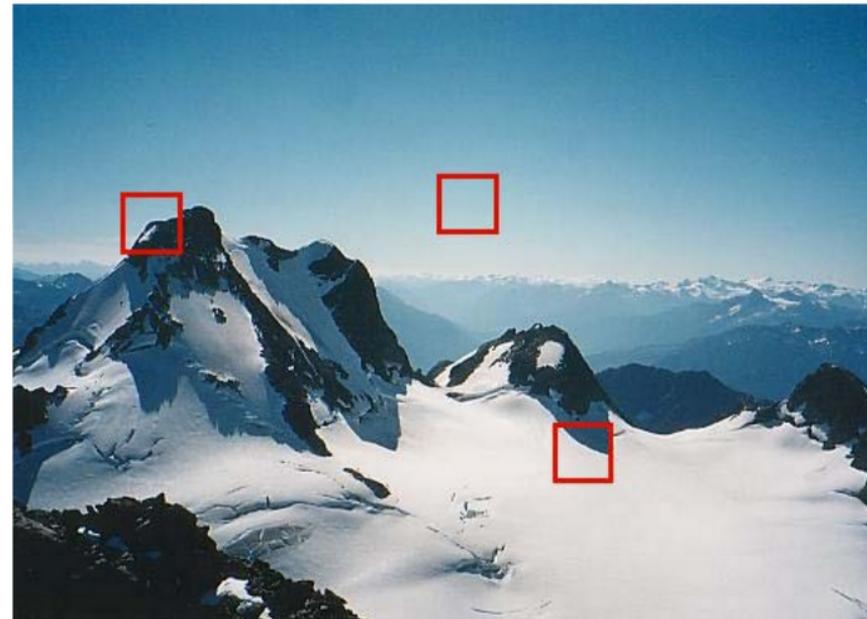
Harris uses a Gaussian window: $w(x, y) = G(x, y, \sigma_I)$ where σ_I is called the “integration” scale

window function $w(x, y)$: Gaussian



$$A_W = \begin{bmatrix} \sum_{x,y} w(x, y) f_x^2 & \sum_{x,y} w(x, y) f_x f_y \\ \sum_{x,y} w(x, y) f_y f_x & \sum_{x,y} w(x, y) f_y^2 \end{bmatrix} = \sum_{x,y} w(x, y) \begin{bmatrix} f_x^2 & f_x f_y \\ f_y f_x & f_y^2 \end{bmatrix}$$

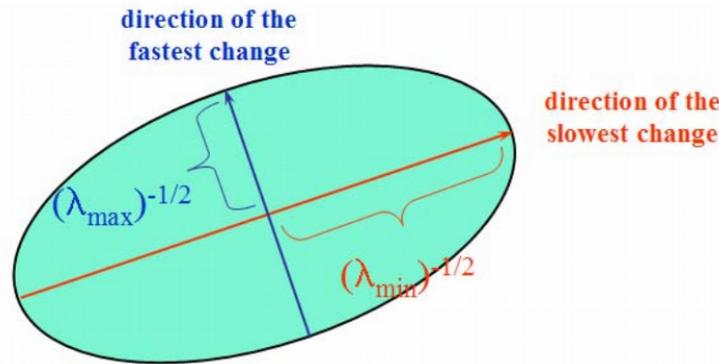
Harris Detector



$$A_W = \sum_{x,y} \begin{bmatrix} f_x^2 & f_x f_y \\ f_y f_x & f_y^2 \end{bmatrix}$$

- Describes the gradient distribution (i.e., local structure) inside window!
- Does not depend on $\Delta x, \Delta y$

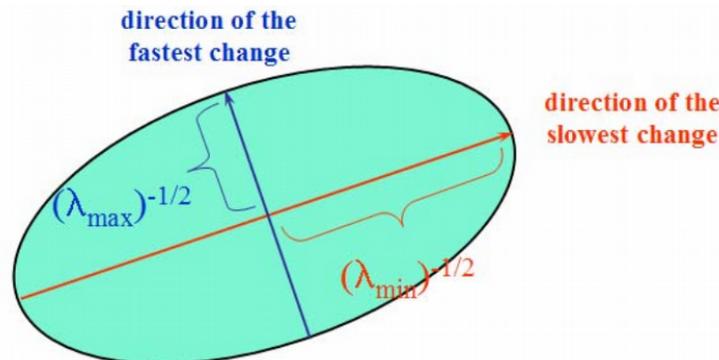
Harris Detector



Ellipse equation: $[\Delta x \Delta y] A_W \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = const$

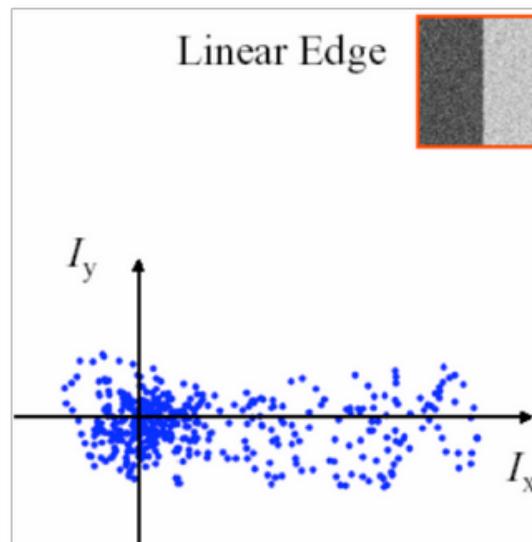
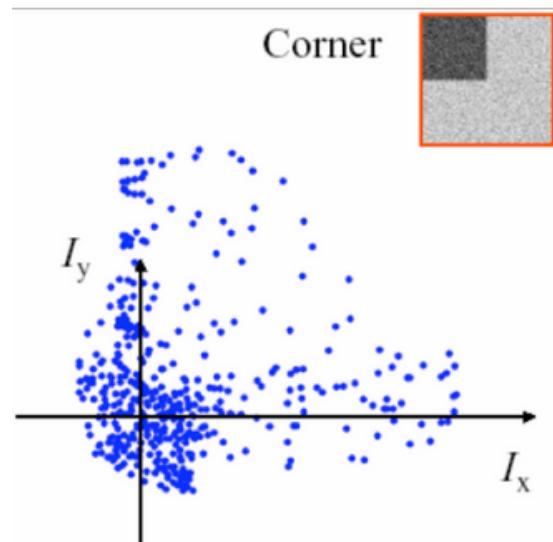
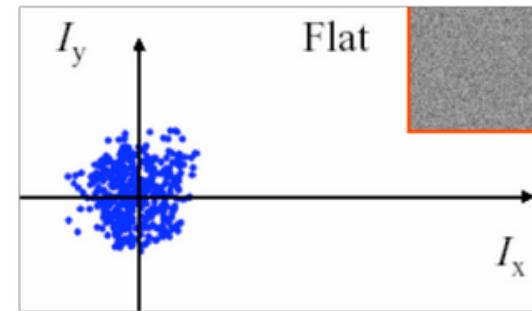
- Since M is symmetric, we have: $A_W = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R$
- We can visualize A_W as an ellipse with axis lengths determined by the eigenvalues and orientation determined by R

Eigenvalues and eigenvector of correlation matrix A



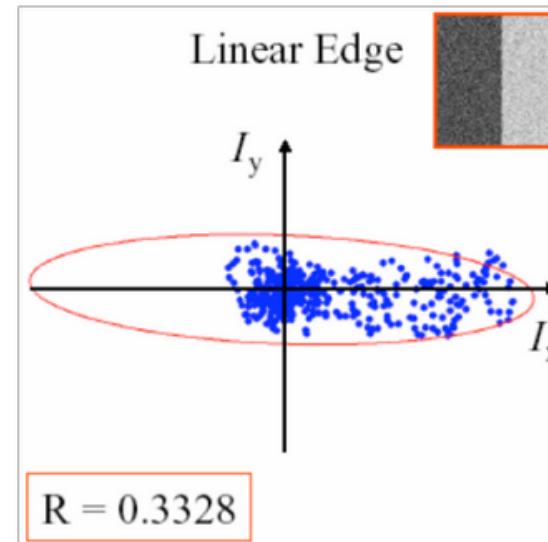
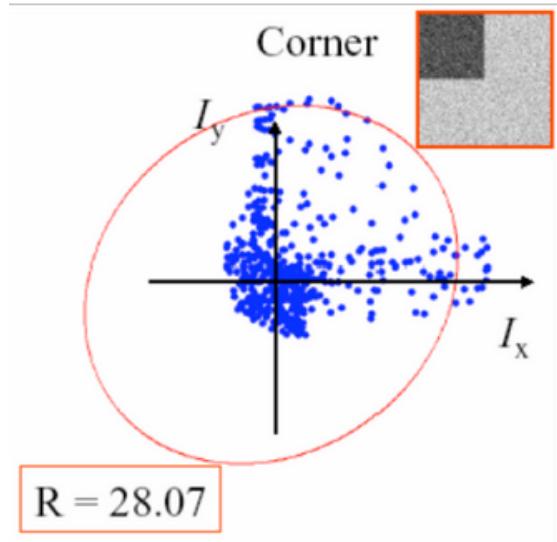
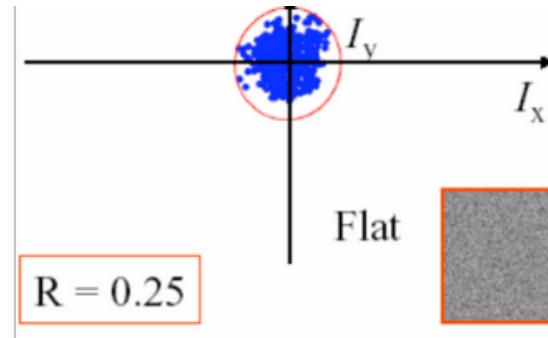
- Matrix A has two eigenvalues (λ_0, λ_1) and two eigenvector directions,
- Eigenvectors encode edge direction
- Eigenvalues encode edge strength
- Larger uncertainty depends on the smaller eigenvalue - task is to find maxima in the smaller eigenvalue (to locate good features),

Distribution of f_x and f_y



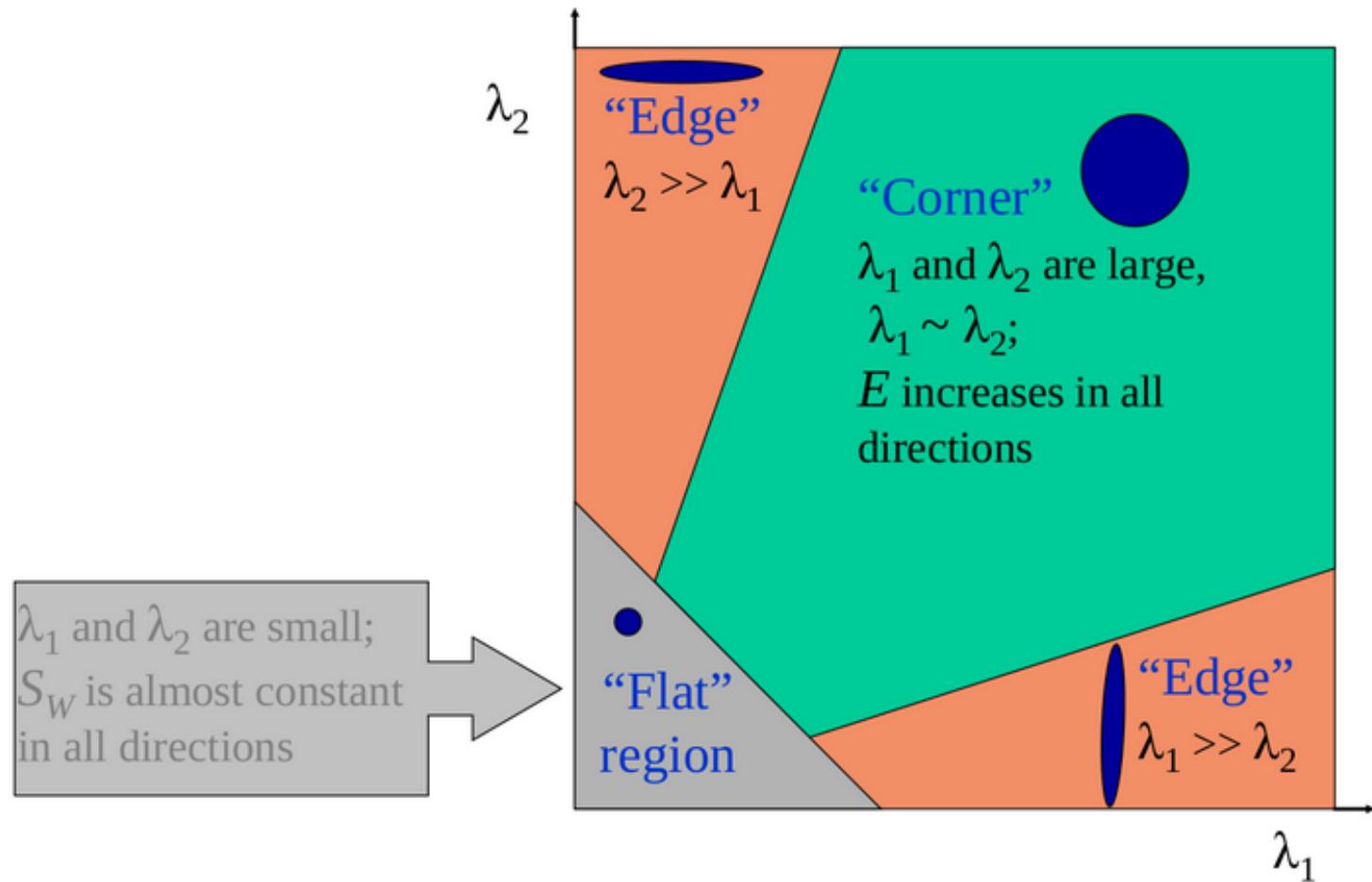
- The distribution of the x and y derivatives is very different for all three types of patches

Distribution of f_x and f_y



- The distribution of the x and y derivatives can be characterized by the shape and size of principal component ellipse.

Classification of image points using eigenvalues of A_W



Harris Detector - measure of corner response:

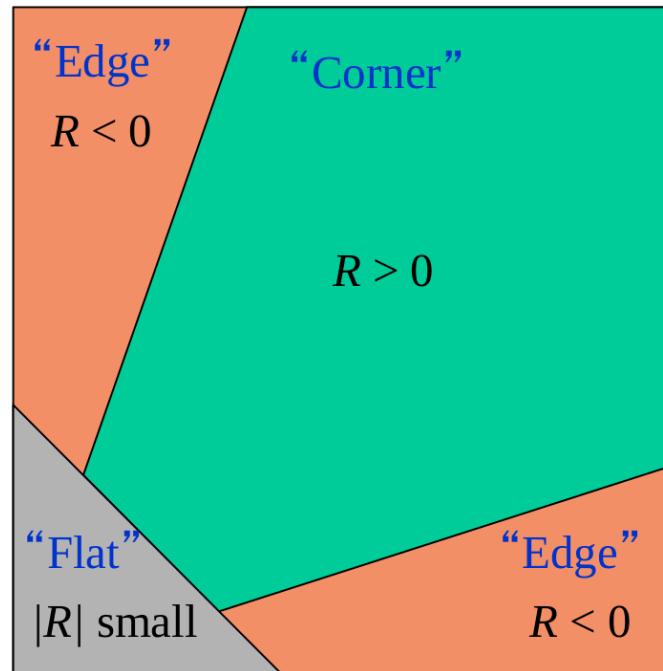
$$R = \det M - k(\text{trace}M)^2$$

$$\det M = \lambda_1 \lambda_2$$

$$\text{trace}M = \lambda_1 + \lambda_2$$

- (k – empirical constant, $k = 0.04 – 0.06$)
- Shi-Tomasi variation: use $\min\{\lambda_1 \lambda_2\}$ instead of R

Harris Detector: Mathematics



- R depends only on eigenvalues of M
- R is large for a corner
- R is negative with large magnitude for an edge
- $|R|$ is small for a flat region

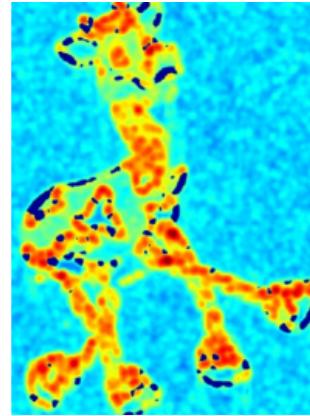
Harris corner detector algorithm

- Compute image gradients I_x I_y for all pixels
- For each pixel
 - Compute $M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix}$
 - Compute $R = \det M - k(\text{trace}M)^2$
- Find points with large corner response function R ($R >$ threshold)
- Take the points of **locally maximum R** as the detected feature points
(ie, pixels where R is bigger than for all the 4 or 8 neighbors).

Harris Detector: Example



(a)



(b)



(c)



(d)



(e)

- (b) Compute corner response R of original image (a)
- (c) Find points with large corner response: $R > \text{threshold}$
- (d) Take only the points of local maxima of R
- (e) Map corners on the original image

Harris Detector – Scale Parameters

- The Harris detector requires two scale parameters:
 - (i) **Differentiation scale** σ_D for smoothing prior to the computation of image derivatives,
 - (ii) **Integration scale** σ_I for defining the size of the Gaussian window (i.e., integrating derivative responses).

$$A_W(x, y) \rightarrow A_W(x, y, \sigma_I, \sigma_D)$$

- Typically,

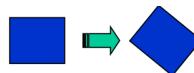
$$\sigma_I = \gamma \sigma_D$$

Invariance to Geometric/Photometric Changes

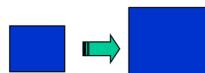
Is the Harris detector invariant to geometric and photometric changes?

- Geometric

- Rotation



- Scale

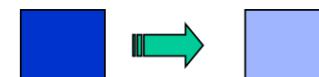


- Affine

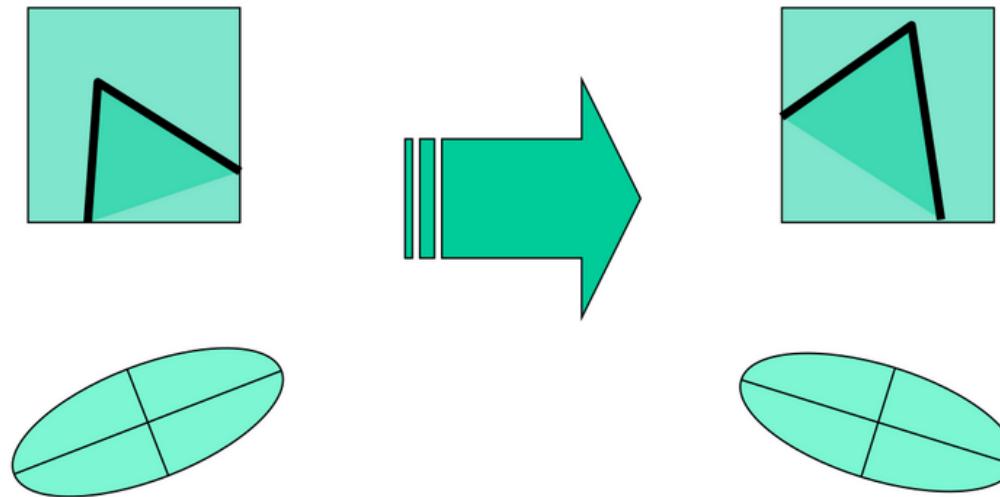


- Photometric

- Affine intensity change: $I(x, y) \rightarrow aI(x, y) + b$



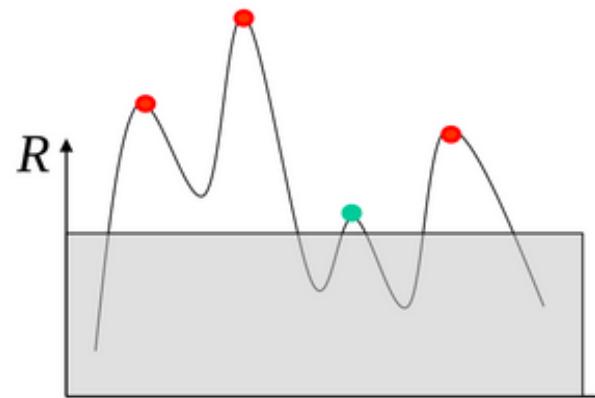
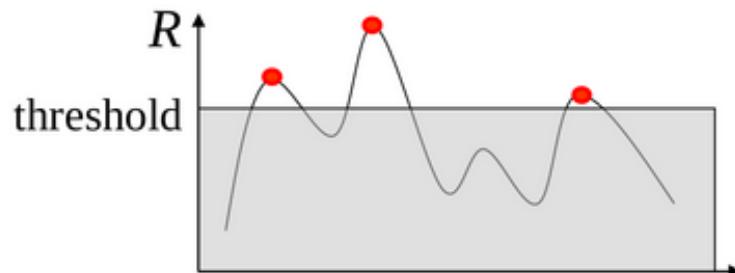
Harris Detector: Rotation Invariance



- Ellipse rotates but its shape (i.e. eigenvalues) remains the same
- Corner response R is invariant to image rotation

Harris Detector: Photometric Changes

Affine intensity change

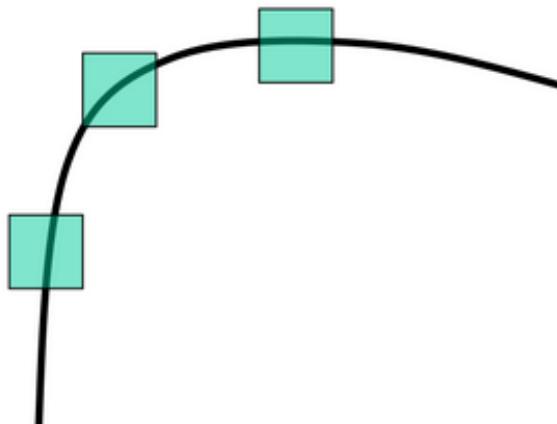


- Only derivatives are used => invariance to intensity shift
 $I(x, y) \rightarrow I(x, y) + b,$
- Intensity scale: $I(x, y) \rightarrow aI(x, y),$
- **Partially invariant to affine intensity change.**

Harris Detector: Scale Invariance

Affine intensity change

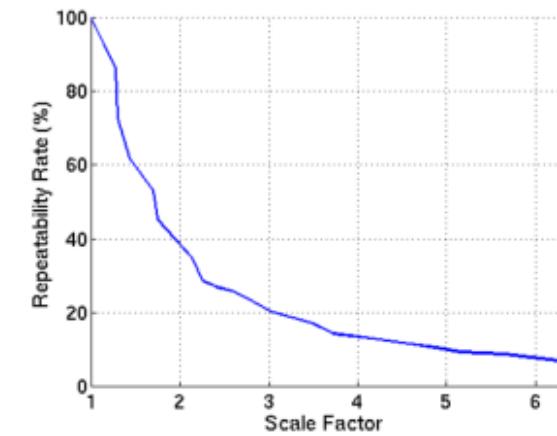
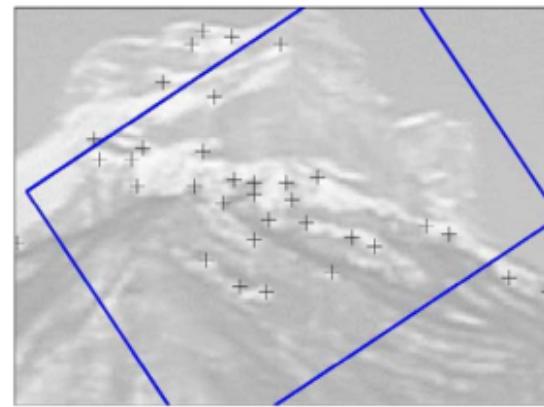
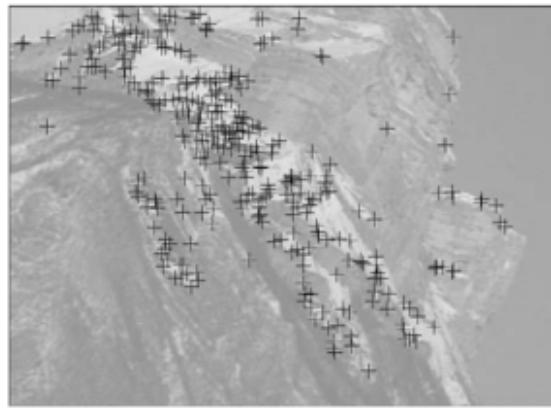
- Scaling



Corner

- All points will be classified as edges,
- **Not invariant to scaling (and affine transforms)**

Harris Detector: Disadvantages



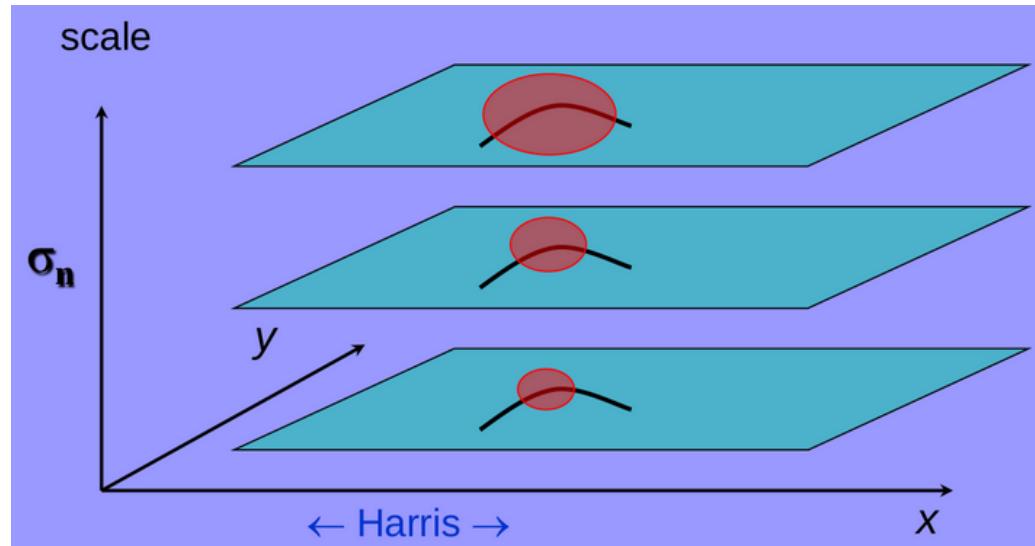
Sensitive to:

- Scale change,
- Significant viewpoint change,
- Significant contrast change.

How to handle scale changes?

- A_W must be adapted to scale changes.
- If the scale change is known, we can adapt the Harris detector to the scale change (i.e., set properly σ_I , σ_D).
- What if the scale change is unknown?

Multi-scale Harris Detector

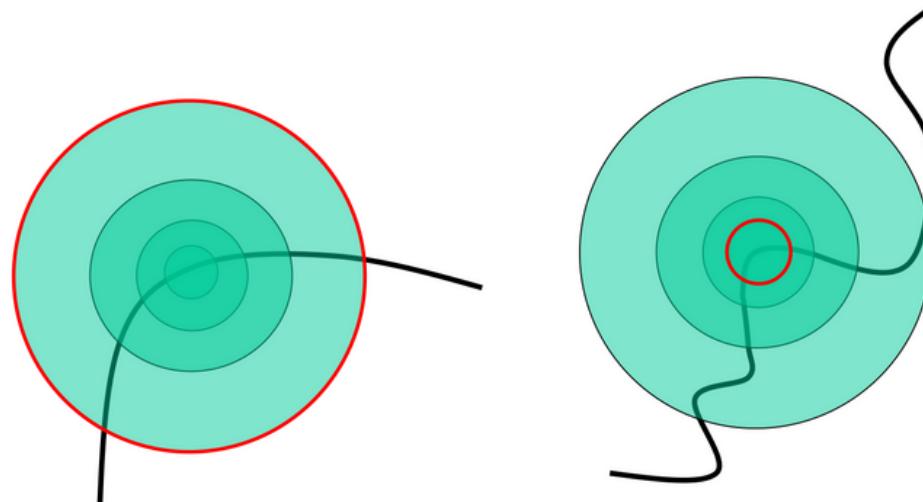


- Detects interest points at varying scales.

$$R(A_W) = \det(A_W(x, y, \sigma_I, \sigma_D)) - \alpha \cdot \text{trace}^2(A_W(x, y, \sigma_I, \sigma_D))$$

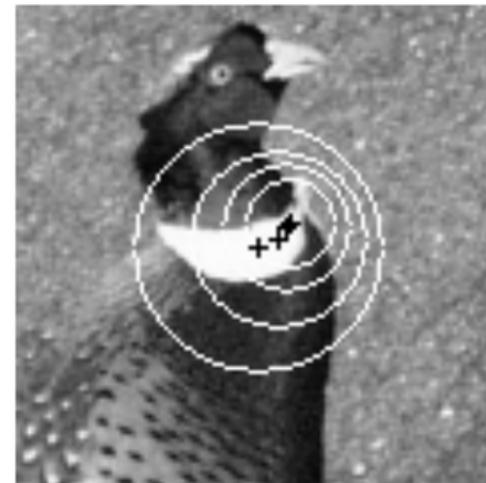
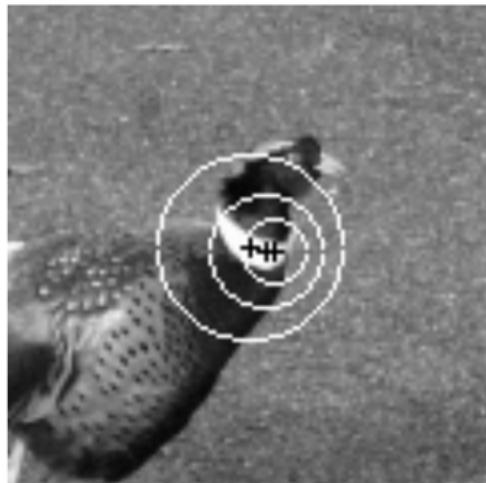
How to handle scale changes?

How do we choose corresponding circles independently in each image?



- The size of the circle corresponds to the scale at which the point was detected

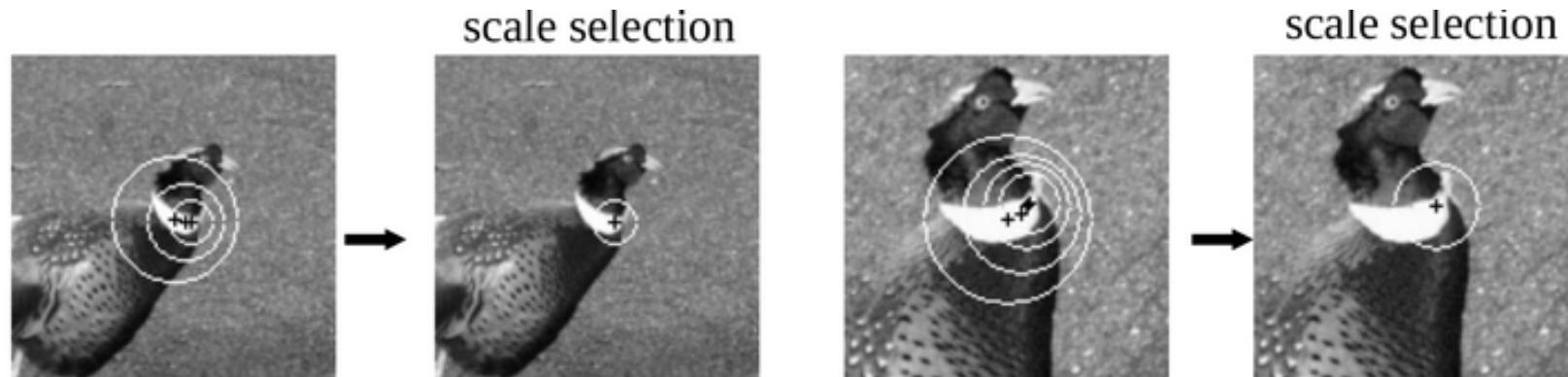
Problems in scale changes?



Problems:

- There will be many points representing the same structure, complicating matching!
- Note that point locations shift as scale increases.

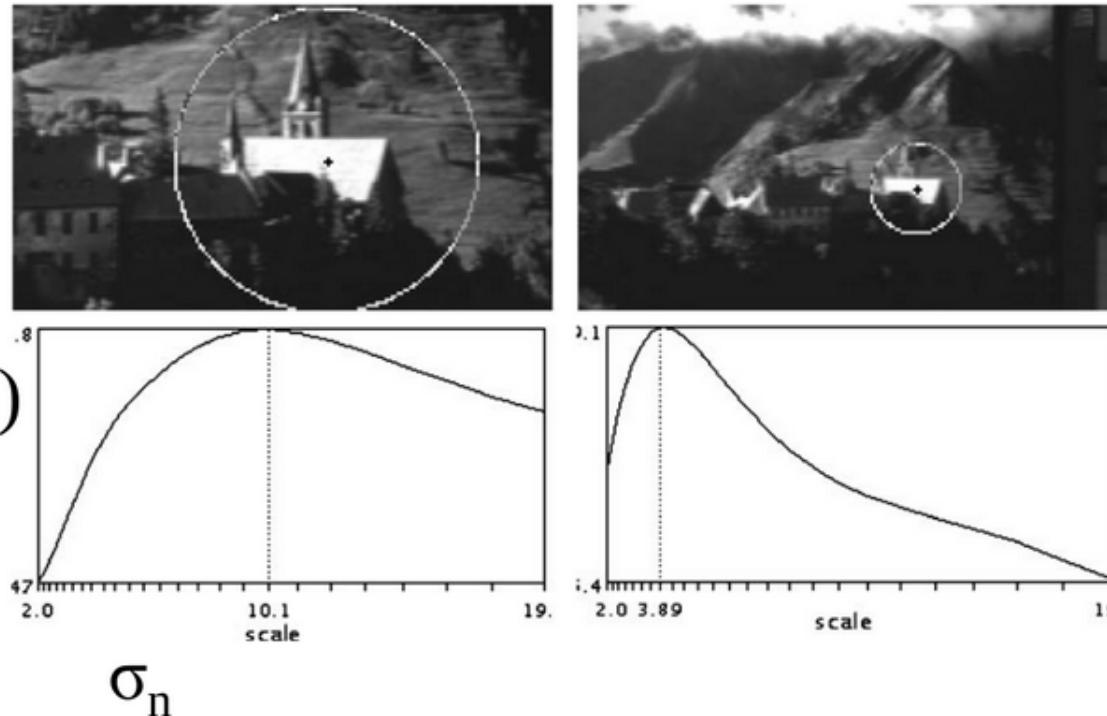
How to handle scale changes?



Problems:

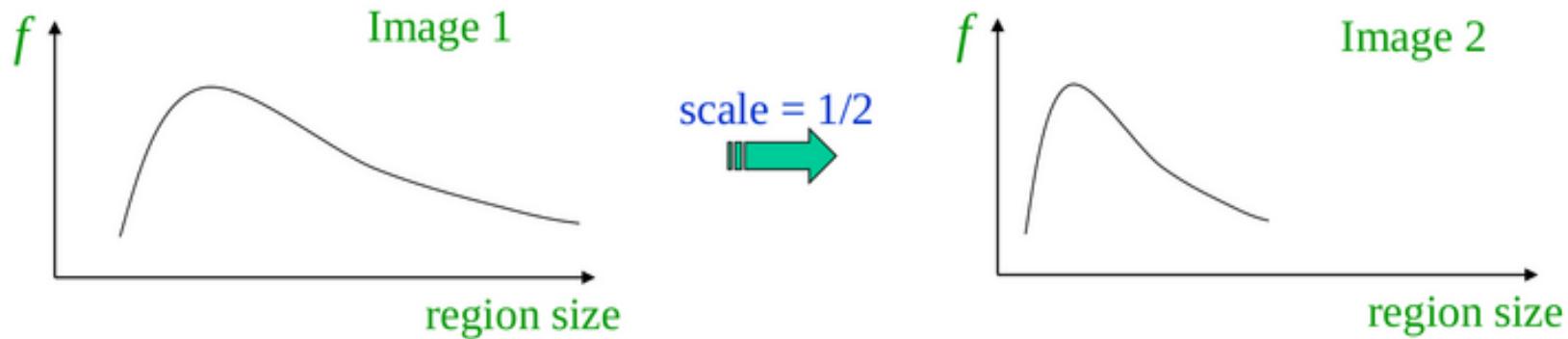
- Alternatively, use scale selection to find the characteristic scale of each feature.
- Characteristic scale depends on the feature's spatial extent (i.e., local neighborhood of pixels).
- Only a subset of the points computed in scale space are selected.

Automatic Scale Selection



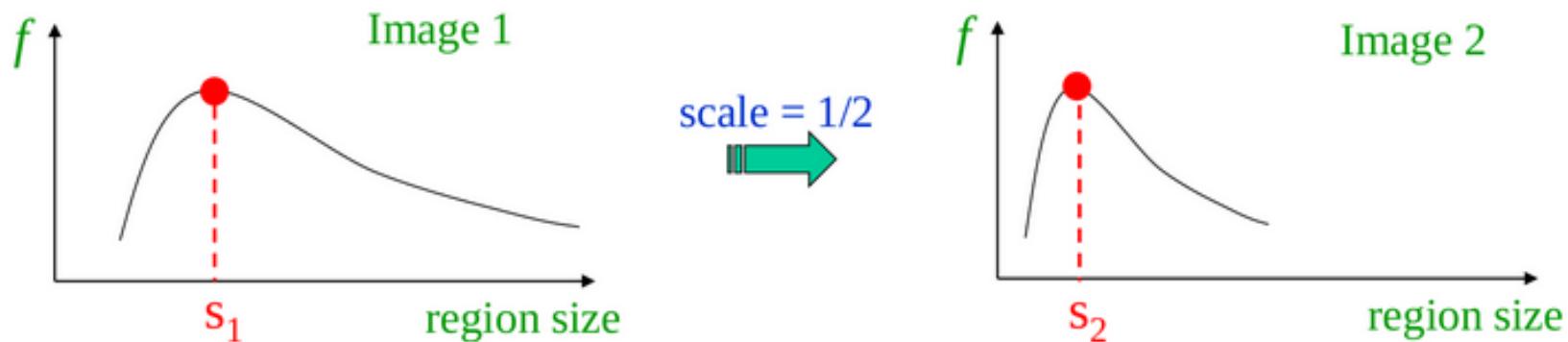
- Design a function $F(x, \sigma_n)$ which provides some local measure,
- Select points at which $F(x, \sigma_n)$ is maximal over σ_n ,
- **max of $F(x, \sigma_n)$ corresponds to characteristic scale**

Automatic Scale Selection



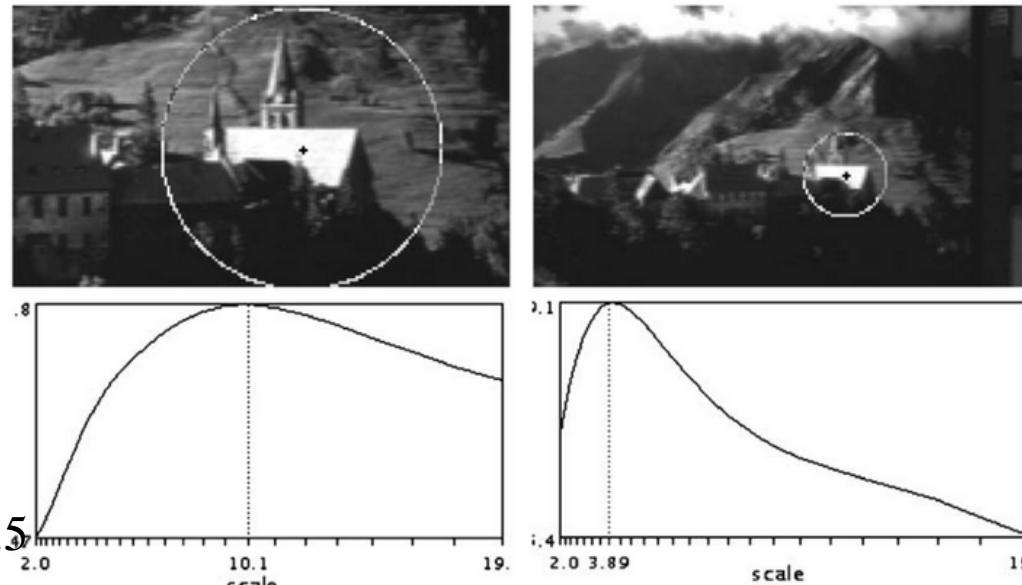
- Design a function on the region, which is “scale invariant” (the same for corresponding regions, even if they are at different scales)
- For a point in one image, we can consider it as a function of region size (patch width)

Automatic Scale Selection - common approach



- Take a local maximum of this function
- Observation: region size, for which the maximum is achieved, should be invariant to image scale.
- **Important: this scale invariant region size is found in each image independently!**

Automatic Scale Selection



- Characteristic scale is relatively **independent** of the image scale.
- The **ratio** of the scale values corresponding to the max values, is equal to the scale factor between the images.
- Scale selection allows for finding spatial extend that is **covariant** with the image transformation.

Automatic Scale Selection



What local measures should we use?

- Should be rotation invariant,
- Should have one stable sharp peak

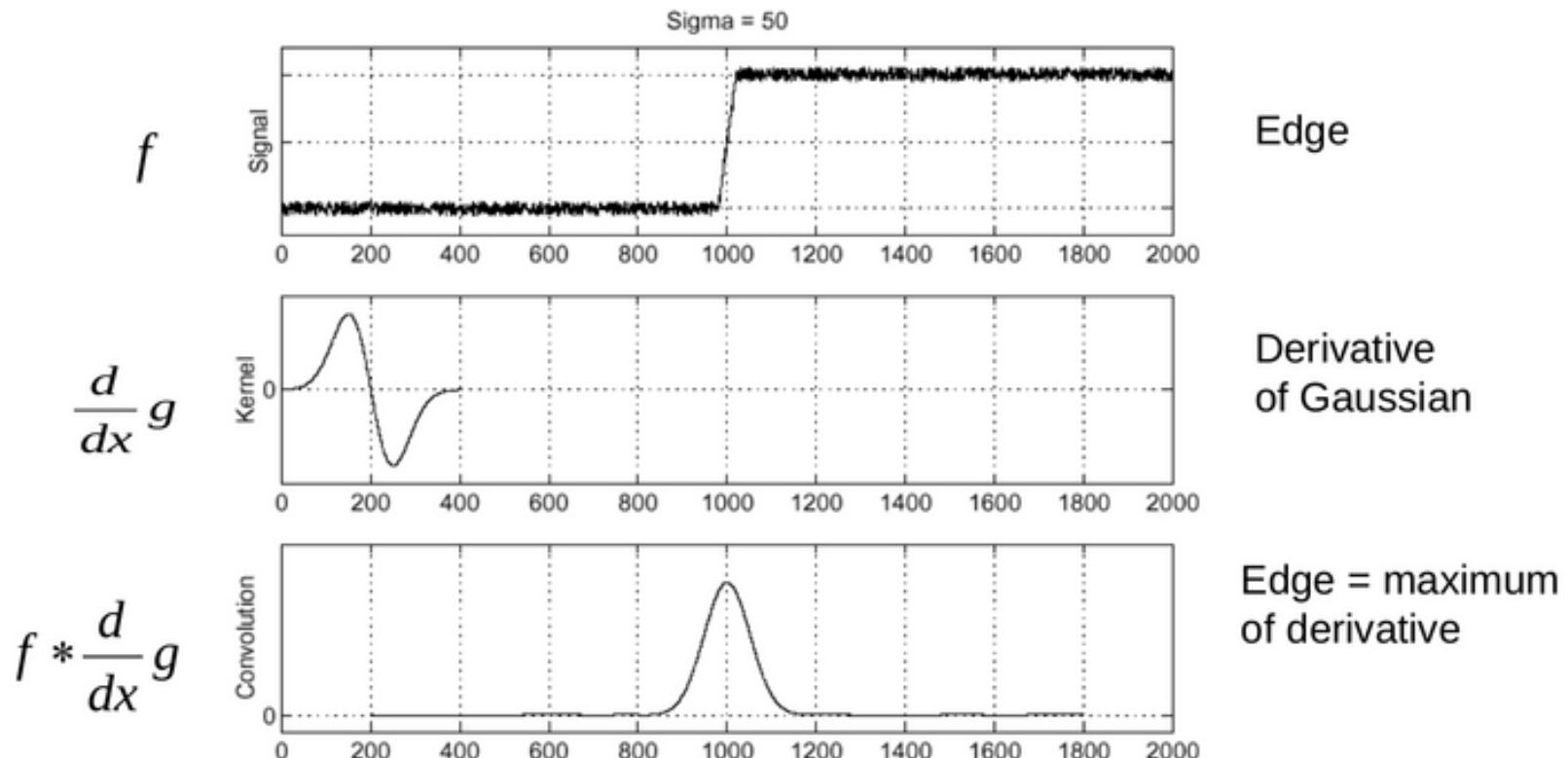
How should we choose $F(x, \sigma_n)$?

Typically, $F(x, \sigma_n)$ is defined using derivatives, e.g.:

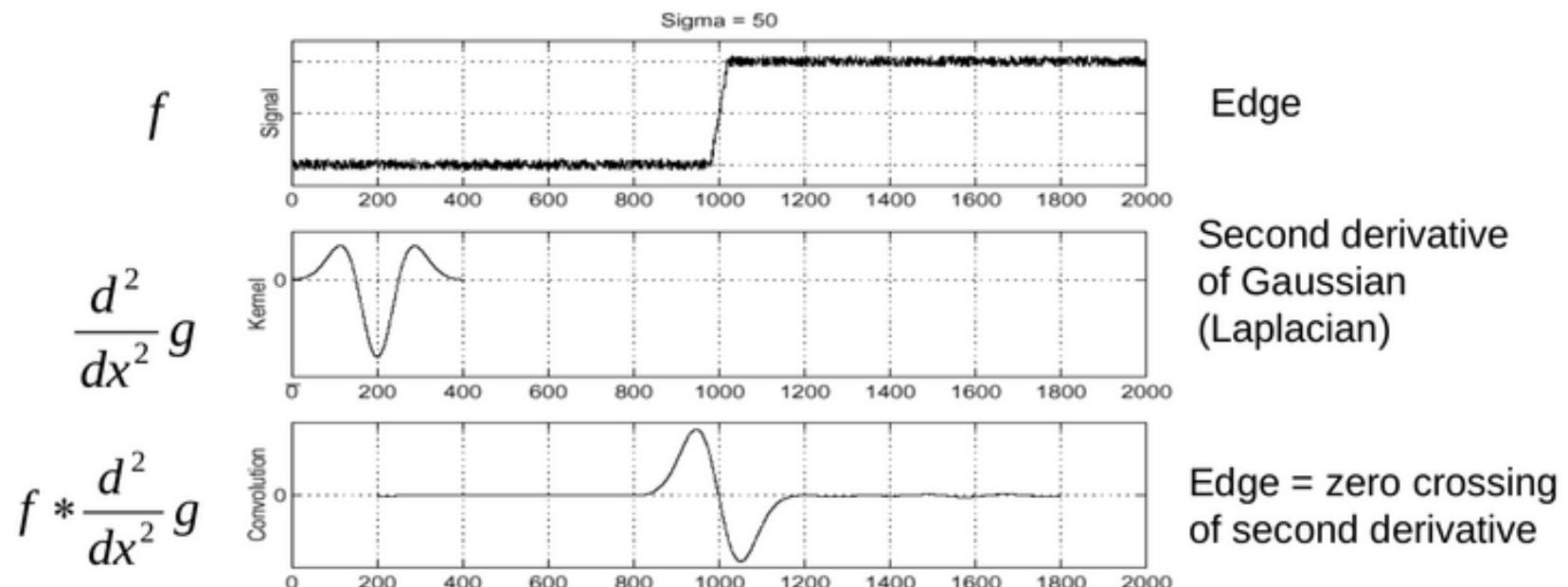
- Square gradient: $\sigma^2(L_x^2(x, \sigma) + L_y^2(x, \sigma))$
- LoG: $|\sigma^2(L_{xx}(x, \sigma) + L_{yy}(x, \sigma))|$
- DoG: $|I(x) \star G(\sigma_{n-1}) - I(x) \star G(\sigma_n)|$
- Harris function: $\det(A_W) - \alpha \cdot \text{trace}^2(A_W)$

LoG yielded best results in a evaluation study; DoG was second best.

Recall: Edge detection Using 1st derivative

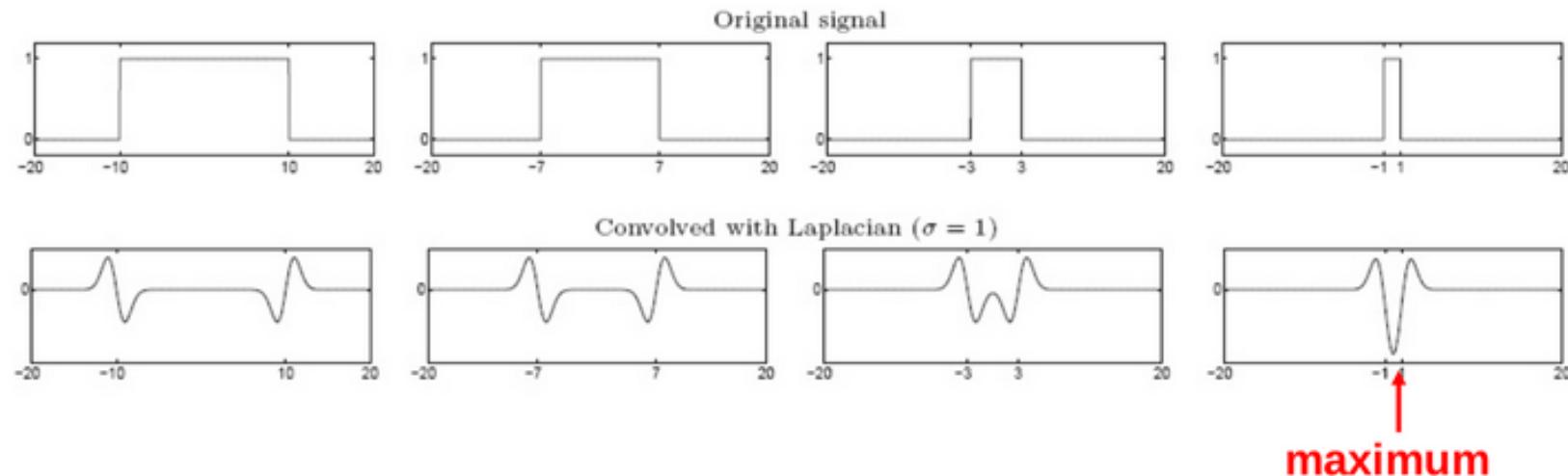


Recall: Edge detection Using 2nd derivative



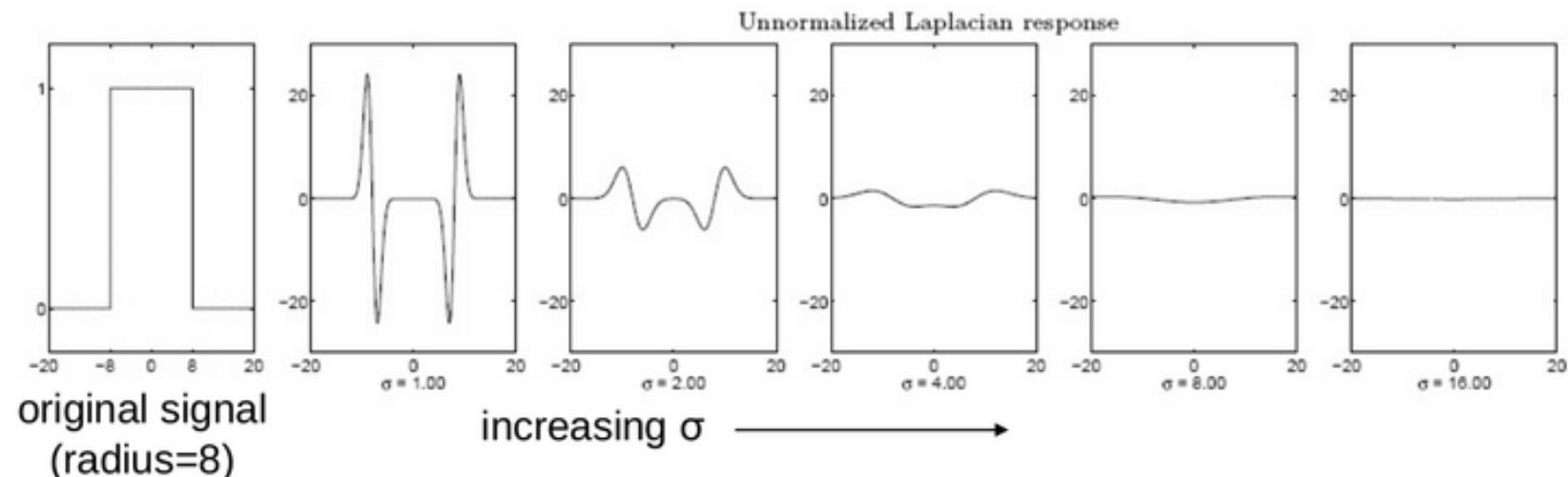
From edges to blobs (i.e., small regions)

Blob = superposition of two edges



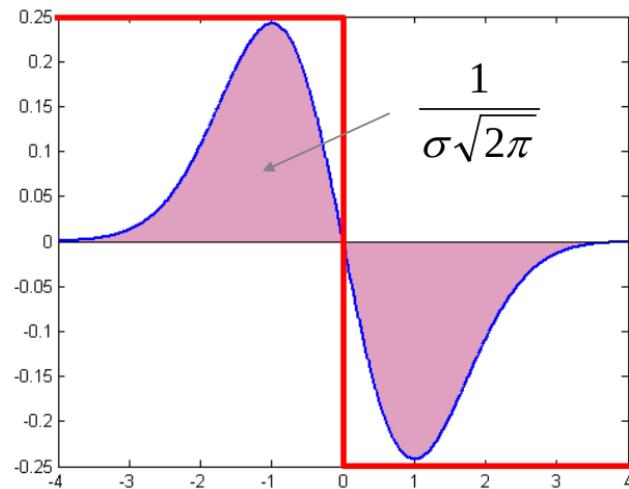
Spatial selection: the magnitude of the Laplacian response will achieve a maximum (absolute value) at the center of the blob, provided the scale of the Laplacian is “matched” to the scale of the blob (e.g., spatial extent)

How could we find the spatial extent of a blob using LoG?



