Computer Vision

Department of Computer Engineering
Razi University

Dr. Abdolhossein Fathi

References and Evaluation

- **Textbook:**
  - D.A. Forsyth and J. Ponce, "Computer Vision: A Modern Approach", Prentice Hall, 2012.
  - E.R. Davies, "Computer and Machine Vision: Theory, Algorithms, Practicalities", Academic Press, 2012.
  - R. Szeliski "Computer Vision: Algorithms and Applications", Springer, 2010.
  - M. Sonka, V. Hlavac, R. Boy, "Image Processing. Analysis. and Machine Vision", Third Edition, Tomson, 2008.

- **Score Details:**
  - Presentation 15%
  - Project 15%
  - Exercise 10%
  - Final Exam 60%

Contents

- Introduction to Computer Vision
- Principles of Image Formation
- Light and Color
- Image Filtering
- Local Features Analysis
  - Local Features Descriptor (Review)
- Texture
- Segmentation and Clustering
- Object Detection and Tracking
- Stereo Vision
- Learning and Classification (review)

Razi University

Introduction to Computer Vision

What is computer vision?

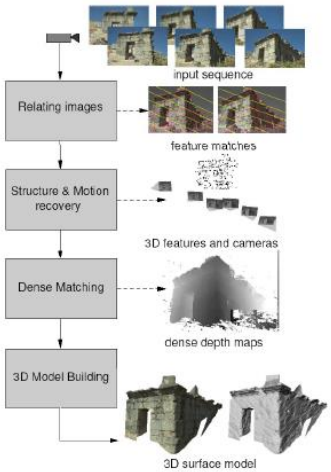
- **Automatic understanding of images and video**
  - **Computing properties of the 3D world from visual data *(measurement)***
  - **Algorithms and representations to allow a machine to recognize objects, people, scenes, and activities. *(perception and interpretation)***

Vision for measurement

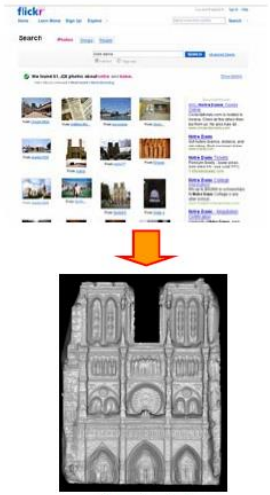
Real-time stereo

Pollefeys et al.