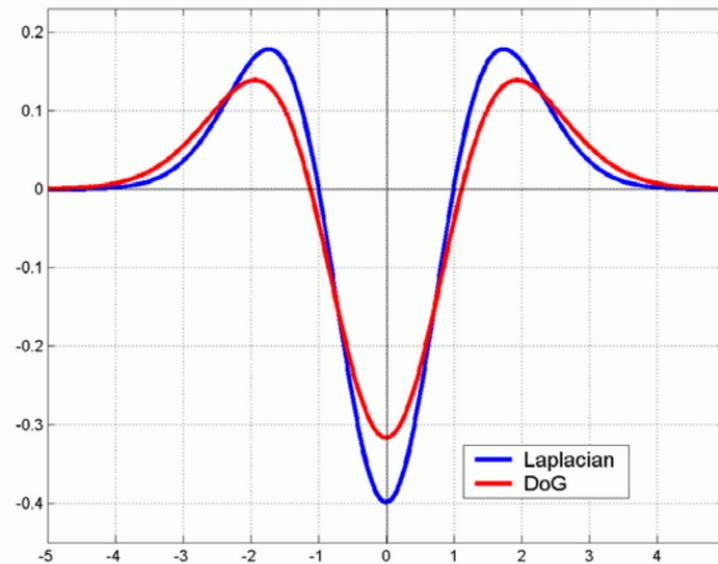
Find the characteristic scale of the blob by convolving it with Laplacian filters at several scales and looking for the maximum response.

Scale normalization



- The response of a derivative of Gaussian filter to a perfect step edge decreases as σ increases
- To keep response the same (scale-invariant), must multiply Gaussian derivative by σ
- Laplacian is the second Gaussian derivative, so it must be multiplied by σ^2

Efficient implementation using DoG



- LoG can be approximated by DoG:

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k - 1)\sigma^2 \nabla^2 G$$

- Note that DoG already incorporates the σ^2 scale normalization.

Efficient implementation using DoG

- Gaussian-blurred image

$$L(x, y, \sigma) = G(x, y, \sigma) \star I(x, y)$$

- The result of convolving an image with a difference-of Gaussian is given by:

$$D(x, y, k\sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) \star I(x, y) = L(x, y, k\sigma) - L(x, y, \sigma)$$

Computer Vision

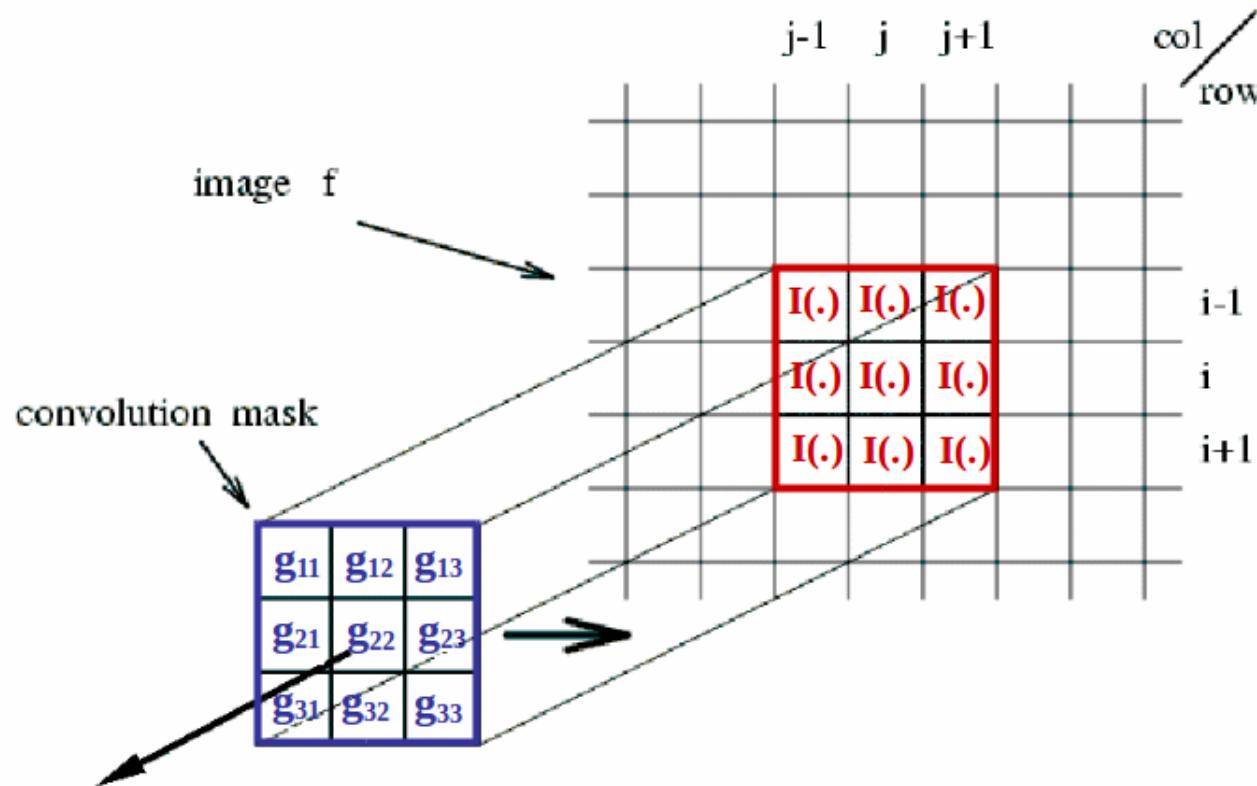
Convolution neuron networks - lecture 6

Adam Szmigiełski

aszmigie@pjwstk.edu.pl

materials: *ftp(public) : //aszmigie/WMAEnglish*

linear filter - convolution



$$f(i,j) = g_{11} I(i-1,j-1) + g_{12} I(i-1,j) + g_{13} I(i-1,j+1) + \\ g_{21} I(i,j-1) + g_{22} I(i,j) + g_{23} I(i,j+1) + \\ g_{31} I(i+1,j-1) + g_{32} I(i+1,j) + g_{33} I(i+1,j+1)$$

Linear filtration

$$\begin{array}{|c|c|c|} \hline 10 & 5 & 3 \\ \hline 4 & 5 & 1 \\ \hline 1 & 1 & 7 \\ \hline \end{array} \otimes \begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 0 & 0.5 & 0 \\ \hline 0 & 1.0 & 0.5 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline & 7 & \\ \hline & & \\ \hline \end{array}$$

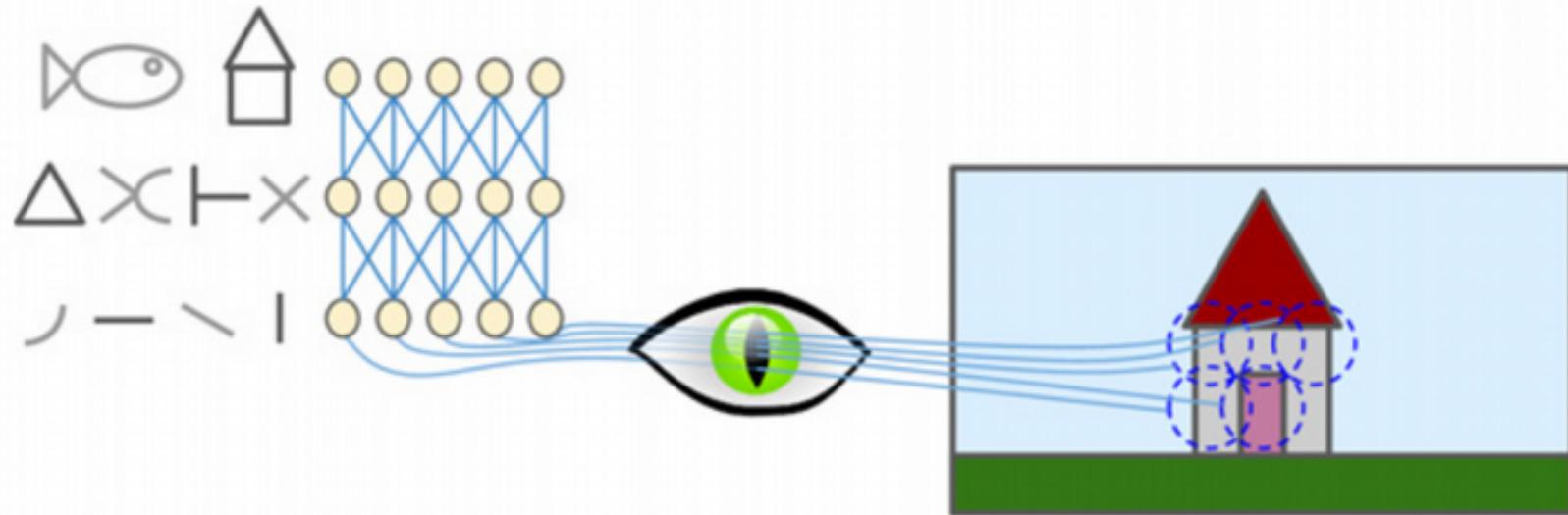
Jądro splotu

- Linear is the simplest and most useful
- Turns every pixel into a linear neighbors combination.
- The function (method) for the linear combination is called the convolution kernel.

Cconvolutional Neural Networks - CNN

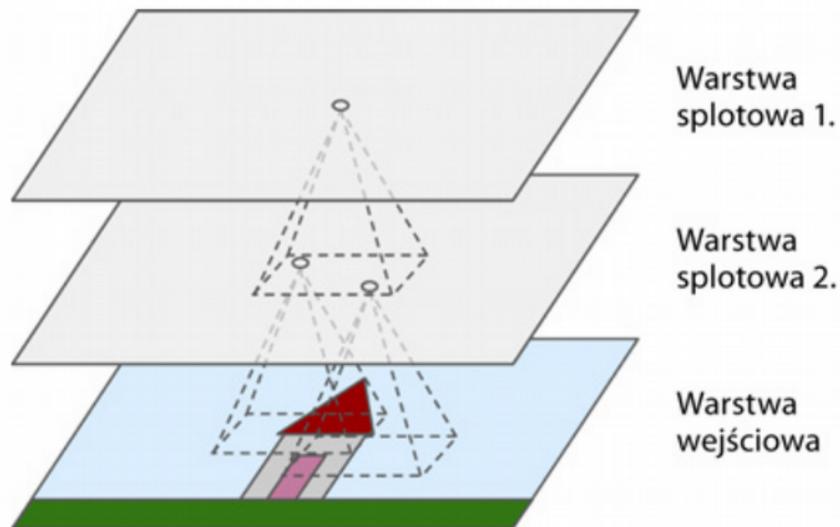
- They are the result of research on the visual cortex and have been used for image recognition since the 1980s.
- They form the basis of image search services, smart cars, automated movie classification systems, etc.
- They are also effective in other tasks, such as voice recognition or natural language processing.

Architecture of the visual cortex



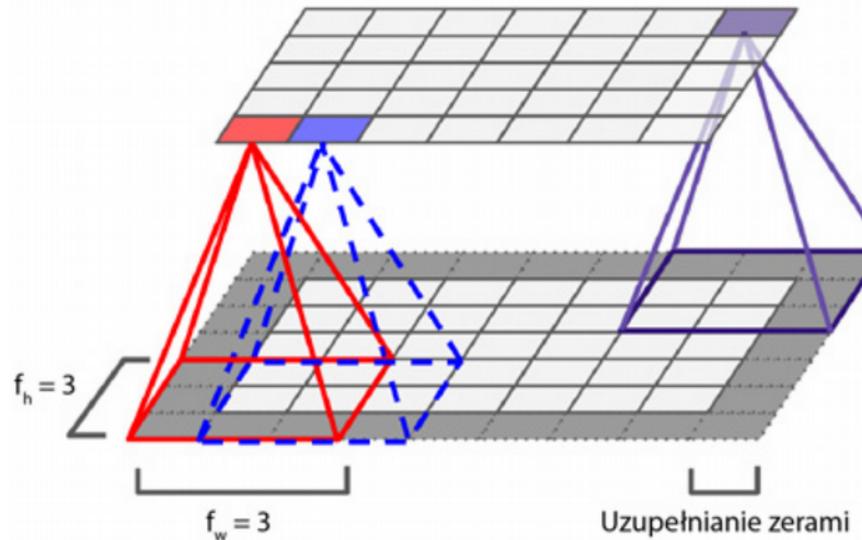
- Neurons form **local reception fields** - they react only to visual stimuli located in a specific region,
- Some neurons may have the same reception field, but they only react to images consisting of horizontal lines, and other vertical ones,
- Neurons that detect more complex shapes (which are a combination of more general patterns) are found on the output of neighboring neurons.

Convolution layer



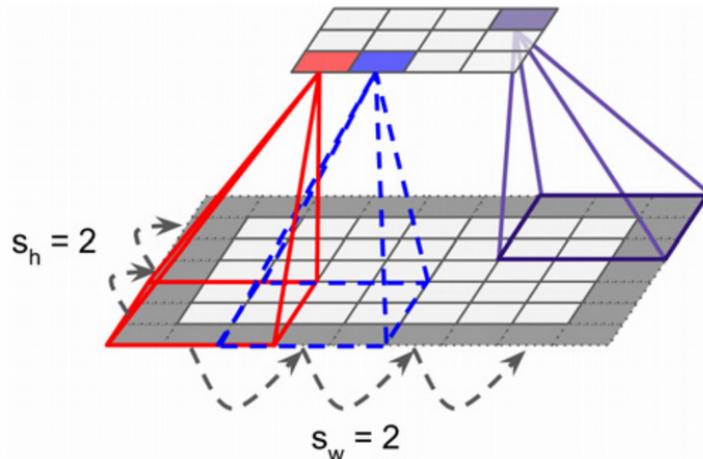
- Neurons in the first convolution layer are not connected to each pixel of the input image, but only to the pixels in their reception field
- In turn, each neuron in the second convolution layer connects only to neurons located in a small area of the first layer.

2D Convolution



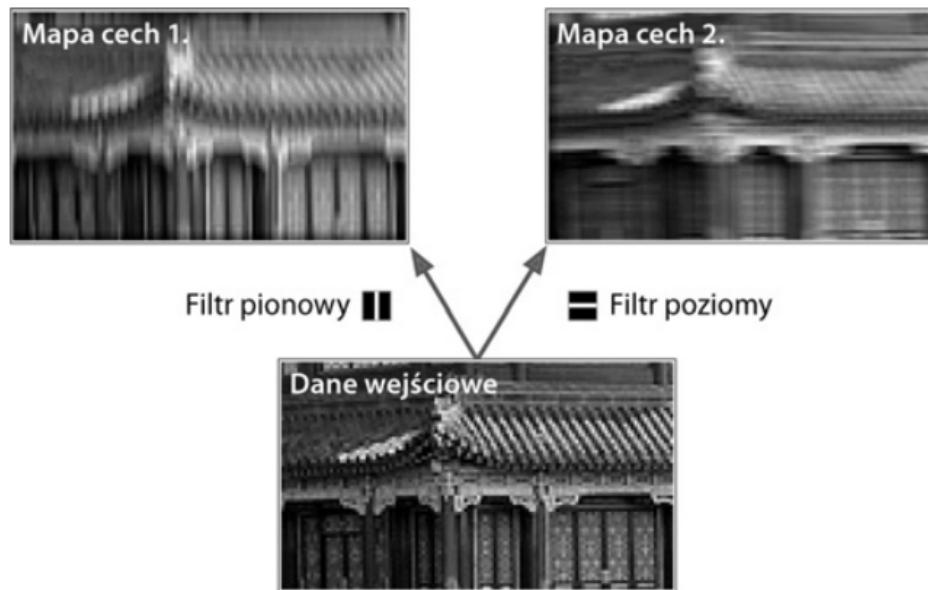
- Connected to the neuron outputs of the previous layer located in rows from i to $i + f_h - 1$ and columns from j to $j + f_w - 1$, where f_h and f_w mean, respectively, the height and width of the reception field
- In order to obtain the same dimensions of each layer, zeros (zero padding) are most often added around the inputs

Separation of reception fields



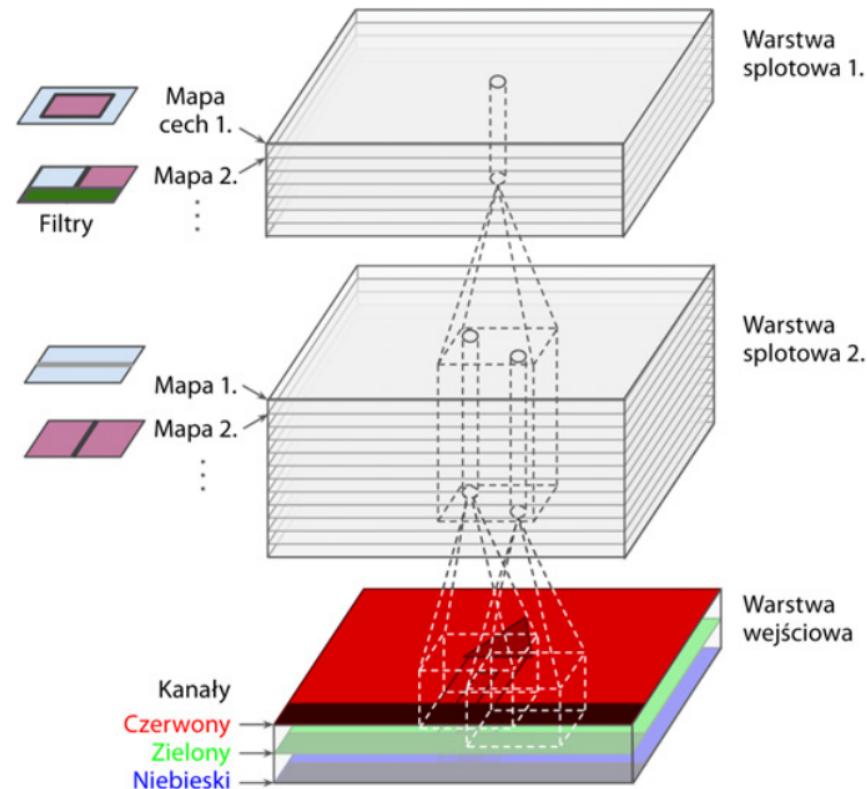
- The distance between two consecutive fields is bf step
- The input layer with dimensions 5×7 (plus zero padding) connects to the size layer 3×4 using the reception fields that are squares 3×3 and a step 2
- The neuron located in the i rows and the j column of the top layer connects to the output of the bottom layer neurons located in rows from $i \times s_h$ to $i \times s_h + f_h - 1$ and in columns from $j \times s_w$ to $j \times s_w + f_w - 1$, where s_h and s_w values of steps in columns and rows.

Filters



- The neuron scales can be presented as a small image with the size of the reception field,
- The first filter (a black square with a white vertical line) passing through its center (this is a 7×7 matrix filled with zeros in addition to the middle column that contains ones). The second filter (the middle line is arranged horizontally),
- A layer filled with neurons using the same filter gives us a feature map,

Stacks of features map



- For one feature map, all neurons have the same parameters,
- Other feature maps may have different parameter values,
- The convolution layer simultaneously applies various filters to the inputs, thanks to which it is able to detect many features.

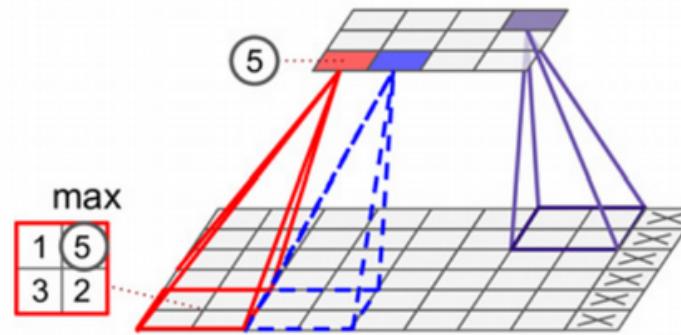
The values of the initial neuron in the convolution layer

$$z_{i,j,k} = b_k \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k'=0}^{f_{n'}-1} x_{i',j',k'} \cdot w_{u,v,k',k} \quad (1)$$

where $i' = i \times s_h + u$ and $j' = j \times s_w + v$

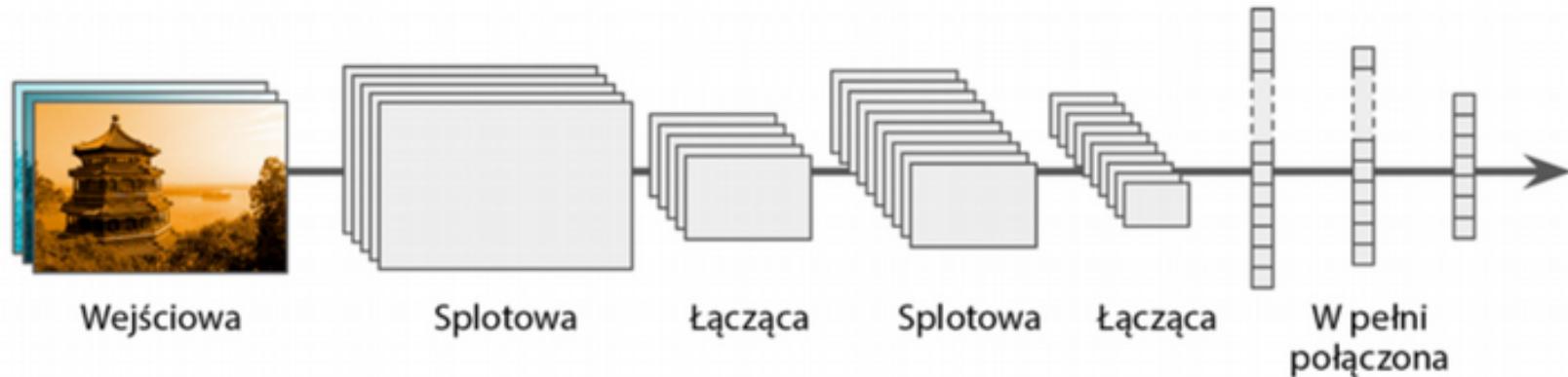
- $z_{i,j,k}$ - neuron output in row i , column j and map feature k convolution layer l ,
- s_h and s_w - vertical and horizontal steps,
- f_h and f_w - height and width of the reception field,
- f_n - number of feature maps in the previous layer ($l - 1$),
- $x_{i,j,k}$ - neuron output in $l - 1$, row i , column j , feature maps k
- b_k - bias for the features map k (in layer l)
- $w_{u,v,k',k}$ - weight in the features map k layers l and its entrance in the row u , column v and features map k

Connecting layer



- Connection kernel: 2×2 , no zeroing
- Does not contain any weights - collects input data using an aggregation function (e.g. maximizing or averaging),
- The purpose of the convolution layer is to subscribe (*subsample*) the input image for optimization purposes,
- Neuron from the connecting layer connects to the outputs of a given number of neurons of the previous layer, in the area of the reception field.

Architecture of convolutional neural networks



- Typical CNN architectures consist of several layers of a convolution, connecting layer, after it several CONVOLUTION layers (+ ReLU), another layer of joining, etc.
- The image gradually decreases, going through subsequent layers of the network, but at the same time its depth increases
- A classic neural network is placed at the top of the network, containing several fully connected layers (+ ReLU), and the last one calculates prediction (eg. softmax logistic regression).

```
# convolution network w keras
from keras import layers
from keras import models
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
model.summary()
```

- The network adopts a tensor with a shape determined by the height of the image, its width and image channels, e.g. (28, 28, 1)
- At the output of each layer *Conv2D* and *MaxPooling2D*, a three-dimensional tensor with shape (height, width, channels) appears.
- As the network sinks, the height and width tend to take smaller values.
- The last step is a dense network with a classifier (tensor (3, 3, 64)).

The result of creating the model

```
Instructions for updating:
```

```
Colocations handled automatically by placer.
```

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_1 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_2 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_3 (Conv2D)	(None, 3, 3, 64)	36928
flatten_1 (Flatten)	(None, 576)	0
dense_1 (Dense)	(None, 64)	36928
dense_2 (Dense)	(None, 10)	650

```
Total params: 93,322
```

```
Trainable params: 93,322
```

```
Non-trainable params: 0
```

Starting and training the model

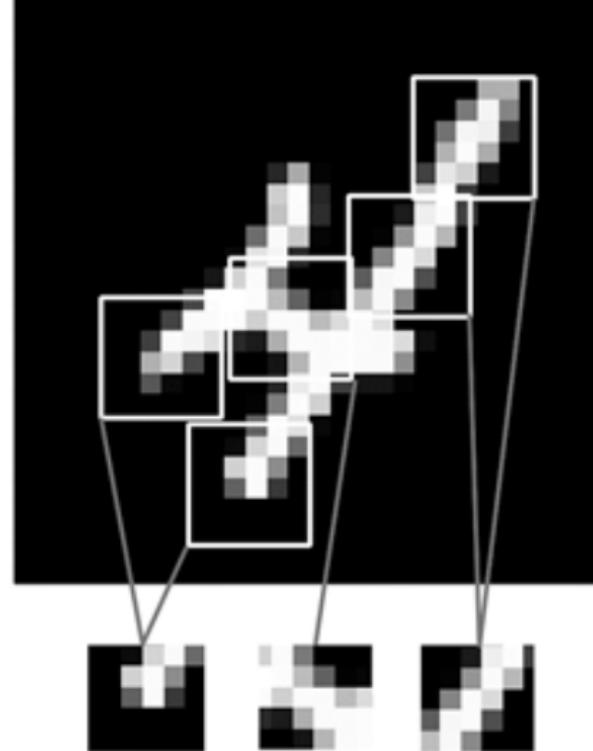
```
from keras.datasets import mnist
from keras.utils import to_categorical
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
model.compile(optimizer='rmsprop', loss='categorical_crossentropy',
metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5, batch_size=64)

test_loss, test_acc = model.evaluate(test_images, test_labels)
print('test_acc = ', test_acc)
```

The result of the program:

```
.....
10000/10000 [=====] - 1s 72us/step
test_acc = 0.9919
```

Convolutional network

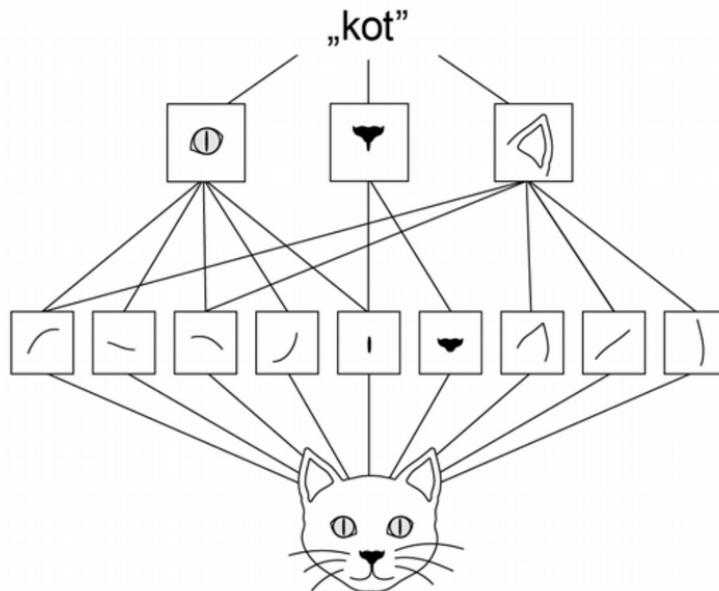


- Images can be broken down into local patterns, e.g. edges and textures,
- The basic difference between the dense connections layer and the convolutional network is that the Dense layers learn the parameters of global parameters in their input spaces,

Properties of convolutional networks

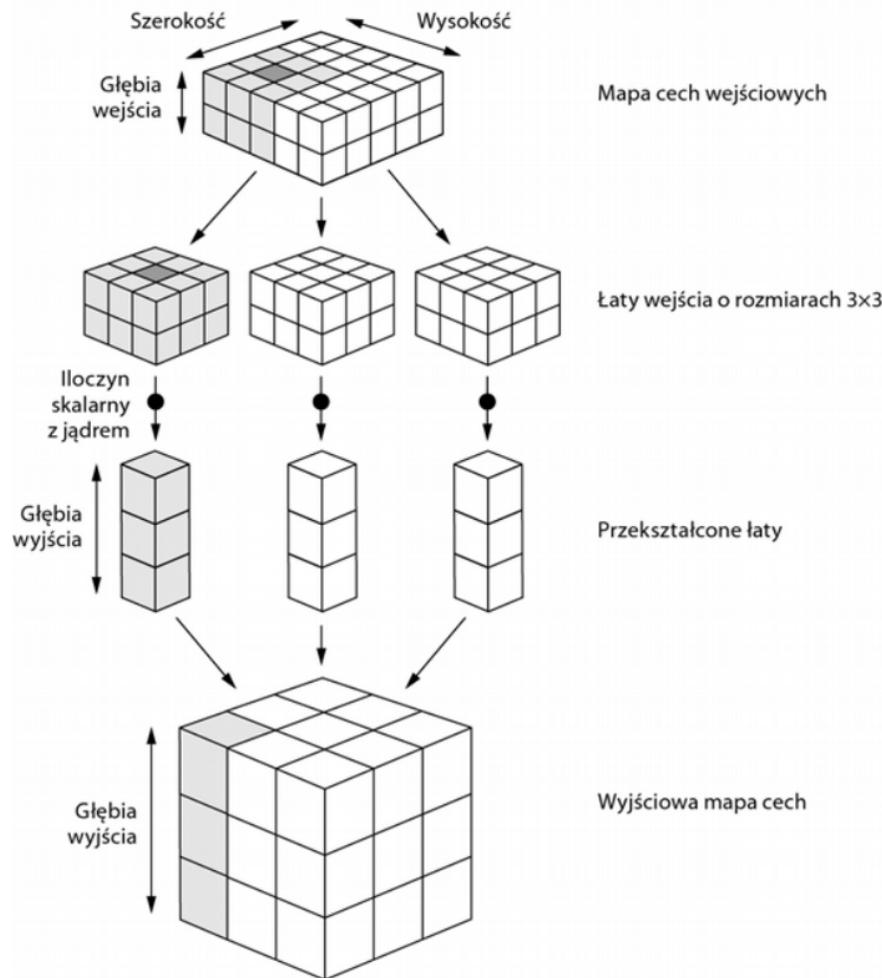
- **Patterns recognized by the network are independent of shifting.** - the image processing algorithm must not bind the shape to its position in the frame.
- **Convolutionary networks can learn spatial hierarchy of patterns.** The first layer of the convolution network is taught small local patterns (eg. edges), the second layer of this network will learn the larger structures consisting of elements recognized by the first layer, etc.

Spatial hierarchy of elements



- They work on three-dimensional tensors known as feature maps, containing two spatial axes defining height and width,
- The third axis is the depth axis, also called the channel axis,
- Extract patches from input features and perform the same operation on all patches to create the output feature map.

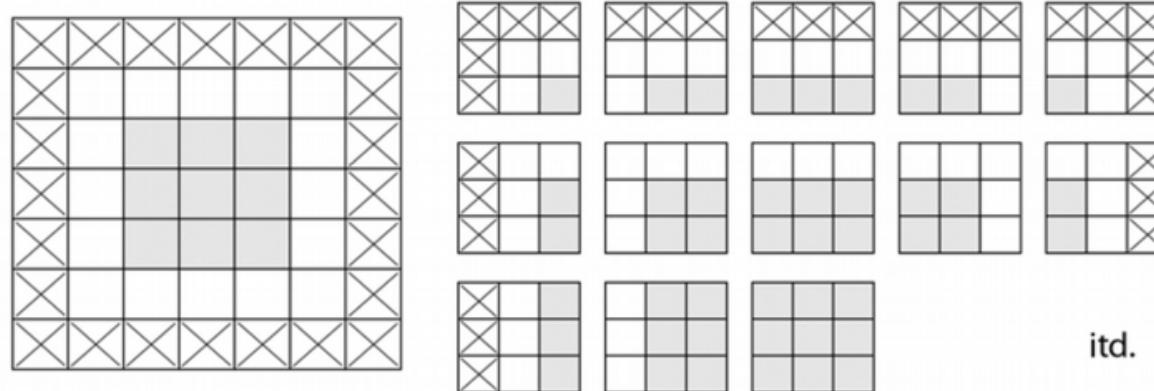
The operation of convolution



The operation of convolution

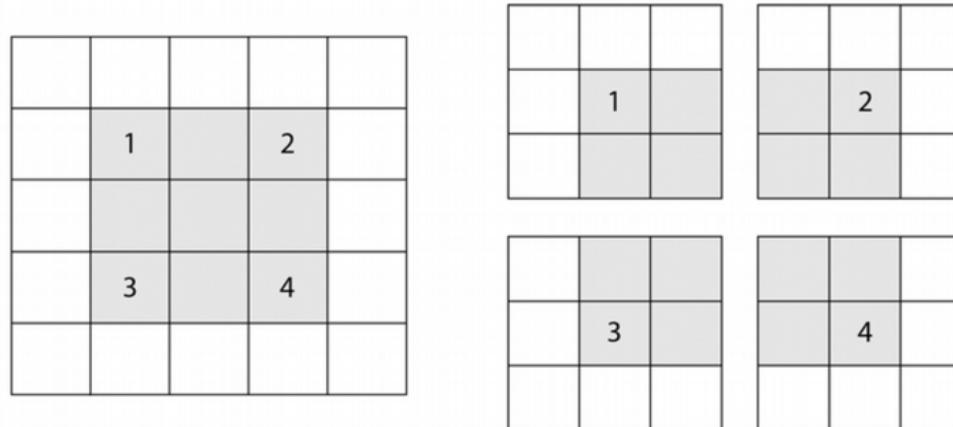
- It involves moving windows with dimensions 3×3 or 5×5 on a three-dimensional map of features.
- Extracts the three-dimensional patch of the surrounding features (*window height, window width, entry depth*)).
- Each such three-dimensional patch is then transformed (through a tensor product with the convolution kernel) to obtain a one-dimensional vector with the shape (*depth of the output*)).
- All these vectors are then rebuilt spatially to create a three-dimensional output map with the shape (*height, width, depth of the output*)).

Border effects and complement



- An example of a feature map with dimensions 5×5 - it contains only 9 fields on which you can set the center of the window with dimensions 3×3
- The feature map will be 3×3 .
- If we wanted to get the output map of features with the same dimensions as the input map, then we would have to use the padding technique.
- It's about adding the right number of rows and columns on each side of the input feature map

Steps of the convolution process



Convolution patches 3×3 at the 2×2 step (without complement).

- The distance between two windows is a convolution parameter - **step** (stride).
- By default, it takes 1, but you can assign a higher value to it, which will lead to the so-called convolutions.
- A step with a value of 2 means that the width and height of the feature map are scaled (they are reduced twice).
- Instead of steps, the **max-pooling** scaling operation is typically used.

Max-pooling operation

- This operation aggressively reduces the resolution of feature maps,
- The *max-pooling* scaling operation is performed using the extraction windows transforming feature maps.
- These windows return the maximum values of each channel - tensor operation max (maximum),
- The *max-pooling* operation is usually performed using windows 2×2 at a step equal to 2 (this is to reduce the feature map twice).
- Convulsions are performed using windows 3×3 with a step parameter equal to 1

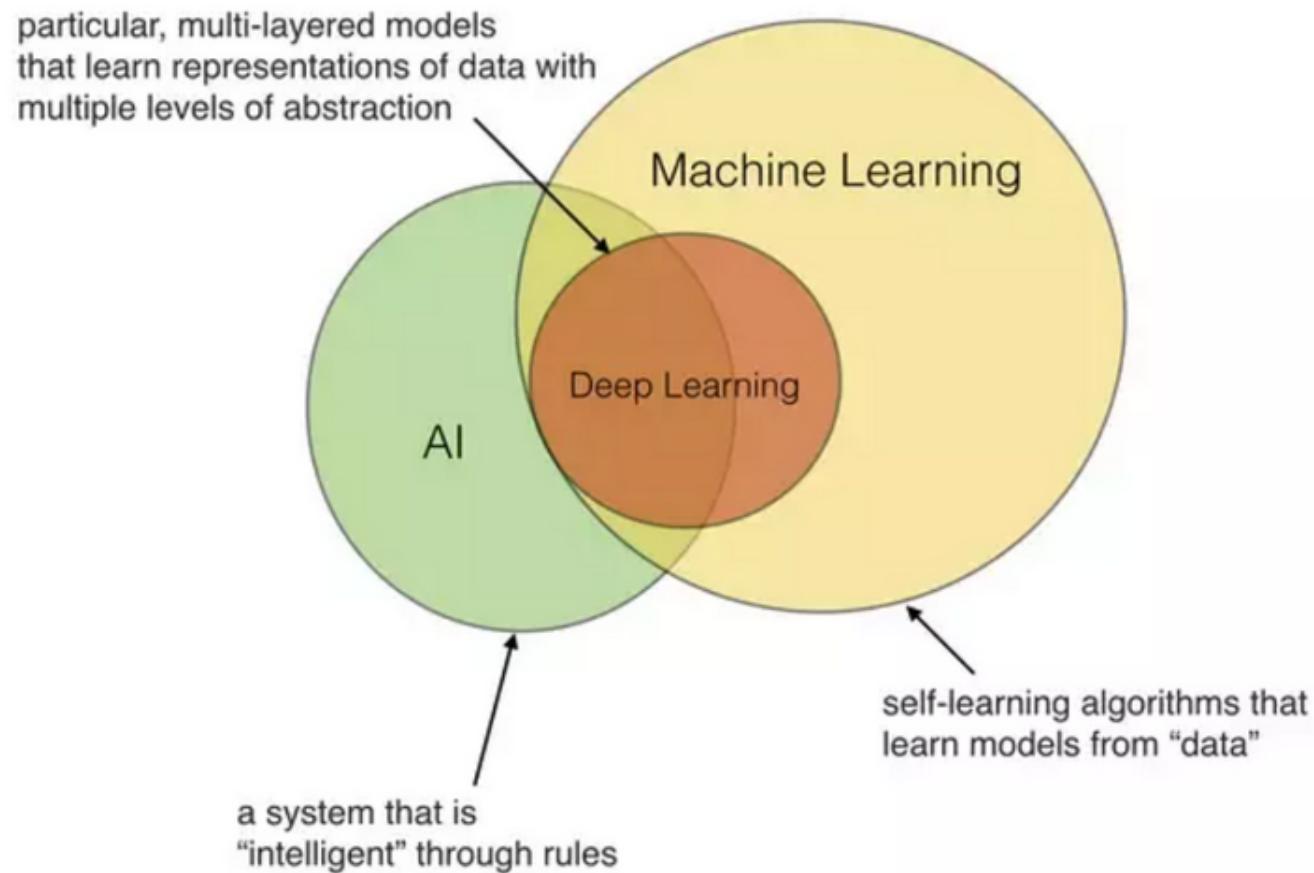
Computer Vision Deep learning - lecture 7

Adam Szmigiełski

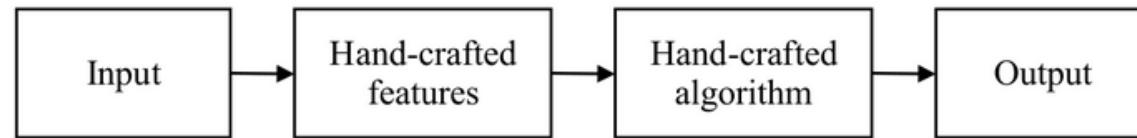
aszmigie@pjwstk.edu.pl

materials: *ftp(public) : //aszmigie/WMAEnglish*

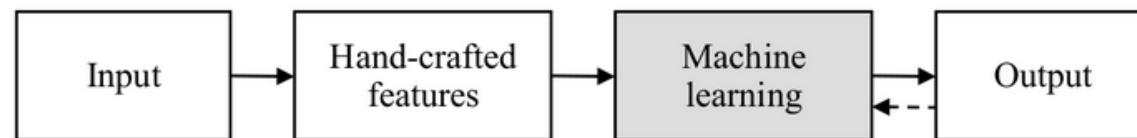
Deep learning



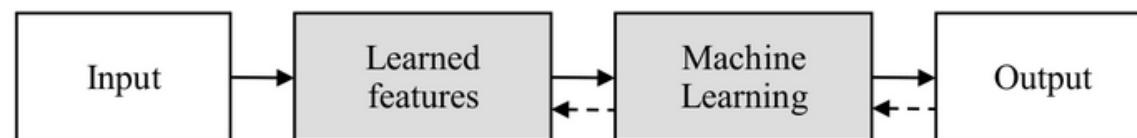
Deep learning in computer vision



(a) Traditional vision pipeline



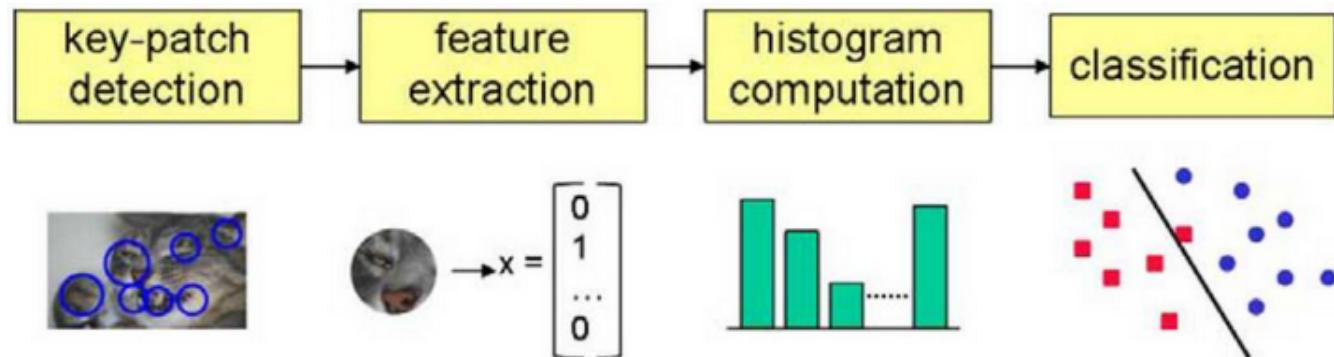
(b) Classic machine learning pipeline



(c) Deep learning pipeline

Currently, deep neural networks are the most popular and widely used machine learning models in computer vision, not just for semantic classification and segmentation, but even for lower-level tasks such as image enhancement, motion estimation, and depth recovery.

Typical processing pipeline



Learning of deep neural network using the standard TensorFlow interface

Learning consists of two phases:

- **Construction phase** - the number of inputs and outputs should be determined, as well as the number of neurons in each layer,
- **Executive phase** - opens the session and starts the init node initializing all variables. Then we enter the main loop - learning the neural network.

Construction phase

- Replacement nodes represent learning data (input) X and target (output) y - node X as $(None, n_{inputs})$, y one-dimensional tensor,

```
X = tf.placeholder(tf.float32, shape=(None, n_inputs), name="X")
y = tf.placeholder(tf.int64, shape=(None), name="y")
```

- The network layer will be described by the input parameters, the number of neurons, the activation function and the name of the layer,

```
def neuron_layer(X, n_neurons, name, activation=None):
    with tf.name_scope(name):
        n_inputs = int(X.get_shape()[1])
        stddev = 2 / np.sqrt(n_inputs)
        init = tf.truncated_normal((n_inputs, n_neurons), stddev=stddev)
        W = tf.Variable(init, name="jadro")
        b = tf.Variable(tf.zeros([n_neurons]), name="obciazenie")
        Z = tf.matmul(X, W) + b
        if activation is not None:
            return activation(Z)
        else:
            return Z
```

- We define the network constructor

```
with tf.name_scope("gsn"):  
    hidden1 = tf.layers.dense(X, n_hidden1, name="ukryta1",  
                             activation=tf.nn.relu)  
    hidden2 = tf.layers.dense(hidden1, n_hidden2, name="ukryta2",  
                             activation=tf.nn.relu)  
    logits = tf.layers.dense(hidden2, n_outputs, name="wyjscia")
```

- Calculating the loss of information (by calculating *cross entropy*)

```
with tf.name_scope("strata"):  
    xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y,  
                                                               logits=logits)  
    loss = tf.reduce_mean(xentropy, name="strata")
```

- Gradient calculation with the optimizer:

```
learning_rate = 0.01  
with tf.name_scope("uczenie"):  
    optimizer = tf.train.GradientDescentOptimizer(learning_rate)  
    training_op = optimizer.minimize(loss)
```

- Assessment of the network's operation:

```
with tf.name_scope("ocena"):  
    correct = tf.nn.in_top_k(logits, y, 1)  
    accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))
```

- Network initialization node:

```
init = tf.global_variables_initializer()  
saver = tf.train.Saver()
```

Executive phase

We import data, open the *TensorFlow* session and start the init node initializing all variables.

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("/tmp/dane/")
n_epochs = 40
batch_size = 50

init = tf.global_variables_initializer()
with tf.Session() as sess:
    init.run()
    for epoch in range(n_epochs):
        for iteration in range(mnist.train.num_examples // batch_size):
            X_batch, y_batch = mnist.train.next_batch(batch_size)
            sess.run(training_op, feed_dict={X: X_batch, y: y_batch})
        acc_train = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
        acc_test = accuracy.eval(feed_dict={X: mnist.validation.images, y: mnist.validation.labels})
        print(epoch, "Dokladnosc uczenia:", acc_train,
              "Dokladnosc testowania:", acc_test)
    save_path = saver.save(sess, "./moj_model_ostateczny.ckpt"
```

The network can be read and used again.

Tuning hyperparameters of the neural network

- **The number of hidden layers** - they can model complicated functions with an exponentially fewer number of neurons than in the case of “shallow” networks (significantly speeds up their learning),
- **Number of neurons forming the hidden layer** - depends on the input and output data required by a specific task (usually the subsequent layers have fewer neurons),
- **Activation functions** - In most cases, in the hidden layers the function ReLU is used, in the output layer the activation function depends on the purpose of the network (softmax for classification, linear for regression).

Deep network learning problems

- The problem of vanishing gradients (or closely related problem of exploding gradients), which concerns deep networks (significantly hinders learning of lower layers of the network),
- Learning an extensive network can be very time-consuming.
- A model containing millions of parameters is significantly exposed to overtraining.

Problems of disappearing / exploding gradients

- The back propagation algorithm runs from the output layer to the input layer and distributes the gradient along the way.
- Gradient values often decrease along with the algorithm's progress to the lower layers of the network - the vanishing gradients problem.
- Sometimes we can find the opposite phenomenon: gradients are constantly increasing, which in many layers of the scale are updated rapidly - the problem of exploding gradients,
- Neural networks show unstable gradients syndrome; individual layers can learn at radically different speeds.

Xavier and He weights initiation

Activation function	Uniform distribution $\langle -r, r \rangle$	Normal distribution
logistic	$r = \sqrt{\frac{6}{n_{inputs} + n_{outputs}}}$	$\sigma = \sqrt{\frac{6}{n_{inputs} + n_{outputs}}}$
tang. hiperbolic	$r = 4 \sqrt{\frac{6}{n_{inputs} + n_{outputs}}}$	$\sigma = 4 \sqrt{\frac{6}{n_{inputs} + n_{outputs}}}$
RELU type	$r = \sqrt{2} \sqrt{\frac{6}{n_{inputs} + n_{outputs}}}$	$\sigma = \sqrt{2} \sqrt{\frac{6}{n_{inputs} + n_{outputs}}}$

- In order to properly signal the direction of information flow, the variance of outputs in a given layer must be equal to the variance of its inputs,
- Gradients must have the same variance before and after passing through the layer in the opposite direction.

Nonsaturation activation functions

- In the case of deep neural networks, the ReLU function performs better than sigmoidal, because it does not saturate for positive values,
- The ReLU function is not perfect - dying ReLUs. During learning, some neurons permanently “get lost”, i.e. the only signal they send is 0,
- It is difficult to re-enter the neuron to the network - the gradient of the ReLU function is 0 with negative values on the input.

Leaky ReLU function



Variants of the ReLU function

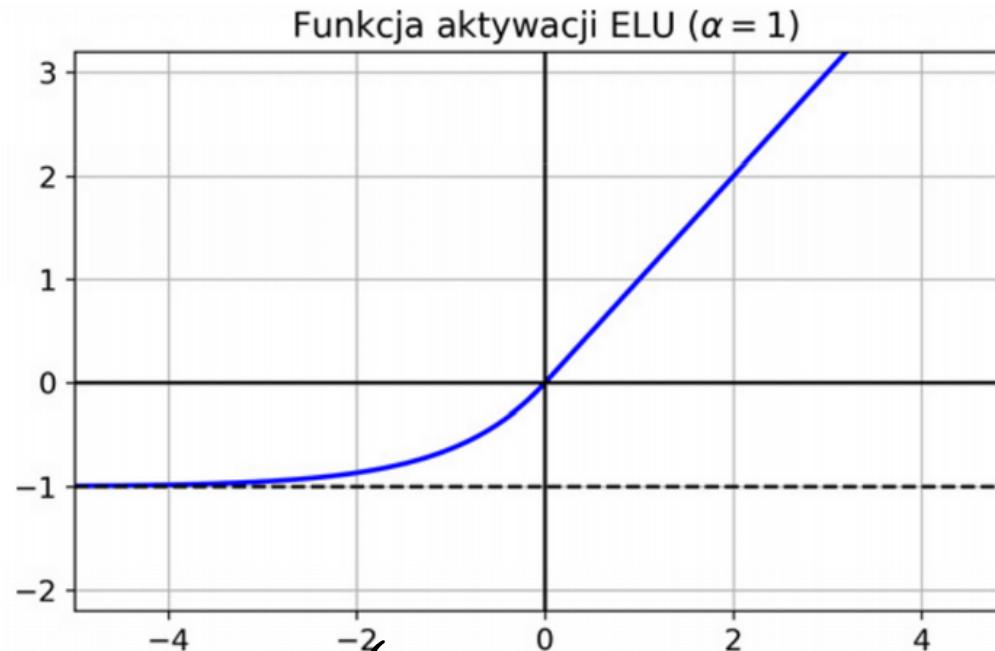
- **Leaky ReLU:**

$$\text{ReLU}_\alpha(z) = \max\{\alpha_z, z\}$$

Hyperparameter α determines the degree of “leakage” function: it determines the slope of the function for $z < 0$ and usually takes the value of 0,01,

- **Random, leaking function ReLU** (called randomized leaky ReLU - RReLU) - the value of the *alpha* hyperparameter is randomly selected in the specified range during learning,
- **Parametric leaky ReLU function** - parameter α “learns” along with the whole model (similar to other parameters, it can be modified in the context of back propagation).

Exponential Linear Unit - ELU



$$ELU_{\alpha}(z) = \begin{cases} \alpha(e^z - 1) & \text{for } z < 0 \\ z & \text{for } z \geq 0 \end{cases}$$

- In the experiments performed, it turned out to be better than all the variations of the ReLU function: the learning time was shorter, and the neural network itself performed better against the test set.

Properties of the ELU activation function

- For $z < 0$ it takes negative values - it solves the problem of disappearing gradients,
- For $z < 0$ it has a non-zero gradient, which is a solution to the problem of “dying” neurons.
- The function is smooth at every point, also in $z = 0$, which allows you to speed up the simple gradient method,
- It is slower calculated from the standard ReLU function and its variations, and during learning this is compensated by a better convergence coefficient. The network from ELU will be slower than the network from ReLU.

Batch normalization

- In deep learning there is a problem of distribution of input changes in each layer during learning, resulting from changes in parameters in the previous layer,
- Before the activation function in each layer, the input data is normalized, and then rescales and shifts the result with two new parameters:
 - one is responsible for scaling
 - second for shifting
- This operation allows the model to determine the optimal scale and average of the input data for each layer.

Batch normalization using the TensorFlow module

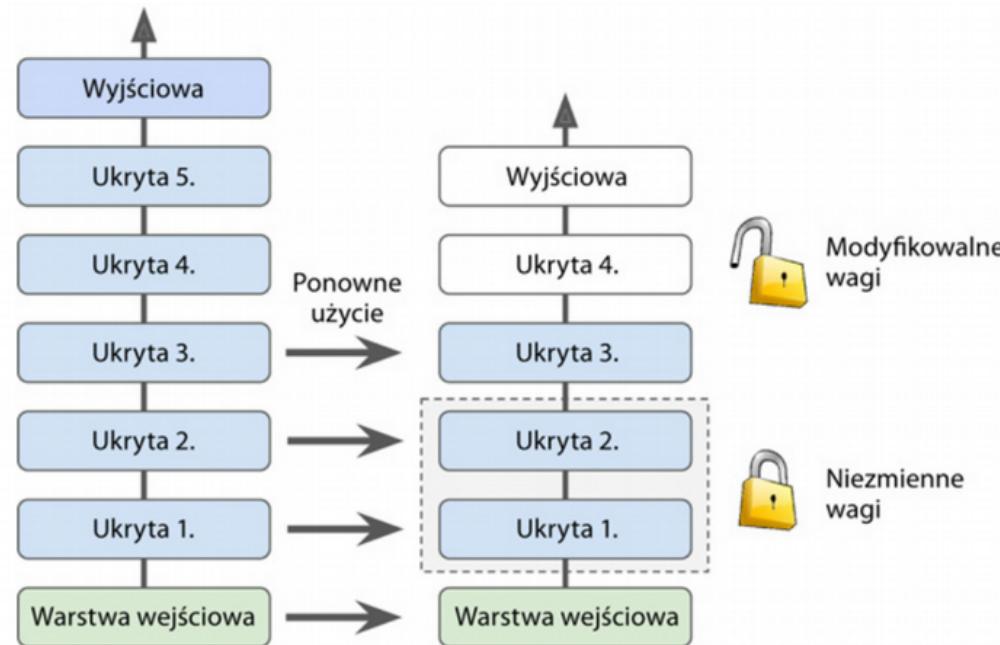
```
hidden1 = tf.layers.dense(X, n_hidden1, name="ukryta1")
bn1 = tf.layers.batch_normalization(hidden1, training=training, momentum=0.9)
bn1_act = tf.nn.elu(bn1)
hidden2 = tf.layers.dense(bn1_act, n_hidden2, name="ukryta2")
bn2 = tf.layers.batch_normalization(hidden2, training=training, momentum=0.9)
bn2_act = tf.nn.elu(bn2)
logits_before_bn = tf.layers.dense(bn2_act, n_outputs, name="wyjscia")
logits = tf.layers.batch_normalization(logits_before_bn,
                                       training=training, momentum=0.9)
```

- The TensorFlow module contains function *tf.nn.batch_normalization()*, which centers and normalizes input data,
- The function *tf.layers.batch_normalization()* itself calculates the mean and standard deviation (using the training data of minibatches) while passing them to the function as parameters.
- In batch normalization, no activation function is assigned - this is used after every normalization.

Gradient clipping

- A popular technique for limiting the problem of exploding gradients is to cut gradients in the back propagation stage in such a way that they never exceed a certain threshold,

Multiple use of ready-made layers



- Learning very large networks is expensive.
- You can use an existing network adapted to a similar task and use its lower layers - **transfer learning**,
- It significantly speeds up the learning process, but also requires significantly less learning data.

Freezing of lower layers

```
with tf.name_scope("gsn"):  
    hidden1 = tf.layers.dense(X, n_hidden1, activation=tf.nn.relu,  
                             name="ukryta1") # ponownie uzyta, zamrozona  
    hidden2 = tf.layers.dense(hidden1, n_hidden2, activation=tf.nn.relu,  
                             name="ukryta2") # ponownie uzyta, zamrozona  
    hidden2_stop = tf.stop_gradient(hidden2)  
    hidden3 = tf.layers.dense(hidden2_stop, n_hidden3, activation=tf.nn.relu,  
                             name="ukryta3") # ponownie uzyta, niezamrozona  
    hidden4 = tf.layers.dense(hidden3, n_hidden4, activation=tf.nn.relu,  
                             name="ukryta4") # nowa!  
    logits = tf.layers.dense(hidden4, n_outputs, name="wyjscia") # nowa!
```

- By inserting the *stop gradient()* layer into the graph. Any layers beneath it will be frozen.

Model repositories

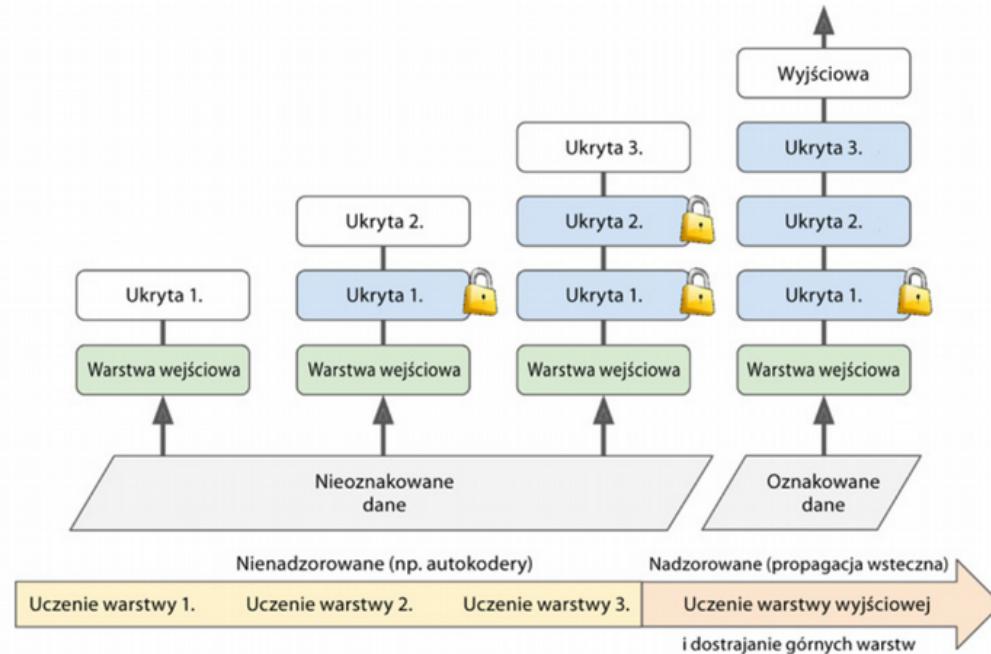
- Module *TensorFlow* has its own repository on the site
[https : //github.com/tensorflow/models](https://github.com/tensorflow/models)

There are networks designed for image classification, such as VGG, Inception or ResNet, ready-made models, as well as tools that allow you to download the most popular learning data files

- Repository Model Zoo Caffe company:
[https : //github.com/BVLC/caffe/wiki/Model – Zoo](https://github.com/BVLC/caffe/wiki/Model-Zoo)

There are many models of digital image recognition available (eg LeNet, AlexNet, ZFNet, GoogLeNet, VGGNet, Inception) trained on a variety of data sets (such as ImageNet, Places Database, CIFAR10, etc.).

Unsupervised initial learning



- For unmarked teaching data, you can try to train the model layer by layer, using the algorithm of unattended detection of features (eg Boltzmann machines or autocoders).
- Each subsequent layer is learned using the results obtained from the previous layers,
- After learning all layers, the network tunes backward propagation.

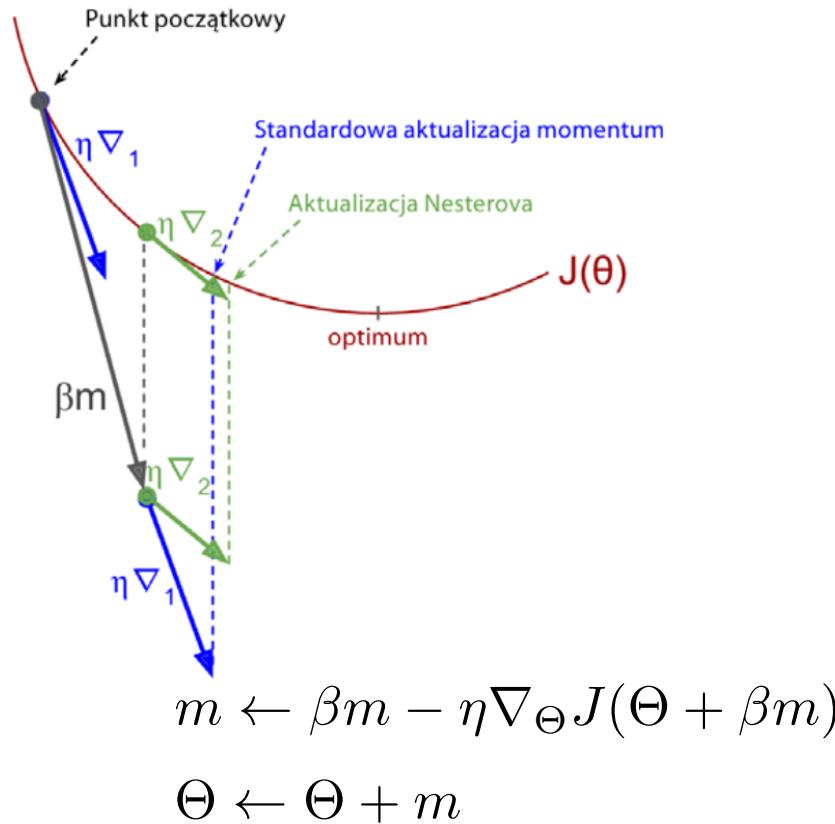
The momentum - optimization algorithm

$$m \leftarrow \beta m - \eta \nabla_{\Theta} J(\Theta)$$

$$\Theta \leftarrow \Theta + m$$

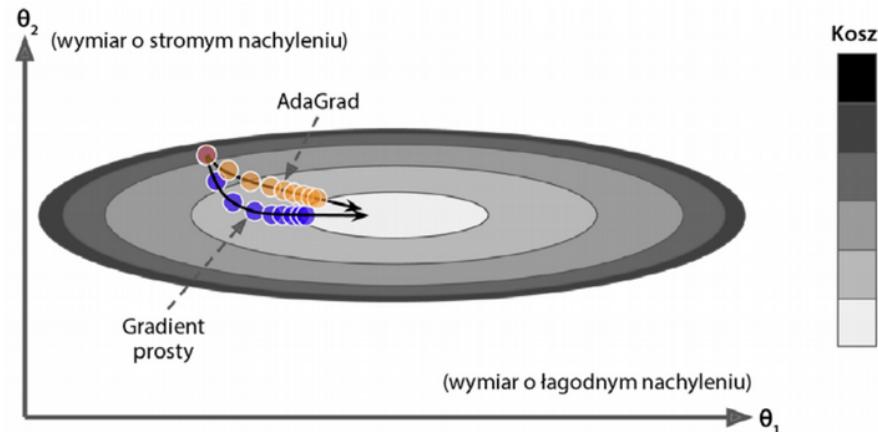
- In the *simple gradient method*, the Θ weights are updated by subtracting the $J(\Theta)$ cost function gradient from weights - $\nabla_{\Theta} J(\Theta)$ multiplied by η learning factor. Earlier gradients are not taken into account here,
- The *momentum* optimization takes into account the previous gradients: in each run the local gradient is subtracted from the **moment vector** m (multiplied by the learning factor η).
- The β parameter simply called moment (or momentum) prevents excessive momentum.

Accelerated gradient drop (Nesterov algorithm)



The only difference compared to momentum optimization is the measurement of the gradient using the expression $\Theta + \beta m$, and not Θ .

Algorithm AdaGrad



$$s \leftarrow s + \nabla_{\Theta} J(\Theta) \otimes \nabla_{\Theta} J(\Theta)$$

$$\Theta \leftarrow \Theta - \eta \nabla_{\Theta} J(\Theta) \oslash \sqrt{s + \epsilon}$$

- Where the \otimes symbol denotes the multiplication of individual elements, the \oslash symbol denotes division by individual elements, and the ϵ parameter is the smoothing element used to avoid dividing operations by 0,
- This is done by gradually reducing the gradient vector along the steepest directions

Algorithm RMSProp

$$\begin{aligned}s &\leftarrow \beta s + (1 - \beta) \nabla_{\Theta} J(\Theta) \otimes \nabla_{\Theta} J(\Theta) \\ \Theta &\leftarrow \Theta - \eta \nabla_{\Theta} J(\Theta) \oslash \sqrt{s + \epsilon}\end{aligned}$$

- The AdaGrad algorithm never reaches convergence with the global minimum,
- This problem is solved by the **RMSProp** algorithm, by collecting only gradients from the most current waveforms,
- The distribution factor β usually has a value of 0.9.

Optimization Adam - Adaptive Moment Estimation

$$m \leftarrow \beta_1 m - (1 - \beta_1) \nabla_{\Theta} J(\Theta)$$

$$s \leftarrow \beta_2 s + (1 - \beta_2) \nabla_{\Theta} J(\Theta) \otimes \nabla_{\Theta} J(\Theta)$$

$$m \leftarrow \frac{m}{1 - \beta_1^t}$$

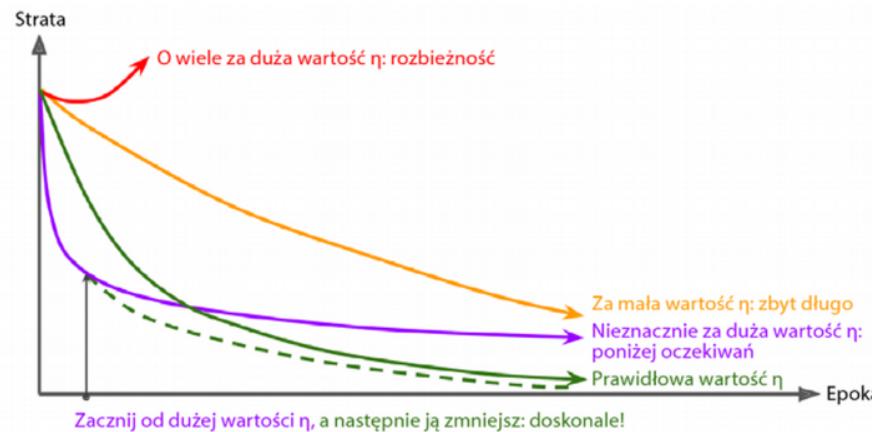
$$s \leftarrow \frac{s}{1 - \beta_2^t}$$

$$\Theta \leftarrow \Theta + \eta m \oslash \sqrt{s + \epsilon}$$

where t - step number (starting from 1).

- The Adam algorithm combines the concept of momentum optimization and RMSProp optimizer,
- from the momentum optimization takes the exponential distribution of the average of previous gradients,
- from the RMSProp optimizer to track the exponential distribution of the average of the previous squares of gradients

Scheduling the learning coefficient

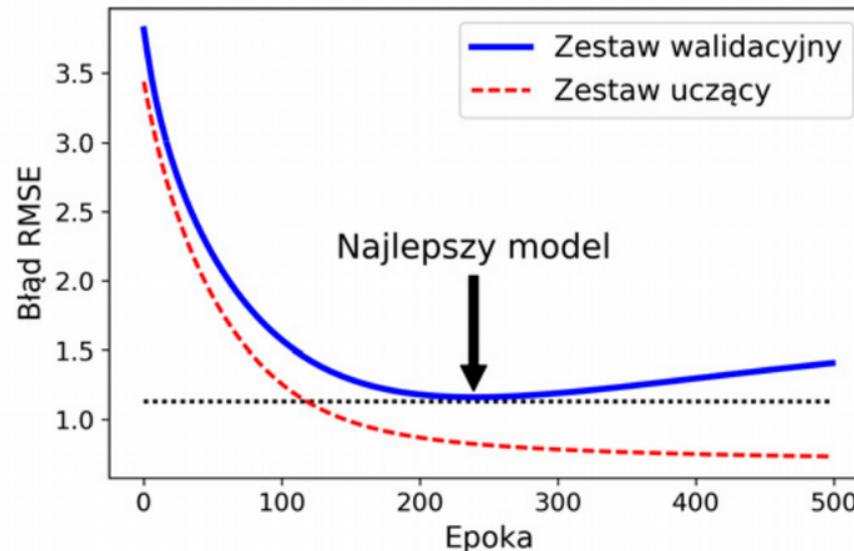


- **Top, interval, constant learning factor** - often requires additional value determination and time of change.
- **Performance scheduling** - we measure the validation error every N of the waveforms and when the error value ceases to decrease, we reduce the learning factor
- **Exponential scheduling** - we set the learning factor based on the t function
- **Power Scheduling** - similar to exponential, but the value of the learning factor decreases much slower.

Methods of protection against overfitting

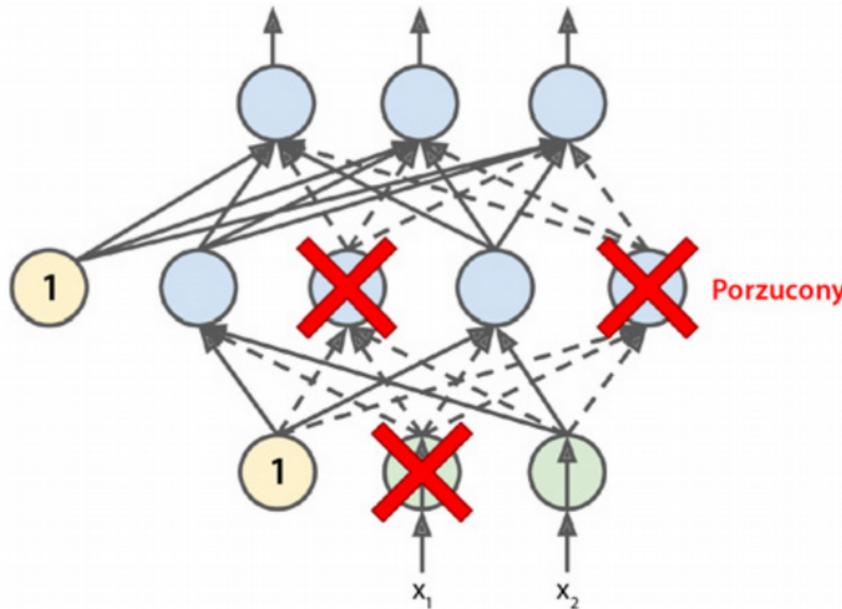
- **Early stopping** - when the performance of the model decreases against the validation set,
- **Regularization L_1 and L_2** - can be used in neural networks to limit connection weights,
- **Dropping** - in individual training courses, the neuron can be temporarily “droped”, i.e. completely omitted,
- **Augmentation of data** - it is about creating new teaching examples from already existing samples.

Early stopping



- The end of learning at the moment of reaching the minimum validation error - early stopping,
- As the subsequent epochs pass, the algorithm learns, and its prediction error against the training set decreases in a natural way, just like the error of prediction against validation data.
- After some time, the forecasting error ceases to decrease and begins to increase again (the moment of overfitting).

Dropping

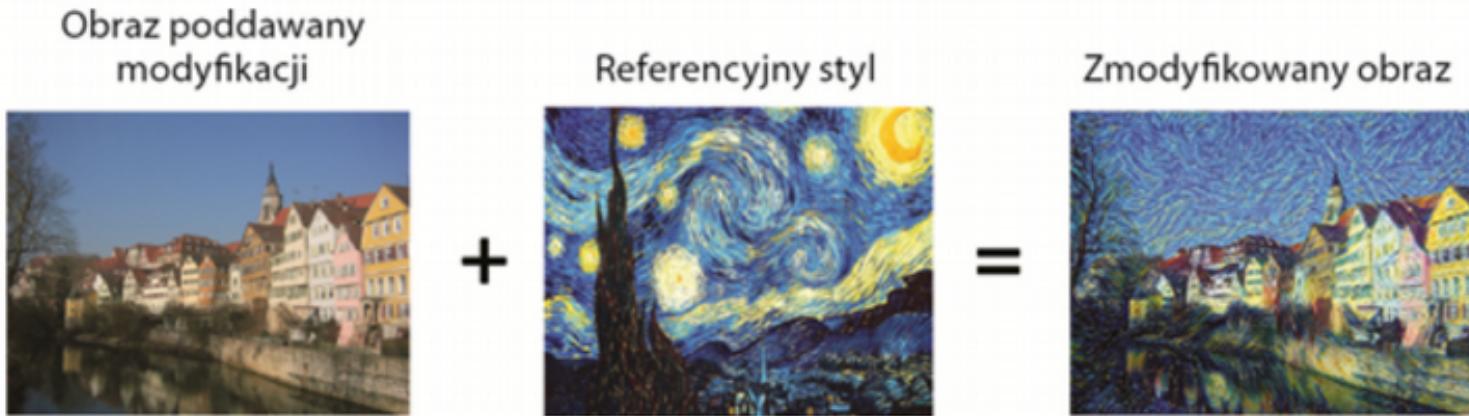


- In individual training cycles, each neuron (in addition to the initial layer) can be temporarily omitted in the learning process,
- In each cycle pass it may be active again,
- The p hyperparameter is called the dropout rate and usually has a value of 50%.
- After finishing learning, neurons stoped to be dropped.

DeepDream

- The DeepDream algorithm attempts to maximize the activation of the whole layer, not the selected filter, which results in simultaneous mixing of the visualization of many features.
- Generating the image does not start with an empty, slightly noisy input image - the finished image is processed at the input, which results in applying effects to the previously created image,
- Input images are processed at different scales referred to as octaves, which aims to improve the quality of visualization.

Neural style transfer



- Neural style transfer involves applying a reference image style to process another image while preserving its contents.
- The concept **style** refers to the textures, colors and presentation of the things shown in the image.
- **Content** is the high-level macrostructure of the image.

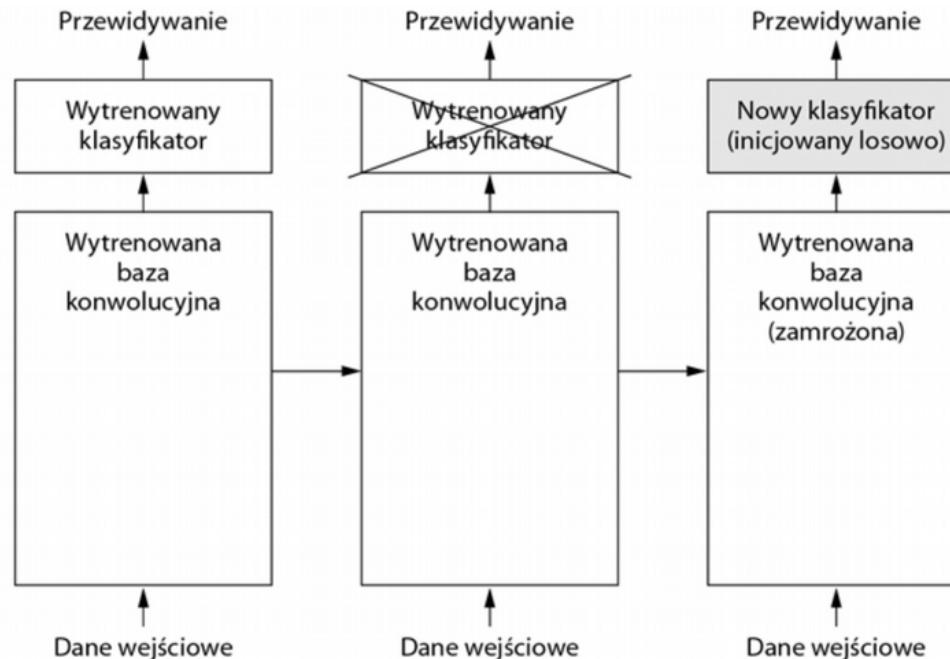
Training a convolutional neural network on a small data set