Structure from motion

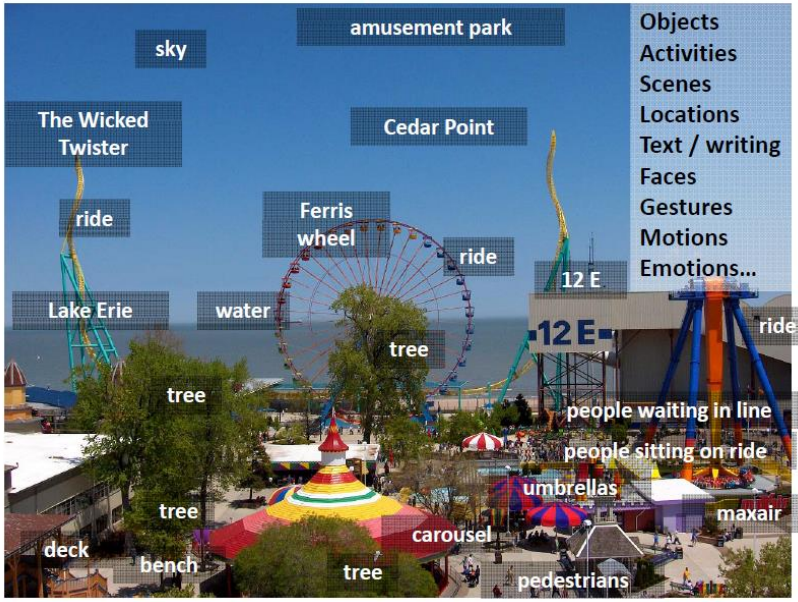


Multi-view stereo for community photo collections

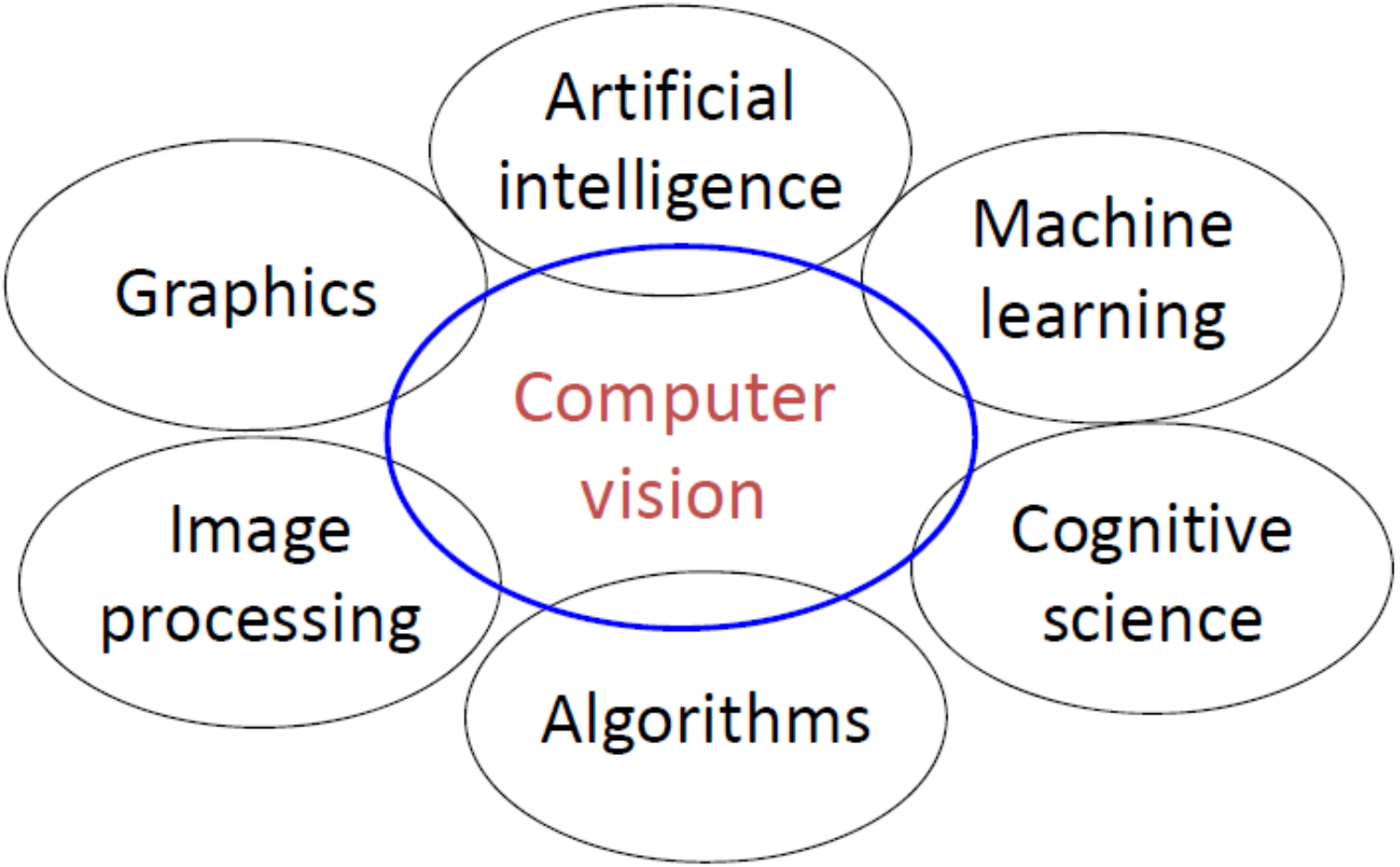


Goesele et al.

Vision for perception, interpretation



Related Disciplines



Why Vision?

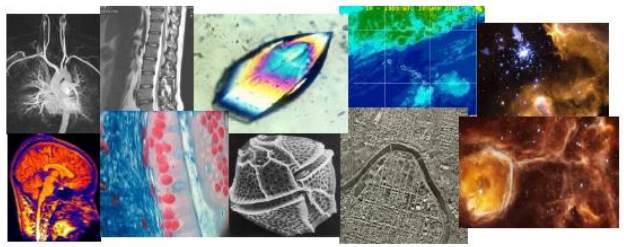
- **As image sources multiply, so do applications**
  - **Relieve humans of boring, easy tasks**
  - **Enhance human abilities: human-computer interaction, visualization**
  - **Perception for robotics / autonomous agents**
  - **Organize and give access to visual content**

Personal photo albums

Movies, news, sports

- Images and video are everywhere!