- A common situation during private work on the problems of image analysis,
- *Small number of samples* can mean a different number - from a few hundred to several dozen,
- Deep learning works mainly when it is possible to access a large amount of data,
- The algorithms of this learning can independently select useful traits from the training data set, but require a large training data set.
- A collection of several hundred examples may be sufficient if the model is small and subject to regularization, and the task will be simple.

Data augmentation technique

- Excessive matching results from too few samples on which the model can learn,
- Data Augmentation is a technique for generating more training elements of a data set by augmenting samples by random transformations that return images,
- In Keras augmentation is assisted by the instance of *ImageDataGenerator*.

Extraction of features



- The extraction of features consists in using the representation learned by the network earlier in order to extract the features of interest from new samples.
- These attributes are then passed through a new classifier trained from scratch.

Freezing a layer or set of layers

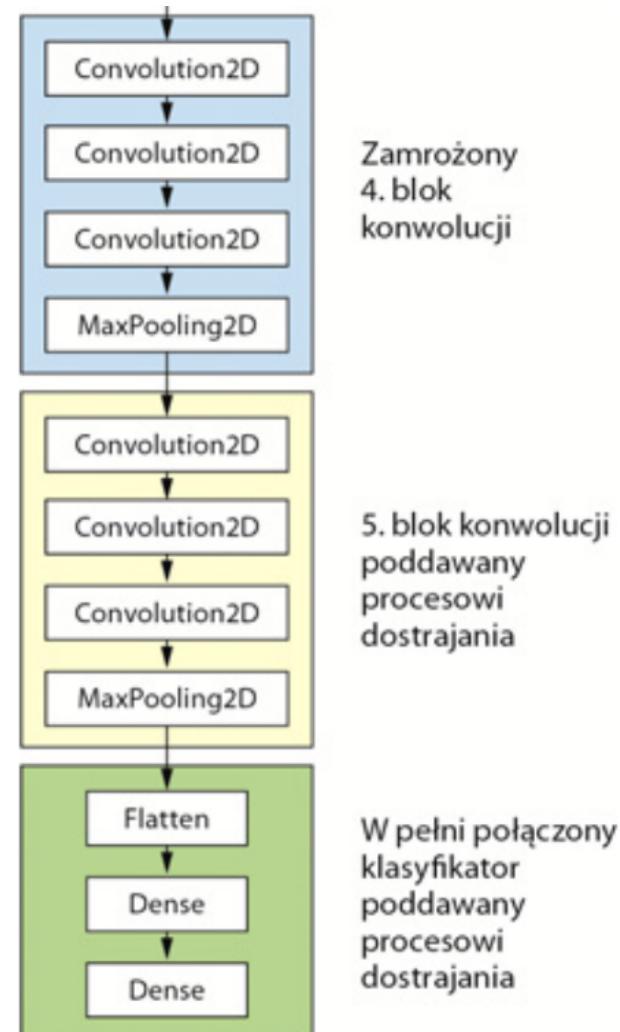
- Freezing the layer or set of layers consists in preventing the update of their weights in the training process.
- If freezing representations previously learned by the convolutional base will be modified during training.
- Dense layers at the top are initialized randomly, which would make them go to major changes to all network parameters, and this would effectively destroy the previously learned representations.
- In the Keras package, the network freezes up by assigning *False* to the attribute *trainable*

Tuning

Tuning is a technique of reusing models that complement the extraction of features.

- It involves defrosting several top layers of frozen model base used to extract traits and training it together with a new part of the model,
- Most often this part of the model is a fully connected classifier,
- This process is referred to as tuning because it modifies partially previously trained more abstract model representations in order to adapt them to the current problem.

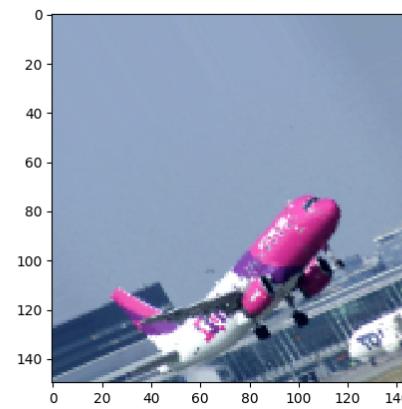
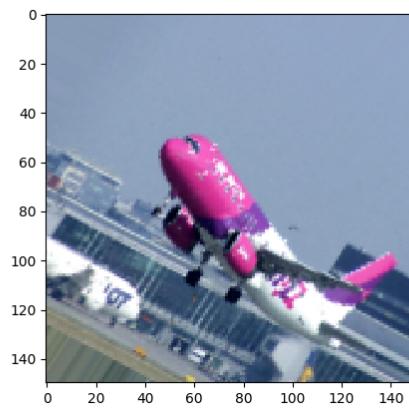
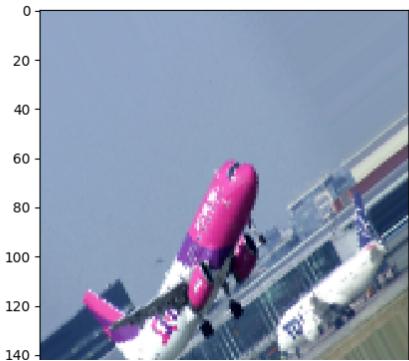
Tuning



Visualization of the learning effects of convolutional neural networks

- **Visualization of intermediate output data** - a technique useful for understanding how consecutive layers of a convolutional network transform input data and what individual network filters do.
- **Visualization of convolutional network filters** - a technique that enables a precise understanding of a graphic pattern or graphic concept that each of the convolutional network filters reacts to.
- **Visualization of heat maps for activation of image classes** - a technique useful for identifying parts of an image identified as belonging to a given class

Augmentation of data



- It involves creating new examples from existing ones,
- To varying degrees, slightly move, rotate and resize.

Deep learning - data generation

- generating text using the LSTM network,
- DeepDream - technique of artistic image modification that uses representations learned by convolutional networks neuronowe.

Task for laboratory

- Every student is ask to take 10 photos from 3 different categories: cup, keyboard, book,
- Please create a common database (for all students) - standardize the format of all photos,
- Propose a convolutional neural network and teach it to recognize objects,
- Increase the database by using augmentation techniques. Teach the network on an extended dataset. Compare the results.

Computer Vision Architectures of convolution neural networks - lecture 8

Adam Szmigiełski

aszmigie@pjwstk.edu.pl

materials: *ftp(public) : //aszmigie/WMAEnglish*

Popular convolutional network architectures

- RNN - fast RNN, faster RNN
- SSD
- YOLO

LeNet-5 1998

Layer	Type	Maps	Size	Kernel Size	Step	F. Activation
Out	Fully connected	—	10	—	—	RBF
F6	Fully connected	—	84	—	—	tanh
C5	convolutions	120	1×1	5×5	1	tanh
S4	Connecting	16	5×5	2×2	2	tanh
C3	convolutions	16	10×10	5×5	1	tanh
S2	connecting	6	14×14	2×2	2	tanh
C1	convolutions	6	28×28	5×5	1	tanh
In	Input	1	32×32	—	—	—

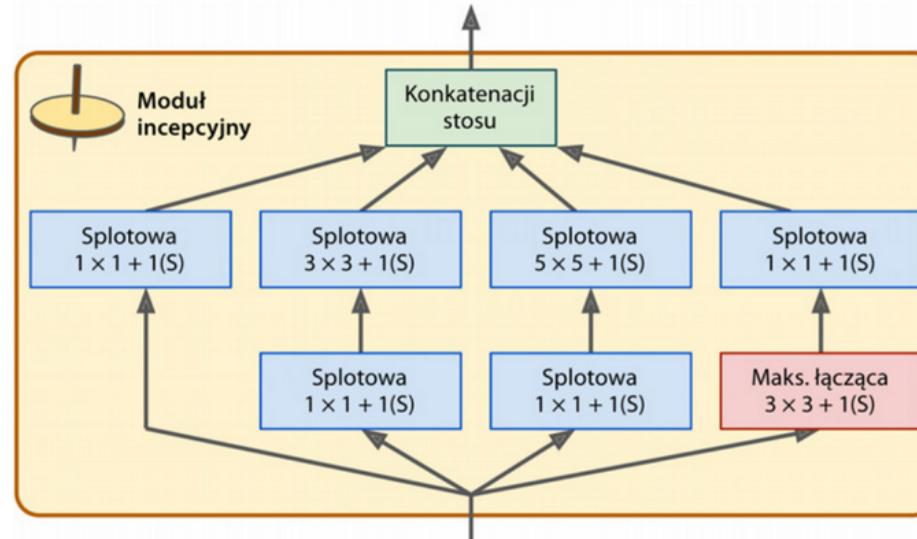
- Została stworzona przez Yanna LeCuna w 1998 roku do rozpoznawania odręcznie pisanych cyfr (MNIST)
- Architektura LeNet-5 stanowi prawdopodobnie najbardziej znany przykład sieci CNN.

AlexNet 2012

Layer	Type	Maps	Size	Kernel Size	Step	Build.	with zeros	F. activation
Out	Fully connected	—	1000	—	—	—	—	Softmax
F9	Fully connected	—	4096	—	—	—	—	ReLU
F8	Fully connected	—	4096	—	—	—	—	ReLU
C7	convolutions	256	13×13	3×3	1	SAME	ReLU	ReLU
C6	convolutions	384	13×13	3×3	1	SAME	ReLU	ReLU
C5	convolutions	384	13×13	3×3	1	SAME	ReLU	ReLU
S4	connecting	256	13×13	3×3	2	VALID	—	—
C3	convolutions	256	27×27	5×5	1	SAME	ReLU	ReLU
S2	connecting	96	27×27	3×3	2	VALID	—	—
C1	convolutions	96	55×55	11×11	4	SAME	ReLU	ReLU
In	Output	3(RGB)	224×224	—	—	—	—	—

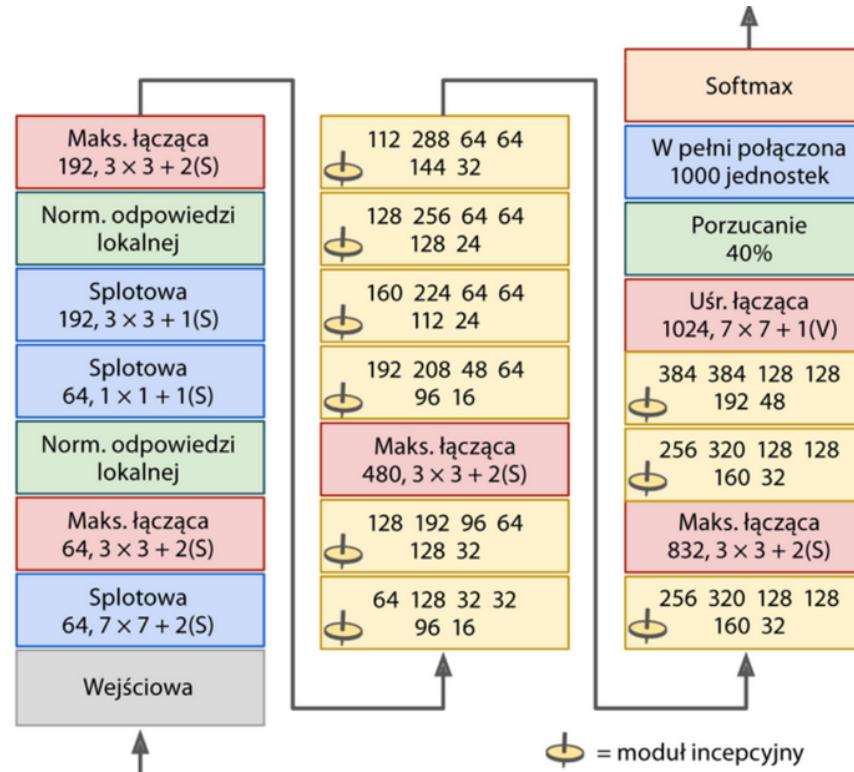
- In order to reduce overtraining, the drop-off method was used on the outputs of the F8 and F9 layers, and they generated data,
- Local response normalization (LRN) was used in layers C1 and C3 - neurons with the highest weights (the most activating) inhibit neurons located in the same position,

GoogLeNet 2014



- The creation of such an architecture was possible due to the introduction of subnets called Inception modules
- The second set of convolution layers contains kernels of different sizes (1×1 , 3×3 , and 5×5), which allows them to catch patterns at different scales.

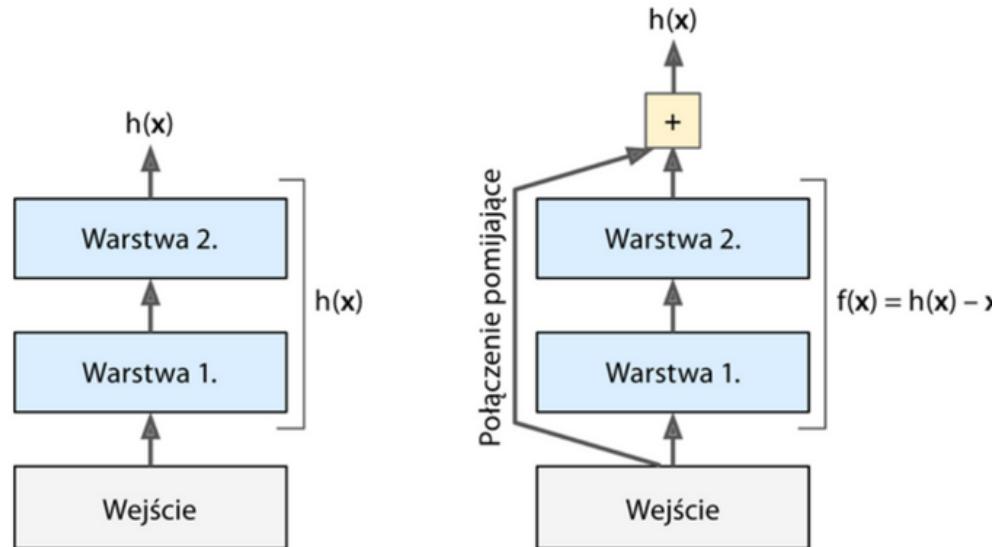
GoogLeNet network architecture



- The first two layers reduce the height and length of the image four times,
- The local response normalization layer instructs previous layers to learn to recognize very different characteristics

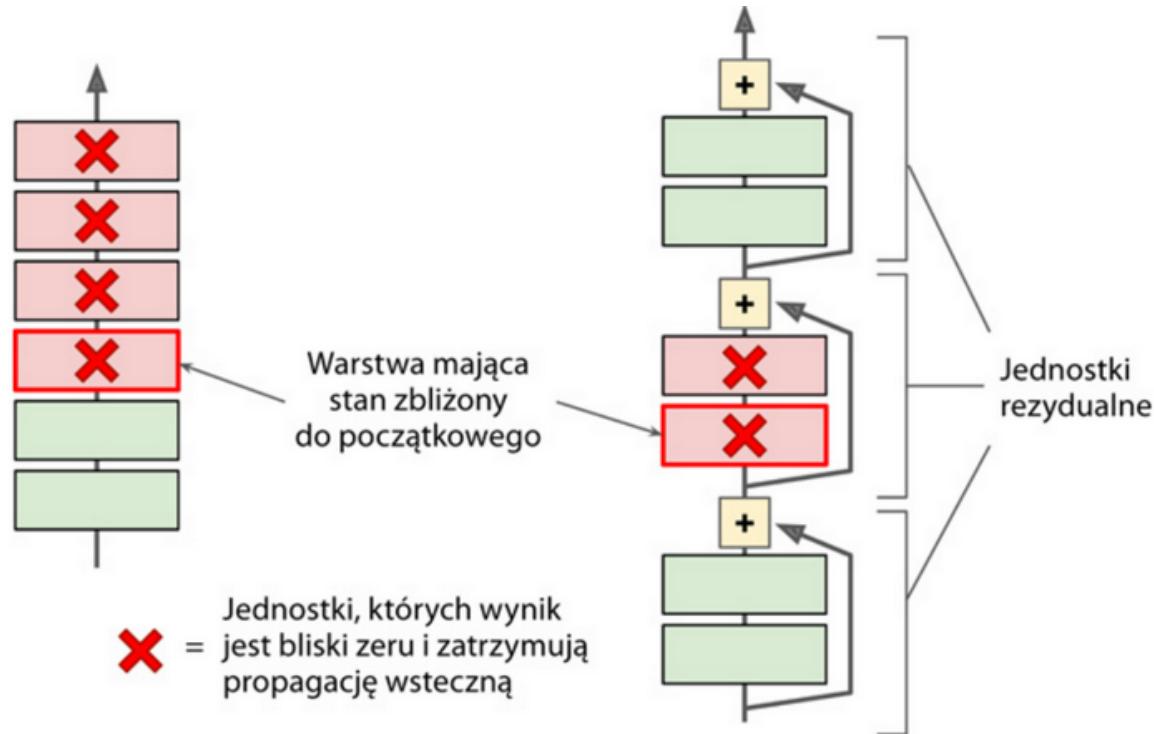
- Two convolution layers, the first of which acts as the boundary layer.
- Local response normalization layer,
- Maximizing blending layer reduces the image dimensions twice as much,
- A tall stack of nine inception modules that
- Dropping is used to regularize, then a fully connected layer using the softmax activation function to display the estimated probabilities of the examples belonging to a given class.

ResNet 2015 - Residual Network



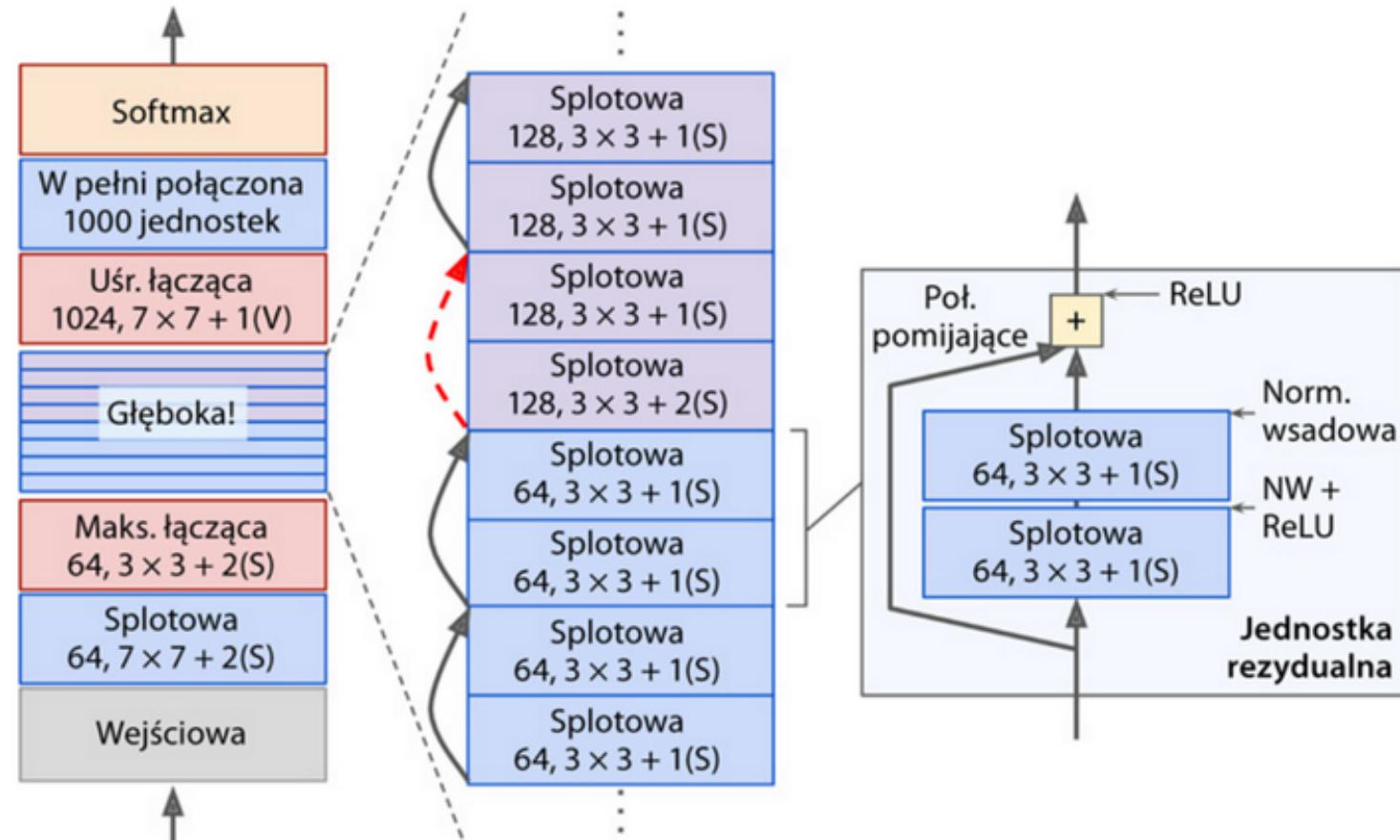
- Deep architecture was used - consisting of 152 layers.
- **connections bypassing** (also called shortcut connections) used
- If we add x input to the network output (skipping connection), the network maps the function $f(x) = h(x) - x$ - **residual learning**.

Deep residual network



- A deep residual network can be viewed as a stack of residual units, or small neural networks containing a bypass connection.

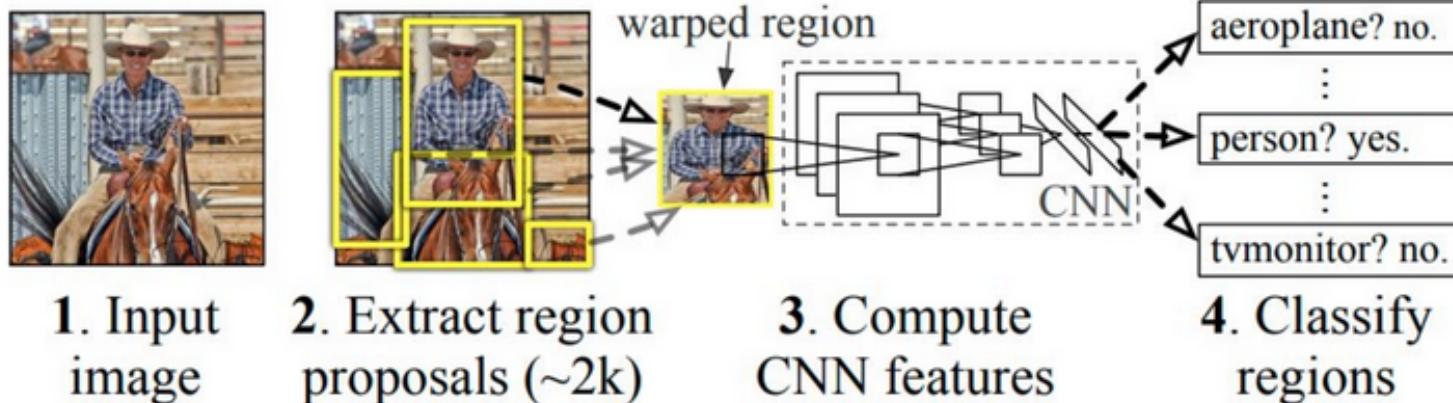
Architecture of the ResNet network



Object detection using convolutional networks

- R-CNN (Regions with CNN features) - fast RNN, faster RNN
- SSD (Single Shot Multibox Detector)
- YOLO (You Only Look Once)

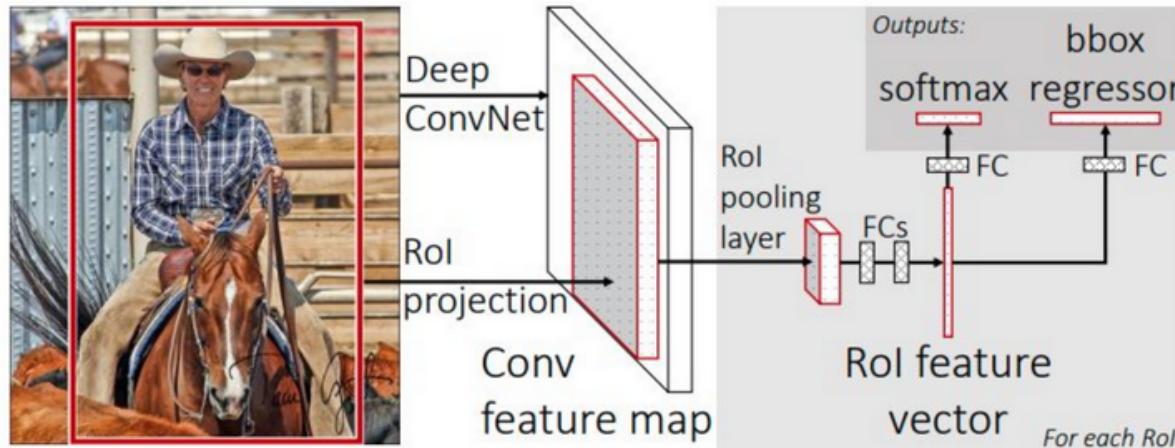
Region convolutional neural network - R-CNN



- **Region proposal** - generating regions where the object can be located,
- **Feature selection module** - feature extraction from each region,
- **Classifier** - classifies features into one of the known classes using a classifier.

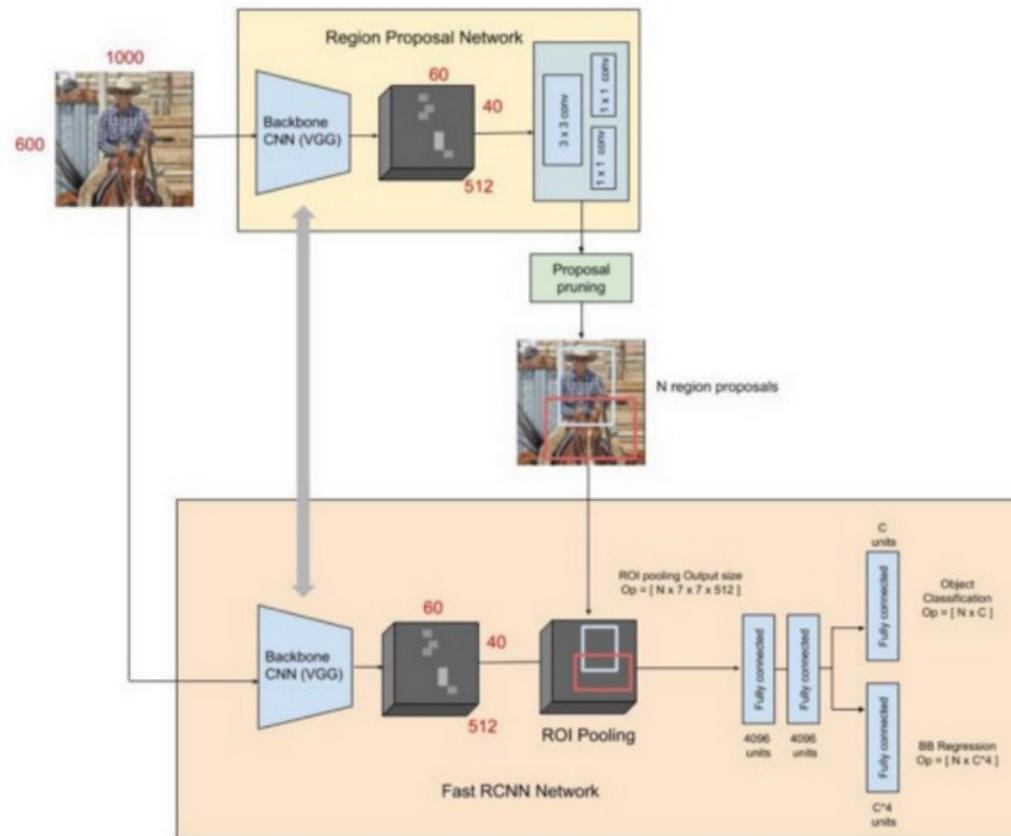
Long training (approx. 2000 region suggestions for each image). Searching for regions is not a learning algorithm - it can lead to wrong propositions and this error will not be corrected.

Architecture of the Fast R-CNN



- Single model to learn region and classification proposals at once,
- Uses a pre-trained network to select features.
- At the end of the network is a Region of Interest Pooling Layer (RoI Pooling) that extracts the features for each proposed region.
- Then this data is interpreted in a layer fully connected to two outputs - classifier and **bounding box**

Architecture of the Faster R-CNN model

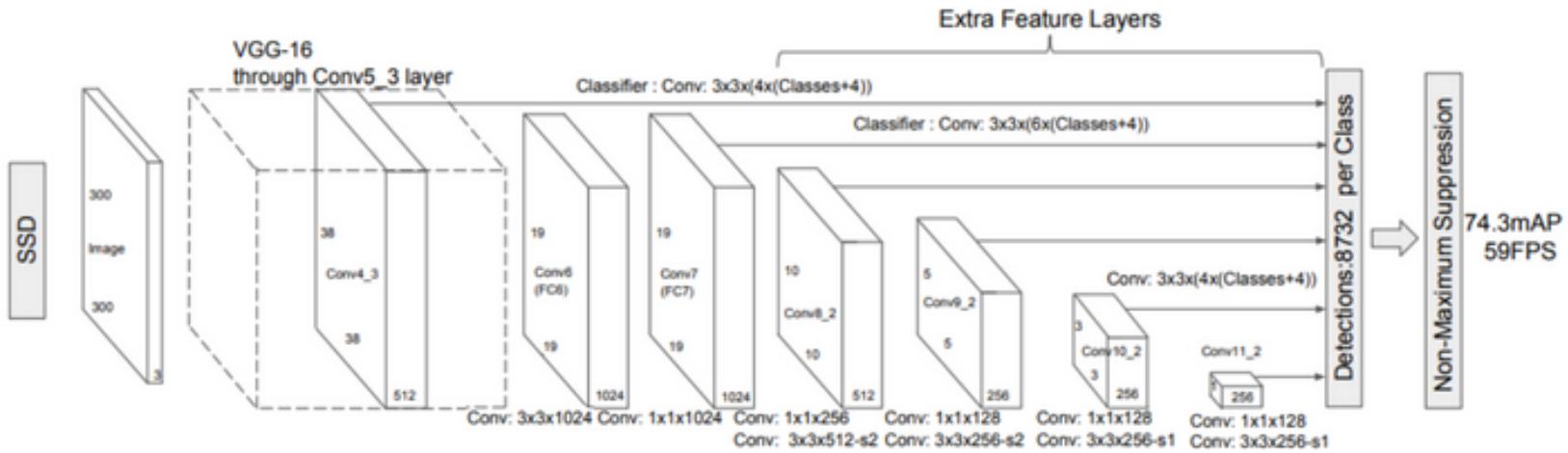


- The main innovation of Faster R-CNN is the application of the Convolutional Network for Region Proposal Network (RPN),
- The RPN process begins with passing the entire image through the

convolutional layers.

- By using RPN instead of selective search, we create a fully learning network that can also share parameters between the convolutional layers in the RPN and the Fast R-CNN detector.
- In Faster R-CNN, the ROI Pooling layer takes the region suggestions not from the selective search results but from the RPN process

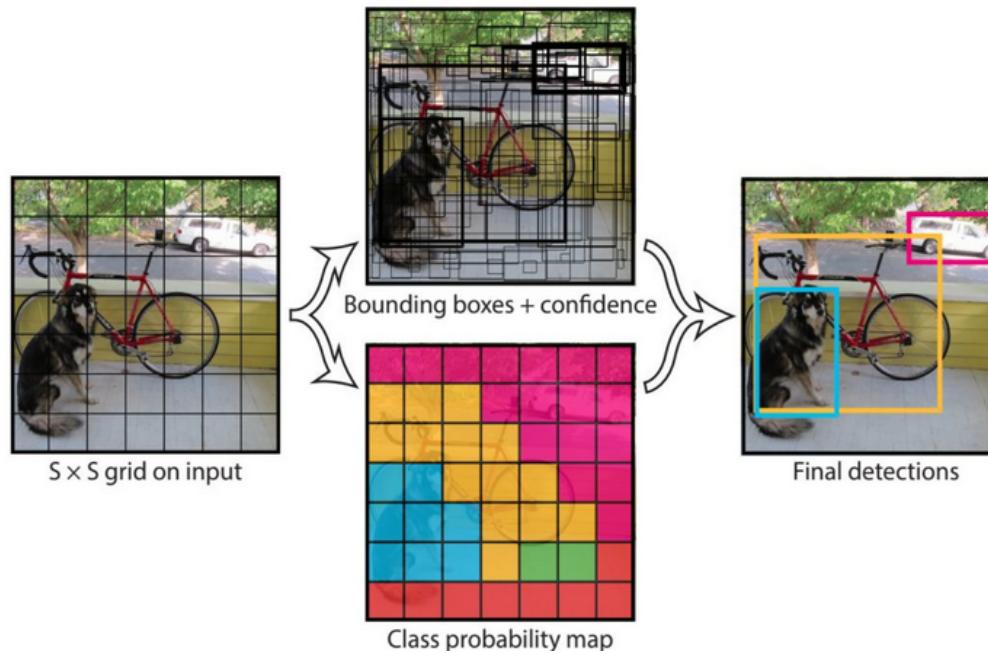
Single Shot Multibox Detector SSD 2016



- SSD is an object detection method based on the use of a single deep network,
 - At the time of prediction, the network generates results for the presence of each category in the default areas
 - The network combines predictions from multiple feature maps with different resolutions,
- SSD has two backbone components and SSD layers - Extra Feature Layers.

- The base model is usually a pre-trained image classification network,
- SSD does not use the panes moving across the image, it splits the image into a grid and each the cell in this mesh is responsible for detecting the objects
- If no object is detected, this cell becomes a background class cell.
- Tools such as the anchor box and receptive field are used, to easily detect many objects within one cell.

YOLO - You Only Look Once 2016



- <https://storage.googleapis.com/openimages/web/index.html>
- It uses features trained by deep neural networks in the Darknet framework.
- It is a fully convolutional network (FCN) - it is indifferent to the size of the input image.

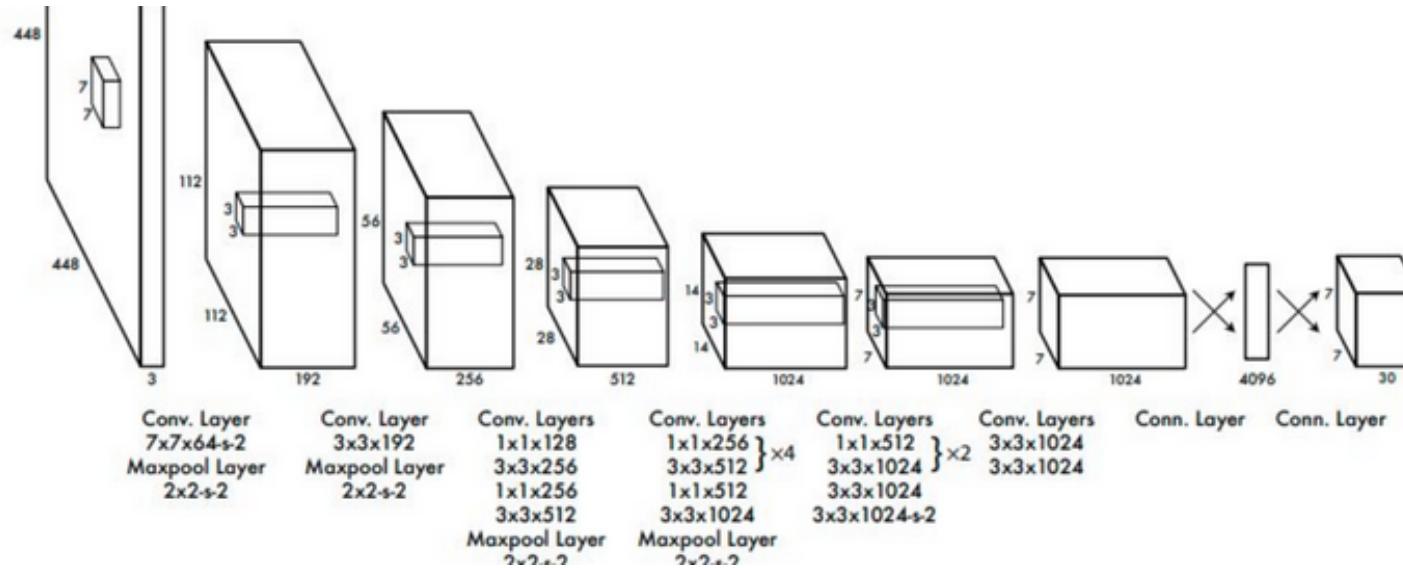
YOLO network performance

- The image is split into cells that form the $S \times S$ grid.
- The input to the YOLO network is the image, and the output is the coordinates x, y of the center of the area containing the object - the bounding box (its bounding box) in this image, its width, height, and the probability for the class detected,
- If the center of an object falls within a grid cell, that cell becomes responsible for detecting that object.
- Each cell detects the B envelope and their confidence scores.
- Confidence intervals define the uncertainty of the detection of the object and the envelope,
- Once you have all possible boundaries, get rid of those that have a low probability of owning an object.

- For this, the non-maximum suppression algorithm is used to calculate the Jaccard index:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

YOLOv1

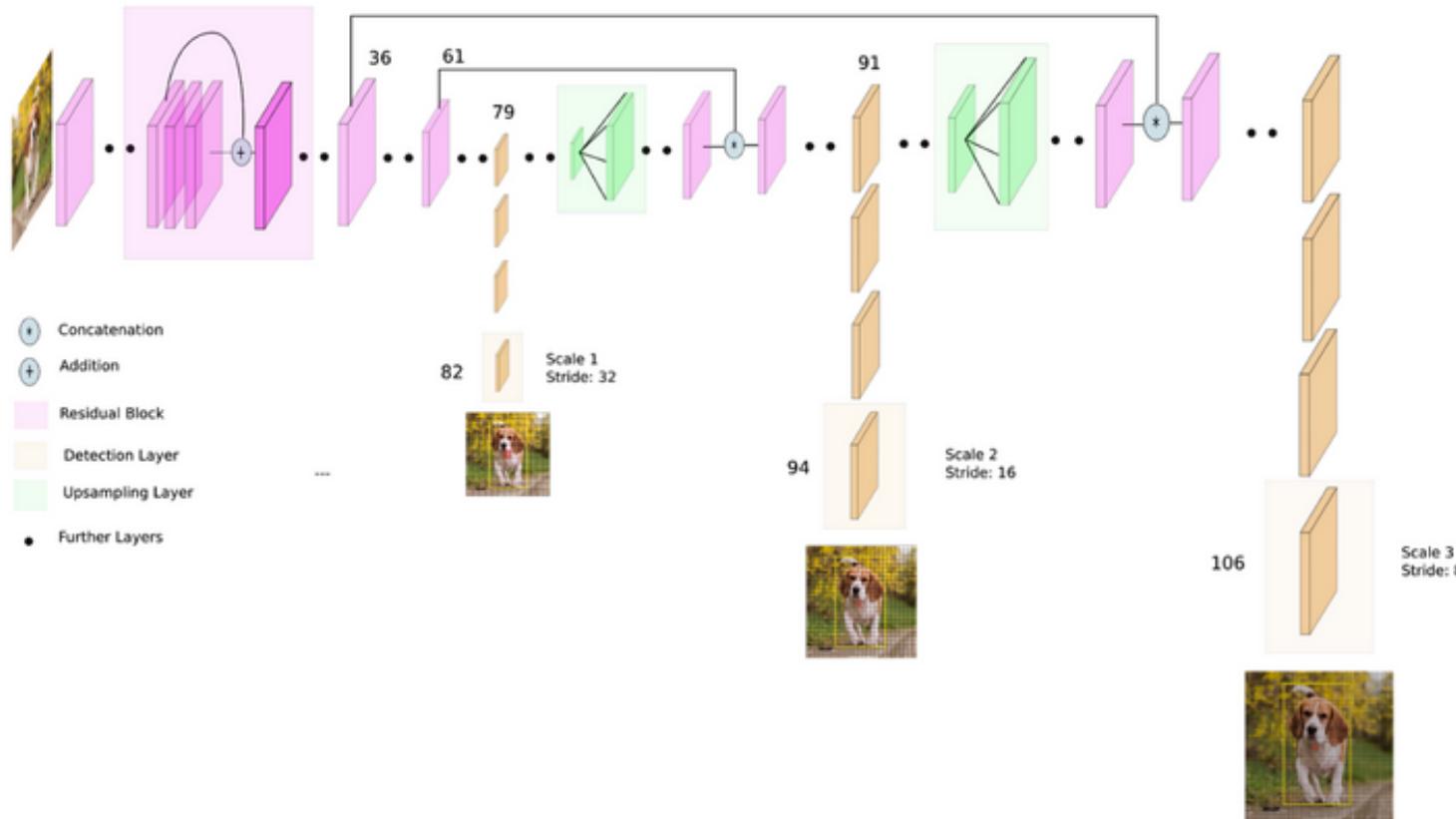


- The network has 24 convolution layers and 2 fully interconnected layers, 2 reducing layers (max-pooling).
- It is a convolutional network that simultaneously predicts multiple areas and class probabilities for those areas.
- YOLOv1 produces poorer results when the image contains many small objects, such as a flock of birds

YOLOv2 - YOLO9000 2016

- In the second version of the algorithm, YOLO detection is based on the also improved, new *Darknet19* model,
- Has fewer convolution layers and 5 reducing layers and a softmax layer,
- Envelope base anchors have been introduced: in YOLOv2 the envelopes have been introduced, as e.g. in Faster R-CNN - the number of envelopes per image increased from 98 to over a thousand,
- The detection of small objects has been improved - it creates detections on the map of features with dimensions of 13×13 ,
- Training with images at different scales.

YOLOv3 - 2018



- YOLOv3 is based on the improved *Darknet53* architecture - it has 53 convolutional layers,
- Predictions at different scales - the third version network predicts envelopes at three different scales of the image,

Machine Vision

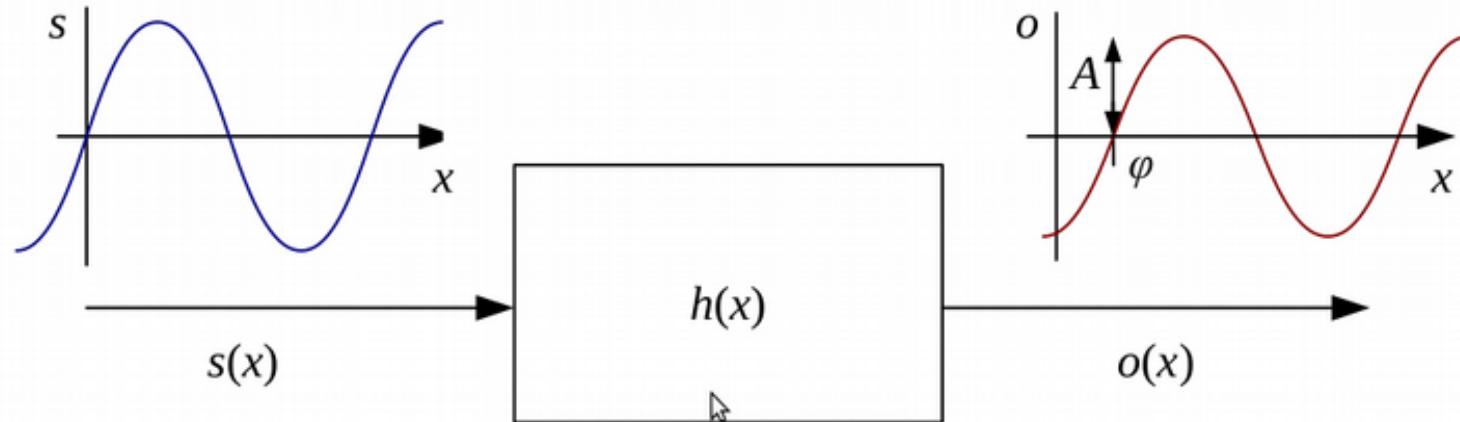
Frequency methods in Image Processing - lecture 9

Adam Szmigielski

aszmigie@pjwstk.edu.pl

materials: *ftp(public) : //aszmigie/WMAEnglish*

The Fourier transform (SYC course repetition)



- The Fourier Transform as the response of a filter $h(x)$ to an input sinusoid $s(x) = e^{j\omega x}$ yielding an output sinusoid $o(x) = h(x) \star s(x) = Ae^{j\omega x + \phi}$.
- If we convolve the sinusoidal signal $s(x)$ with a filter whose impulse response is $h(x)$, we get another sinusoid of the same frequency but different magnitude A and phase ϕ_0 ,

$$o(x) = h(x) \star s(x) = A \sin(\omega x + \phi_0) = Ae^{j\omega x + \phi_0}$$

Fourier transform - definition

The Fourier transform is simply a tabulation of the magnitude and phase response at each frequency,

$$H(\omega) = \mathcal{F}(h(x)) = Ae^{j\phi}$$

i.e., it is the response to a complex sinusoid of frequency ω passed through the filter $h(x)$. The Fourier transform pair is also often written as

$$h(x) \leftrightarrow^{\mathcal{F}} H(\omega)$$

Fourier transform exist both in the continuous domain,

$$H(\omega) = \int_{-\infty}^{\infty} h(x) \cdot e^{-j\omega x} dx$$

and in the discrete domain.

Discrete Fourier Transform (DFT) and Fast Fourier Transform (FFT)

The discrete form of the Fourier transform is known as the *Discrete Fourier Transform (DFT)*:

$$H(k) = \sum_{x=0}^{N-1} h(x)e^{-j\frac{2\pi k x}{N}}$$

where N is the length of the signal or region of analysis. These formulas apply both to filters, such as $h(x)$, and to signals or images, such as $s(x)$ or $g(x)$.

- Formula can be evaluated for any value of k , it only makes sense for values in the range $k \in < -\frac{N}{2}, \frac{N}{2} >$. This is because larger values of k alias with lower frequencies,
- DFT takes $O(N^2)$ operations to evaluate. Fortunately, there exists a faster algorithm called the Fast Fourier Transform (FFT), which requires only $O(N \log_2 N)$ - it involves a series of $\log_2 N$ stages.

Two-dimensional Fourier transforms

Instead of just specifying a horizontal or vertical frequency ω_x or ω_y , we can create an oriented sinusoid of frequency $\{\omega_x, \omega_y\}$

$$s(x, y) = \sin(\omega_x x + \omega_y y).$$

The corresponding two-dimensional Fourier transforms are then

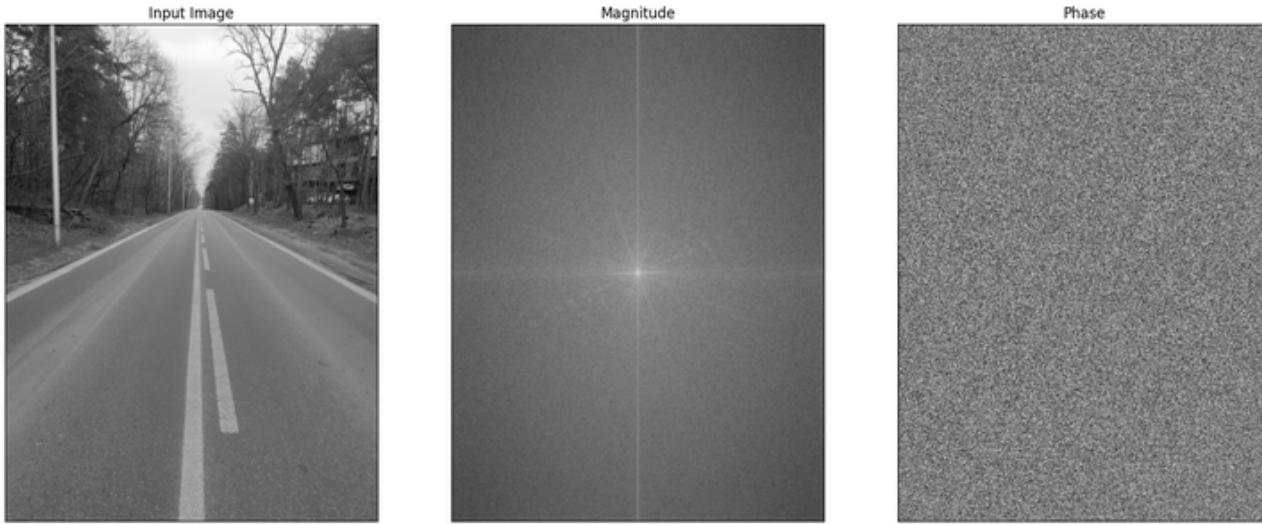
$$H(\omega_x, \omega_y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(x, y) \cdot e^{-j(\omega_x x + \omega_y y)} dx dy$$

and in the discrete domain,

$$H(k_x, k_y) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} h(x, y) e^{-j2\pi \frac{k_x x + k_y y}{MN}}$$

where M and N are the width and height of the image.

Image transformation - example



```
img = cv.imread('droga4.jpg',0)
f = np.fft.fft2(img)
fshift = np.fft.fftshift(f)
faza = np.arctan(np.abs(f).imag/f.real)
magnitude_spectrum = 20*np.log(np.abs(fshift))

plt.subplot(131),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(132),plt.imshow(magnitude_spectrum, cmap = 'gray')
plt.title('Magnitude'), plt.xticks([]), plt.yticks([])
plt.subplot(133),plt.imshow(faza, cmap = 'gray')
plt.title('Phase'), plt.xticks([]), plt.yticks([])
plt.show()
```

Image transformation - example

$$H(k_x, k_y) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} h(x, y) e^{-j2\pi \frac{k_x x + k_y y}{MN}}$$

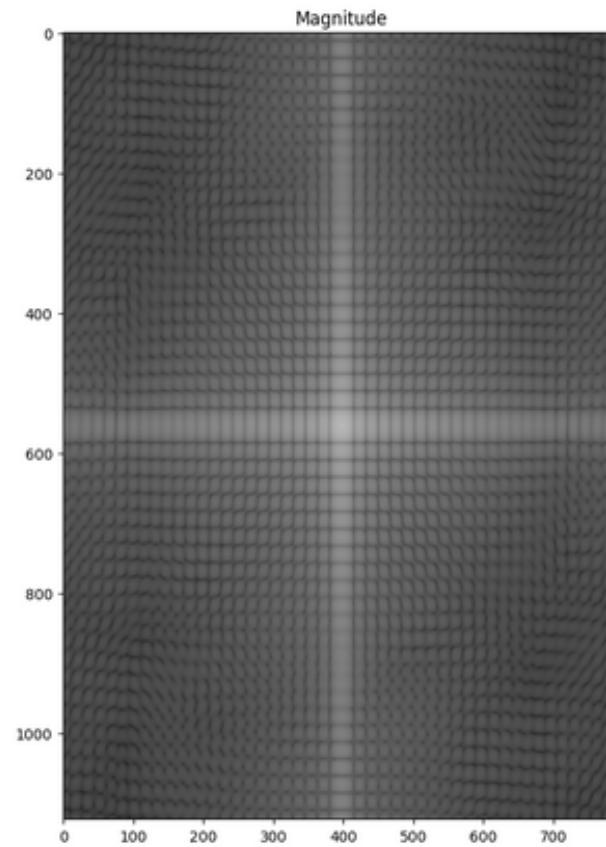
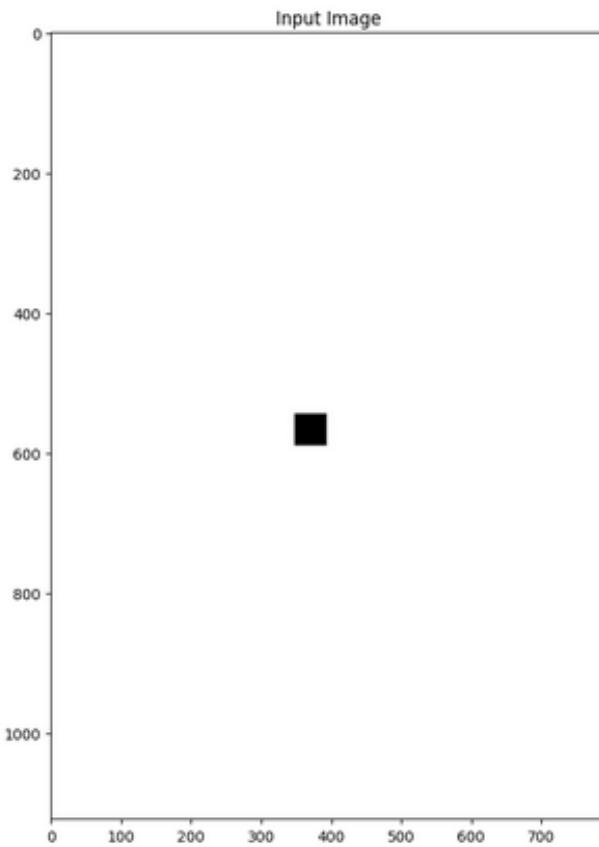
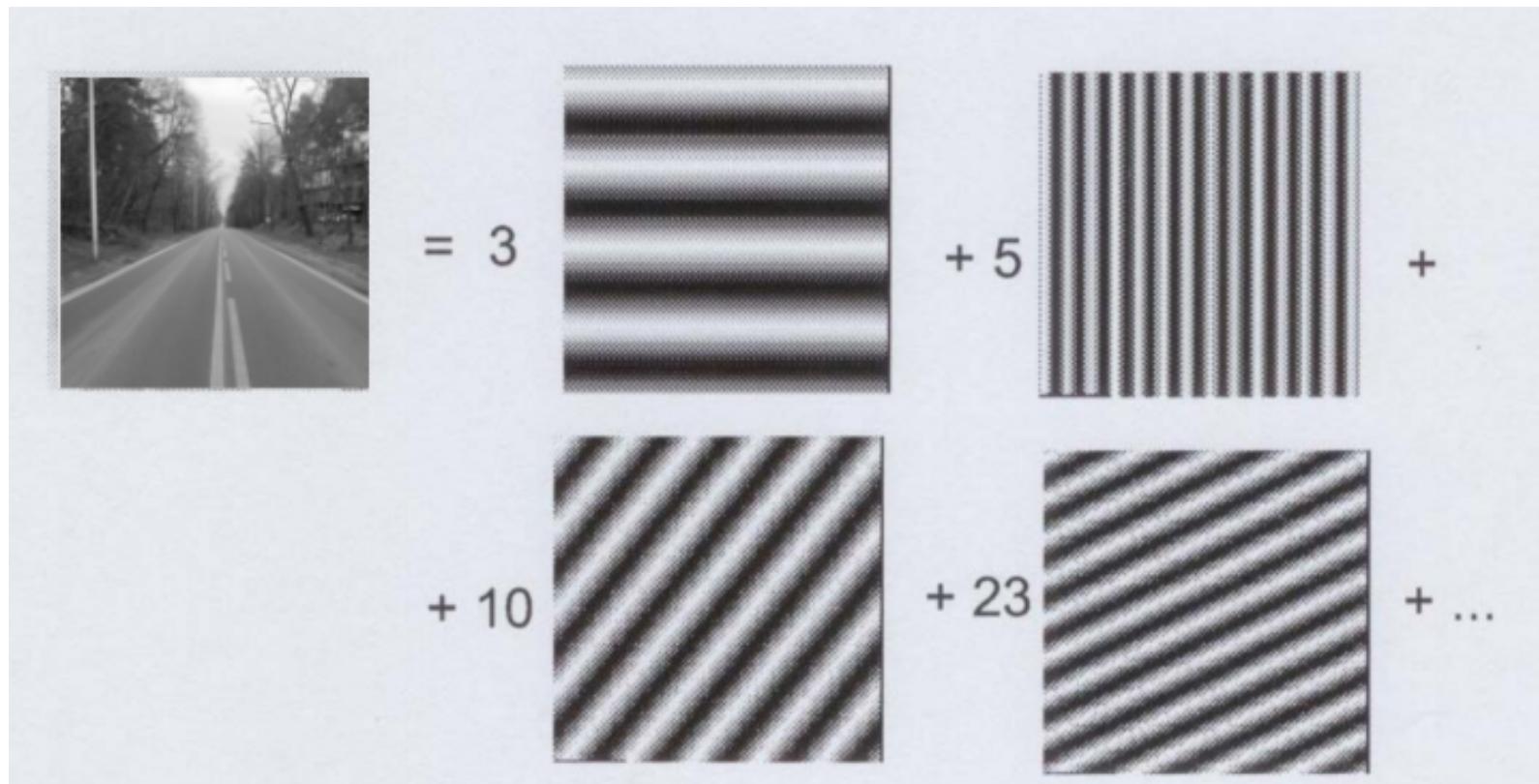
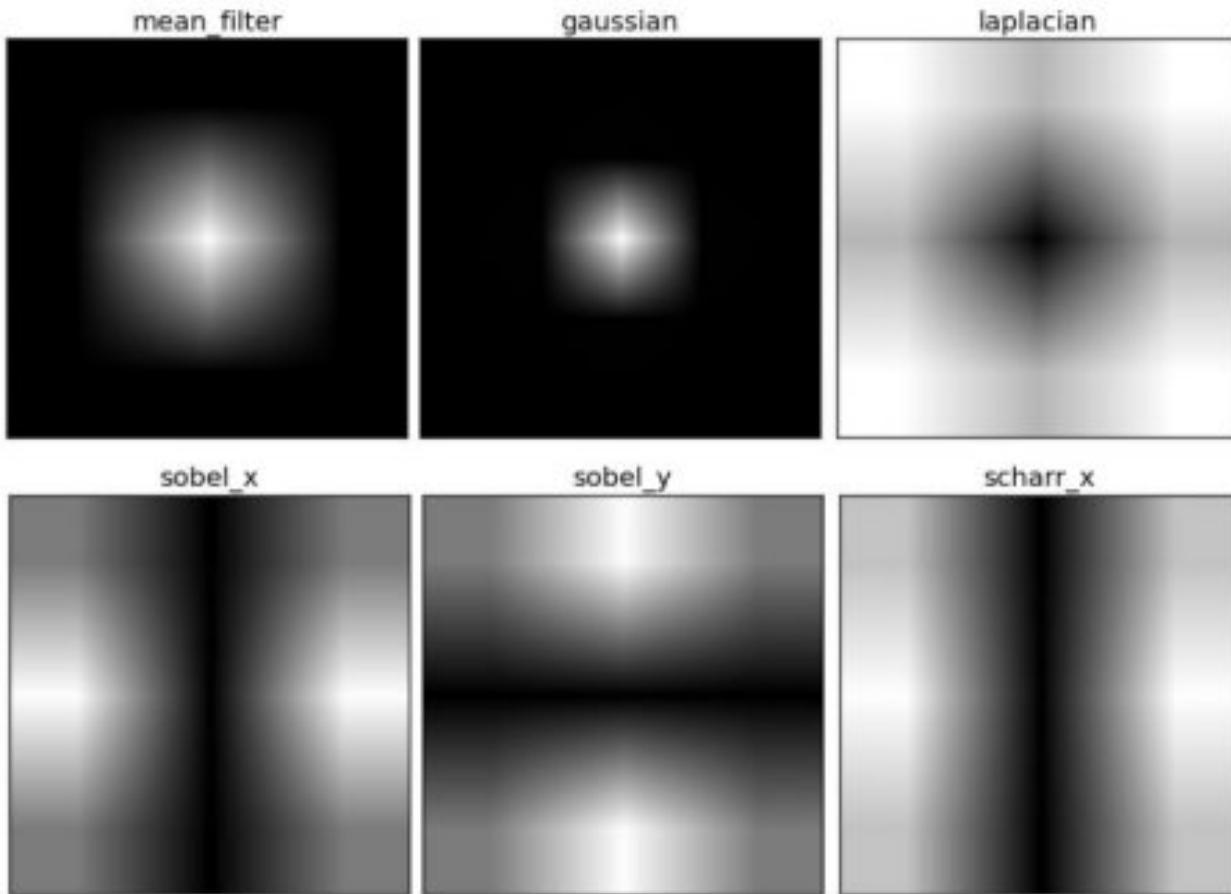


Image decomposition

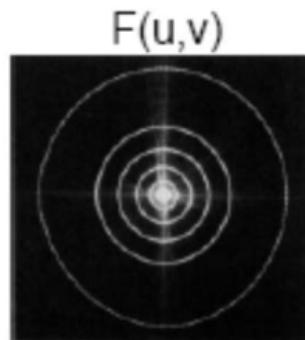


Filter spectrum



https://docs.opencv.org/master/de/dbc/tutorial_py_fourier_transform.html

Fourier filtering

 $f(x,y)$  $F(u,v)$

spatial domain

frequency domain

$$f(x,y) \longrightarrow F(u,v)$$

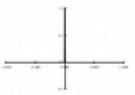
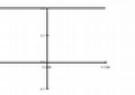
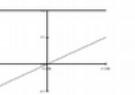
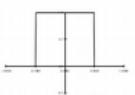
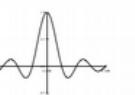
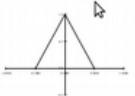
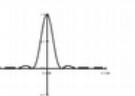
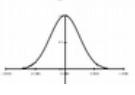
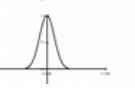
 $g(x,y)$  $H(u,v)$

$$G(u,v) = F(u,v) \cdot H(u,v)$$

filter

$$g(x,y) \longleftarrow G(u,v)$$

Fourier transform pairs

Name	Signal		Transform
impulse		$\delta(x)$	\Leftrightarrow 1 
shifted impulse		$\delta(x - u)$	$\Leftrightarrow e^{-j\omega u}$ 
box filter		$\text{box}(x/a)$	$\Leftrightarrow \text{asinc}(a\omega)$ 
tent		$\text{tent}(x/a)$	$\Leftrightarrow \text{asinc}^2(a\omega)$ 
Gaussian		$G(x; \sigma)$	$\Leftrightarrow \frac{\sqrt{2\pi}}{\sigma} G(\omega; \sigma^{-1})$ 

- *Impulse*: The impulse response has a constant (all frequency) transform.
- *Shifted impulse*: The shifted impulse has unit magnitude and linear phase.
- *Box filter*: The *box* (moving average) filter

$$\text{box}(x) = \begin{cases} 1 & \text{if } |x| \leq 1 \\ 0 & \text{else} \end{cases}$$

has a *sinc* Fourier transform,

$$\text{sinc}(\omega) = \frac{\sin(\omega)}{\omega}$$

which has an infinite number of side lobes. Conversely, the sinc filter is an ideal low- pass filter. For a non-unit box, the width of the box a and the spacing of the zero crossings in the $\text{sinc } \frac{1}{a}$ are inversely proportional.

- *Tent*: The piecewise linear tent function,

$$\text{tent}(x) = \max\{0, 1 - |x|\},$$

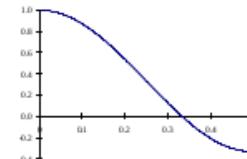
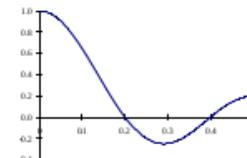
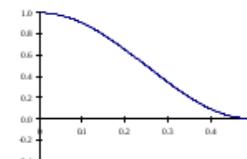
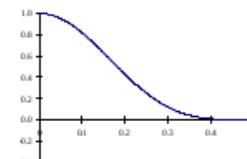
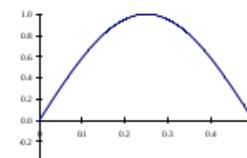
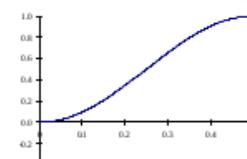
has a sinc^2 Fourier transform.

- *Gaussian*: The (unit area) Gaussian of width σ ,

$$G(x, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

has a (unit height) Gaussian of width σ^{-1} as its Fourier transform.

Fourier transforms of the separable kernels

Name	Kernel	Transform	Plot
box-3	$\frac{1}{3} [1 \ 1 \ 1]$	$\frac{1}{3}(1 + 2 \cos \omega)$	
box-5	$\frac{1}{5} [1 \ 1 \ 1 \ 1 \ 1]$	$\frac{1}{5}(1 + 2 \cos \omega + 2 \cos 2\omega)$	
linear	$\frac{1}{4} [1 \ 2 \ 1]$	$\frac{1}{2}(1 + \cos \omega)$	
binomial	$\frac{1}{16} [1 \ 4 \ 6 \ 4 \ 1]$	$\frac{1}{4}(1 + \cos \omega)^2$	
Sobel	$\frac{1}{2} [-1 \ 0 \ 1]$	$\sin \omega$	
corner	$\frac{1}{2} [-1 \ 2 \ -1]$	$\frac{1}{2}(1 - \cos \omega)$	

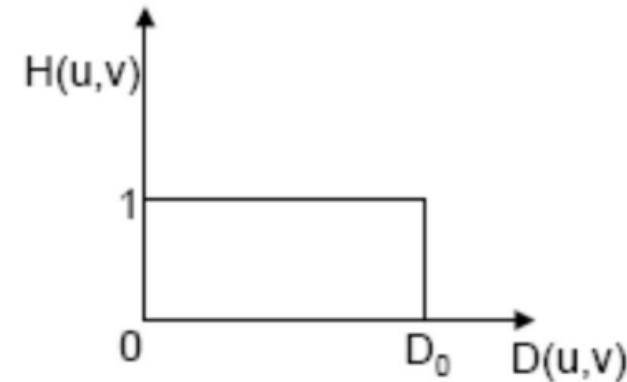
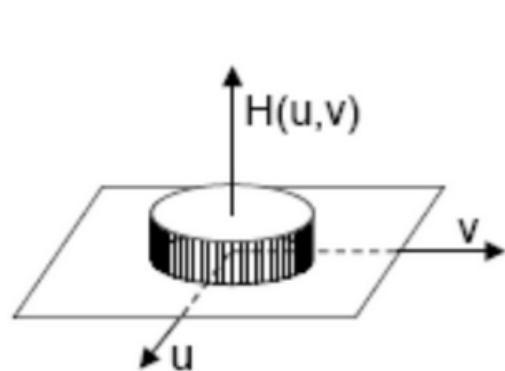
 Δ

$H(u, v)$ – Ideal Low Pass Filter

$$H(u, v) = \begin{cases} 1 & D(u, v) \leq D_0 \\ 0 & D(u, v) > D_0 \end{cases}$$

$$D(u, v) = \sqrt{u^2 + v^2}$$

D_0 = cut off frequency

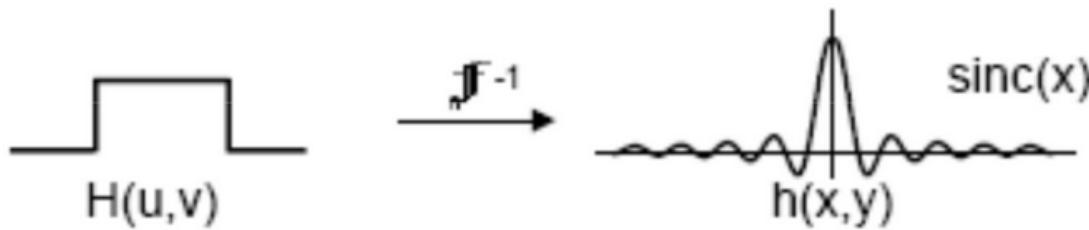


The Ringing Problem

$$G(u,v) = F(u,v) \cdot H(u,v)$$

↓
Convolution Theorem

$$g(x,y) = f(x,y) * h(x,y)$$



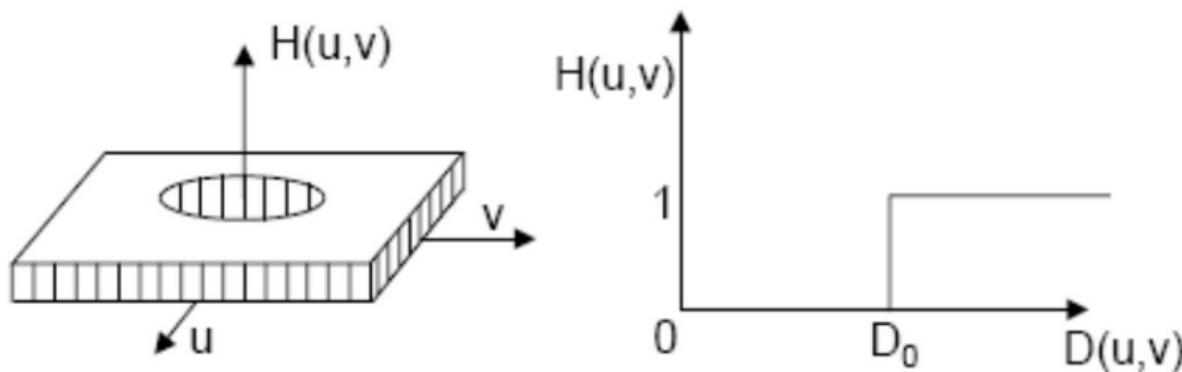
$\uparrow D_0$ → ↓ Ringing radius + blur

Image Sharpening – High Pass Filter

$$H(u,v) = \begin{cases} 0 & D(u,v) \leq D_0 \\ 1 & D(u,v) > D_0 \end{cases}$$

$$D(u,v) = \sqrt{u^2 + v^2}$$

D_0 = cut off frequency



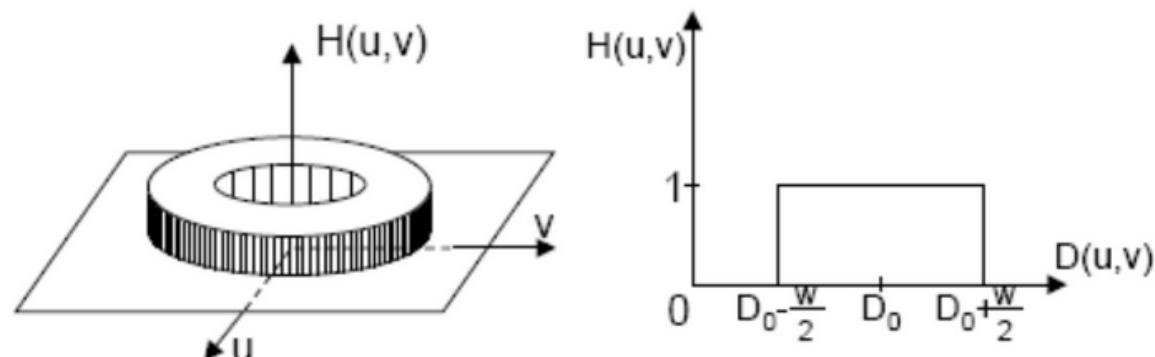
Band Pass Filtering

$$H(u,v) = \begin{cases} 0 & D(u,v) \leq D_0 - \frac{w}{2} \\ 1 & D_0 - \frac{w}{2} \leq D(u,v) \leq D_0 + \frac{w}{2} \\ 0 & D(u,v) > D_0 + \frac{w}{2} \end{cases}$$

$$D(u,v) = \sqrt{u^2 + v^2}$$

D_0 = cut off frequency

w = band width



Properties of Fourier Transform $F(\omega) = \mathcal{F}\{f(x)\}$

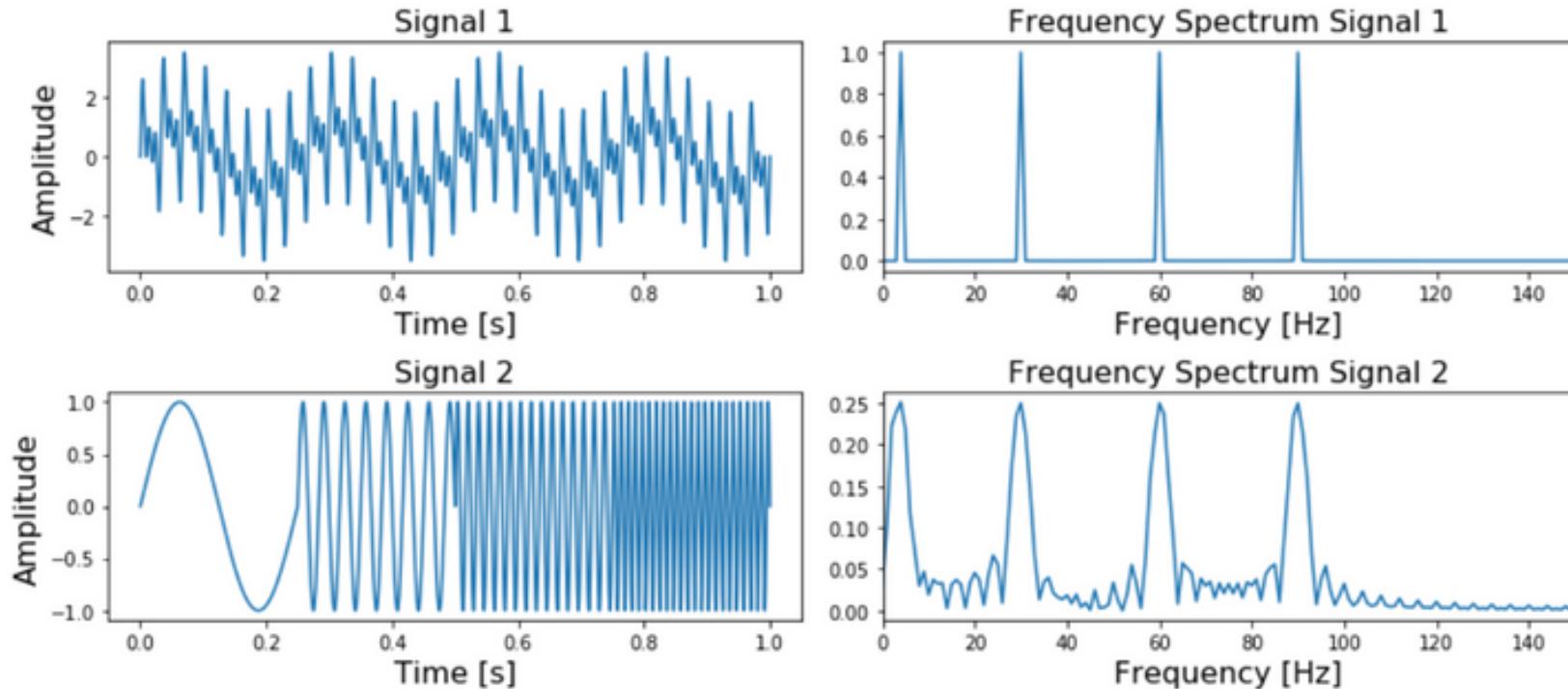
Property	Signal	Transform
superposition	$f_1(x) + f_2(x)$	$F_1(\omega) + F_2(\omega)$
shift	$f(x - x_0)$	$F(\omega)e^{-j\omega x_0}$
reversal	$f(-x)$	$F^*(\omega)$
convolution	$f(x) * h(x)$	$F(\omega)H(\omega)$
correlation	$f(x) \otimes h(x)$	$F(\omega)H^*(\omega)$
multiplication	$f(x)h(x)$	$F(\omega) * H(\omega)$
differentiation	$\hat{\Leftrightarrow} f'(x)$	$j\omega F(\omega)$
domain scaling	$f(ax)$	$1/aF(\omega/a)$
real images	$f(x) = f^*(x) \Leftrightarrow F(\omega) = F(-\omega)$	
Parseval's Theorem	$\sum_x [f(x)]^2$	$= \sum_\omega [F(\omega)]^2$

- *Superposition:* The Fourier transform of a sum of signals is the sum of their Fourier transforms (Fourier transform is a linear operator).
- *Shift:* The Fourier transform of a shifted signal is the transform of the original signal multiplied by a linear phase shift.
- *Reversal:* The Fourier transform of a reversed signal is the complex conjugate of the signal's transform.
- *Convolution:* The Fourier transform of a pair of convolved signals is the

product of their transforms.

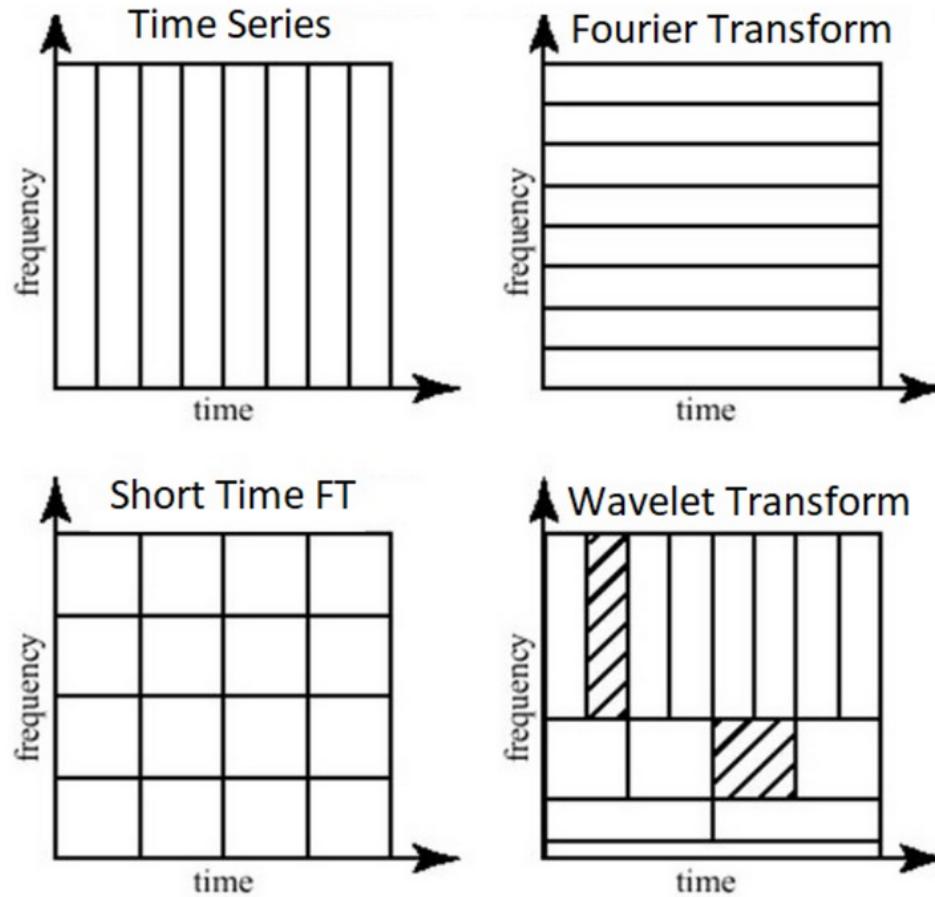
- *Correlation*: The Fourier transform of a correlation is the product of the first transform times the complex conjugate of the second one.
- *Multiplication*: The Fourier transform of the product of two signals is the convolution of their transforms.
- *Differentiation*: The Fourier transform of the derivative of a signal is that signal's transform multiplied by the frequency (differentiation linearly emphasizes (magnifies) higher frequencies)
- *Domain scaling*: The Fourier transform of a stretched signal is the equivalently compressed (and scaled) version of the original transform and vice versa.
- *Real images*: The Fourier transform of a real-valued signal is symmetric around the origin.
- *Parseval's Theorem*: The energy (sum of squared values) of a signal is the same as the energy of its Fourier transform.

Spatial and frequency domain description



- Top - signal containing four different frequencies (4, 30, 60 and 90Hz) which are present at all times,
- Bottom - the same four frequencies, only the first one is present in the first quarter of the signal.

Short-Time Fourier Transform

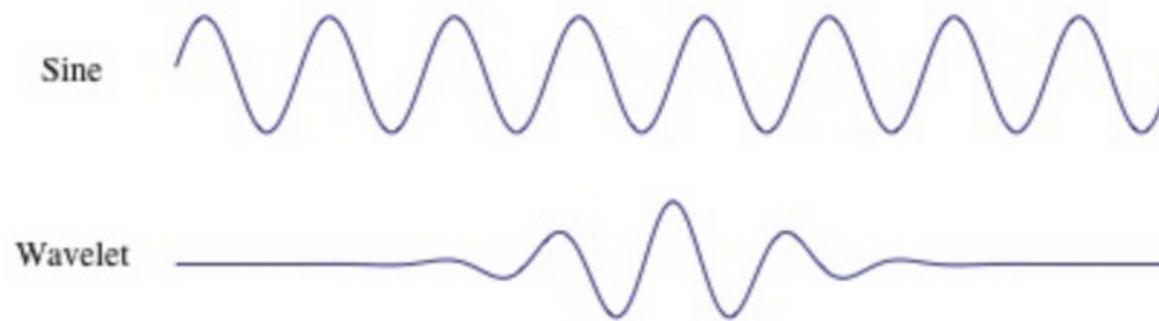


- Original signal is splitted into several parts of equal length by using a sliding window before applying the Fourier Transform.

Limits of the Fourier Transform - uncertainty principle

- The smaller we make the size of the window the more we will know about where a frequency has occurred in the signal, but less about the frequency value itself.
- The larger we make the size of the window the more we will know about the frequency value and less about the time.

Wavelet Transform



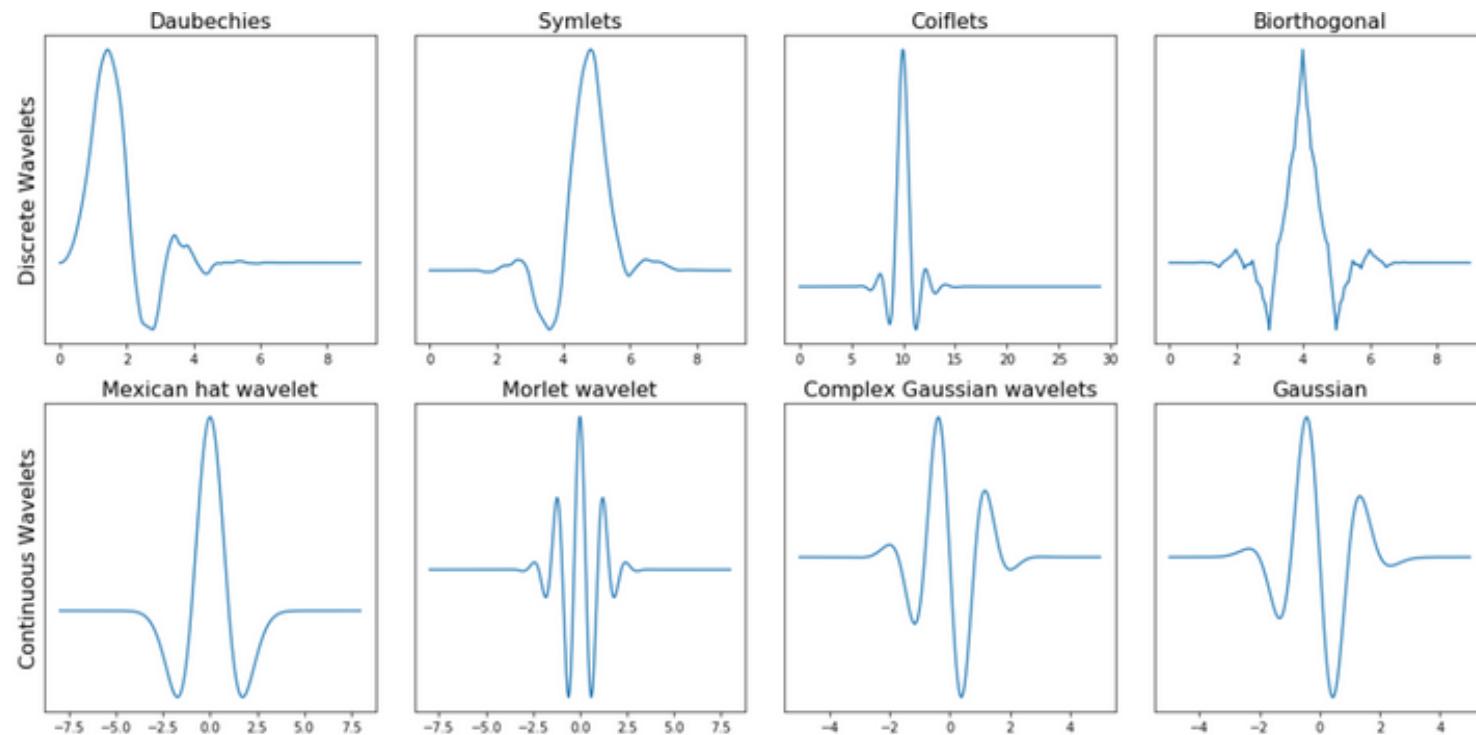
- **Fourier Transform** uses a series of sine-waves with different frequencies to analyze a signal.
- **Wavelet Transform** uses a series of functions called wavelets, each with a different scale.
- The sine-wave is infinitely long and the Wavelet is localized in time.

Wavelet procedure



- Since the Wavelet is localized, we can multiply our signal with the wavelet at different locations,
- We start with the beginning of our signal and move the wavelet towards the end of the signal (convolution),
- we can scale it such that it becomes larger and repeat the process.

Families of wavelets



A wavelet must have:

- **Finite energy** - inner product between the wavelet and the signal always exists.
- **zero mean** - inverse of the wavelet transform can also be calculated.

Continuous Wavelet Transform

Continuous Wavelet Transform is described by the following equation:

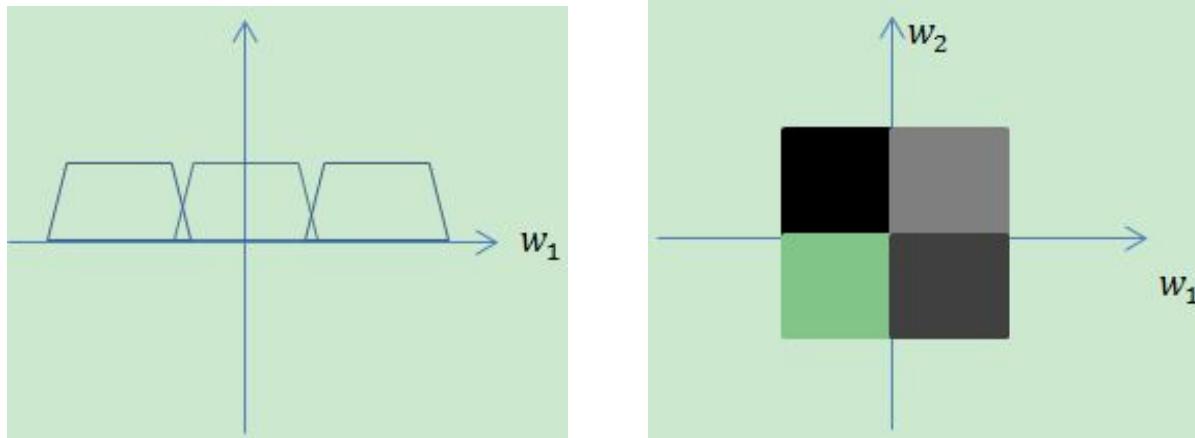
$$X_w(\tau, s) = \frac{1}{\sqrt{|s|}} \int_{-\infty}^{\infty} x(t)\psi\left(\frac{t-\tau}{s}\right)dt$$

where $\psi(t)$ is the continuous mother wavelet which gets scaled by a factor of s and translated by a factor of τ .

A wavelet must have:

- **Finite energy** - inner product between the wavelet and the signal always exists.
- **zero mean** - inverse of the wavelet transform can also be calculated.

Discrete Wavelet Transform as filter-bank.



- Discrete Wavelet Transform is always implemented as a filter-bank,
- This means that it is implemented as a cascade of high-pass and low-pass filters.

Pyramids and wavelets

- Can be used to reduce the size of an image (to speed up the execution of an algorithm or to save on storage space or transmission time),
- Sometimes we do not even know what the appropriate resolution for the image should be.
- Since we do not know the scale at which the object will appear, we need to generate a whole pyramid of differently sized images and scan each one for possible object.

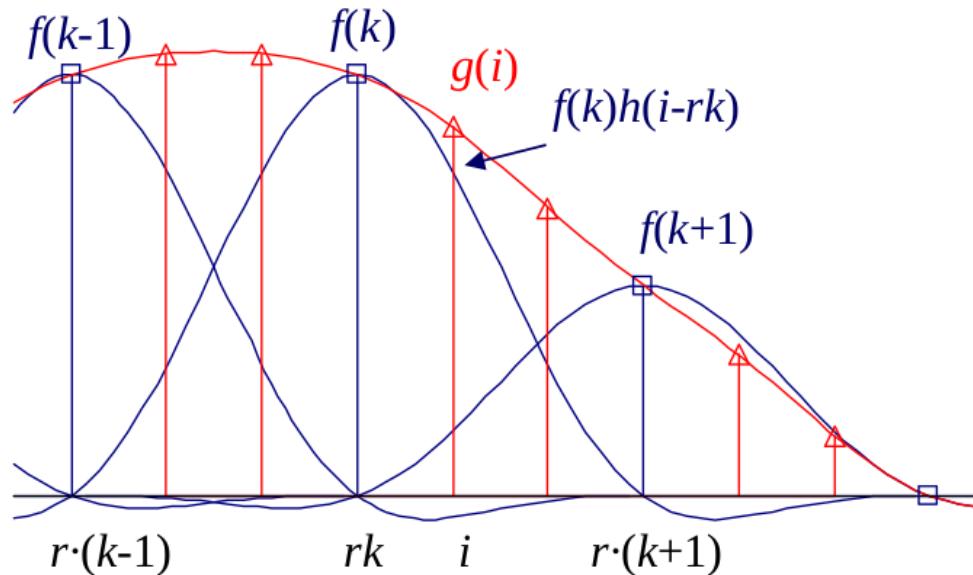
Interpolation

In order to *interpolate (or upsample)* an image to a higher resolution, we need to select some interpolation kernel with which to convolve the image,

$$g(i, j) = \sum_{k,l} f(k, l)h(i - rk, j - rl)$$

- This formula is related to the *discrete convolution* except that we replace k and l in $h()$ with rk and rl , where r is the *upsampling rate*

Interpolation - example



- Signal interpolation: $g(i) = \sum_k f(k)h(i - rk)$ - weighted summation of input values, where $f(k)$ are samples and $h(i - rk)$ is kernel.
- Interpolation can be viewed as the superposition of sample weighted interpolation kernels, one centered at each input sample k .
- What kinds of kernel make good interpolators?

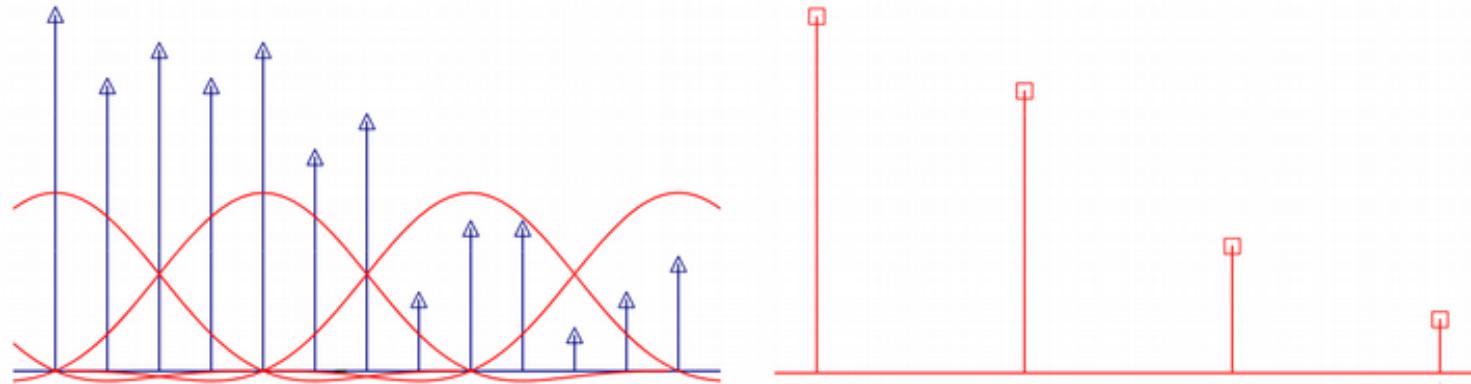
Decimation

To perform decimation, we first convolve the image with a low-pass filter (to avoid aliasing) and then keep every r^{th} sample.

$$g(i, j) = \sum_{k,l} f(k, l)h(ri - k, rj - l)$$

- In practice, we usually only evaluate the convolution at every r^{th} sample,
- While interpolation can be used to increase the resolution of an image, decimation (downsampling) is required to reduce the resolution.

Decimation - example

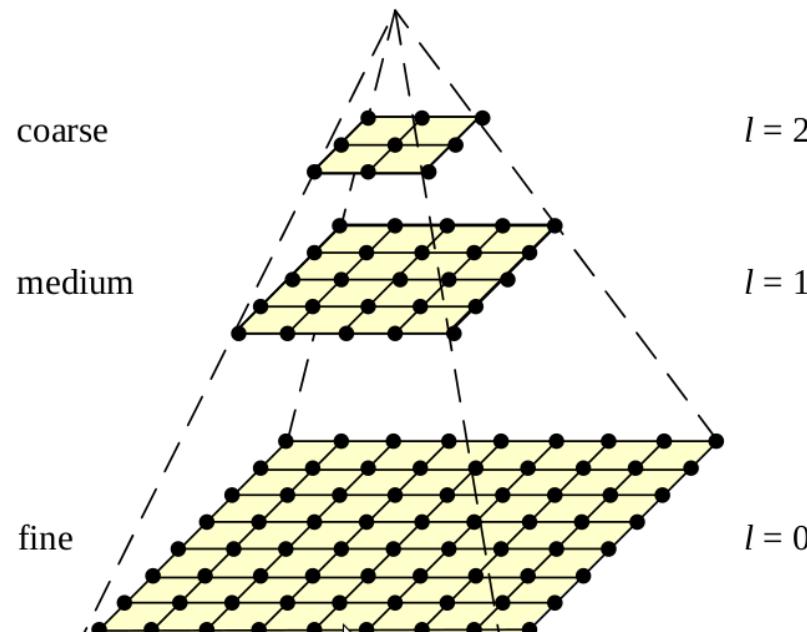


Signal decimation: (a) the original samples are (b) convolved with a
low-pass filter before being downsampled.

Multi-resolution representations

- Pyramids can be used to accelerate coarse-to-fine search algorithms, to look for objects or patterns at different scales, and to perform multi-resolution blending operations.
- Because adjacent levels in the pyramid are related by a sampling rate $r = 2$, this kind of pyramid is known as an octave pyramid

Image pyramid

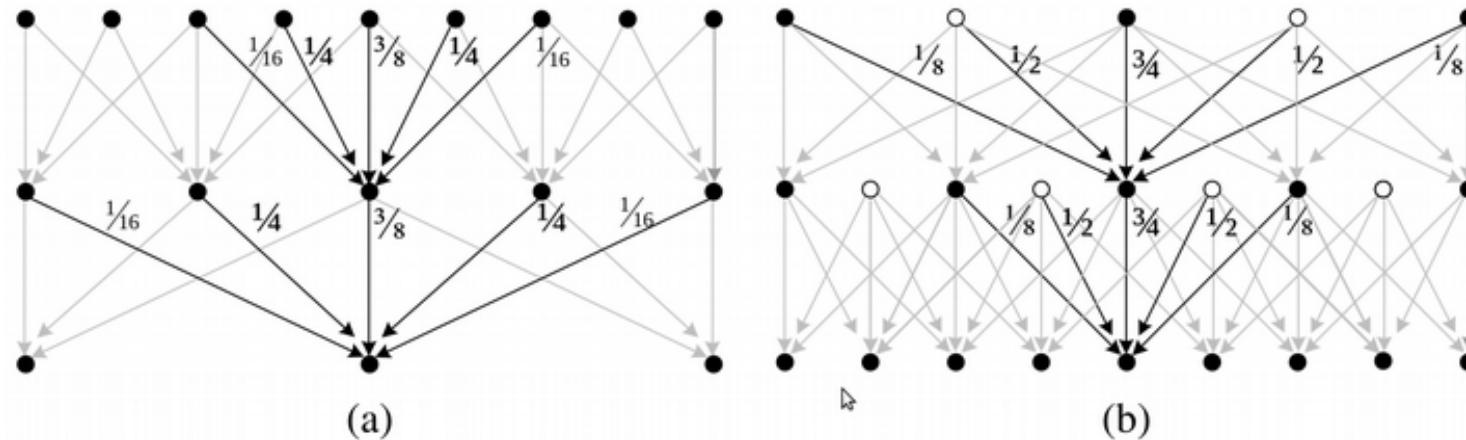


Each level has half the resolution (width and height)

- five-tap kernel of the form $|c|b|a|b|c|$, with $b = 1/4$ and $c = 1/4-a/2$, which results in the familiar binomial kernel,

$$\frac{1}{16} |1|4|6|4|1|$$

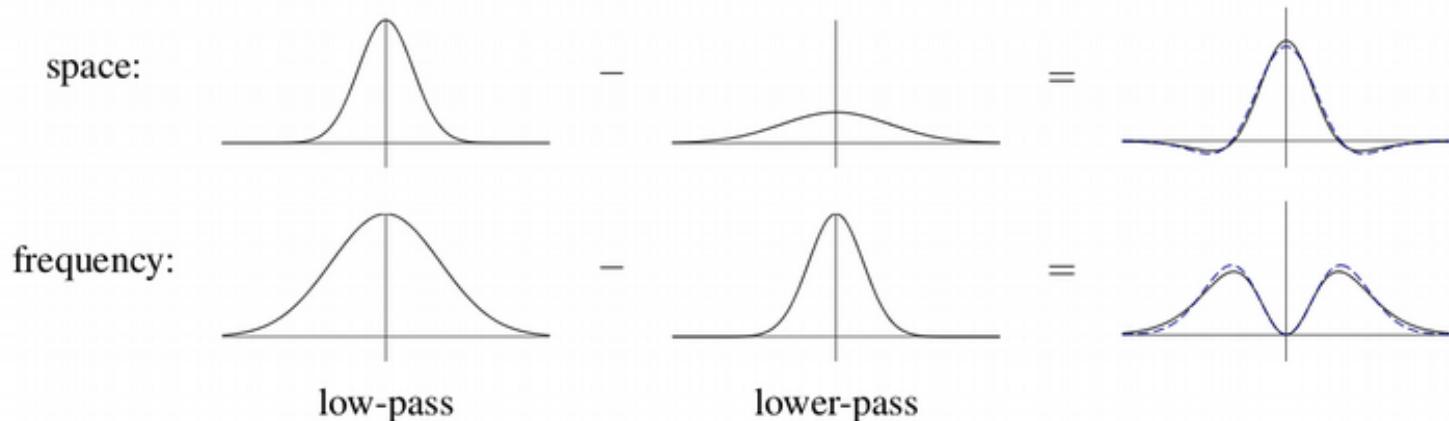
Gaussian pyramid



The Gaussian pyramid as a signal processing diagram:

- (a) The analysis
 - (b) re-synthesis stages are shown as using similar computations. The white circles indicate zero values inserted by the $\uparrow 2$ upsampling operation.
 - The reconstruction filter coefficients are twice the analysis coefficients. The computation is shown as flowing down the page, regardless of whether we are going from coarse to fine or vice versa.

Laplacian and Gaussian Filters



- The difference of two low-pass filters results in a band-pass filter. The dashed blue lines show the close fit to a half-octave Laplacian of Gaussian.
- Differences of Gaussian and Laplacians of Gaussian look in both space and frequency.
- The term Laplacian is a bit of a misnomer, since their band-pass images are really differences of (approximate) Gaussians

$$DoG\{I; \sigma_1, \sigma_2\} = G_{\sigma_1} * I - G_{\sigma_2} * I = (G_{\sigma_2} - G_{\sigma_1}) * I.$$

A Laplacian of Gaussian is actually its second derivative,

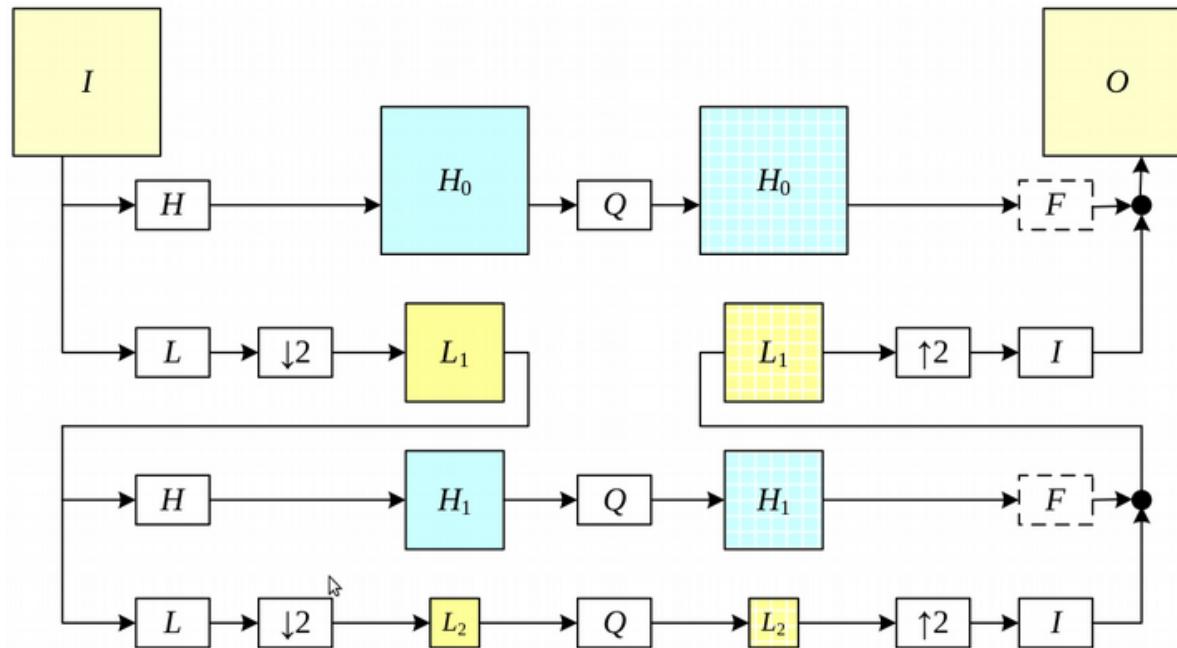
$$LoG\{I; \sigma\} = \nabla^2(G_\sigma * I) = (\nabla^2 G_\sigma) * I$$

where

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

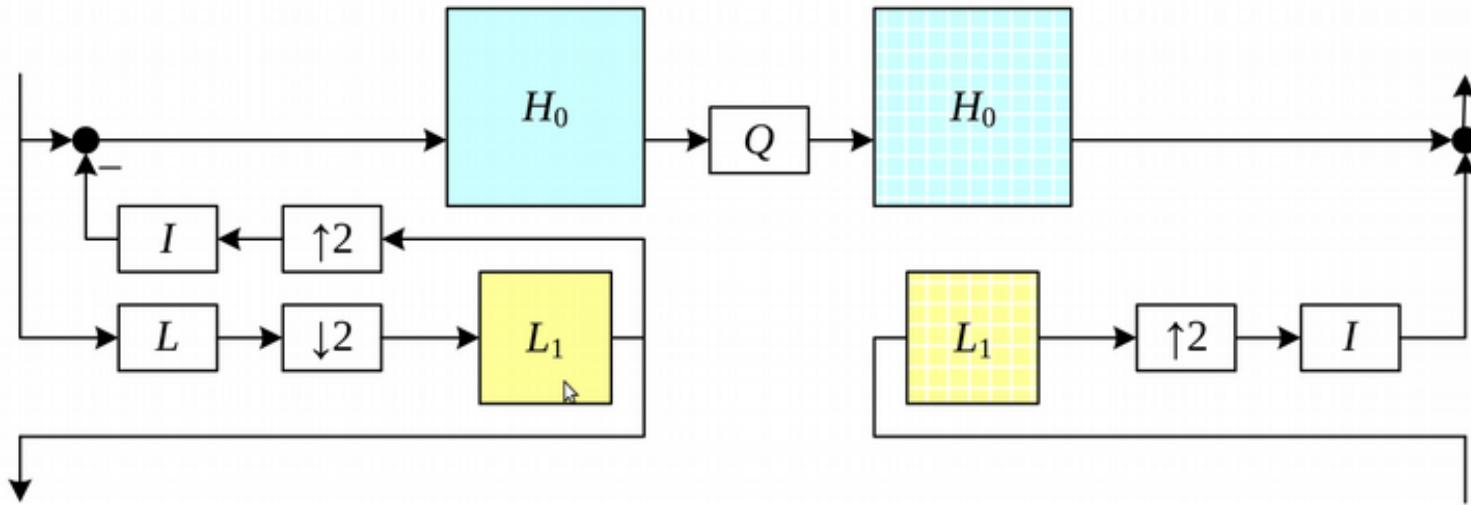
is the Laplacian (operator) of a function.

The Laplacian pyramid - the conceptual flow



- The conceptual flow of images through processing stages: images are high-pass and low-pass filtered, and the low-pass filtered images are processed in the next stage of the pyramid.
- During reconstruction, the interpolated image and the (optionally filtered) high-pass image are added back together. The Q box indicates quantization.

The Laplacian pyramid - the computational flow

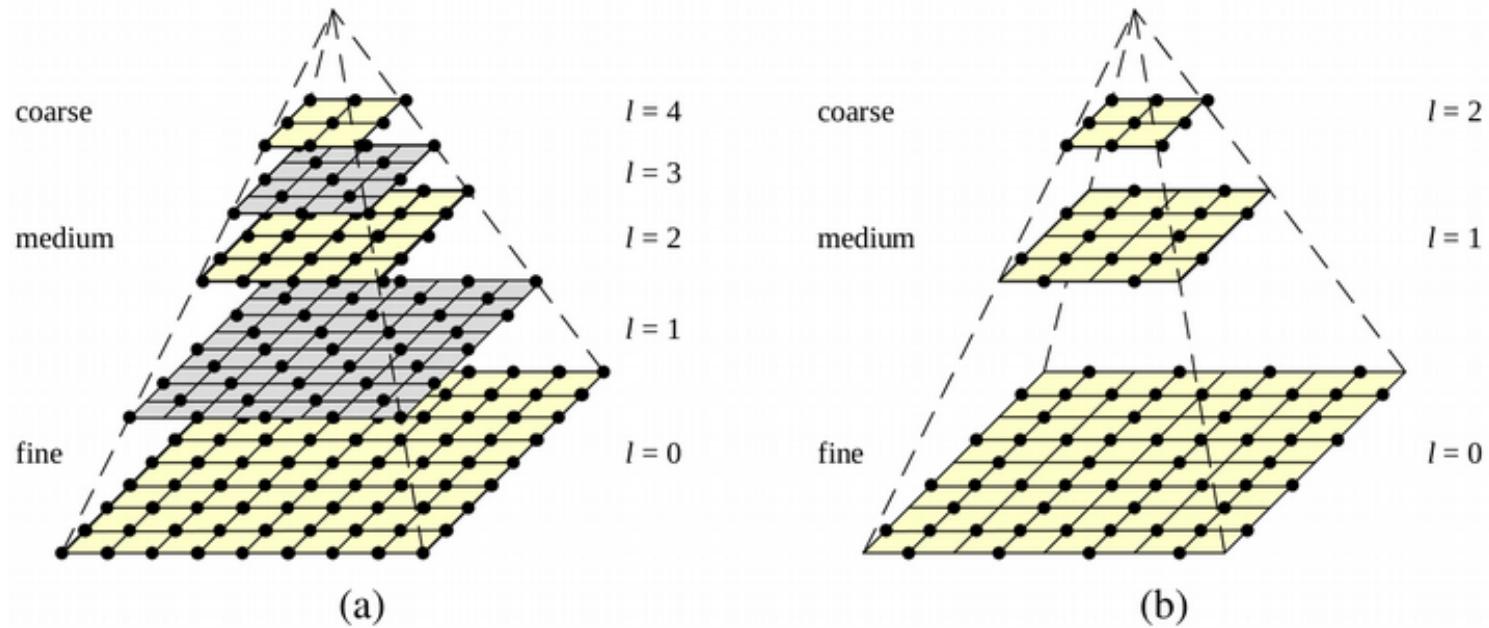


- The actual computation of the high-pass filter involves first interpolating the downsampled low-pass image and then subtracting it.
- This results in perfect reconstruction when Q is the identity.
- The high-pass (or band-pass) images are typically called Laplacian images, while the low-pass images are called Gaussian images.

Wavelets

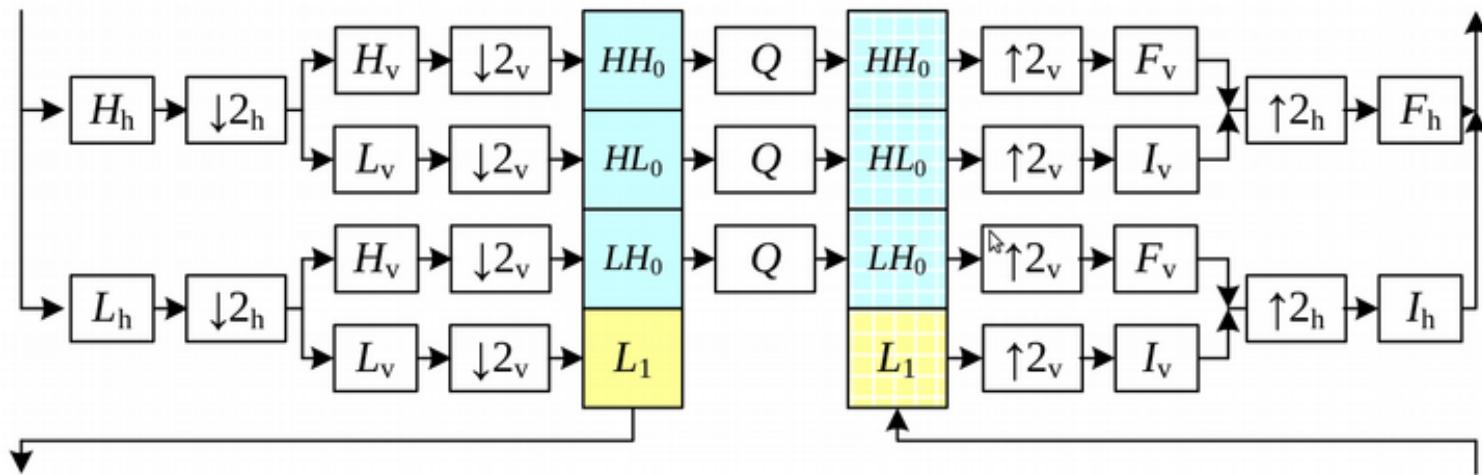
- Wavelets are filters that localize a signal in both space and frequency and are defined over a hierarchy of scales,
- Wavelets provide a smooth way to decompose a signal into frequency components without blocking and are closely related to pyramids,
- Wavelets were originally developed in the applied math and signal processing communities and were introduced to the computer vision community,
- Wavelets are widely used in the computer graphics community to perform multi-resolution geometric processing.

Multiresolution pyramids



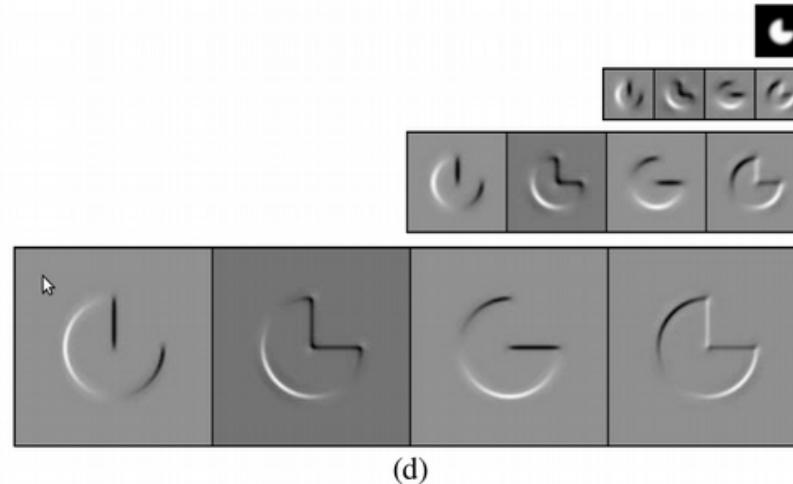
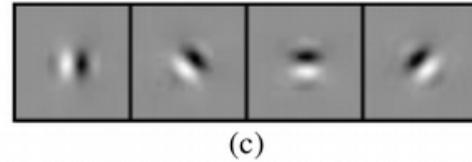
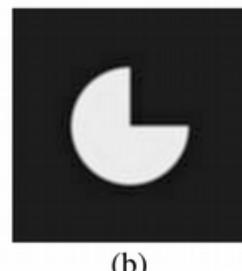
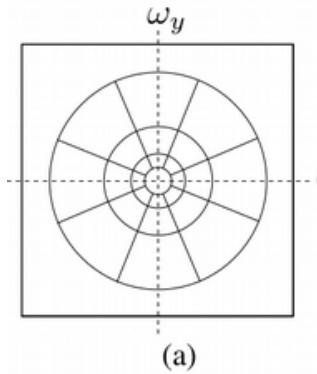
- (a) pyramid with half-octave (quincunx) sampling (odd levels are colored gray for clarity).
- (b) wavelet pyramid—each wavelet level stores $\frac{3}{4}$ of the original pixels (usually the horizontal, vertical, and mixed gradients),

Two-dimensional wavelet decomposition



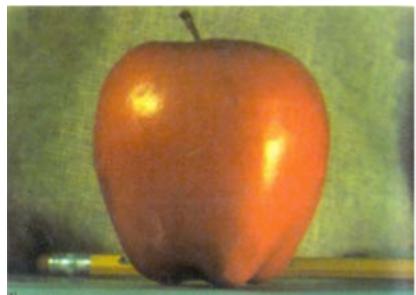
- Separable implementation, which involves first performing the wavelet transform horizontally and then vertically.
- The I and F boxes are the interpolation and filtering boxes required to re-synthesize the image from its wavelet components.

Steerable shiftable multiscale transforms - example



- (a) radial multi-scale frequency domain decomposition,
- (b) original image,
- (c) a set of four steerable filters,
- (d) the radial multi-scale wavelet decomposition.

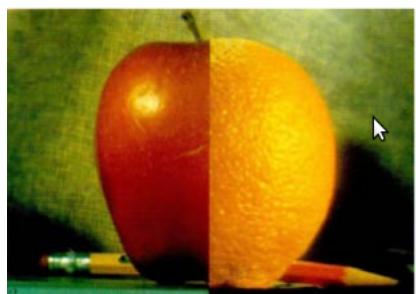
Application: Image blending



(a)



(b)



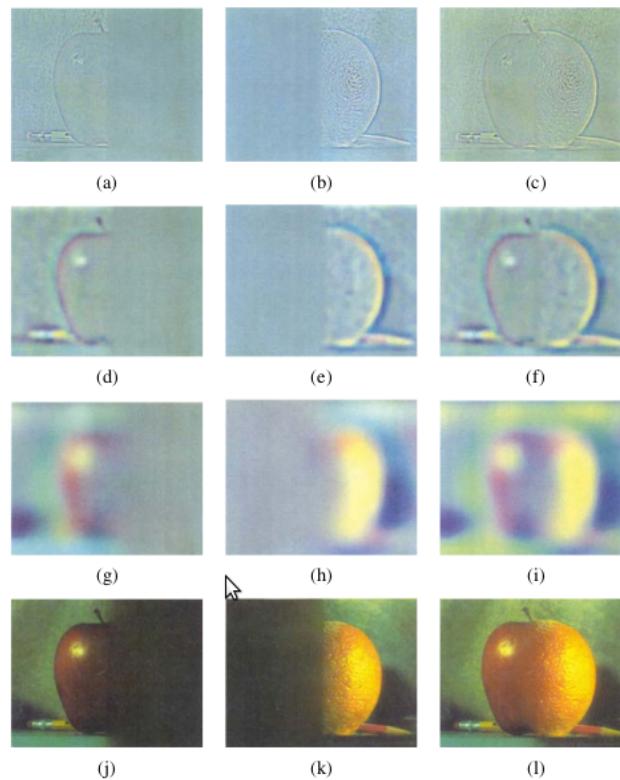
(c)



(d)

- (a) original image of apple,
- (b) original image of orange,
- (c) regular splice,
- (d) pyramid blend

Laplacian pyramid blending details



The first three rows show the high, medium, and low frequency parts of the Laplacian pyramid (taken from levels 0, 2, and 4). The left and middle columns show the original apple and orange images weighted by the smooth interpolation functions, while the right column shows the averaged contributions.

OpenCV library code - procedure

- https :
//docs.opencv.org/master/dc/dff/tutorial_py_pyramids.html*
1. Load the two images of apple and orange
 2. Find the Gaussian Pyramids for apple and orange (in this particular example, number of levels is 6)
 3. From Gaussian Pyramids, find their Laplacian Pyramids
 4. Now join the left half of apple and right half of orange in each levels of Laplacian Pyramids
 5. Finally from this joint image pyramids, reconstruct the original image.

task for labs

1. Apply the Fourier transform FFT (or DFT) to the example. Display the received phase and amplitude component,
2. Empirically check Paseval's theorem for the image and its module,
3. Check the operation of the pyramid algorithm for the example in opencv,
4. For an example photo of the sea and the boat, try using the pyramid algorithm to place the boat at sea (photomontage)

Computer Vision Spatial aspects in computer vision - lecture 10

Adam Szmigiełski

aszmigie@pjwstk.edu.pl

materials: *ftp(public) : //aszmigie/MWREnglish*

Geometric primitives - 2D points

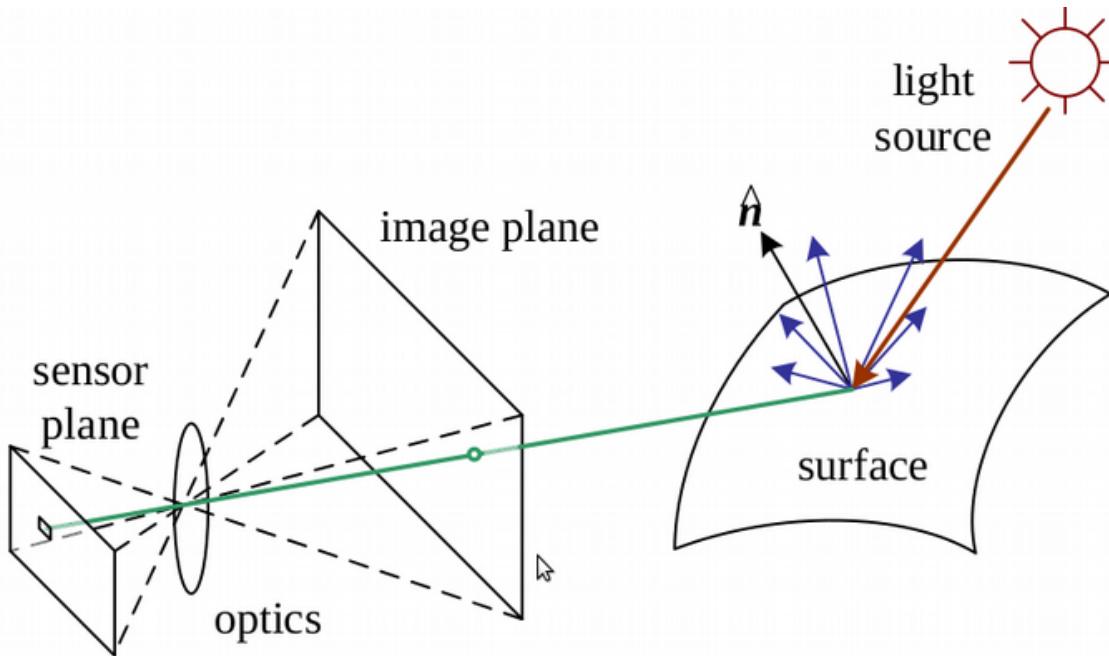
Geometric primitives form the basic building blocks used to describe three-dimensional shapes.

- **2D points** (pixel coordinates in an image) can be denoted using a pair of values, $x = (x, y) \in R^2$
 $2D$ points can also be represented using homogeneous coordinates,

$$\tilde{x} = (\tilde{x}, \tilde{y}, \tilde{w}) = \tilde{w}\bar{x} \in P^2$$

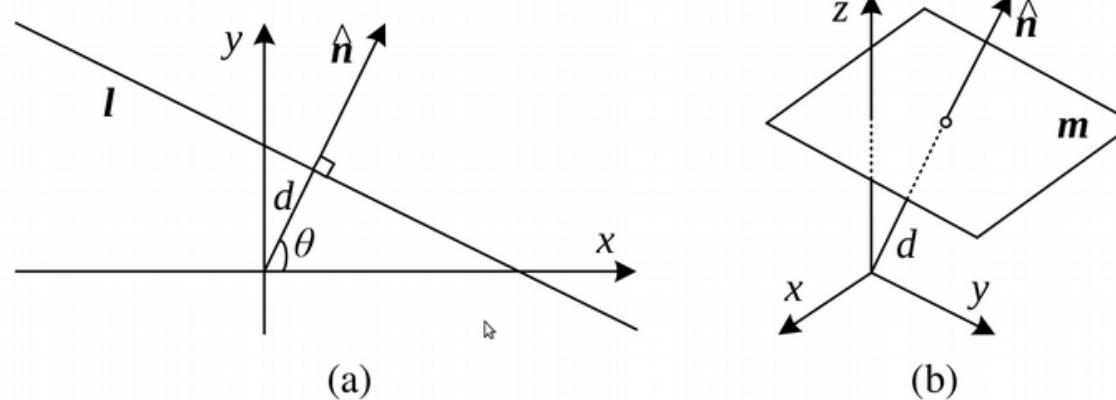
where vectors \tilde{x} that differ only by scale \tilde{w} are considered to be equivalent, $\bar{x} = (x, y, 1)$ is the *augmented vector* and P^2 is called the *2D projective space*.

Photometric image formation



- **Lighting** to produce an image, the scene must be illuminated with one or more light sources.
- Light sources can generally be divided into point and area light sources. A point light source originates at a single location in space, potentially at infinity (e.g., the sun).

Geometric primitives - 2D lines



- **2D lines** can also be represented using homogeneous coordinates $\tilde{l} = (a, b, c)$. The corresponding line equation is

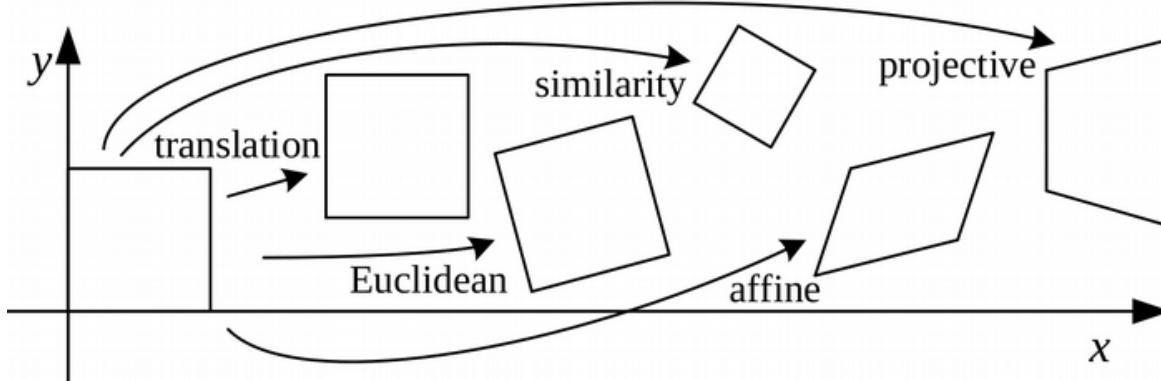
$$\tilde{x} \cdot \tilde{l} = ax + by + c = 0.$$

We can normalize the line equation vector so that

$l = (\hat{n}_x, \hat{n}_y, d) = (\hat{n}, d)$ with $\|\hat{n}\| = 1$ this case, \hat{n} is the normal vector perpendicular to the line and d is its distance to the origin.

- The combination (Θ, d) is also known as *polar coordinates*.

2D transformations



- **2D translations** can be written as $x' = x + t$ or

$$x' = \begin{bmatrix} I & t \end{bmatrix} \cdot \bar{x},$$

where I is the 2×2 identity matrix.

- **Rotation + translation.** This transformation is also known as 2D rigid body motion or the 2D Euclidean transformation. It can be written as

$$x' = \begin{bmatrix} R & t \end{bmatrix} \bar{x}$$

where

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

is an orthonormal rotation matrix with $R \cdot R^T = I$ and $|R| = 1$.

- **Scaled rotation**, also known as the similarity transform, this transformation can be expressed as $x' = sRx + t$ where s is an arbitrary scale factor. It can also be written as

$$x' = \begin{bmatrix} sR & t \end{bmatrix} \bar{x} = \begin{bmatrix} a & -b & t_x \\ b & a & t_y \end{bmatrix} \bar{x}$$

where we no longer require that $a^2 + b^2 = 1$. The similarity transform preserves angles between lines.

- **Affine transformation** is written as $x' = A\bar{x}$ where A is an arbitrary 2×3 matrix, i.e.,

$$x' = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix} \bar{x}$$

Parallel lines remain parallel under affine transformations (does not keep angles between stight lines and distance between points)

- **Projective transformation**, also known as a perspective transform or homography, operates on homogeneous coordinates,

$$\tilde{x}' = \tilde{H}\tilde{x},$$

where \tilde{H} is an arbitrary 3×3 matrix. \tilde{H} is only defined up to a scale, and that two \tilde{H} matrices that differ only by scale are equivalent.

Hierarchy of 2D coordinate transformations

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \mathbf{I} & \mathbf{t} \end{bmatrix}_{2 \times 3}$	2	orientation	
rigid (Euclidean)	$\begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}_{2 \times 3}$	3	lengths	
similarity	$\begin{bmatrix} s\mathbf{R} & \mathbf{t} \end{bmatrix}_{2 \times 3}$	4	angles	
affine	$\begin{bmatrix} \mathbf{A} \end{bmatrix}_{2 \times 3}$	6	parallelism	
projective	$\begin{bmatrix} \tilde{\mathbf{H}} \end{bmatrix}_{3 \times 3}$	8	straight lines	

- Each transformation also preserves the properties (similarity preserves not only angles but also parallelism and straight lines).
- The 2×3 matrices are extended with a third $[0^T 1]$ row to form a full 3×3 matrix for homogeneous coordinate transformations.

Forward warping (or mapping)

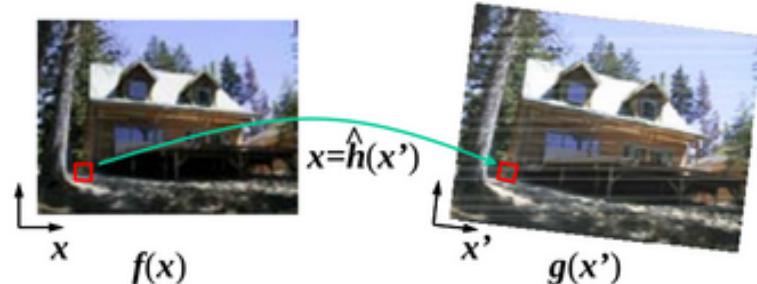


- (a) a pixel $f(x)$ is copied to its corresponding location $x' = h(x)$ in image $g(x')$;
- (b) detail of the source and destination pixel locations.
- Forward warping algorithm for transforming an image $f(x)$ into an image $g(x')$:

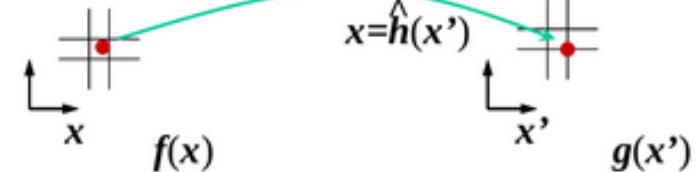
```

procedure forwardWarp( f , h , out g ):
    For every pixel x in f(x)
        1. Compute the destination location x' = h(x).
        2. Copy the pixel f(x) to g(x').
    