Surveillance and security



Medical and scientific images

Why Vision?

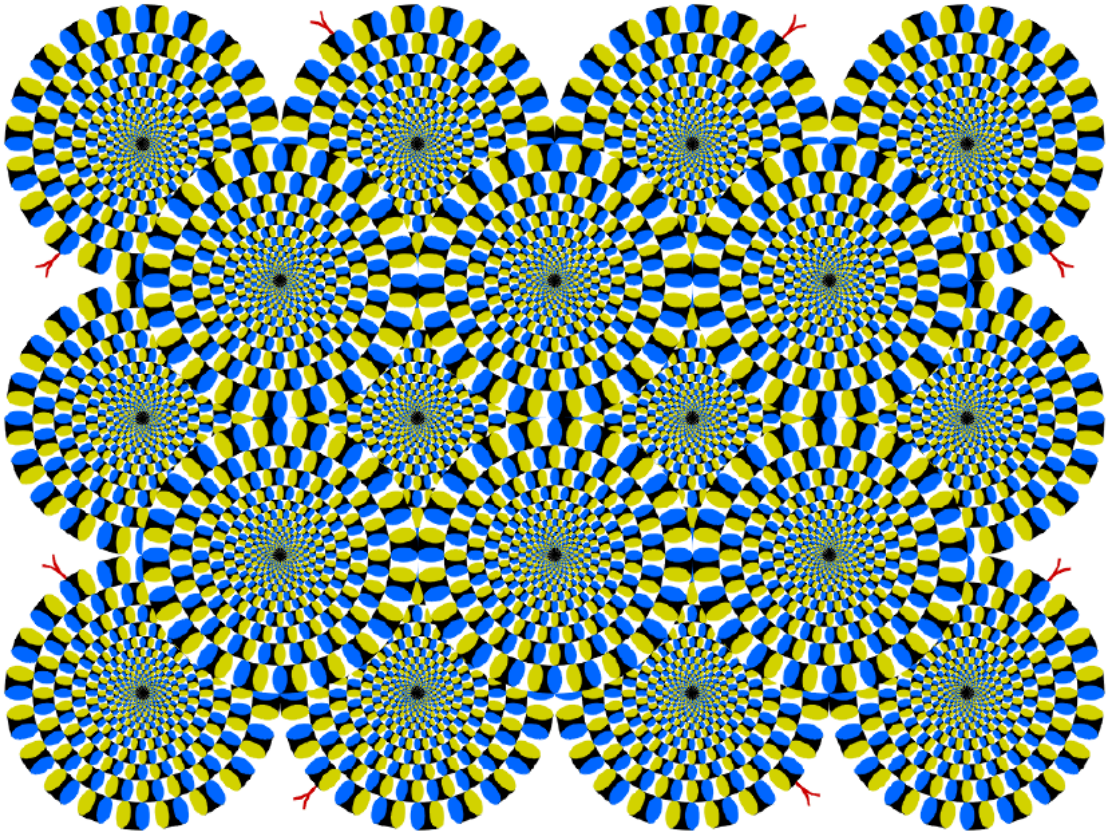
- **Every picture tells a story**
  - **Goal of computer vision is to write computer programs that can interpret images.**

- **Can computers match (or beat) human vision?**
  - **Yes and no (but mostly no!)**
    - **humans are much better at "hard" things**
    - **computers can be better at "easy" things**

Why Vision?

- **Human perception has its shortcomings…**

Sinha and Poggio, *Nature*, 1996



Three Stages of Computer Vision

- low-level image → image
- mid-level image → features
- high-level features → analysis

Low-Level

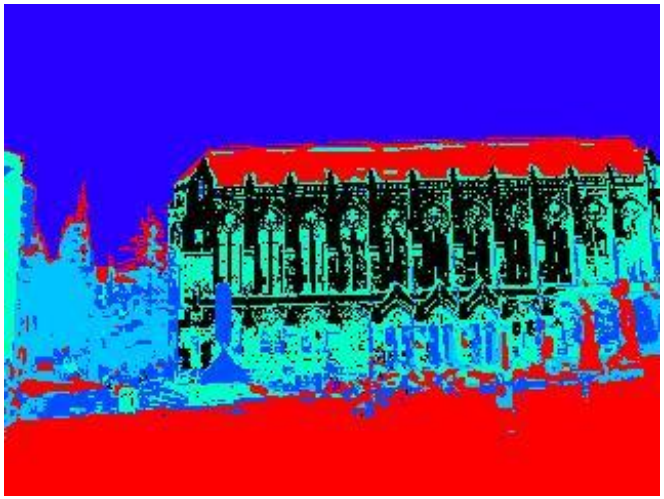
sharpening

blurring

Mid-level

original color image

K-means clustering (followed by connected component analysis)



regions of homogeneous color

data structure

Low-Level

original image

Canny

edge image

Mid-Level

edge image

ORT

data structure

circular arcs and line segments

Low- to High-Level

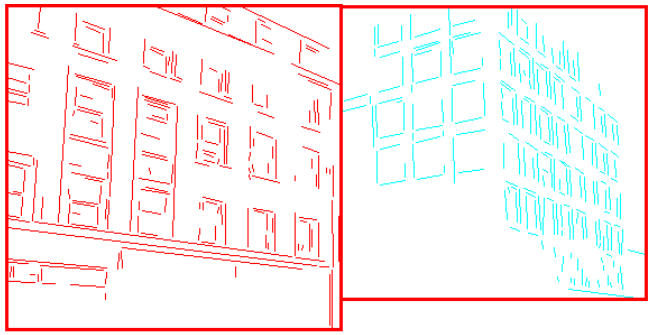
low-level

edge image

mid-level

consistent line clusters

high-level



Application of Computer Vision

Earth viewers (3D modeling)

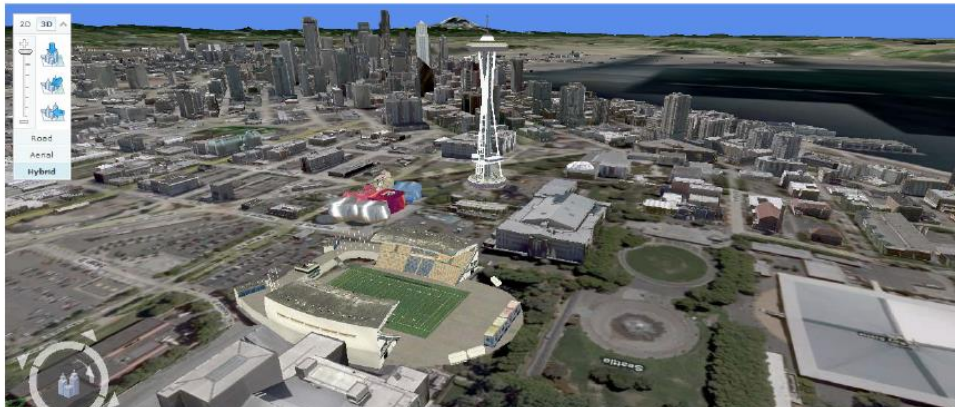
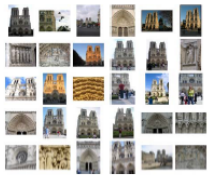
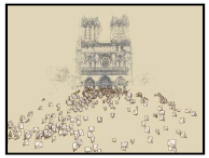


Photo Tourism overview



Input photographs

Scene reconstruction



Relative camera positions and orientations

Point cloud

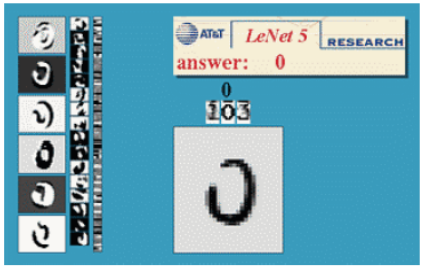
Sparse correspondence

Optical character recognition (OCR)

Technology to convert scanned docs to text

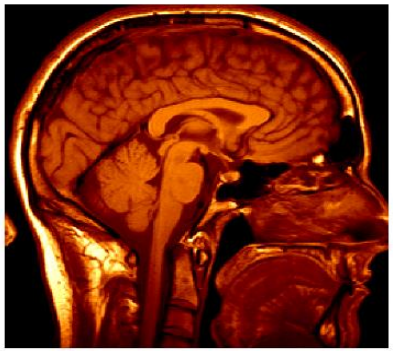
- If you have a scanner, it probably came with OCR software

Medical imaging



Digit recognition, AT&T labs
http://www.research.att.com/~yann/

License plate readers
http://en.wikipedia.org/wiki/Automatic_number_plate_recognition