```

Inverse warping (or mapping)



(a)



(b)

- (a) a pixel $g(x')$ is sampled from its corresponding location $x = \hat{h}(x')$ in image $f(x)$;
- (b) detail of the source and destination pixel locations.
- Inverse warping algorithm for creating an image $g(x')$ from an image $f(x)$ using the parametric transform $x' = h(x)$:

```

procedure inverseWarp( f , h , out g ):
    For every pixel x' in g(x')
        1. Compute the source location x = h^(x')
        2. Resample f(x) at location x and copy to g(x')

```

2D and 3D feature-based alignment

Feature-based alignment is the problem of estimating the motion between two or more sets of matched 2D or 3D points.

Transform	Matrix	Parameters p	Jacobian J
translation	$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$	(t_x, t_y)	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
Euclidean	$\begin{bmatrix} c_\theta & -s_\theta & t_x \\ s_\theta & c_\theta & t_y \end{bmatrix}$	(t_x, t_y, θ)	$\begin{bmatrix} 1 & 0 & -s_\theta x - c_\theta y \\ 0 & 1 & c_\theta x - s_\theta y \end{bmatrix}$
similarity	$\begin{bmatrix} 1 + a & -b & t_x \\ b & 1 + a & t_y \end{bmatrix}$	(t_x, t_y, a, b)	$\begin{bmatrix} 1 & 0 & x & -y \\ 0 & 1 & y & x \end{bmatrix}$
affine	$\begin{bmatrix} 1 + a_{00} & a_{01} & t_x \\ a_{10} & 1 + a_{11} & t_y \end{bmatrix}$	$(t_x, t_y, a_{00}, a_{01}, a_{10}, a_{11})$	$\begin{bmatrix} 1 & 0 & x & y & 0 & 0 \\ 0 & 1 & 0 & 0 & x & y \end{bmatrix}$
projective	$\begin{bmatrix} 1 + h_{00} & h_{01} & h_{02} \\ h_{10} & 1 + h_{11} & h_{12} \\ h_{20} & h_{21} & 1 \end{bmatrix}$	$(h_{00}, h_{01}, \dots, h_{21})$	

- For simplicity we reduce problem to global *parametric transformations*,
- Jacobians of the 2D coordinate transformations $x' = f(x; p)$.

2D alignment using least squares

Given a set of matched feature points $\{x_i, x'_i\}$ and a planar parametric transformation of the form $x' = f(x; p)$

How can we estimate the parameters p ?

- The usual way is least squares (LS to minimize the sum of squared errors)

$$E_{LS} = \sum_i \|r_i\|^2 = \sum_i \|f(x_i; p) - x'_i\|^2,$$

where

$$r_i = f(x_i; p) - x'_i = \hat{x}'_i - \tilde{x}'_i$$

is the residual between the measured location \hat{x}'_i and its corresponding predicted location $\tilde{x}'_i = f(x_i; p)$.

Linear motion model

For linear operations (translation, similarity, and affine) relationship between the amount of motion $\Delta x = x' - x$ and the unknown parameters p ,

$$\Delta x = x' - x = J(x)p,$$

where $J = \frac{\partial f}{\partial p}$ is the Jacobian of the transformation f with respect to the motion parameters p .

- In linear least squares problem:

$$E_{LS} = \sum_i \|J(x_i)p - \Delta x'_i\|^2 = p^T A p - 2p^T b + c$$

where $A = \sum_i J^T(x_i)J(x_i)$ is the Hessian and $b = \sum_i J^T(x_i) \Delta x_i$

- The minimum can be found by solving

$$A \cdot p = b$$

Robust least squares

More robust version of LS required analysis of outliers between the correspondences:

- In this case, it is preferable to use an M-estimator, which involves applying a robust penalty function $\rho(r)$ to the residuals:

$$E_{RLS}(\Delta p) = \sum_i \rho(||r_i||)$$

- Instead of squaring them. We can take the derivative of this function with respect to p and set it to 0

$$\sum_i \psi(||r_i||) \frac{\partial ||r_i||}{\partial p} = \sum_i \frac{\psi(||r_i||)}{||r_i||} r_i^T \frac{\partial r_i}{\partial p} = 0$$

where $\psi(r) = \rho'(r)$ is the derivative of ρ and is called the *influence function*.

Iteratively Reweighted Least Squares (IRLS)

- If we introduce **weight function** $w(r) = \frac{\psi(r)}{r}$,
- We observe that finding the stationary point of $E_{RLS}(\Delta p) = \sum_i \rho(||r_i||)$ is equivalent to minimizing IRLS:

$$E_{IRLS} = \sum_i w(||r_i||) ||r_i||^2,$$

where the $w(||r_i||)$ is inverse proportional to squared standard deviation $\sim \frac{1}{\sigma_i^2}$.

RANDom SAmple Consensus - RANSAC

A better approach is to find correspondences with a dominant motion estimate.

- It starts by selecting (at random) a subset of k correspondences, which is used to compute an initial estimate for p .
- The residuals of the full set of correspondences are then computed as:

$$r_i = \tilde{x}_i' (x_i; p) - \hat{x}_i'$$

where \tilde{x}_i' are the estimated (mapped) locations and \hat{x}_i' are the sensed (detected) feature point locations.

- The RANSAC technique counts the number of inliers that are within ϵ of their predicted location (whose $\|r_i\| \leq \epsilon$).

Implementation of RANSAC

- The value of ϵ is application dependent (often around 1–3 pixels),
- Least median of squares finds the median value of the $||r_i||^2$ values,
- The random selection process is repeated S times and the sample set with the largest number of inliers (or with the smallest median residual) is kept as the final solution.
- The initial parameter p is passed on to the next data fitting stage.
- When the number of measurements is large the most plausible points can be selected.

Implementation of RANSAC - number of trials

- To ensure that the random sampling has a good chance of finding a true value we have to make the sufficient number of trials S .
- The likelihood in one trial that all k random samples are inliers is p_k .
- Let p be the probability of valid correspondence and P the total probability of success after S trials. The likelihood that S such trials will all fail is

$$1 - P = (1 - p^k)^S$$

- and the required minimum number of trials is

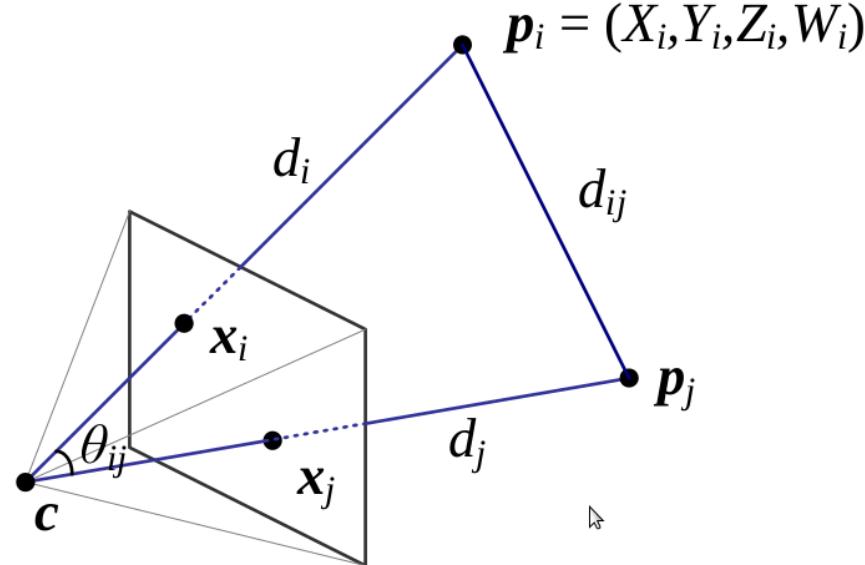
$$S = \frac{\log(1 - P)}{\log(1 - p^k)}$$

Pose estimation

A particular instance of feature-based alignment, which occurs very often, is estimating an object's 3D pose from a set of 2D point projections.

- This **pose estimation** problem is also known as **extrinsic calibration** (as opposed to the *intrinsic calibration* of internal camera parameters such as focal length etc.)
- The problem of recovering pose from three correspondences is known as the perspective-3-point-problem (P3P)

Linear algorithms



The simplest way to recover the pose of the camera is to form a set of linear equations from the camera matrix of perspective projection,

$$x_i = \frac{p_{00}X_i + p_{01}Y_i + p_{02}Z_i + p_{03}}{p_{20}X_i + p_{21}Y_i + p_{22}Z_i + p_{23}}$$

$$y_i = \frac{p_{10}X_i + p_{11}Y_i + p_{12}Z_i + p_{13}}{p_{20}X_i + p_{21}Y_i + p_{22}Z_i + p_{23}}$$

where (x_i, y_i) are the measured 2D feature locations and (X_i, Y_i, Z_i) are the known 3D feature locations.

Linear algorithms

- In order to compute the 12 unknowns in P , at least six correspondences between 3D and 2D locations must be known.
- Once the entries in P have been recovered, it is possible to recover both the intrinsic calibration matrix K and the rigid transformation (R, t) :

$$P = K[R|t].$$

- Since K is by convention upper-triangular, both K and R can be obtained from the front 3×3 sub-matrix of P using RQ factorization,
- In the case when the camera is already calibrated the matrix K is known,
- Visual angle θ_{ij} between any pair of 2D points \hat{x}_i and \hat{x}_j must be the same as the angle between their corresponding 3D points p_i and p_j .

Linear algorithms

- Given a set of corresponding $2D$ and $3D$ points $\{(\hat{x}_i, p_i)\}$, where the \hat{x}_i are unit directions obtained by transforming $2D$ pixel measurements x_i to unit norm $3D$ directions \hat{x}_i through the inverse calibration matrix K

$$\hat{x}_i = \frac{K^{-1}x_i}{\|K^{-1}x_i\|},$$

the unknowns are the distances d_i from the camera origin c to the $3D$ points p_i , where

$$p_i = d_i \hat{x}_i + c$$

- Once the individual estimates of the d_i distances we can generate a $3D$ structure consisting of the scaled point directions $d_i \hat{x}_i$, to obtain the desired pose estimate.

Iterative algorithms

The most accurate way to estimate pose is to directly minimize the squared reprojection error for the 2D points using non-linear least squares.

- We can write the projection equations as

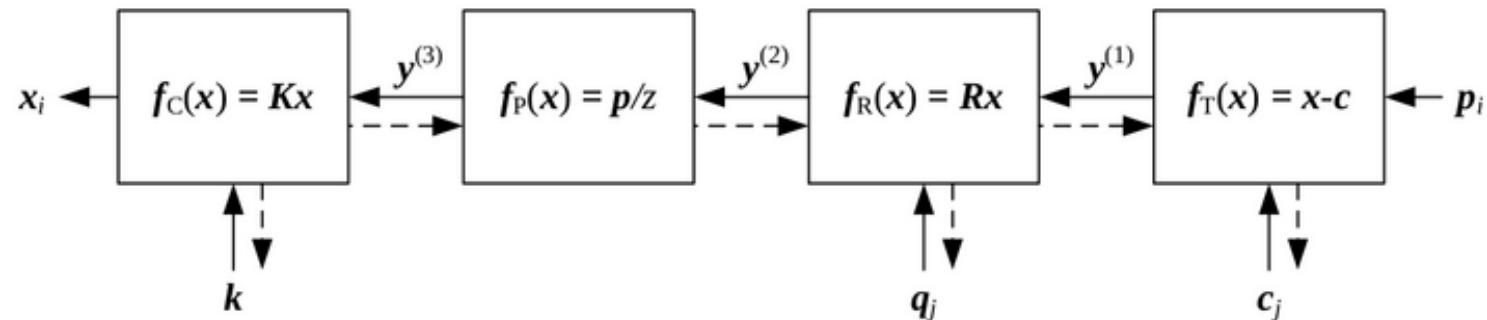
$$x_i = f(p_i; R, t, K)$$

and iteratively minimize the linearized reprojection errors:

$$E_{NLP} = \sum_i \rho\left(\frac{\partial f}{\partial R} \Delta R + \frac{\partial f}{\partial t} \Delta t + \frac{\partial f}{\partial K} \Delta K - r_i\right)$$

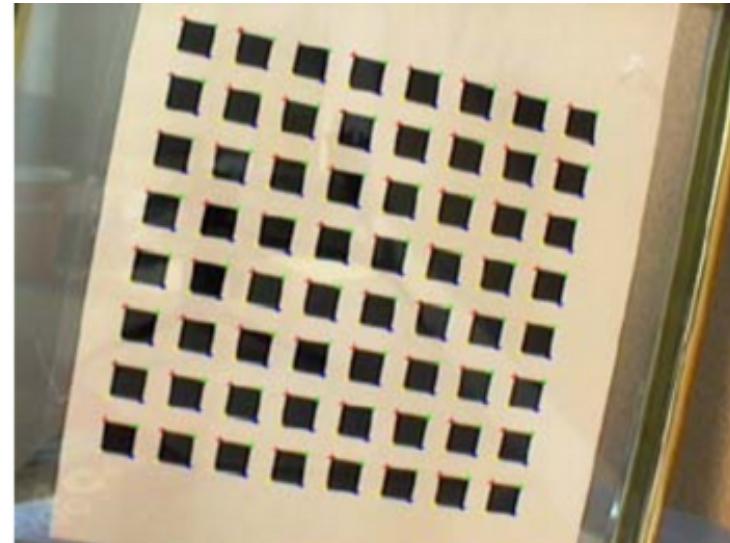
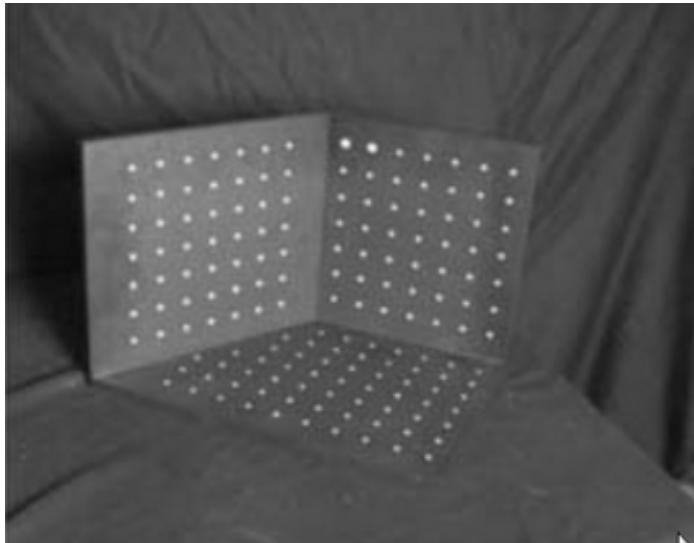
where $r_i = \tilde{x}_i - \hat{x}_i$ is 2D error in predicted position.

Implementation of iterative algorithms



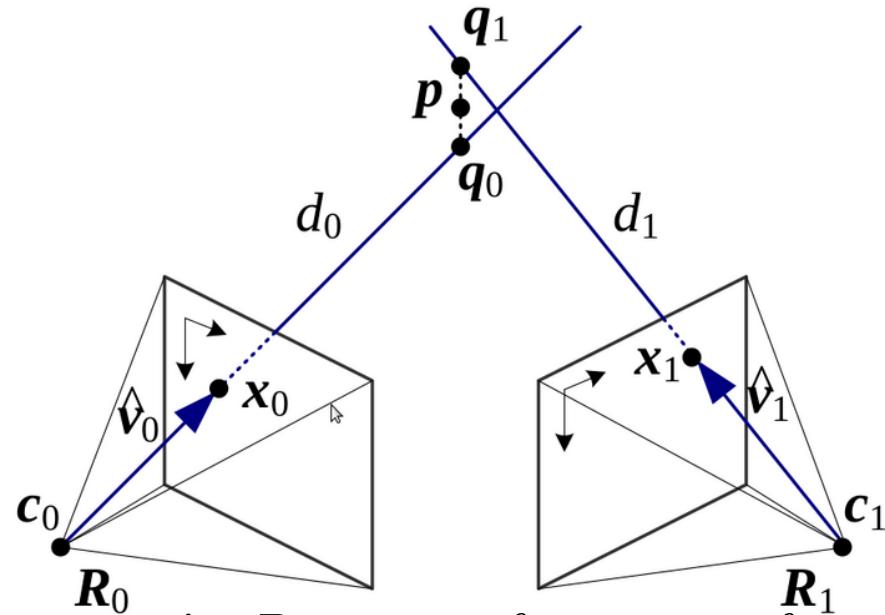
- A set of chained transforms for projecting a 3D point p_i to a 2D measurement x_i through a series of transformations $f^{(k)}$, each of which is controlled by its own set of parameters.
- The dashed lines indicate the flow of information as partial derivatives are computed during a backward pass.

Geometric intrinsic calibration



- **Calibration patterns** - use of a calibration pattern or set of markers is one of the more reliable ways to estimate a camera's intrinsic parameters,
- **Planar calibration patterns** - a good way to perform calibration is to move a planar calibration target in a controlled fashion through the workspace volume.

Triangulation



The problem of determining a point's 3D position from a set of corresponding image locations and known camera positions is known as *triangulation*.

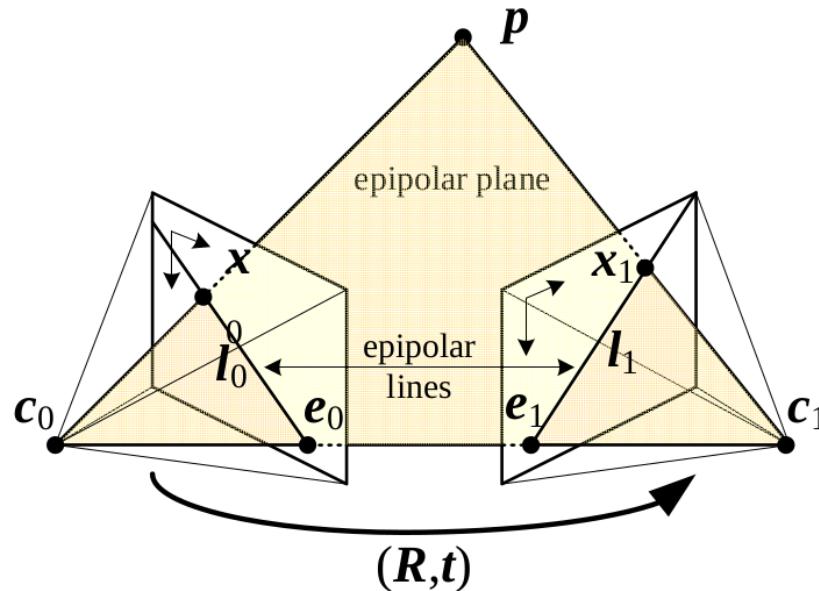
- This problem is the reverse to the pose estimation,
- To solve this problem we need to find the 3D point p that lies closest to all of the 3D rays corresponding to the 2D matching feature locations $\{x_j\}$ observed by cameras $\{P_j = K_j[R_j|t_j]\}$, where $t_j = -R_j c_j$ and c_j is the j th camera center.

Triangulation

- The nearest point to p on this ray, which we denote as q_1 , minimizes the distance $\|c_1 + d_1 \hat{v}_1 - p\|^2$,
- The optimal value for p , which lies closest to all of the rays, can be computed as a regular least squares problem by summing over all the r_j^2 and finding the optimal value of p :

$$p = \left[\sum_i (I - \hat{v}_j \hat{v}_j^T) \right]^{-1} \left[\sum_i (I - \hat{v}_j \hat{v}_j^T) c_j \right].$$

Two-frame structure from motion



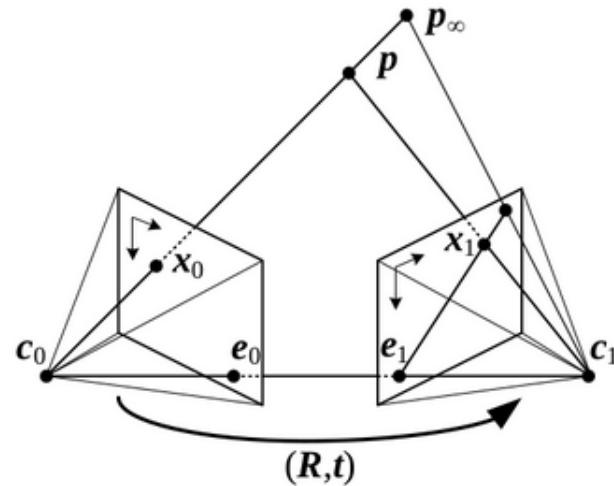
In that task we simultaneous recovery of 3D structure and pose from image correspondences.

- Relative position can be encoded by a rotation R and a translation t ,
- The vectors $t = c_1 - c_0$, $p - c_0$ and $p - c_1$ are co-planar and define the basic epipolar constraint expressed in terms of the pixel measurements x_0 and x_1 .

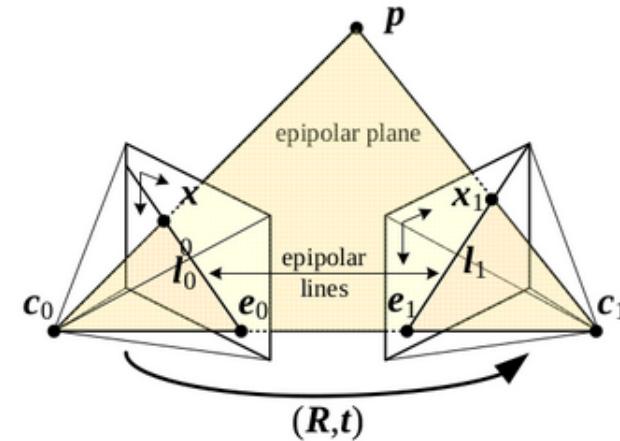
Stereo matching

Stereo matching is the process of taking two or more images and estimating a $3D$ model of the scene by finding matching pixels in the images and converting their $2D$ positions into $3D$ depths.

Epipolar geometry



(a)



(b)

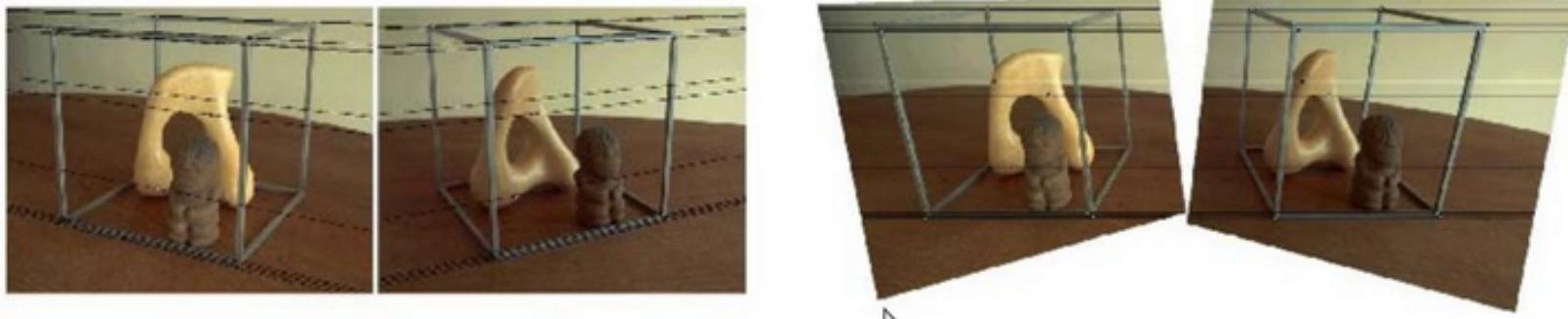
- (a) epipolar line segment corresponding to one ray. Pixel x_0 in one image projects to an epipolar line segment in the other image.
- (b) corresponding set of epipolar lines and their epipolar plane.

Rectification

The epipolar geometry for a pair of cameras is implicit in the relative pose and calibrations of the cameras, and can be computed from point matches.

- One way to do this is to use a general correspondence algorithm, such as *optical flow*.
- A more efficient algorithm can be obtained by rectifying the input images so that corresponding horizontal scanlines are epipolar lines,
- Afterwards, it is possible to match horizontal scanlines independently or to shift images horizontally while computing matching scores.

Rectification - example



- A simple way to rectify the two images is to first rotate both cameras so that they are looking perpendicular to the line joining the camera centers c_0 and c_1 ,
- Next, to determine the desired twist around the optical axes, make the up vector perpendicular to the camera center line.
- Finally, re-scale the images, if necessary, to account for different focal lengths,

Standard rectified geometry

is employed in a lot of stereo camera setups and stereo algorithms, and leads to a very simple inverse relationship between 3D depths Z and disparities d :

$$d = f \frac{B}{Z}$$

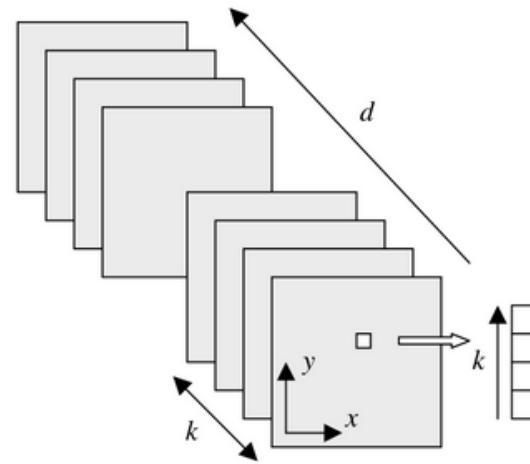
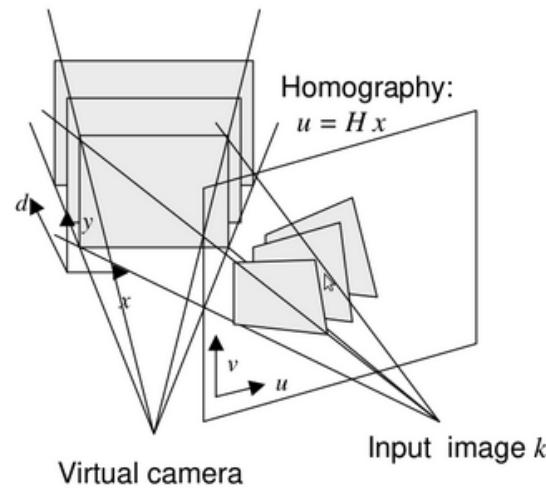
where f is the focal length (measured in pixels), B is the baseline.

- The relationship between corresponding pixel coordinates in the left and right images:

$$x' = x + d(x, y), y' = y$$

- The task of extracting depth from a set of images then becomes one of estimating the **disparity map** $d(x, y)$.
- After rectification, we can easily compare the similarity of pixels at corresponding locations (x, y) and $(x', y') = (x + d, y)$.

Plane sweep



An alternative to pre-rectifying the images before matching is to **sweep a set of planes** through the scene and to measure the photoconsistency of different images as they are re-projected onto these planes

- The set of planes seen from a virtual camera induces a set of homographies in any other source (input) camera image.
- The warped images from all the other cameras can be stacked into a generalized disparity space volume $\tilde{I}(x, y, d, k)$, disparity d of camera k , (x, y) - pixel location

3D to 2D projections

We can do this using a linear $3D$ to $2D$ projection matrix.

- An orthographic projection simply drops the z component of the three-dimensional coordinate p to obtain the $2D$ point x .

$$x = [I_{2 \times 2} | 0]p$$

- Often images need to be scaled by s to fit onto an image sensor (eg. cm to pixels). For this reason, scaled orthography is actually more commonly used,

$$x = [sI_{2 \times 2} | 0]p$$

Perspective

The most commonly used projection in computer graphics and computer vision is true $3D$ perspective

- Here, points are projected onto the image plane by dividing them by their z component
- In homogeneous coordinates, the projection has a simple linear form

$$\tilde{x} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tilde{p}$$

- Into normalized device coordinates in the range $(x, y, z) \in [-1, -1] \times [-1, 1] \times [0, 1]$, and then rescales these coordinates to integer pixel coordinates using a viewport transformation
- The (initial) perspective projection is then represented using a 4×4 matrix

$$\tilde{x} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{-z_{far}}{z_{range}} & \frac{z_{near} \cdot z_{far}}{z_{range}} \\ 0 & 0 & 1 & 0 \end{bmatrix} \tilde{p}$$

where z_{near} and z_{far} are the near and far z *clipping planes* and $z_{range} = z_{far} - z_{near}$.

- If we set $z_{near} = 1$, $z_{far} = \inf$, the third element of the normalized screen vector becomes the inverse depth (*disparity*)
- It is then convenient to be able to map disparity value d directly back to a $3D$ location using the inverse of a 4×4 matrix.

- We can do this if we represent perspective projection using a full-rank 4×4 matrix:

$$\tilde{P} = \begin{bmatrix} K & 0 \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} = \tilde{K} \cdot E$$

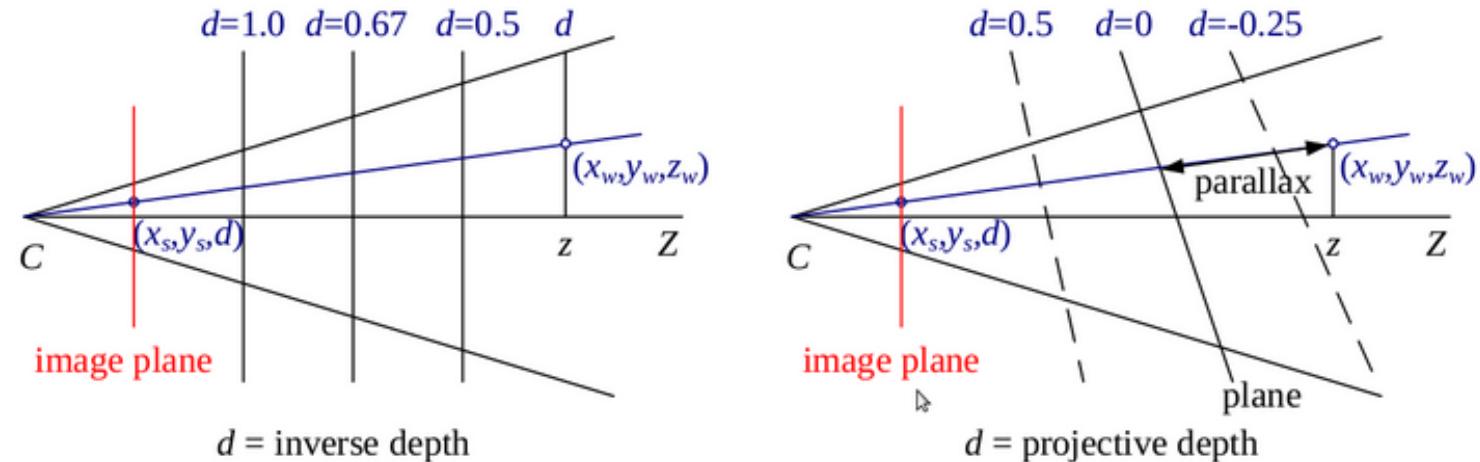
where E is a $3D$ rigid-body (Euclidean) transformation and \tilde{K} is the full-rank calibration matrix.

- The 4×4 camera matrix \tilde{P} can be used to map directly from $3D$ world coordinates $\overline{p_w} = (x_w, y_w, z_w, 1)$ to screen coordinates (plus disparity), $x_s = (x_s, y_s, 1, d)$,

$$s_s \sim \tilde{P} \overline{p_w}$$

where \sim indicates equality up to scale.

Plane plus parallax (projective depth)

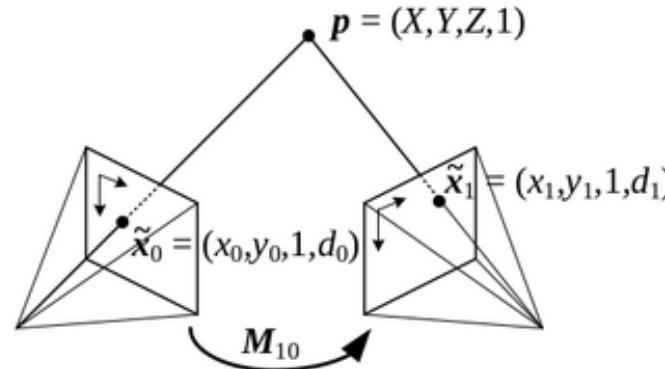


- In general, when using the 4×4 matrix \tilde{P} we have the freedom to remap the last row to whatever suits our purpose,
- Let the last row of \tilde{P} be $p_3 = s_3[\hat{n}_0|c_0]$, where $\|\hat{n}_0\| = 1$, then we have:

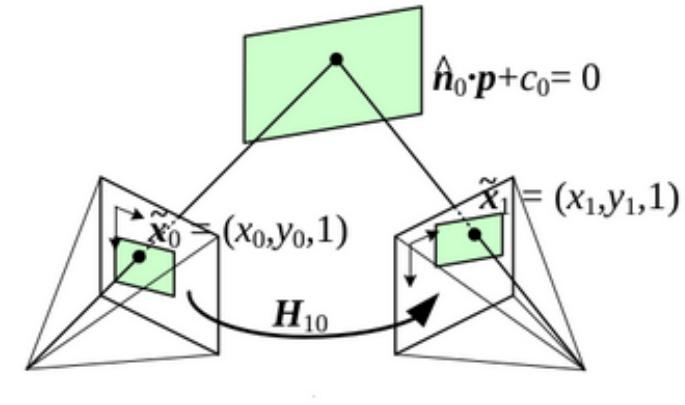
$$d = \frac{s_3}{z}(\hat{n}_0 p_w + c_0),$$

where $z = p_2 \cdot \overline{p_w} = r_z \cdot (p_w - c)$ is the distance of p_w from the camera center C along the optical axis Z .

Mapping from one camera to another



(a)



(b)

- Using the full rank 4×4 camera matrix $\tilde{P} = \tilde{K}E$ we can write the projection from world to screen coordinates as

$$\tilde{x}_0 \sim \tilde{K}_0 E_0 p = \tilde{P}_0 p.$$