3D imaging
MRI, CT

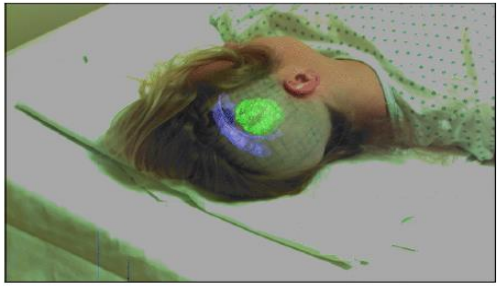
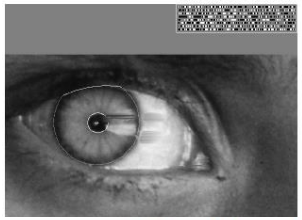


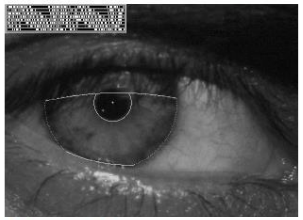
Image guided surgery
Grimson et al., MIT

Application of Computer Vision

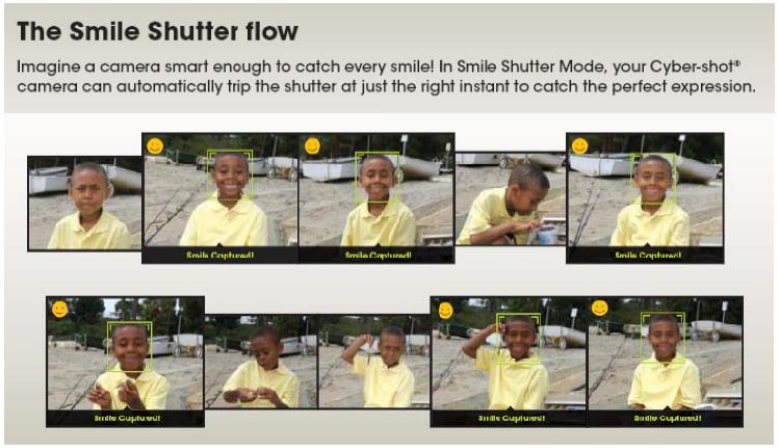
Vision-based biometrics

"*How the Afghan Girl was Identified by Her Iris Patterns*" Read the story





Smile detection?



Login without a password...

Fingerprint scanners on many new laptops, other devices

Face recognition systems now beginning to appear more widely
http://www.sensiblevision.com/

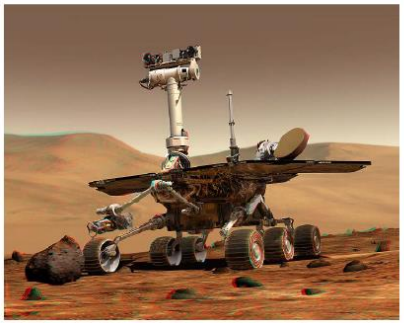
Face detection

Application of Computer Vision

Smart cars

Special effects: shape capture

Robotics



NASA's Mars Spirit Rover
http://en.wikipedia.org/wiki/Spirit_rover

http://www.robocup.org/

Sports

Sportvision first down line
Nice explanation on www.howstuffworks.com

Principles of Image Formation

Image Formation
Historical context

- **Pinhole model:** Mozi (470-390 BCE), Aristotle (384-322 BCE)
- **Principles of optics (including lenses):** Alhacen (965-1039 CE)
- **Camera obscura:** Leonardo da Vinci (1452-1519), Johann Zahn(1631-1707)
- **First photo:** Joseph Nicephore Niepce (1822)
- **Photographic film** (Eastman, 1889)
- **Cinema** (Lumière Brothers, 1895)
- **Color Photography** (LumièreBrothers, 1908)
- **Television** (Baird, Farnsworth, Zworykin, 1920s)
- **First consumer camera with CCD**:Sony Mavica (1981)
- **First fully digital camera:** Kodak DCS100 (1990)

Physical parameters of image formation

- **Geometric**
  - **Type of projection**
  - **Camera pose**
- **Optical**
  - **Sensor's lens type**
  - **focal length, field of view, aperture**
- **Photometric**
  - **Type, direction, intensity of light reaching sensor**
  - **Surfaces' reflectance properties**
- **Sensor**
  - **sampling, etc.**