- Assuming that we know disparity value d_0 for a pixel in one image, we can compute the 3D point location p using

$$p \sim E_0^{-1} \tilde{K}_0^{-1} \tilde{x}_0.$$

Computer Vision

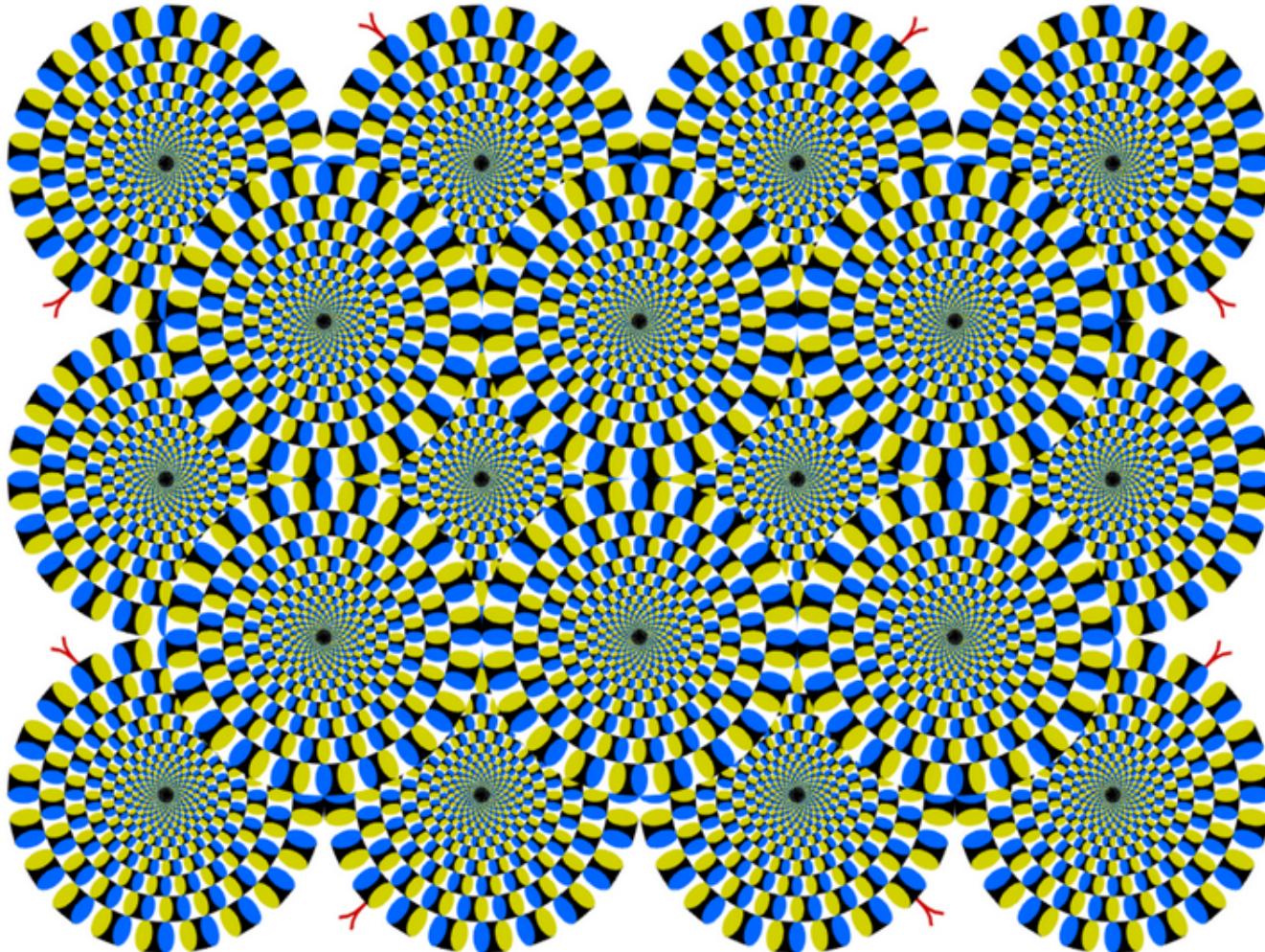
**Vision and movement
lecture 11**

Adam Szmigelski

aszmigie@pjwstk.edu.pl

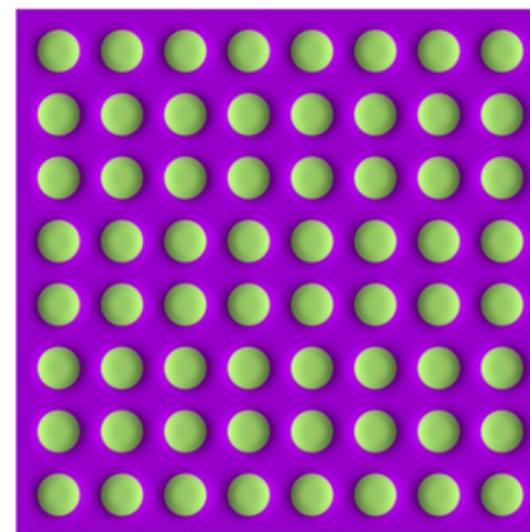
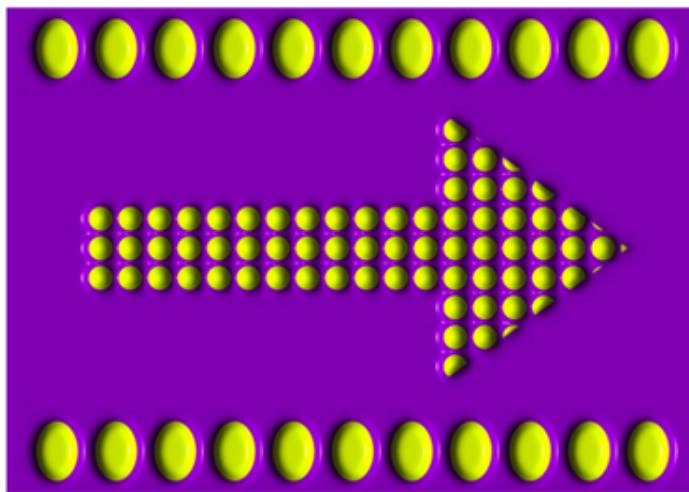
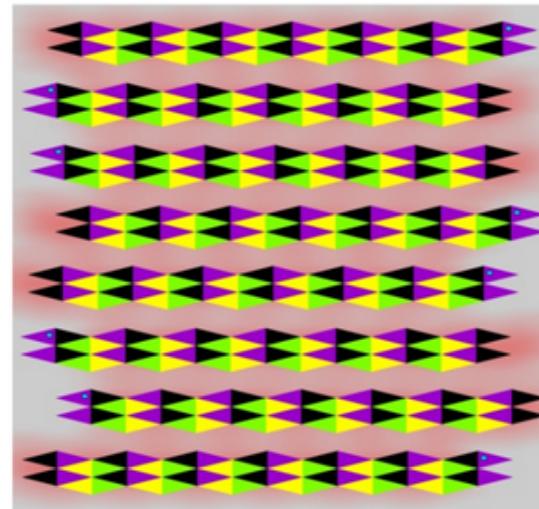
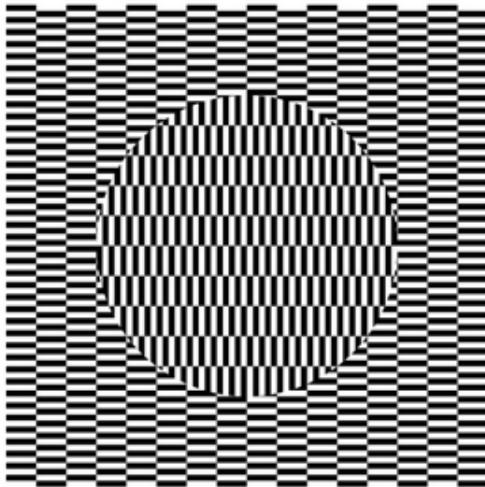
materials: *ftp(public) : //aszmigie/MWR*

Seeing motion from a static picture?

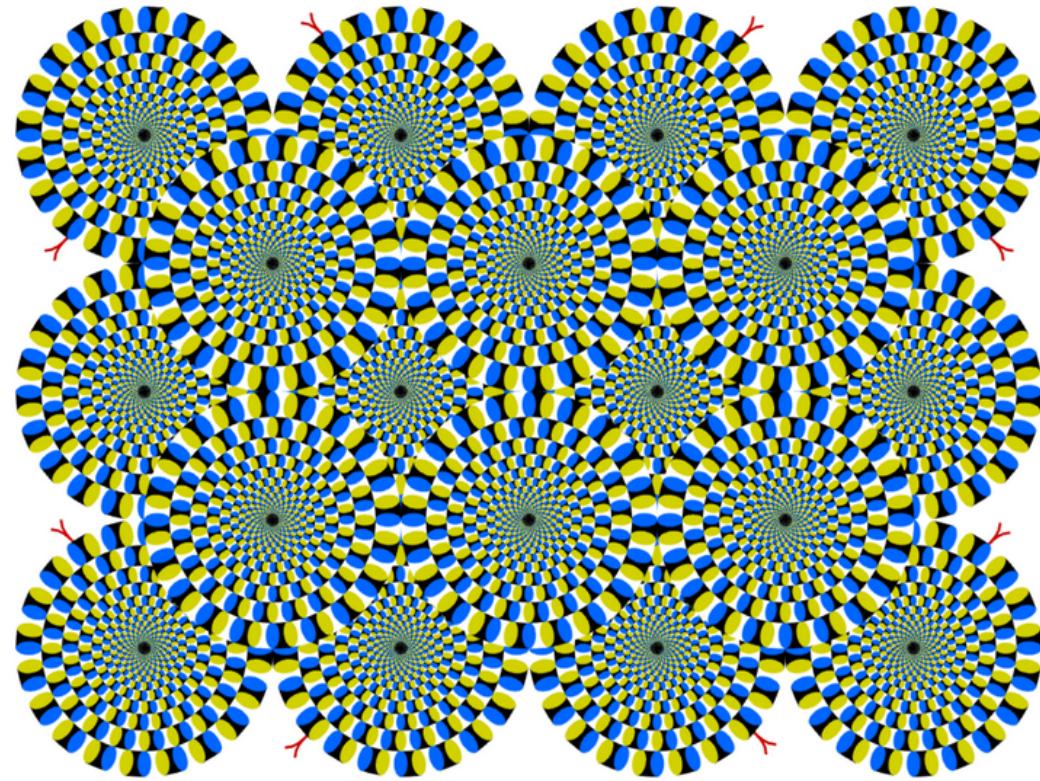


<http://www.ritsumei.ac.jp/~akitaoka/index-e.html>

More examples



How is this possible?



- The true mechanism is to be revealed,
- Data suggest that illusion is related to some component of eye movements,
- We don't expect computer vision to "see" motion from these stimuli.

Motion and perceptual organization

1. Even “impoverished” motion data can evoke a strong percept.

We live in a moving world



- Perceiving, understanding and predicting motion is an important part of our daily lives.

The cause of motion

- Three factors in imaging process
 - Light,
 - Object,
 - Camera.
- Varying either of them causes motion
 - Static camera, moving objects (surveillance),
 - Moving camera, static scene (3D capture),
 - Moving camera, moving scene (sports, movie),
 - Static camera, moving objects, moving light (time lapse).

Motion scenarios (priors)



Static camera, moving scene



Moving camera, static scene



Moving camera, moving scene



Static camera, moving scene, moving light

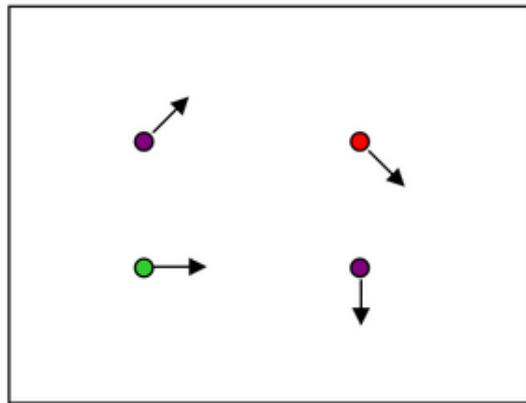
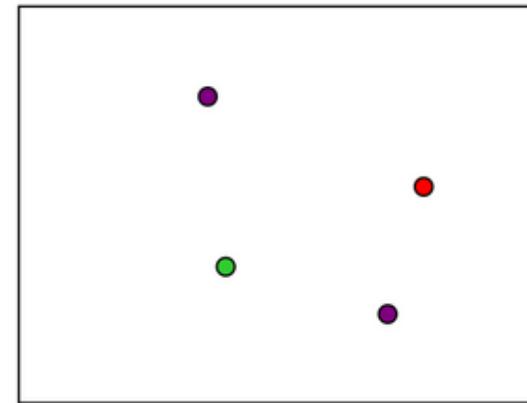
Recovering motion

1. **Feature-tracking** - Extract visual features (corners, textured areas) and “track” them over multiple frames
2. **Optical flow** - Recover image motion at each pixel from spatio-temporal image brightness variations (optical flow)

Feature tracking

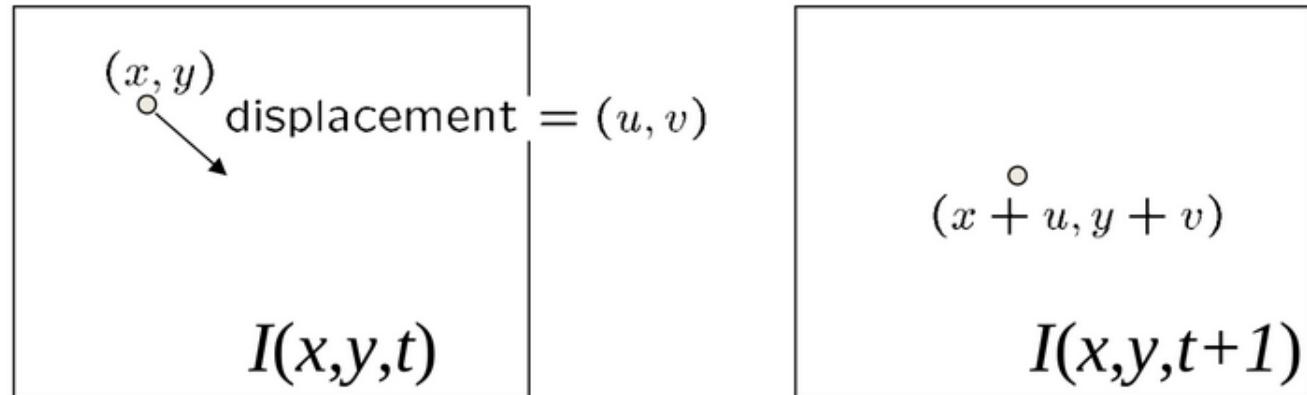
1. Figure out which features can be tracked,
2. Efficiently track across frames,
3. Some points may change appearance over time (e.g., due to rotation, moving into shadows, etc.)
4. Drift: small errors can accumulate as appearance model is updated,
5. Points may appear or disappear: need to be able to add/delete tracked points.

Feature tracking

 $I(x,y,t)$  $I(x,y,t+1)$

- Given two subsequent frames, estimate the point translation
- Key assumptions of Lucas-Kanade Tracker
 - Brightness constancy: projection of the same point looks the same in every frame
 - Small motion: points do not move very far
 - Spatial coherence: points move like their neighbors

The brightness constancy constraint



- Brightness Constancy Equation:

$$I(x, y, t) = I(x + u, y + v, t + 1)$$

Take Taylor expansion of $I(x + u, y + v, t + 1)$ at (x, y, t) to linearize the right side:

$$I(x + u, y + v, t + 1) \approx I(x, y, t) + I_x \cdot u + I_y \cdot v + I_t$$

Image derivative along x Difference over frames

$$I(x + u, y + v, t + 1) - I(x, y, t) = +I_x \cdot u + I_y \cdot v + I_t$$

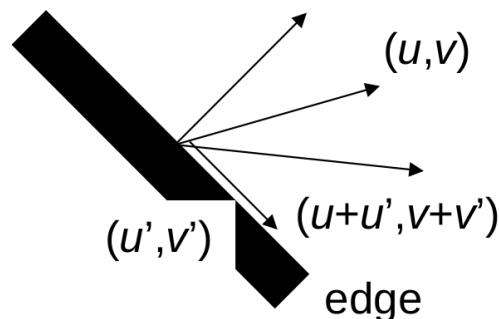
$$I_x \cdot u + I_y \cdot v + I_t \approx 0 \rightarrow \nabla I \cdot [u \ v]^T + I_t = 0$$

The brightness constancy constraint

Can we use this equation to recover image motion (u, v) at each pixel?

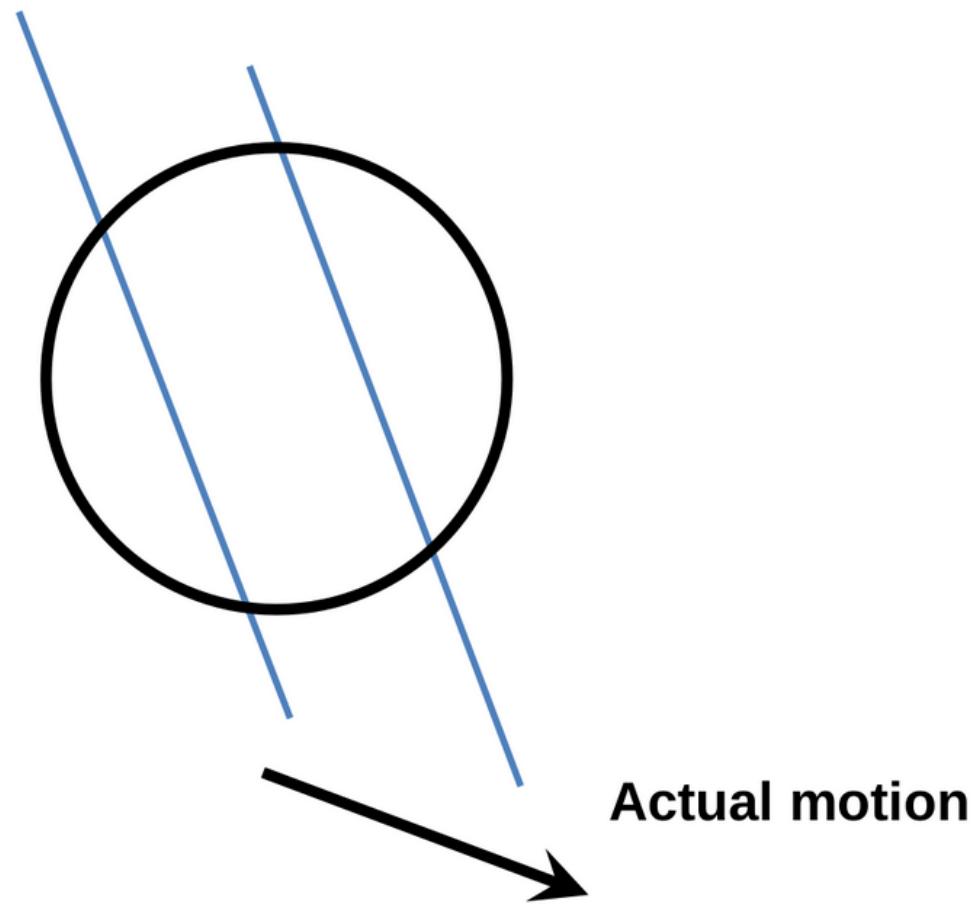
$$\nabla I \cdot [uv]^T + I_t = 0$$

gradient

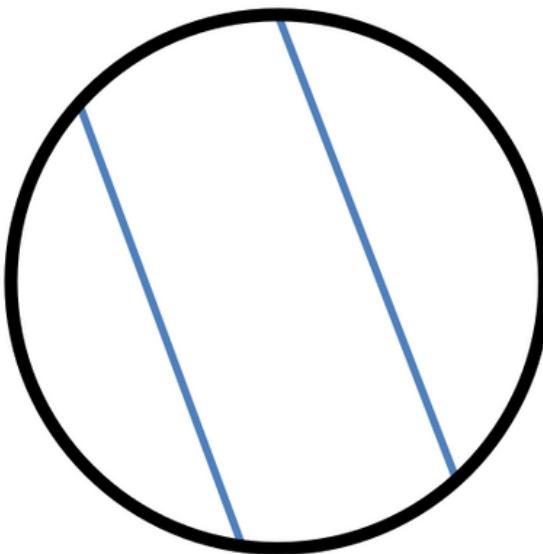


- How many equations and unknowns per pixel ?
- One equation (this is a scalar equation!), two unknowns (u, v) The component of the motion perpendicular to the gradient (i.e., parallel to the edge) cannot be measured
- If (u, v) satisfies the equation, so does $(u + u', v + v')$ if $\nabla I \cdot [u'v']^T = 0$

The aperture problem



The aperture problem



Perceived motion

The barber pole illusion



Working around the aperture problem

Two methods:

- Horn and Shunck method - global method Horn and Shunck, “Determining optical flow”,
- Lucas-Kanade method - local (semi-local) method, uses linear algebra

Horn and Shunck

- The optical flow $(u(x, y), v(x, y))$ at pixel (x, y) is undetermined in the direction of the edge at (x, y) ,
- Image intensities can be noisy,
- Mitigate these problems, by assuming that the underlying optical flow is smooth, i.e. the optical flow vectors at adjacent pixels are similar.
- Leads to following energy functional to be minimized:

$$J(u, v) = \int \int_{\Omega} (I_x u + I_y v + I_t)^2 + \lambda(u_x^2 + u_y^2 + v_x^2 + v_y^2) dx dy$$

Regularization

$$J(u, v) = \underbrace{\int \int_{\Omega} (I_x u + I_y v + I_t)^2}_{\text{data}} + \underbrace{\lambda \underbrace{(u_x^2 + u_y^2 + v_x^2 + v_y^2)}_{\text{regularizer}}}_{\text{Regularization parameter}} dxdy$$

- The addition of the smoothness constraint in the under-constrained problem is called as regularization,
- The parameter $\lambda \geq 0$ performs a weighting between the regularizer and data fidelity term. A larger λ means more weight to the regularizer, and a smaller λ means more weight to the data fidelity term.
- It is usually a user-specified parameter

Horn and Shunck: Basic update equations

$$J(u, v) = \int \int_{\Omega} (I_x u + I_y v + I_t)^2 + \lambda(u_x^2 + u_y^2 + v_x^2 + v_y^2) dx dy$$

Replace the integral by a summation:

$$J(u_{i,j}, v_{i,j}) = \sum_{i=1}^N \sum_{j=1}^M (I_x u_{i,j} + I_y v_{i,j} + I_t)^2 + \lambda((u_{i,j+1} - u_{i,j})^2 + (u_{i+1,j} - u_{i,j})^2 + (v_{i,j+1} - v_{i,j})^2 + (v_{i+1,j} - v_{i,j})^2)$$

Take derivatives of the energy function J in the x direction at pixel (k, l) :

$$\frac{\partial J}{\partial u_{k,l}} = (I_{k,l}^2 + 4\lambda)u_{k,l} + I_x I_y v_{k,l} = 4\lambda u_{k,l} - I_x I_t = 0$$

$$\frac{\partial J}{\partial v_{k,l}} = I_x I_y u_{k,l} + (I_{k,l}^2 + 4\lambda)v_{k,l} = 4\lambda v_{k,l} - I_y I_t = 0$$

Horn and Shunck: Solution

- Note that such a system is extremely expensive to invert.
- Initial condition for u and v – zeros; boundary conditions for derivatives of I - zero across boundary of the image
- Iterative algorithm (Jacobi's method): run for T iterations or until convergence: Update the optical flow at one pixel keeping the optical flow at all other pixels fixed.

Comments on Horn and Shunck

- Optical flow is completely indeterminate in regions where intensity is constant,
- But this method allows for filling in of the flow vectors in such regions, because of the regularizer term,
- The update of u (or v) at pixel (k, l) at iteration $t + 1$ does not depend on $u_{k,l}$ or $v_{k,l}$ in iteration t , but it depends only the neighboring values of u or v ,
- Note that the method banks on the brightness constancy assumption,
- It also assumes that the flow at all points is small in magnitude (otherwise the basic constancy equation doesn't hold),
- It also assumes smoothness of the flow (this can get violated if two independently moving objects come close together in a video frame).

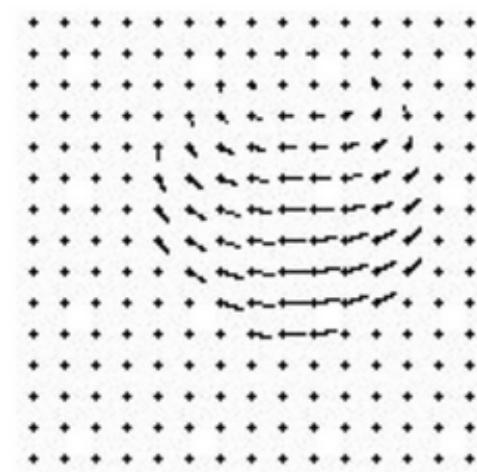
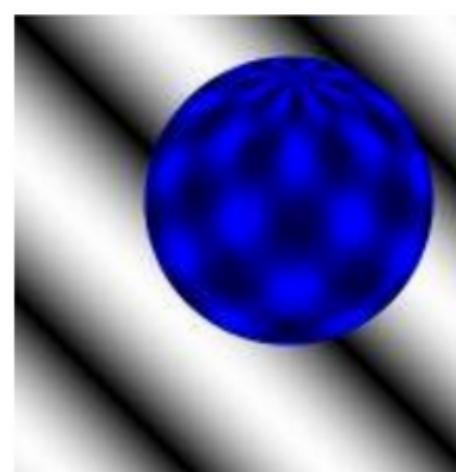
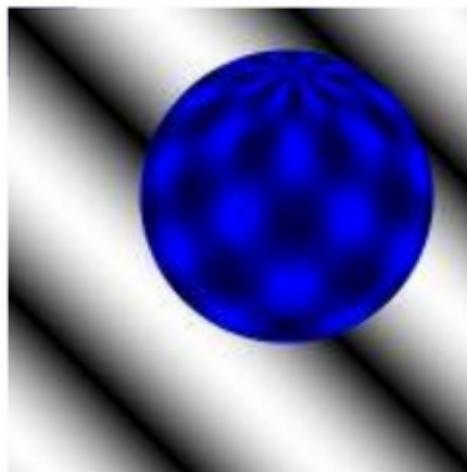
Horn and Shunck: Laplacian

- The original paper by Horn and Shunck suggests the following mask for the Laplacian of u and v :

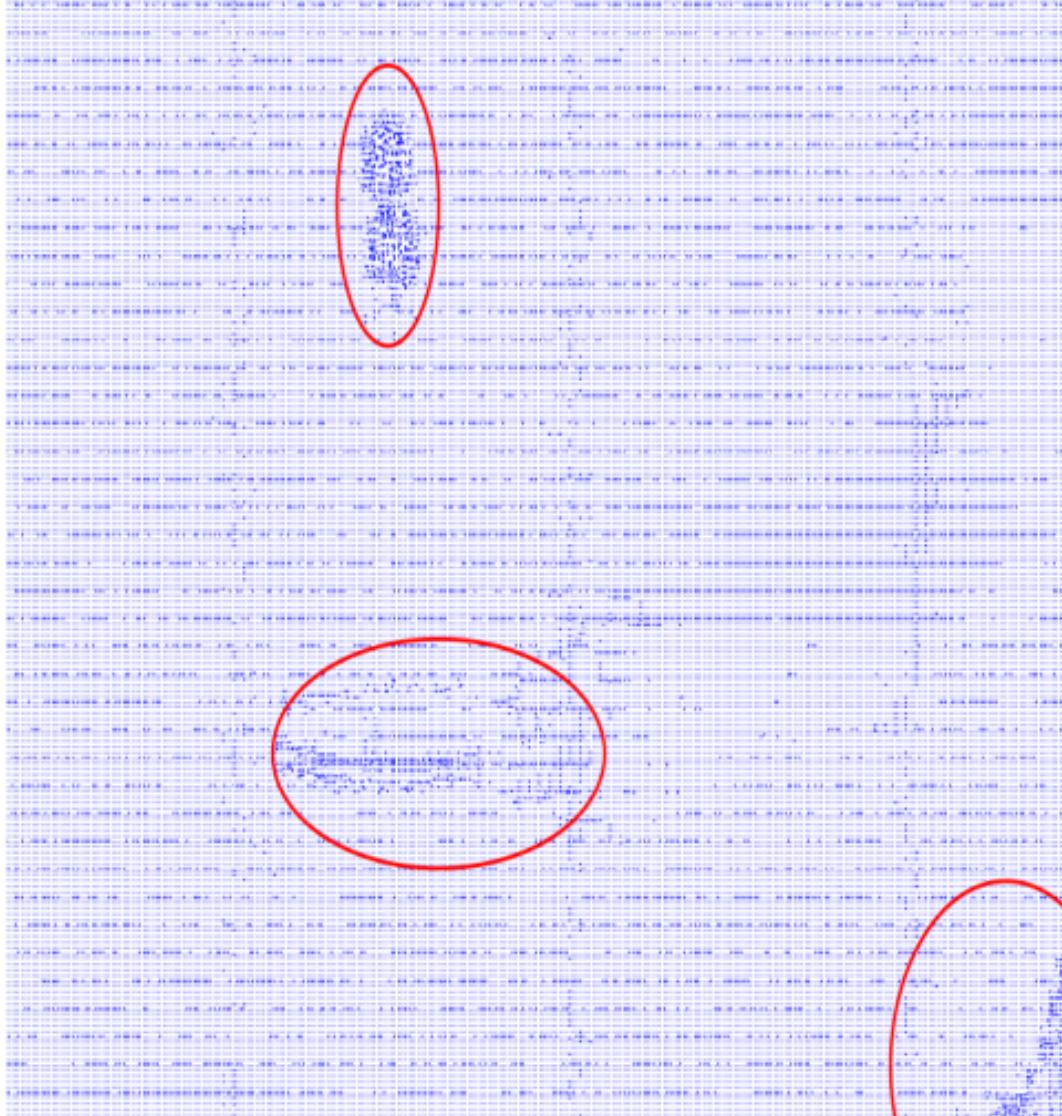
$$12 \cdot \begin{bmatrix} \frac{1}{12} & \frac{1}{6} & \frac{1}{12} \\ \frac{1}{6} & -1 & \frac{1}{6} \\ \frac{1}{12} & \frac{1}{6} & \frac{1}{12} \end{bmatrix}$$

- For image smoothing as well, we applied point-wise masks.
- But here the mask values sum to 0. In smoothing the weights summed to 1.

Horn and Shunck: examples



Horn and Shunck: examples



Lucas-Kanade

- Is a local (semi-local) method - it solves several small problems independently, whereas Horn and Shunck solves one large (global) problem,
- Assume that the optical flow is constant within a small window (say, of size $N \times N$, N is around 5 to 8).
- Uses a least squares approach to solve for the optical flow by combining together several (N^2 in case of $N \times N$ window) brightness constancy equations.

Solving the ambiguity

- Lucas and Kanade - An iterative image registration technique with an application to stereo vision.
- Spatial coherence constraint
- Assume the pixel's neighbors have the same (u, v) - If we use a 5×5 window, that gives us 25 equations per pixel

$$0 = I_t(p_i) + \nabla I(p_i) \cdot [uv]$$

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix} \cdot \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_{25}) \end{bmatrix}$$

Solving the ambiguity

- Least square problem (linear regression)

$$\underbrace{\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix}}_A \cdot \underbrace{\begin{bmatrix} u \\ v \end{bmatrix}}_d = -\underbrace{\begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_{25}) \end{bmatrix}}_b$$

- Least squares solution for d given by $(A^T A)d = A^T b$

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_y I_x & \sum I_y I_y \end{bmatrix} \cdot \begin{bmatrix} u \\ v \end{bmatrix} = -\begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

$$A^T A \cdot d = -A^T b$$

The summations are over all pixels in the $K \times K$ window

Conditions for solvability (Harris corner detector)

Optimal (u, v) satisfies Lucas-Kanade equation

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_y I_x & \sum I_y I_y \end{bmatrix} \cdot \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$
$$A^T A \quad \cdot \quad d \quad = \quad -A^T b$$

When is this solvable? I.e., what are good points to track?

- $A^T A$ should be invertible,
- $A^T A$ should not be too small due to noise,
eigenvalues λ_1 and λ_2 of $A^T A$ should not be too small
- $A^T A$ should be well-conditioned
 $\frac{\lambda_1}{\lambda_2}$ should not be too large (λ_1 = larger eigenvalue)

Errors in Lukas-Kanade

- What are the potential causes of errors in this procedure?
 - Suppose $A^T A$ is easily invertible
 - Suppose there is not much noise in the image
- When our assumptions are violated
 - Brightness constancy is not satisfied,
 - The motion is not small,
 - A point does not move like its neighbors
 - * window size is too large,
 - * what is the ideal window size?

Dealing with larger movements - Iterative refinement

1. Initialize $(x', y') = (x, y)$,
2. Compute (u, v) by

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_y I_x & \sum I_y I_y \end{bmatrix} \cdot \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

3. Shift window by (u, v) : $x' = x' + u; y' = y' + v;$,
4. Recalculate I_t ,
5. Repeat steps 2-4 until small change - Use interpolation for subpixel values.

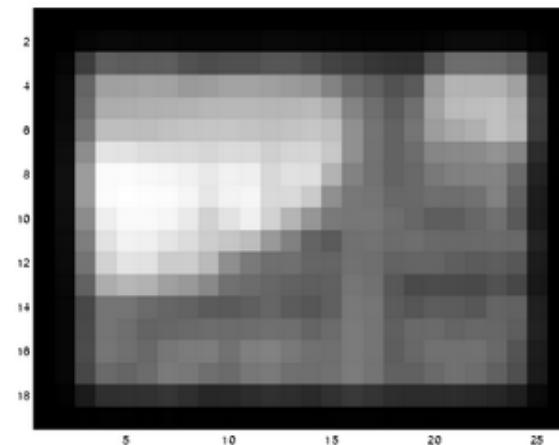
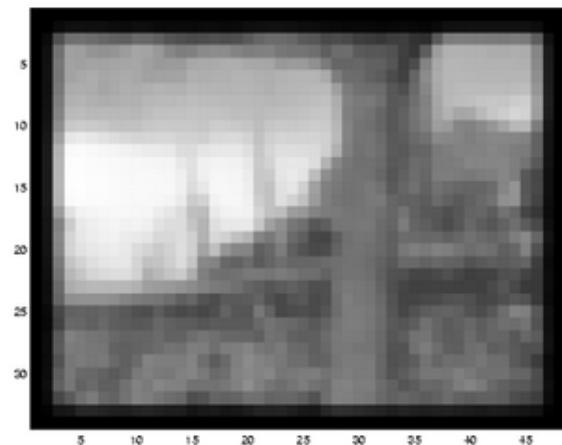
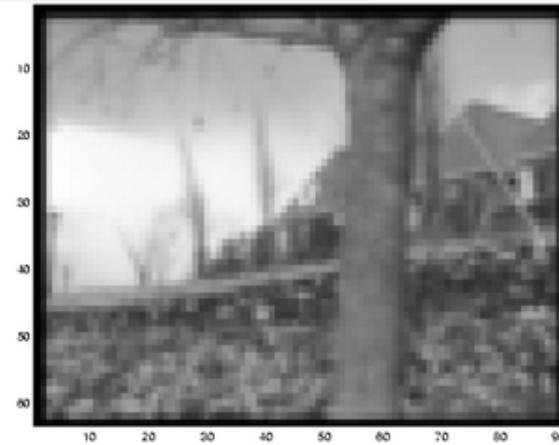
Revisiting the small motion assumption



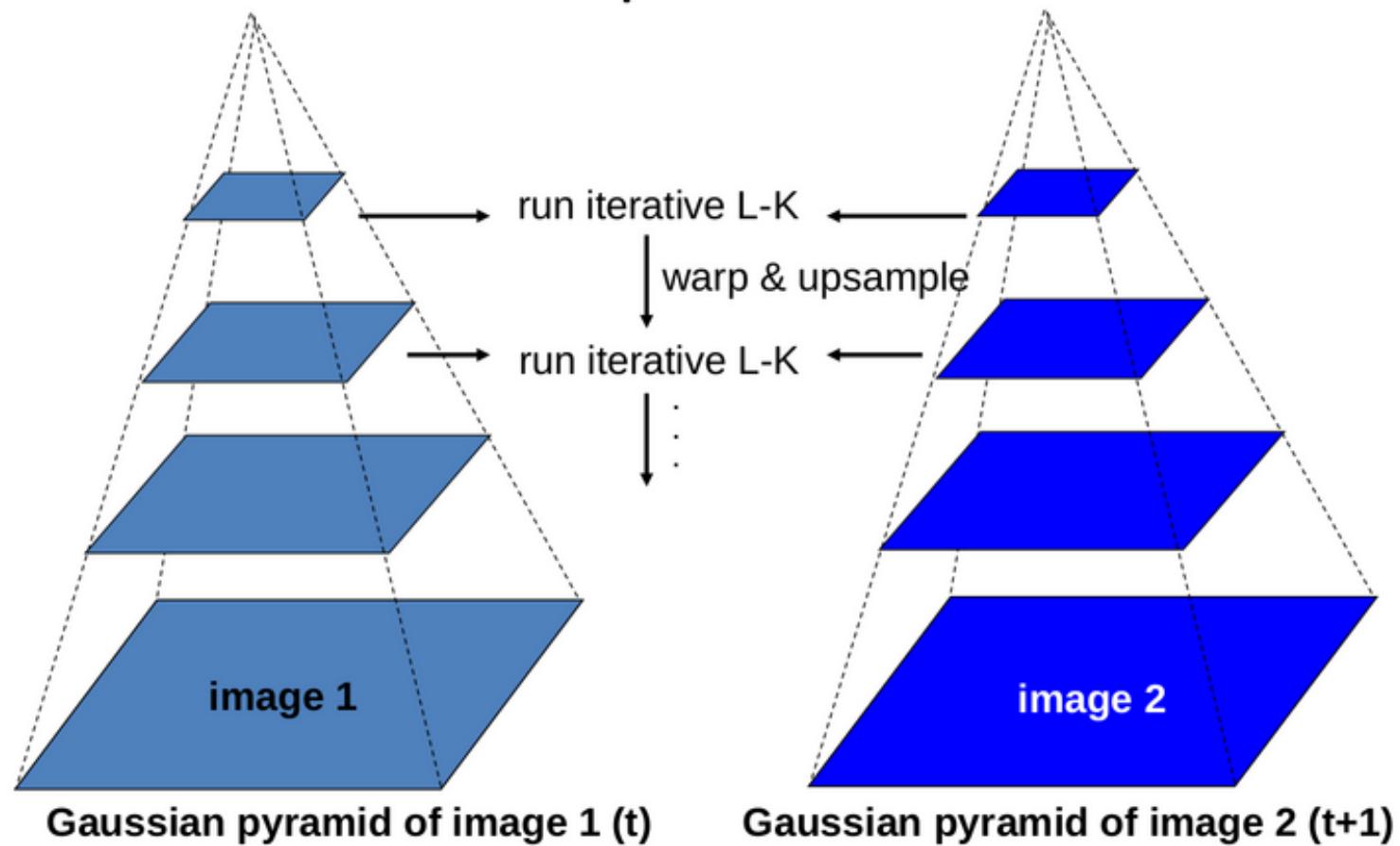
Is this motion small enough?

- Probably not - it's much larger than one pixel (2nd order terms dominate)
- How might we solve this problem?

Reduce the resolution!



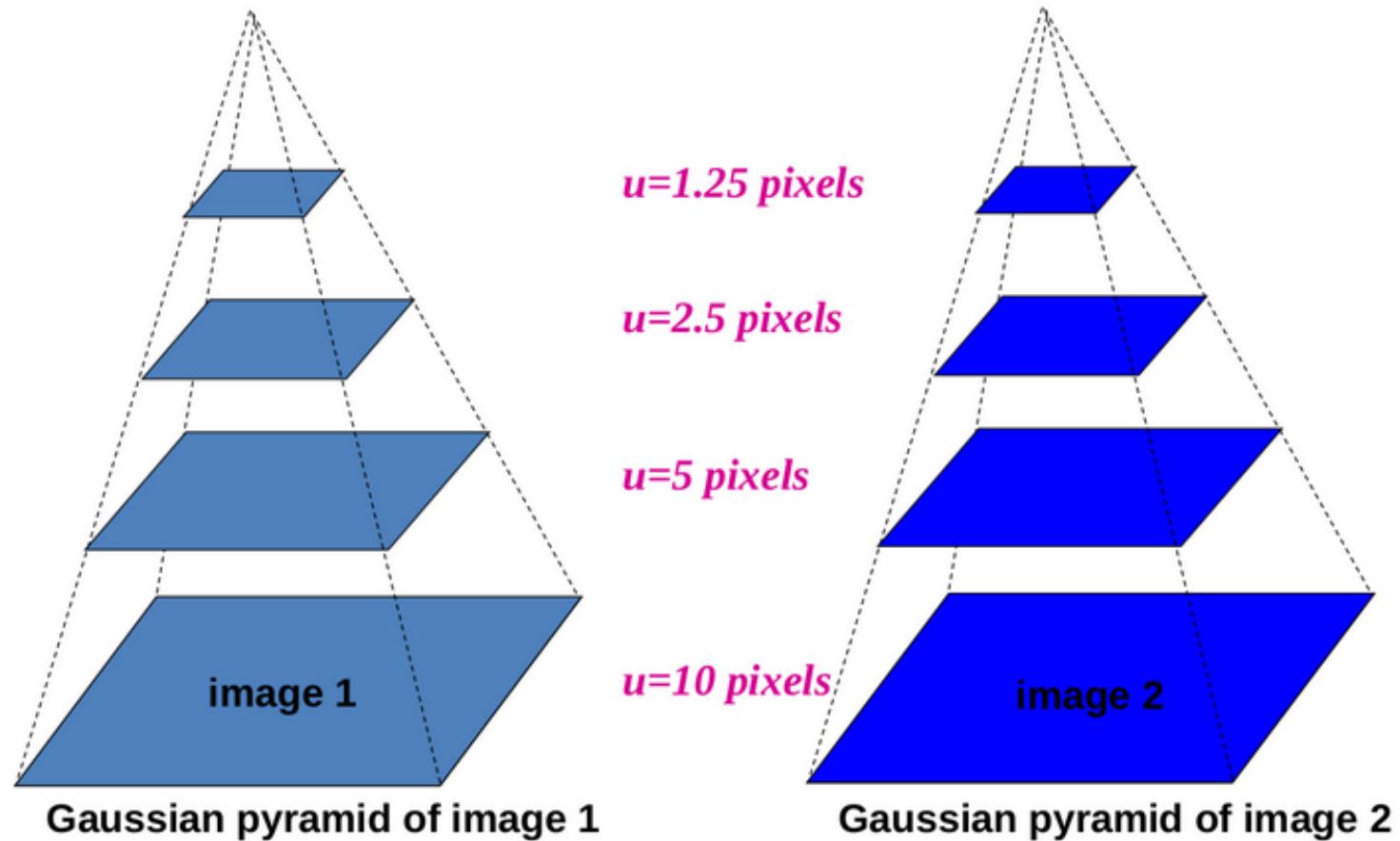
Coarse-to-fine optical flow estimation



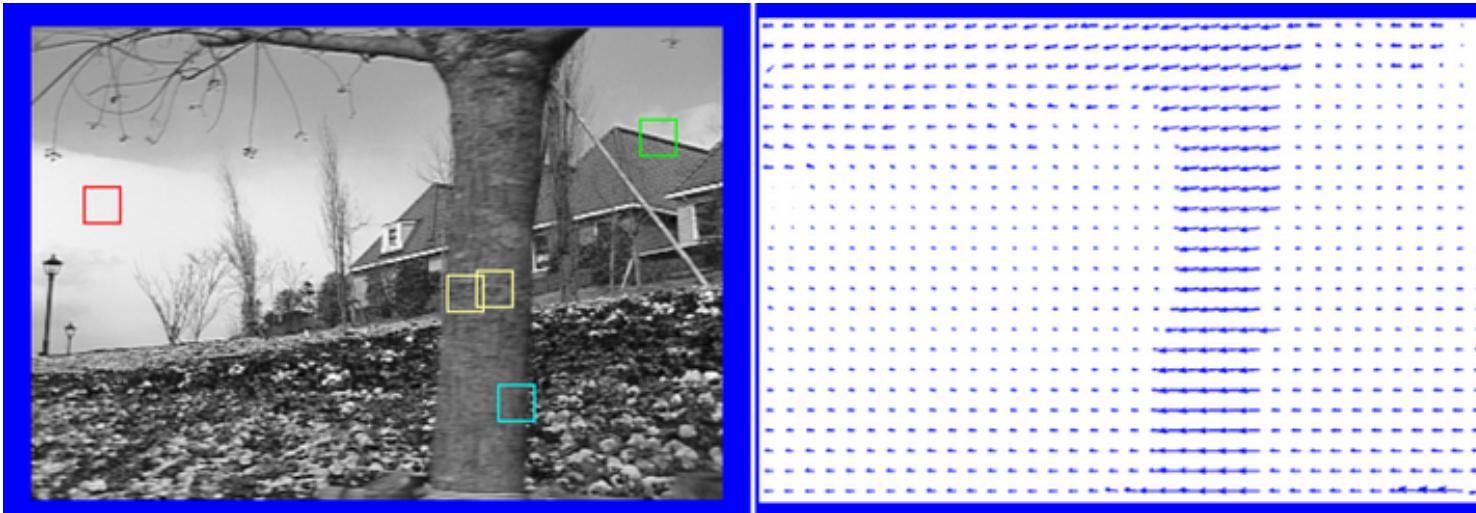
A Few Details

- Top Level
 - Apply L-K to get a flow field representing the flow from the first frame to the second frame,
 - Apply this flow field to warp the first frame toward the second frame,
 - Rerun L-K on the new warped image to get a flow field from it to the second frame.
 - Repeat till convergence.
- Next Level
 - Upsample the flow field to the next level as the first guess of the flow at that level.
 - Apply this flow field to warp the first frame toward the second frame.
 - Rerun L-K and warping till convergence as above.

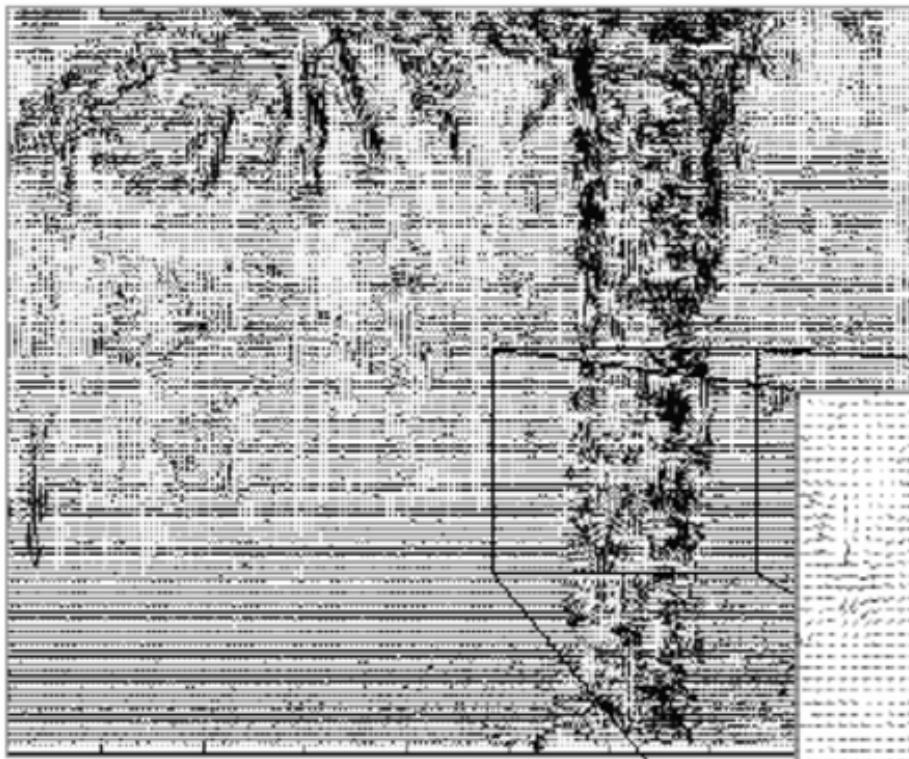
Coarse-to-fine optical flow estimation



Optical flow - example

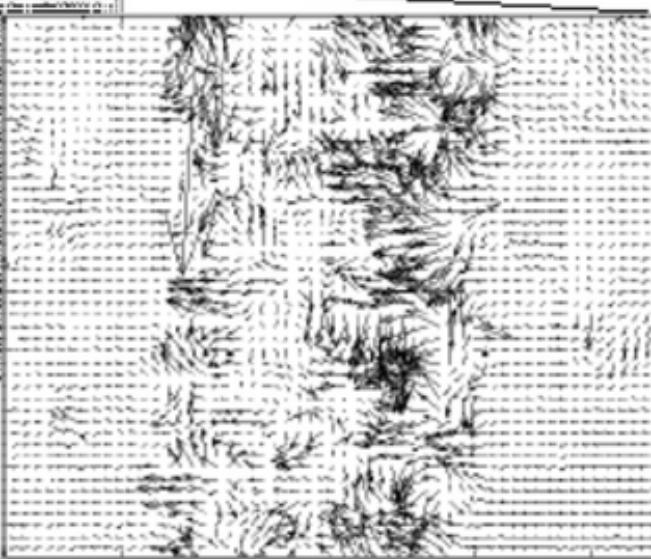


Optical Flow Results

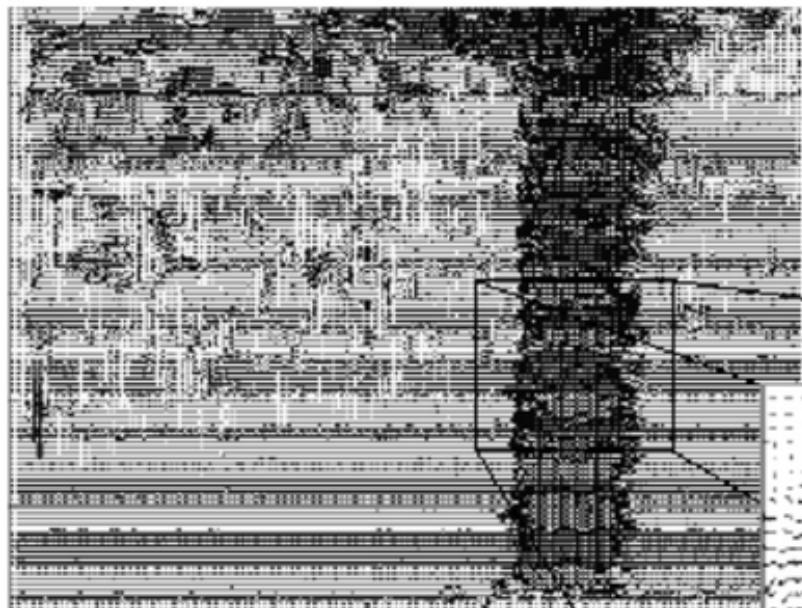


Lucas-Kanade
without pyramids

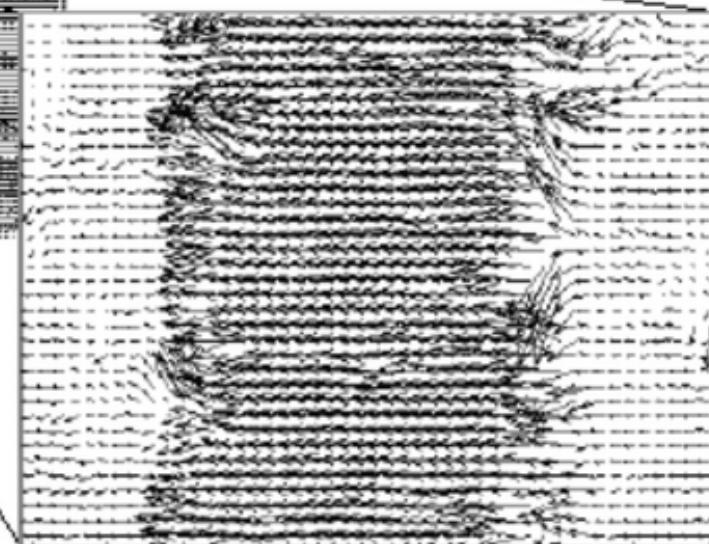
Fails in areas of large motion



Optical Flow Results



Lucas-Kanade with Pyramids



H-S versus L-K

- H-S is global, L-K is local,
- Both have parallelizable implementations,
- H-S gives an optical flow value at all pixels regardless of gradient. L-K requires a post-processing step to interpolate flow in case of regions with aligned gradients or null gradients.
- H-S required selection of λ , L-K requires selection of window-size.
- H-S is more noise-sensitive, as errors in one part of the image propagate all over. In L-K, the optical flow estimates in different windows are generally independent.
- H-S (without modifications) has no inbuilt reliability estimate unlike L-K

Optical Flow versus True Motion

- Consider a uniform sphere rotating about its axis under constant illumination. Optical flow is zero, true motion is not!
- Consider a still sphere under lighting of varying direction. Optical flow is not zero, true motion is zero!

Structure from Motion

- Consider a video sequence of an object undergoing constant rigid motion (translation/rotation).
- Assume orthographic projections.
- Suppose we tracked some N feature points across all the frames (using optical flow!)
- It turns out you can make some estimates of the 3D coordinates of those points in every frame as well as the motion. This is called structure from ‘motion (SfM)

Task for lab

- Given a video sequence containing two independently moving objects,
- Compute optical flow at each frame,
- Extrapolate the trajectory of salient points on each object.
- Will the images of the objects collide?

Computer vision Simultaneous Localization And Mapping (SLAM) - lecture 12

Adam Szmigiełski

aszmigie@pjwstk.edu.pl

materiały: *ftp(public) : //aszmigie/MWREnglish*

Major trends in robotics

- Seventies - classical (deliberate) approach, accurate models, no uncertainty, no sensing.
- Mid-Eighties: reactive approach, rejection of models, pure reliance on sensors, uncertainty (if any) handled by feedback control.
- Mid-Nineties: Hybrid approaches, reactive at low level (fast decision cycle), deliberate at higher levels. Uses sensing mostly at low level, and models at high level.
- Late Ninties: Probabilistic robotics, different way to integrate models and sensor data.

Introduction

- **Localization** Robot needs to estimate its location with respects to objects in its environment (Map provided).
- **Mapping** Robot need to map the positions of objects that it encounters in its environment (Robot position known)
- **(SLAM)** Robot simultaneously maps objects that it encounters and determines its position (as well as the position of the objects) using noisy sensors.
- **Visual SLAM** - SLAM by using visual sensors such as monocular cameras, stereo rigs, RGB-D cameras, DVS, etc

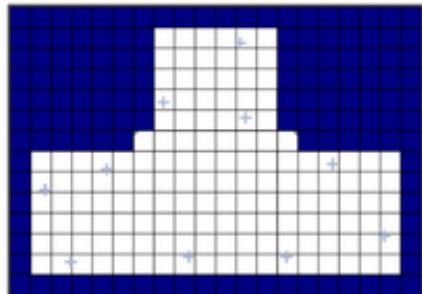
Localization, mapping, SLAM

Tasks	Belief Representation	Probabilistic Models
Localization $P(\text{pose} \mid \text{data})$	Gaussian / Particles	Motion model Measurement model
Mapping $P(\text{map} \mid \text{data})$	Discrete (binary)	Inverse measurement model
SLAM $P(\text{pose, map} \mid \text{data})$	Particles+GridMap (pose, map)	Motion model, measurement model, <i>correspondence model</i>

Three Major Map Models

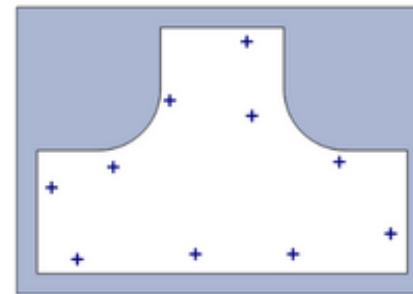
Grid-Based:

Collection of discretized obstacle/free-space pixels



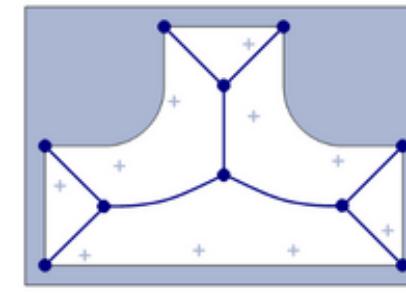
Feature-Based:

Collection of landmark locations and correlated uncertainty



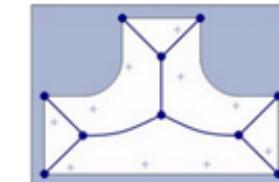
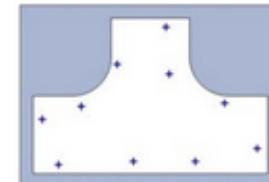
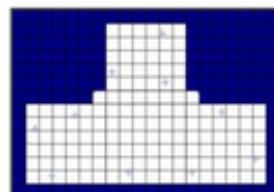
Topological:

Collection of nodes and their interconnections



Three Major Map Models

	Grid-Based	Feature-Based	Topological
Resolution vs. Scale	Discrete localization	Arbitrary localization	Localize to nodes
Computational Complexity	Grid size and resolution	Landmark covariance (N^2)	Minimal complexity
Exploration Strategies	Frontier-based exploration	No inherent exploration	Graph exploration



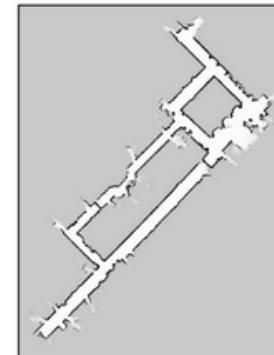
Map Representation Examples

Subway map, city map, landmark-based map

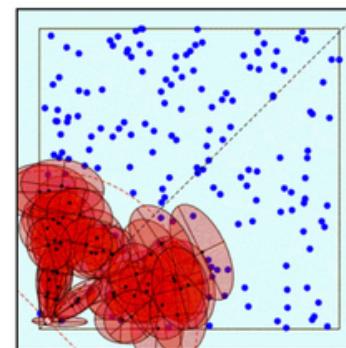
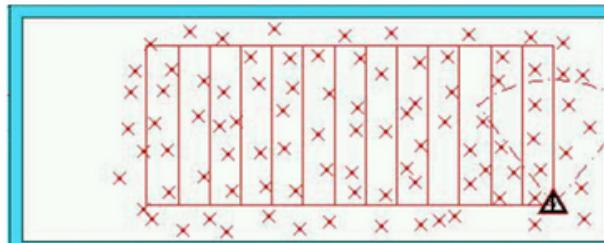


Map Representations in Robotics

- Grid maps or scans, 2d, 3d



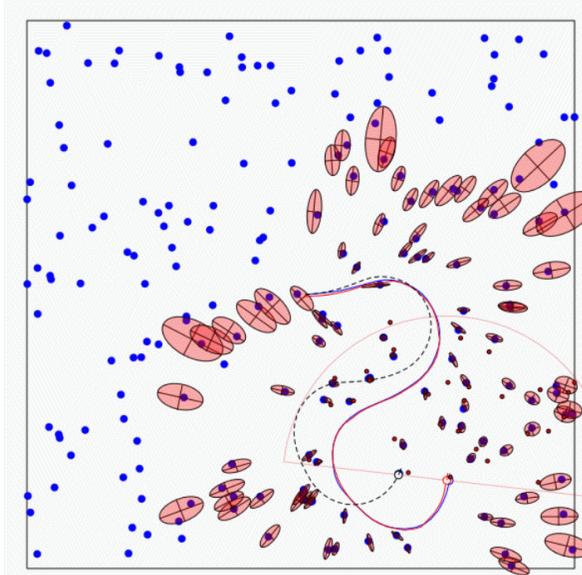
- Landmark-based



SLAM Problem

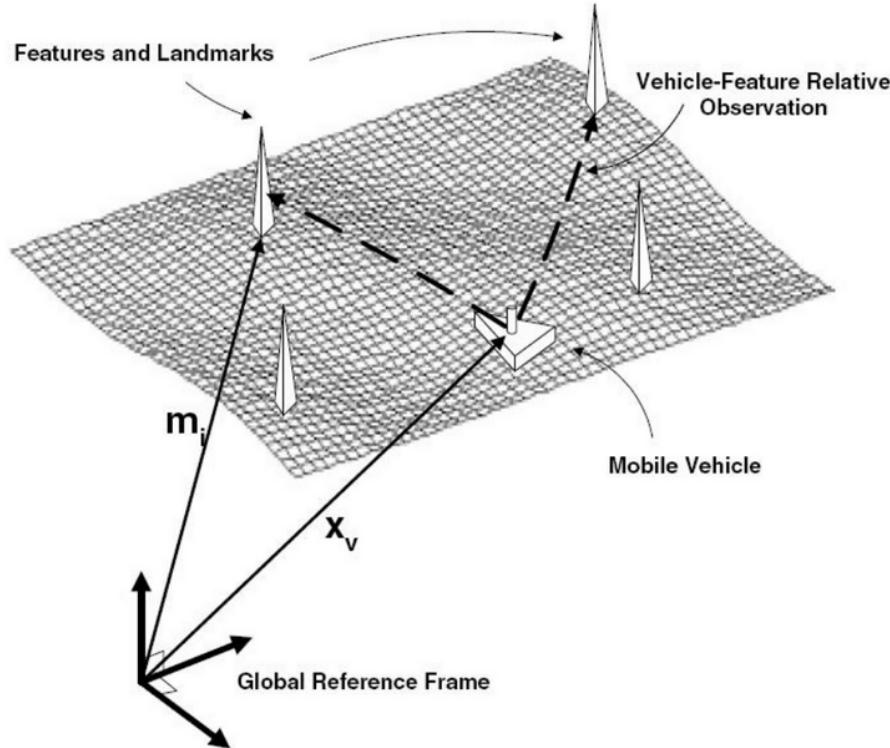
- SLAM is considered a fundamental problem for robots to become truly autonomous
- Large variety of different SLAM approaches have been developed
- The majority uses probabilistic concepts

Feature-Based SLAM



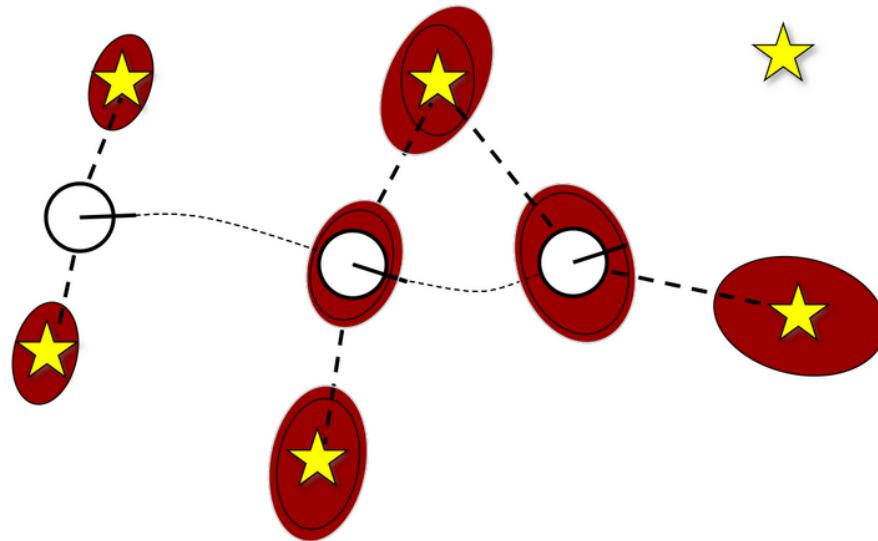
- Given:
 - The robot's controls: $U_{1:k} = \{u_1, u_2, \dots, u_k\}$
 - Relative observations: $Z_{1:k} = \{z_1, z_2, \dots, z_k\}$
- Wanted:
 - Map of features:
 - Path of the robot

Feature-Based SLAM



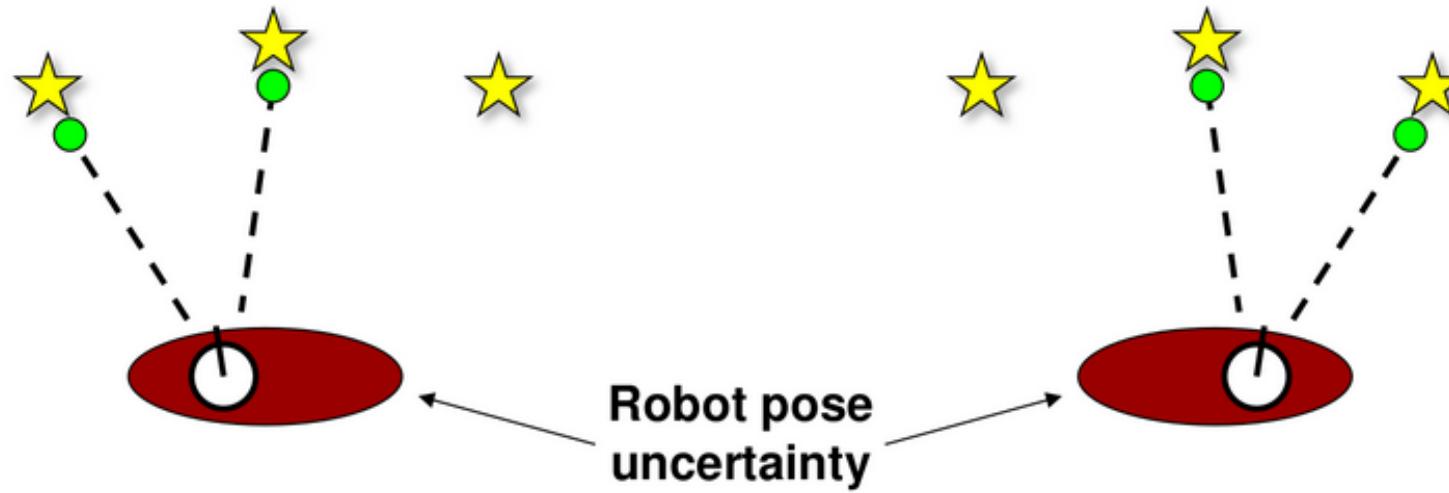
- **Absolute** robot poses,
- **Absolute** landmark positions,
- But only **relative** measurements of landmark.

Why is SLAM a Hard Problem?



- Robot path and map are both **unknown**,
- Errors in map and pose estimates correlated

Why is SLAM a Hard Problem?



- The mapping between observations and landmarks is unknown,
- Picking wrong data associations can have catastrophic consequences (divergence).

SLAM: Simultaneous Localization And Mapping

- Full SLAM:

$$p(x_{0:t}, m | z_{1:t}, u_{1:t})$$

Estimates entire path and map!

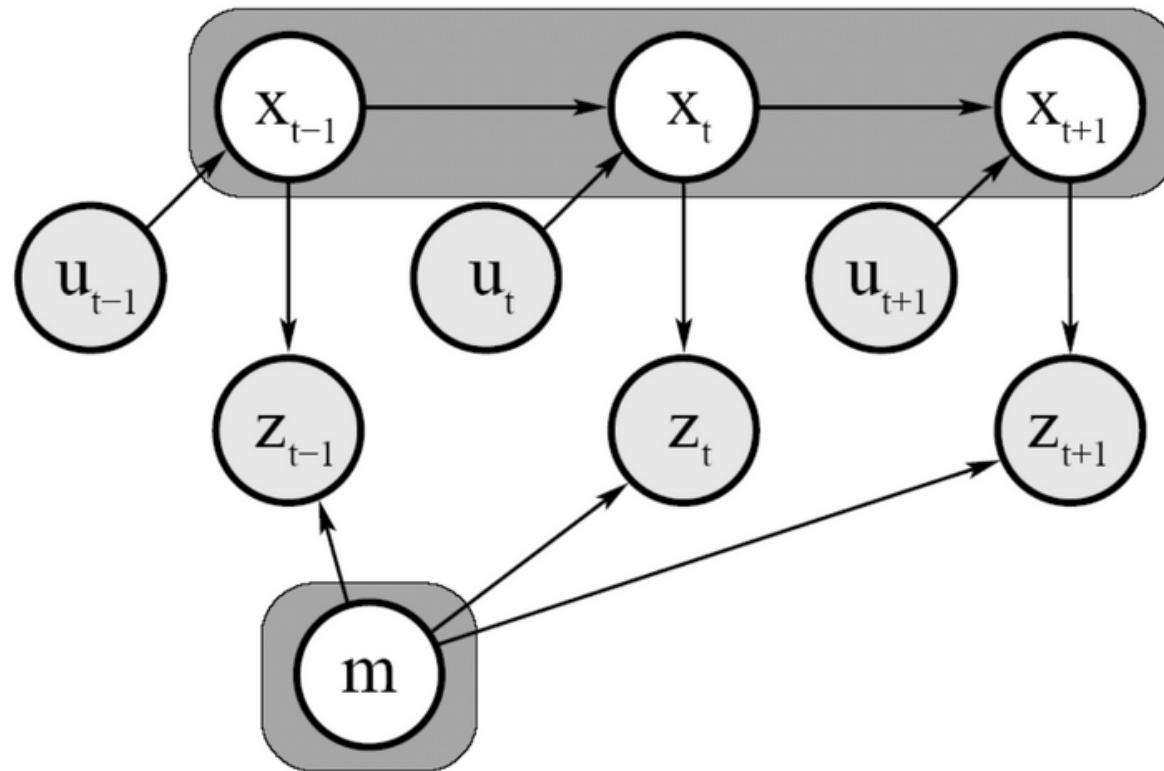
- Online SLAM:

$$p(x_t, m | z_{1:t}, u_{1:t}) = \int \int K \int p(x_{1:t}, m | z_{1:t}, u_{1:t}) dx_1 dx_2 \dots dx_{t-1}$$

Estimates most recent pose and map!

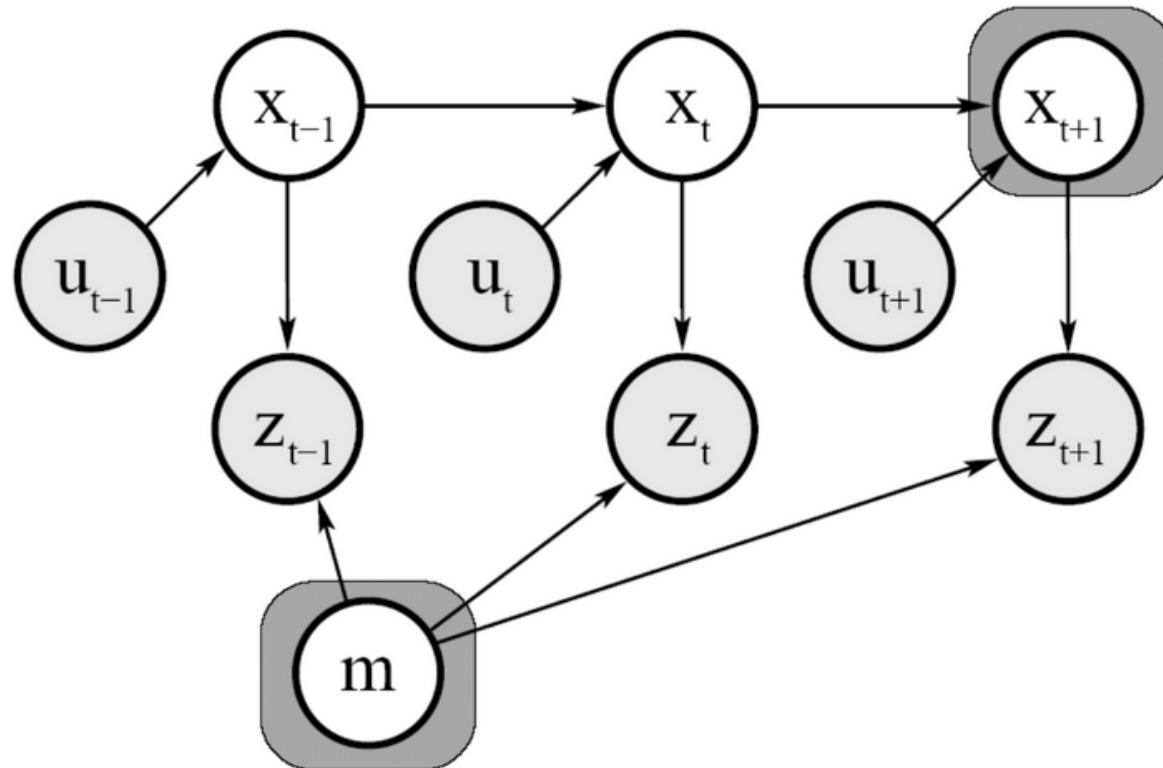
- Integration (marginalization) typically done recursively, one at a time

Graphical Model of Full SLAM



$$p(x_{1:t+1}, m | z_{1:t+1}, u_{1:t+1})$$

Graphical Model of Online SLAM



$$p(x_{t+1}, m | z_{1:t+1}, u_{1:t+1}) = \int \int K \int p(x_{1:t+1}, m | z_{1:t+1}, u_{1:t+1}) dx_1 dx_2 K dx_t$$

EKF SLAM: State representation

- Localization

3x1 pose vector $\mathbf{x}_k = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix}$

3x3 cov. matrix $\Sigma_k = \begin{bmatrix} \sigma_x^2 & \sigma_{xy}^2 & \sigma_{x\theta}^2 \\ \sigma_{yx}^2 & \sigma_y^2 & \sigma_{y\theta}^2 \\ \sigma_{\theta x}^2 & \sigma_{\theta y}^2 & \sigma_\theta^2 \end{bmatrix}$

- SLAM Landmarks simply extend the state. Growing state vector and covariance matrix!

$$\mathbf{x}_k = \begin{bmatrix} \mathbf{x}_R \\ \mathbf{m}_1 \\ \mathbf{m}_2 \\ \vdots \\ \mathbf{m}_n \end{bmatrix} \quad \Sigma_k = \begin{bmatrix} \Sigma_R & \Sigma_{RM_1} & \Sigma_{RM_2} & \cdots & \Sigma_{RM_n} \\ \Sigma_{M_1 R} & \Sigma_{M_1} & \Sigma_{M_1 M_2} & \cdots & \Sigma_{M_1 M_n} \\ \Sigma_{M_2 R} & \Sigma_{M_2 M_1} & \Sigma_{M_2} & \cdots & \Sigma_{M_2 M_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \Sigma_{M_n R} & \Sigma_{M_n M_1} & \Sigma_{M_n M_2} & \cdots & \Sigma_{M_n} \end{bmatrix}$$

EKF SLAM: State representation

Map with n landmarks: $(3+2n)$ -dimensional Gaussian

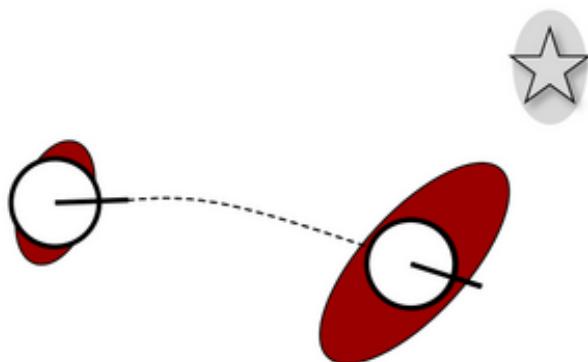
$$\begin{bmatrix} x \\ y \\ \theta \\ m_{1,x} \\ m_{1,y} \\ \vdots \\ m_{n,x} \\ m_{n,y} \end{bmatrix} \underbrace{\left[\begin{array}{ccc|ccccc} \sigma_{xx} & \sigma_{xy} & \sigma_{x\theta} & \sigma_{xm_{1,x}} & \sigma_{xm_{1,y}} & \cdots & \sigma_{xm_{n,x}} & \sigma_{xm_{n,y}} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{y\theta} & \sigma_{ym_{1,x}} & \sigma_{ym_{1,y}} & \cdots & \sigma_{m_{n,x}} & \sigma_{m_{n,y}} \\ \sigma_{\theta x} & \sigma_{\theta y} & \sigma_{\theta\theta} & \sigma_{\theta m_{1,x}} & \sigma_{\theta m_{1,y}} & \cdots & \sigma_{\theta m_{n,x}} & \sigma_{\theta m_{n,y}} \\ \hline \sigma_{m_{1,x}x} & \sigma_{m_{1,x}y} & \sigma_{\theta} & \sigma_{m_{1,x}m_{1,x}} & \sigma_{m_{1,x}m_{1,y}} & \cdots & \sigma_{m_{1,x}m_{n,x}} & \sigma_{m_{1,x}m_{n,y}} \\ \sigma_{m_{1,y}x} & \sigma_{m_{1,y}y} & \sigma_{\theta} & \sigma_{m_{1,y}m_{1,x}} & \sigma_{m_{1,y}m_{1,y}} & \cdots & \sigma_{m_{1,y}m_{n,x}} & \sigma_{m_{1,y}m_{n,y}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \sigma_{m_{n,x}x} & \sigma_{m_{n,x}y} & \sigma_{\theta} & \sigma_{m_{n,x}m_{1,x}} & \sigma_{m_{n,x}m_{1,y}} & \cdots & \sigma_{m_{n,x}m_{n,x}} & \sigma_{m_{n,x}m_{n,y}} \\ \sigma_{m_{n,y}x} & \sigma_{m_{n,y}y} & \sigma_{\theta} & \sigma_{m_{n,y}m_{1,x}} & \sigma_{m_{n,y}m_{1,y}} & \cdots & \sigma_{m_{n,y}m_{n,x}} & \sigma_{m_{n,y}m_{n,y}} \end{array} \right]}_{\Sigma}$$

Can handle hundreds of dimensions

EKF SLAM: Filter Cycle

1. State prediction (odometry)
2. Measurement prediction
3. Measurement
4. Data association
5. Update
6. Integration of new landmarks

EKF SLAM: State Prediction



Odometry:

$$\hat{\mathbf{x}}_R = f(\mathbf{x}_R, \mathbf{u})$$

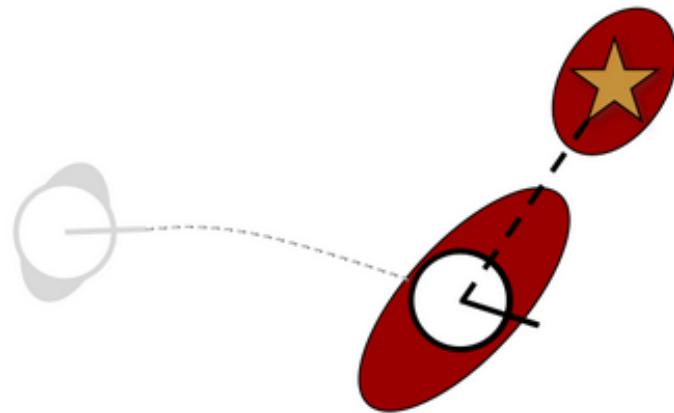
$$\hat{\Sigma}_R = F_x \Sigma_R F_x^T + F_u U F_u^T$$

Robot-landmark cross-covariance prediction:

$$\hat{\Sigma}_{RM_i} = F_x \Sigma_{RM_i}$$

$$\underbrace{\begin{bmatrix} \mathbf{x}_R \\ \mathbf{m}_1 \\ \vdots \\ \mathbf{m}_n \end{bmatrix}}_{\mu} \quad \underbrace{\begin{bmatrix} \Sigma_R & \Sigma_{RM_1} & \dots & \Sigma_{RM_n} \\ \Sigma_{M_1 R} & \Sigma_{M_1} & \dots & \Sigma_{M_1 M_n} \\ \vdots & \ddots & & \vdots \\ \Sigma_{M_n R} & \Sigma_{M_n M_1} & \dots & \Sigma_{M_n} \end{bmatrix}}_{\Sigma}$$

EKF SLAM: Measurement Prediction



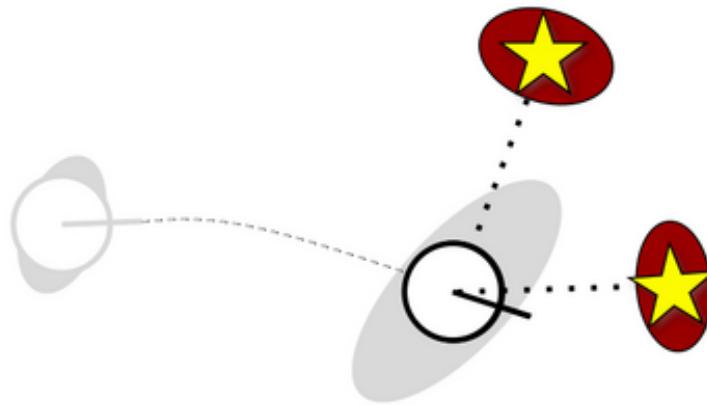
Global-to-local
frame transform h

$$\hat{\mathbf{z}}_k = h(\hat{\mathbf{x}}_k)$$

$$\underbrace{\begin{bmatrix} \mathbf{x}_R \\ \mathbf{m}_1 \\ \vdots \\ \mathbf{m}_n \end{bmatrix}}_{\mu} \quad \underbrace{\begin{bmatrix} \Sigma_R & \Sigma_{RM_1} & \dots & \Sigma_{RM_n} \\ \Sigma_{M_1 R} & \Sigma_{M_1} & \dots & \Sigma_{M_1 M_n} \\ \vdots & \vdots & \ddots & \vdots \\ \Sigma_{M_n R} & \Sigma_{M_n M_1} & \dots & \Sigma_{M_n} \end{bmatrix}}_{\Sigma}$$

EKF SLAM: Obtained Measurement

(x,y) -point landmarks

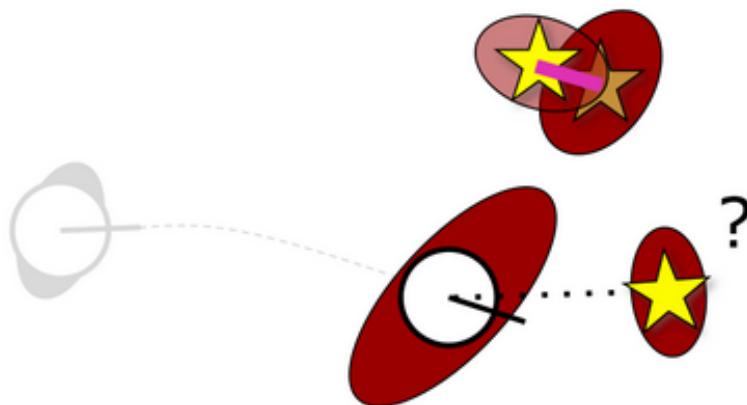


$$\mathbf{z}_k = \begin{bmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \end{bmatrix}$$

$$R_k = \begin{bmatrix} R_1 & 0 \\ 0 & R_2 \end{bmatrix}$$

$$\underbrace{\begin{bmatrix} \mathbf{x}_R \\ \mathbf{m}_1 \\ \vdots \\ \mathbf{m}_n \end{bmatrix}}_{\mu} \quad \underbrace{\begin{bmatrix} \Sigma_R & \Sigma_{RM_1} & \dots & \Sigma_{RM_n} \\ \Sigma_{M_1R} & \Sigma_{M_1} & \dots & \Sigma_{M_1M_n} \\ \vdots & \vdots & \ddots & \vdots \\ \Sigma_{M_nR} & \Sigma_{M_nM_1} & \dots & \Sigma_{M_n} \end{bmatrix}}_{\Sigma}$$

EKF SLAM: Data Association

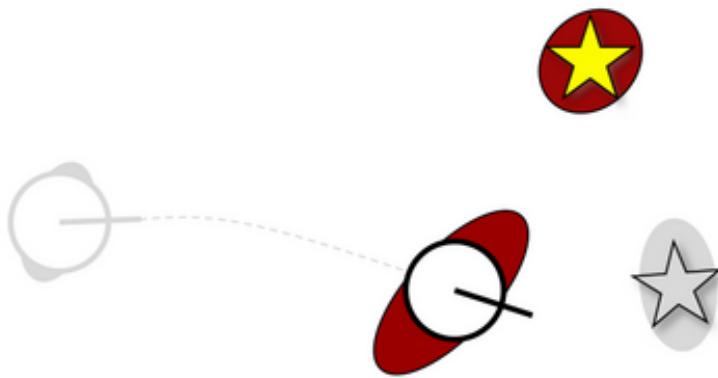


Associates predicted measurements $\hat{\mathbf{z}}_k^i$ with observation \mathbf{z}_k^j

$$\begin{aligned}\nu_k^{ij} &= \mathbf{z}_k^j - \hat{\mathbf{z}}_k^i \\ S_k^{ij} &= R_k^j + H^i \hat{\Sigma}_k H^{iT}\end{aligned}$$

$$\underbrace{\begin{bmatrix} \mathbf{x}_R \\ \mathbf{m}_1 \\ \vdots \\ \mathbf{m}_n \end{bmatrix}}_{\mu} \quad \underbrace{\begin{bmatrix} \Sigma_R & \Sigma_{RM_1} & \dots & \Sigma_{RM_n} \\ \Sigma_{M_1 R} & \Sigma_{M_1} & \dots & \Sigma_{M_1 M_n} \\ \vdots & \vdots & \ddots & \vdots \\ \Sigma_{M_n R} & \Sigma_{M_n M_1} & \dots & \Sigma_{M_n} \end{bmatrix}}_{\Sigma}$$

EKF SLAM: Update Step



The usual Kalman filter expressions

$$K_k = \hat{\Sigma}_k H^T S_k^{-1}$$

$$\mathbf{x}_k = \hat{\mathbf{x}}_k + K_k \nu_k$$

$$C_k = (I - K_k H) \hat{\Sigma}_k$$

$$\underbrace{\begin{bmatrix} \mathbf{x}_R \\ \mathbf{m}_1 \\ \vdots \\ \mathbf{m}_n \end{bmatrix}}_{\mu} \quad \underbrace{\begin{bmatrix} \Sigma_R & \Sigma_{RM_1} & \dots & \Sigma_{RM_n} \\ \Sigma_{M_1 R} & \Sigma_{M_1} & \dots & \Sigma_{M_1 M_n} \\ \vdots & \vdots & \ddots & \vdots \\ \Sigma_{M_n R} & \Sigma_{M_n M_1} & \dots & \Sigma_{M_n} \end{bmatrix}}_{\Sigma}$$

EKF SLAM: New Landmarks



State augmented by

$$\mathbf{m}_{n+1} = g(\mathbf{x}_R, \mathbf{z}_j)$$

$$\Sigma_{M_{n+1}} = G_R \Sigma_R G_R^T + G_z R_j G_z^T$$

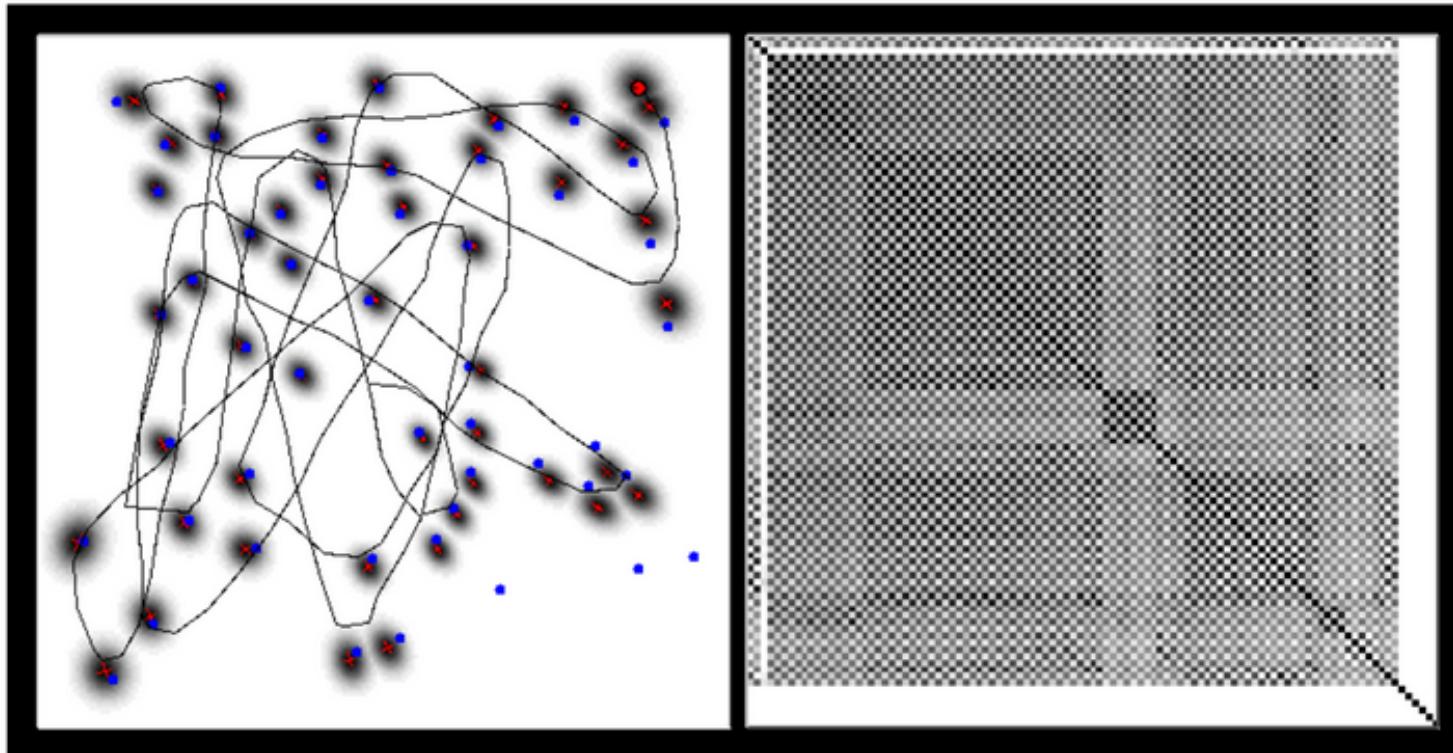
Cross-covariances:

$$\Sigma_{M_{n+1} M_i} = G_R \Sigma_{RM_i}$$

$$\Sigma_{M_{n+1} R} = G_R \Sigma_R$$

$$\left[\begin{array}{c} \mathbf{x}_R \\ \mathbf{m}_1 \\ \vdots \\ \mathbf{m}_n \\ \mathbf{m}_{n+1} \end{array} \right] \left[\begin{array}{ccccc} \Sigma_R & \Sigma_{RM_1} & \dots & \Sigma_{RM_n} & \Sigma_{RM_{n+1}} \\ \Sigma_{M_1 R} & \Sigma_{M_1} & \dots & \Sigma_{M_1 M_n} & \Sigma_{M_1 M_{n+1}} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \Sigma_{M_n R} & \Sigma_{M_n M_1} & \dots & \Sigma_{M_n} & \Sigma_{M_n M_{n+1}} \\ \Sigma_{M_{n+1} R} & \Sigma_{M_{n+1} M_1} & \dots & \Sigma_{M_{n+1} M_n} & \Sigma_{M_{n+1}} \end{array} \right] \underbrace{\Sigma}_{\mu}$$

EKF SLAM



Map

Correlation matrix

EKF SLAM: Correlations Matter

- What if we neglected cross-correlations?

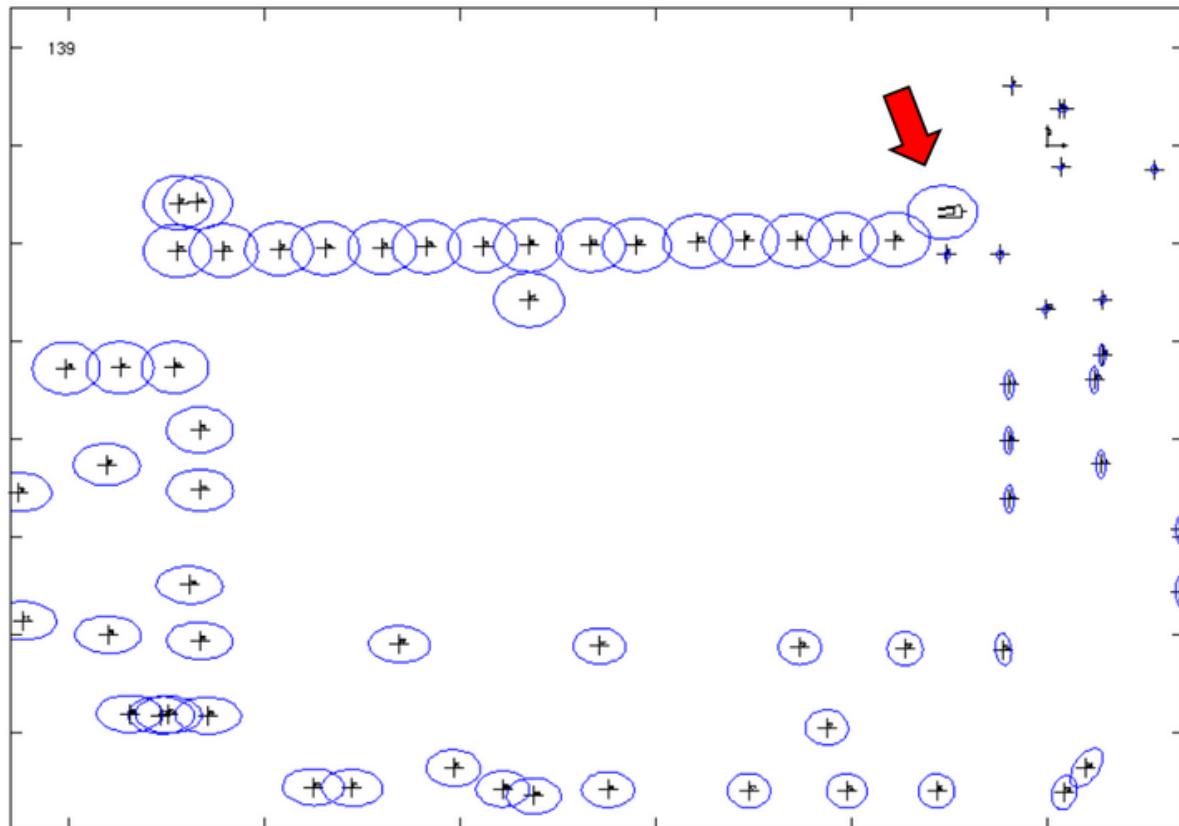
$$\Sigma_k = \begin{bmatrix} \Sigma_R & 0 & \cdots & 0 \\ 0 & \Sigma_{M_1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \Sigma_{M_n} \end{bmatrix} \quad \Sigma_{RM_i} = \mathbf{0}_{3 \times 2}$$
$$\Sigma_{M_i M_{i+1}} = \mathbf{0}_{2 \times 2}$$

- Landmark and robot uncertainties would become overly optimistic
- Data association would fail
- Multiple map entries of the same landmark
- Inconsistent map

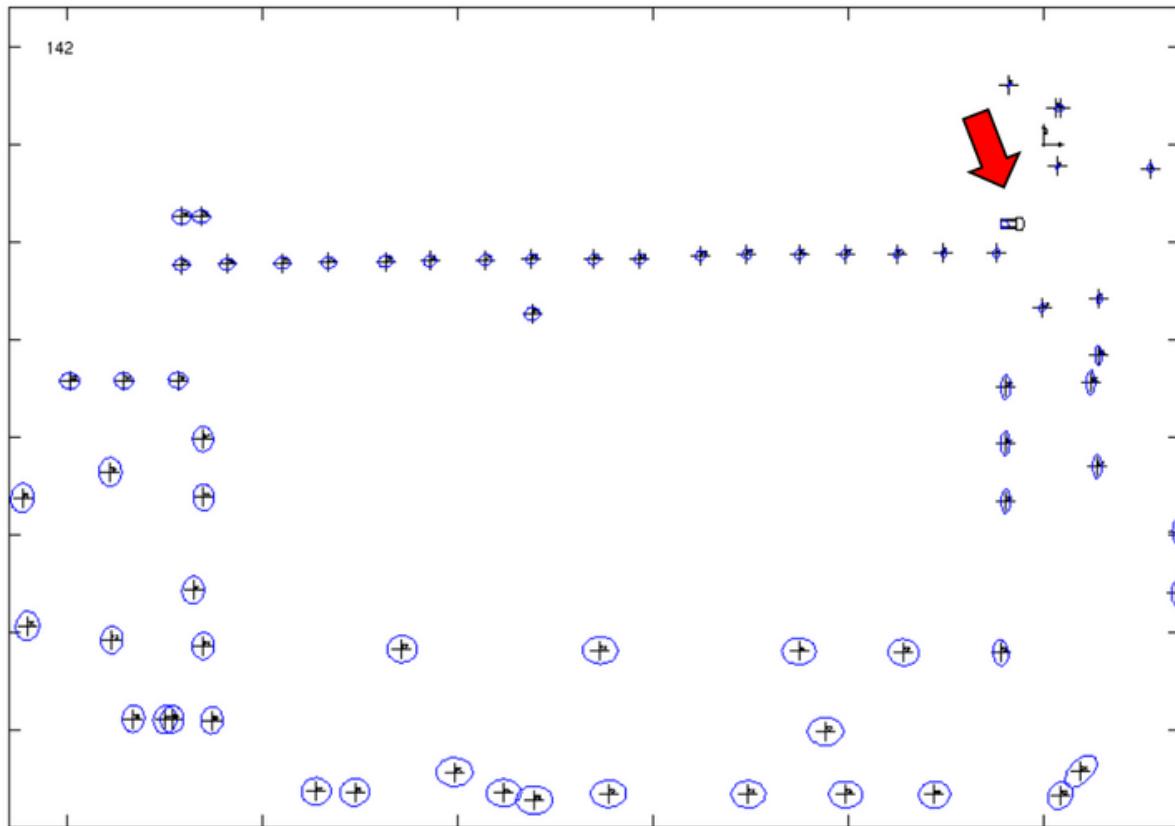
SLAM: Loop Closure

- Recognizing an already mapped area, typically after a long exploration path (the robot “closes a loop”)
- Structurally identical to data association, but
 - high levels of ambiguity
 - possibly useless validation gates
 - environment symmetries
- Uncertainties collapse after a loop closure (whether the closure was correct or not)

SLAM: Loop Closure - Before loop closure:



SLAM: Loop Closure - After loop closure



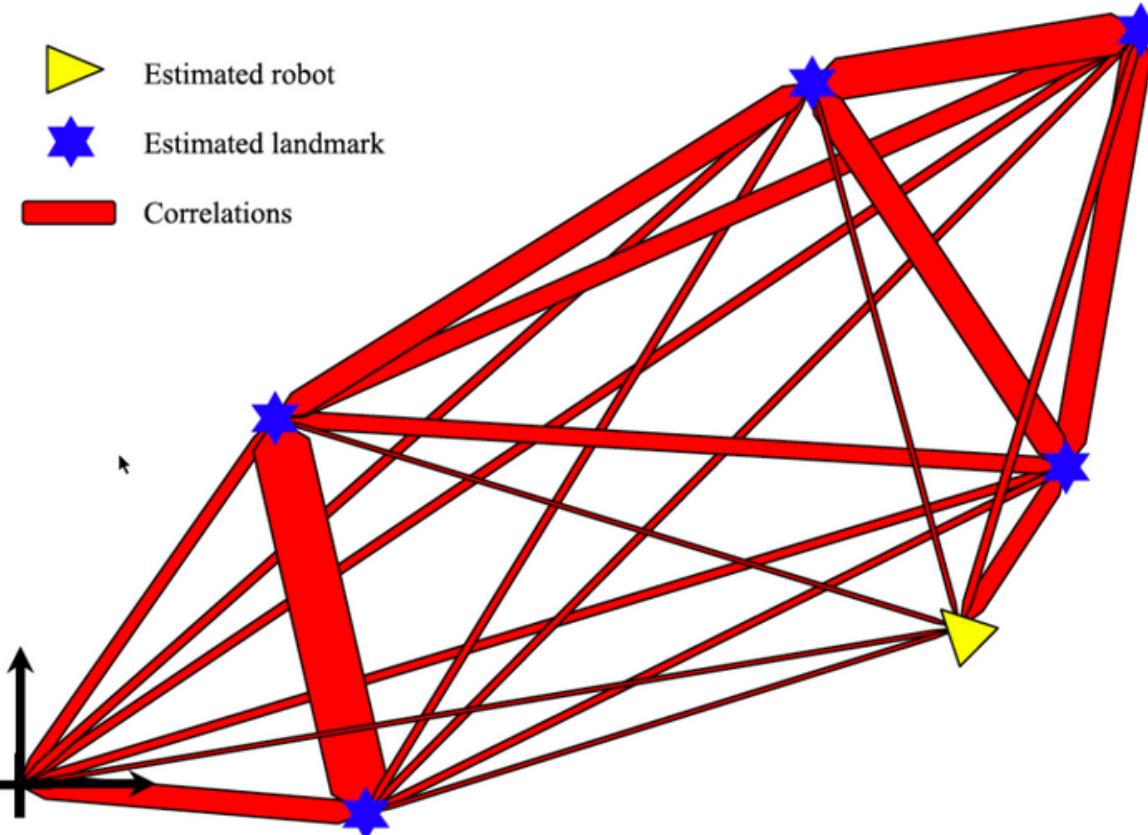
SLAM: Loop Closure

- By revisiting already mapped areas, uncertainties in robot and landmark estimates can be reduced
- This can be exploited when exploring an environment for the sake of better (e.g. more accurate) maps
- Exploration: the problem of where to acquire new information

KF-SLAM Properties (Linear Case)

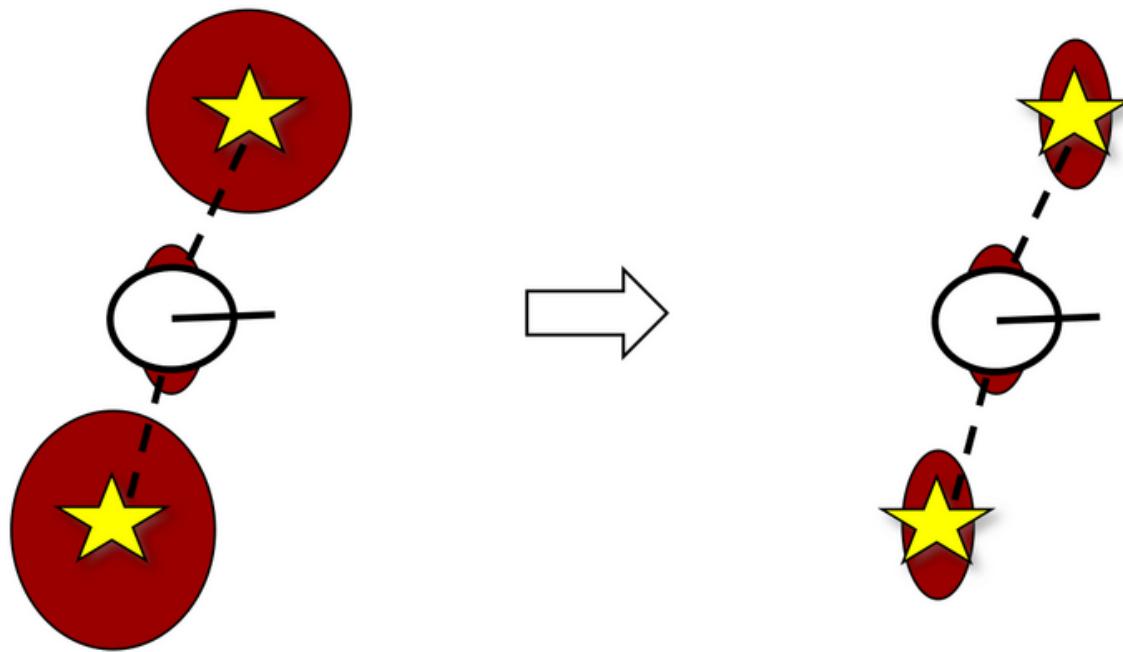
- The determinant of any sub-matrix of the map covariance matrix decreases monotonically as successive observations are made
- When a new landmark is initialized, its uncertainty is maximal
- Landmark uncertainty decreases monotonically with each new observation

KF-SLAM Properties (Linear Case)



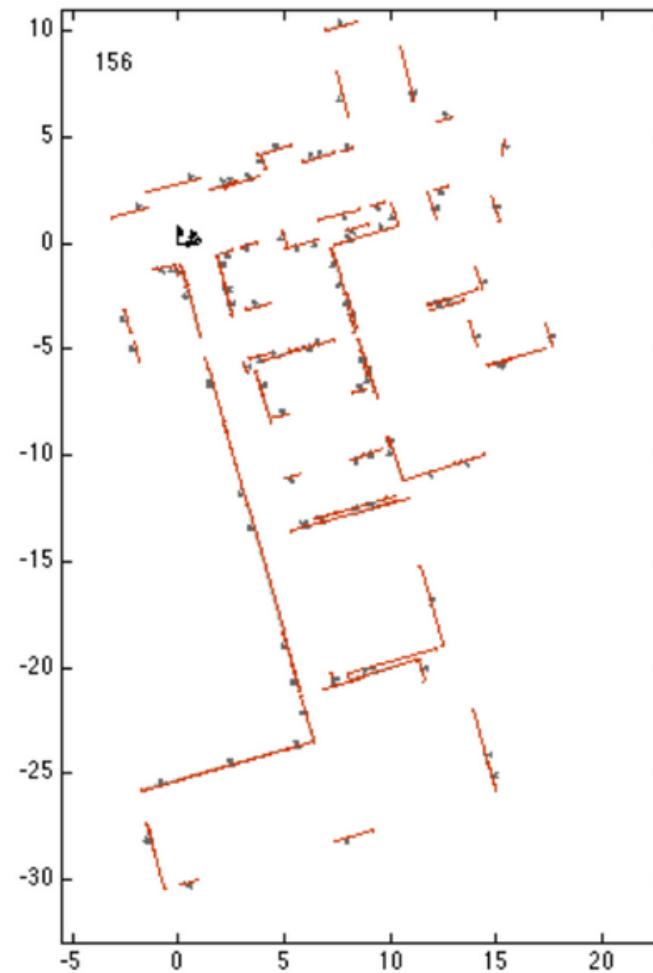
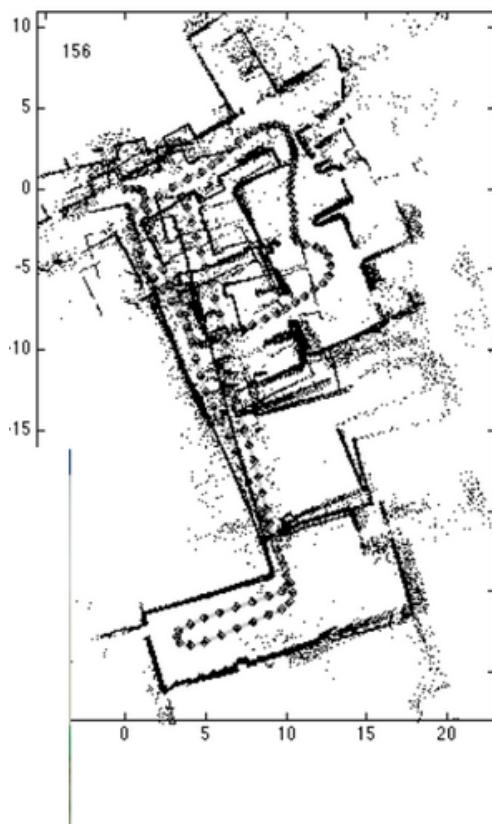
In the limit, the landmark estimates become fully correlated

KF-SLAM Properties (Linear Case)



In the limit, the covariance associated with any single landmark location estimate is determined only by the initial covariance in the vehicle location estimate.

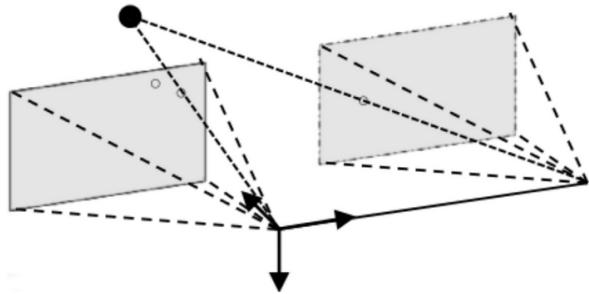
EKF SLAM Example: Line Features



Visual SLAM

- Quality open source systems: LSD-SLAM, ORB-SLAM, SVO, KinectFusion, ElasticFusion
- Commercial products and prototypes: Google Tango, Hololens, Dyson 360 Eye, Roomba 980
- SLAM continues and evolves into generic real-time 3D perception research

Visual Odometry (VO) - Problem Formulation



- An agent is moving through the environment and taking images with a rigidly-attached camera system at discrete times k
- In case of a monocular system, the set of images taken at times k is denoted by

$$I_{l,0:n} = \{I_0, \dots, I_n\}$$

- In case of a stereo system, the set of images taken at times k :

$$I_{l,0:n} = \{I_{l,0}, \dots, I_{l,n}\}$$

$$I_{r,0:n} = \{I_{r,0}, \dots, I_{r,n}\}$$

The coordinate system of the left camera can be used as the origin

Visual Odometry (VO) - Problem Formulation

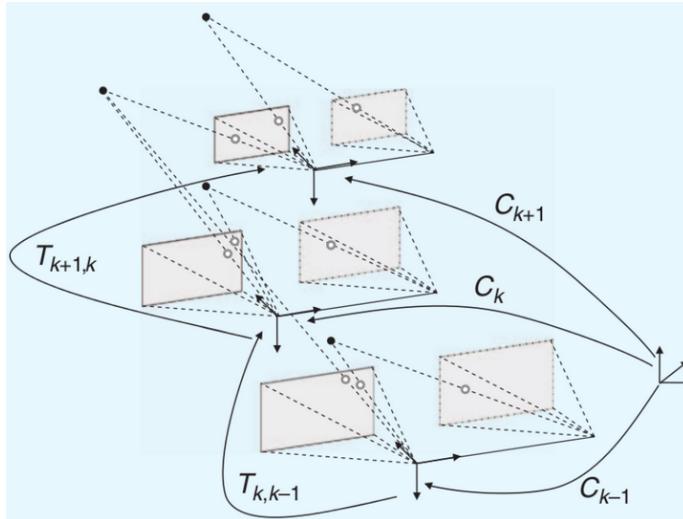


- In case of a RGB-D camera, the set of images taken at times k is denoted by

$$I_{0:n} = \{I_0, \dots, I_n\}, D_{0:n} = \{D_0, \dots, D_n\}$$

- Two camera positions at adjacent time instants $k - 1$ and k are related by the rigid body transformation $T_k = \begin{bmatrix} R_{k-1,k} & t_{k-1,k} \\ 0 & 1 \end{bmatrix}$
- The set $T_{1:n} = \{T_1, \dots, T_n\}$ contains all the subsequent motions

- The set of camera pose $C_{0:n} = \{C_0, \dots, C_n\}$ contains the transformations of the camera - initial coordinate frame at $k = 0$

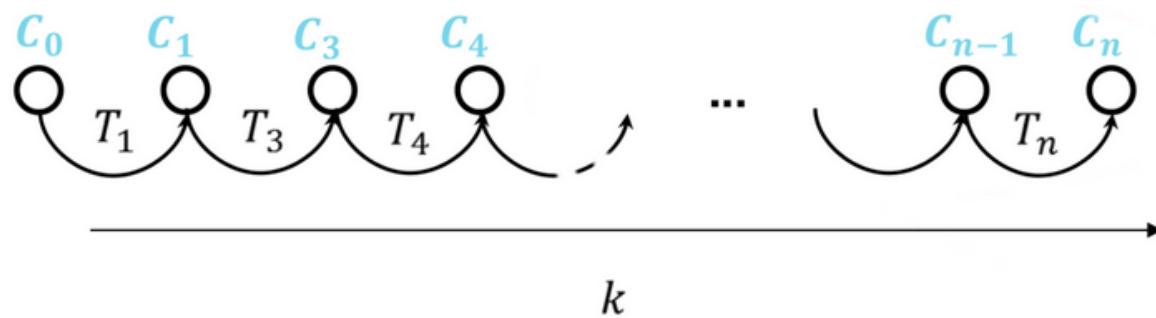


- The current camera pose C_n can be computed by concatenating all the transformations $T_{1:k}$, therefore

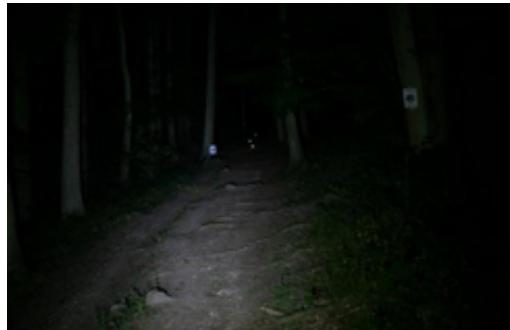
$$C_n = C_{n-1} T_n$$

with C_0 being the camera pose at the instant $k = 0$, which can be arbitrarily set by the user

- The main task of VO is to compute the relative transformations T_k from images I_k and I_{k-1} and then to concatenate these transformation to recover the full trajectory $C_{0:n}$ of the camera
- This means that VO recovers the path incrementally, pose after pose:



Visual Odometry (VO) - Assumptions



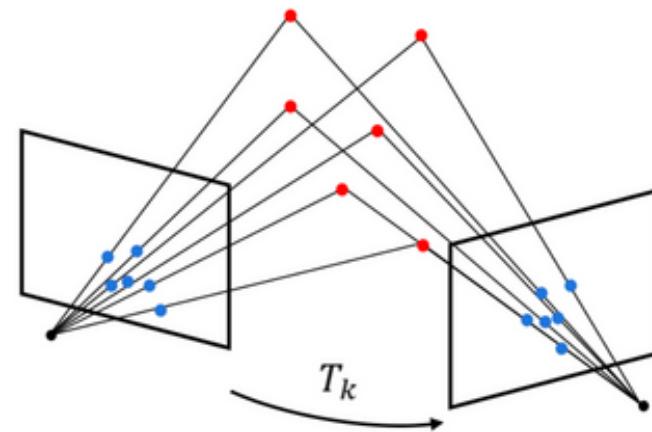
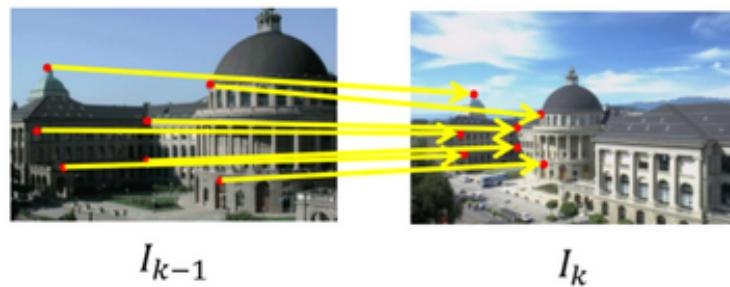
Usual assumptions about the environment:

- Sufficient illumination in the environment
- Dominance of static scene over moving objects
- Enough texture to allow apparent motion to be extracted
- Sufficient scene overlap between consecutive frames

Advantages of Visual odometry

- Contrary to wheel odometry, VO is not affected by wheel slip in uneven terrain or other adverse conditions.
- More accurate trajectory estimates compared to wheel odometry (relative position error 0.1)
- VO can be used as a complement to wheel odometry
 - GPS
 - inertial measurement units (IMUs)
 - laser odometry
- In GPS-denied environments, such as underwater and aerial, VO has utmost importance

Visual odometry (VO) feature-based



- Feature detection
- Feature matching/tracking
- Motion estimation
- Local optimization

Visual Odometry Pipeline

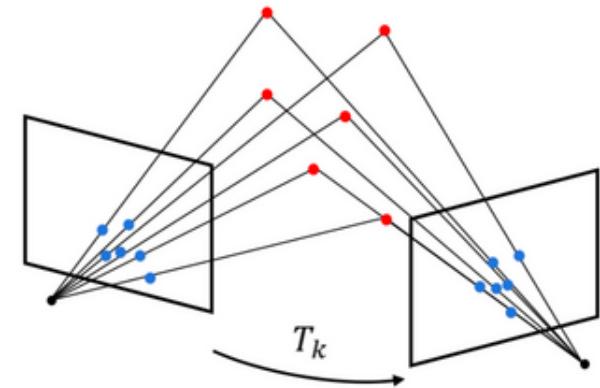
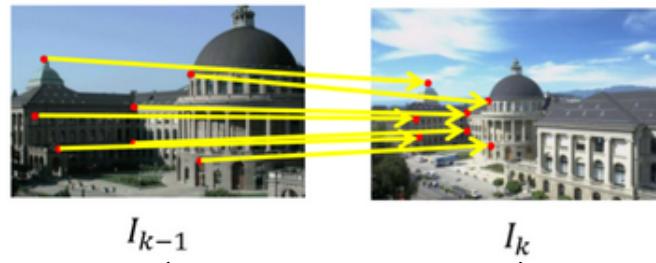
Visual odometry (VO) feature-based: Assumption: camera is well calibrated.

- **Feature detection:** Detect a set of features f_k at time k
(General idea: extract high-contrast areas in the image)
- **Feature matching/Feature tracking** Find correspondences between set of features f_{k-1}, f_k
 - *tracking*: locally search each feature (e.g. by prediction and correlation)
 - *matching*: independently detect features in each image and find correspondences on the basis of a similarity metric
(exploit descriptors such SURF, SIFT, ORB, etc)
- **Motion estimation** Compute transformation T_k between two

images I_{k-1} and I_k from two sets of corresponding features f_{k-1} , f_k . Different algorithms depending on available sensor data:

- An iterative refinement over last m poses can be optionally performed after motion estimation to obtain a more accurate estimate of the local trajectory

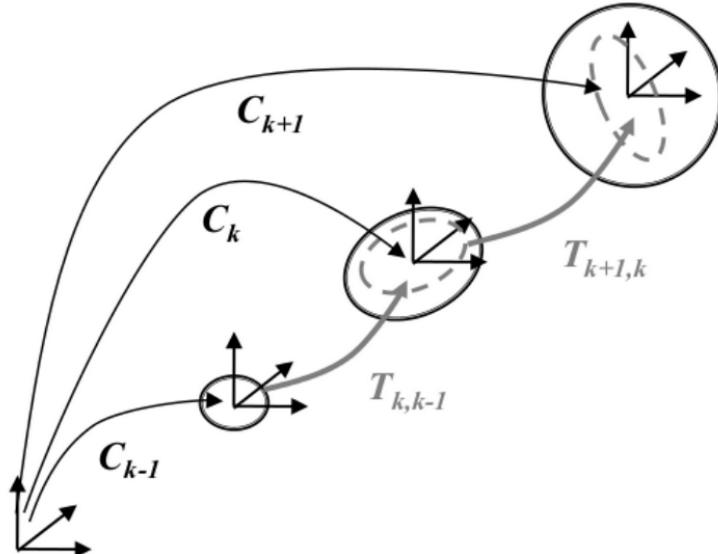
Visual Odometry Pipeline



VO from 2-D to 2-D (feature-based)

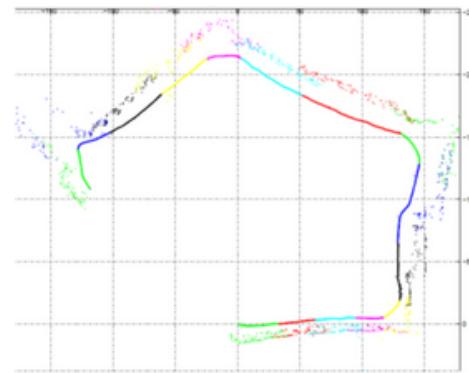
1. Capture new frame I_k
2. Extract and match features between I_{k-1} and I_k
3. Compute essential matrix for image pair I_{k-1} , I_k
4. Decompose essential matrix into R_k and t_k , and form T_k
5. Compute relative scale and rescale t_k accordingly
6. Concatenate transformation by computing $C_k = C_{k-1}T_k$
7. Repeat from 1.

Visual Odometry Drift

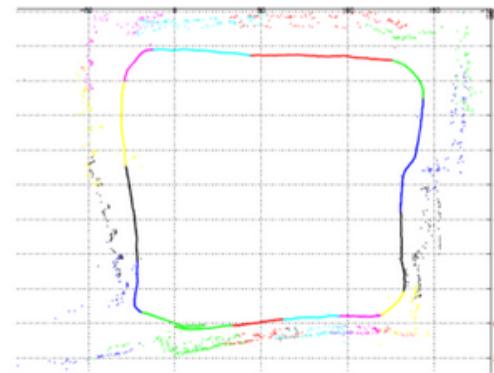


- The errors introduced by each new frame-to-frame motion accumulate over time
- This generates a drift of the estimated trajectory from the real one
- Uncertainty of the camera pose at C_k is a combination of the uncertainty at C_{k-1} (black solid ellipse) and the uncertainty of the transformation $T_{k,k-1}$ (gray dashed ellipse)

Visual SLAM



Before loop closing



After loop closing

- The goal of SLAM in general is to obtain a global and consistent estimate of the robot path and the map. This is done by identifying loop closures.
- When a loop closure is detected, this information is used to reduce the drift in both the map and camera path (global bundle adjustment)
- Conversely, VO aims at recovering a path incrementally, pose after pose, It can potentially use optimization only over the last m pose path (windowed bundle adjustment)

VO vs Visual SLAM

- VO only aims at the local consistency of the trajectory,
- SLAM aims to the global consistency of the trajectory and of the map
- VO can be used as building block of SLAM
- VO is SLAM before closing the loop
- The choice between VO and V-SLAM depends on the tradeoff between performance, consistency and simplicity of implementation
- VO trades off consistency for real-time performance, without the need to keep track of all the previous history of the camera

SLAM Techniques

- EKF SLAM
- FastSLAM
- Graph-based SLAM
- Topological SLAM (mainly place recognition)
- Scan Matching / Visual Odometry (only locally consistent maps)
- Approximations for SLAM: Local submaps, Sparse extended information filters, Sparse links, Thin junction tree filters, etc.