Image Formation

- **Let's design a camera**
  - **Idea 1: put a piece of film in front of an object**
  - **Do we get a reasonable image?**
    - The obtained image will be **blur**

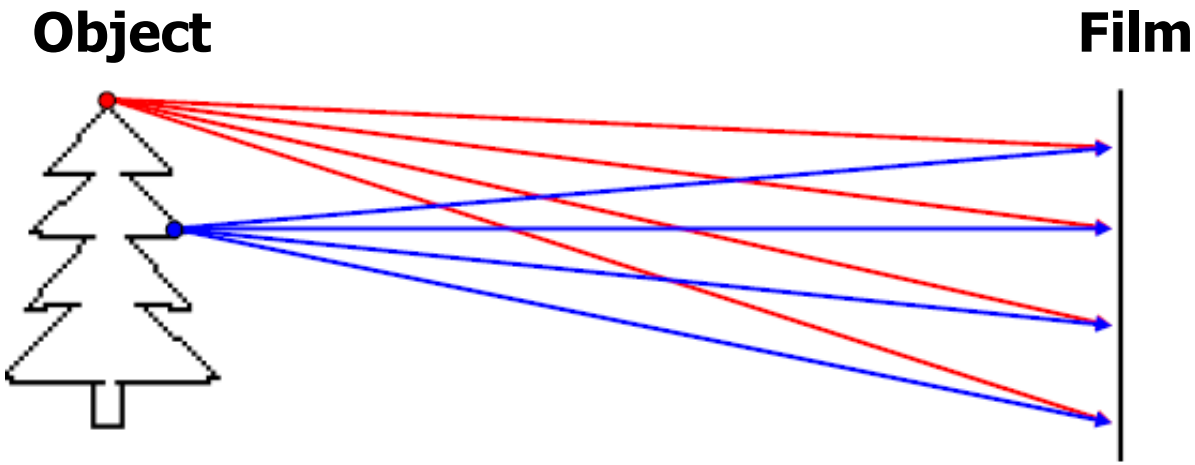


Image Formation
Pinhole camera

- **Add a barrier to block off most of the rays**
  - This reduces blurring
  - The opening is known as the **aperture**
  - How does this transform the image?

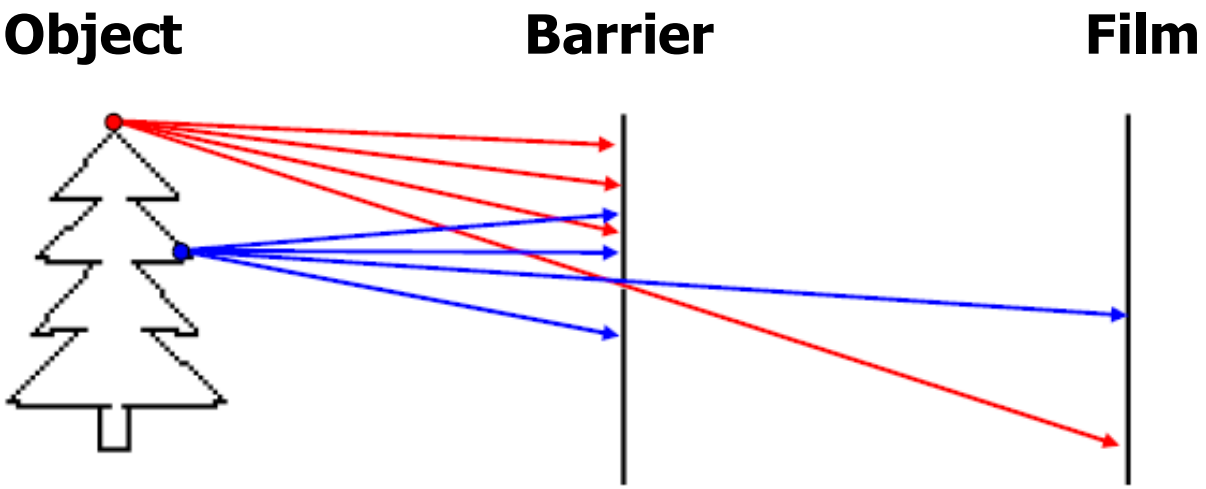


Image Formation
Pinhole camera

- **Pinhole camera is a simple model to approximate imaging process, perspective projection.**
- **If we treat pinhole as a point, only one ray from any given point can enter the camera.**

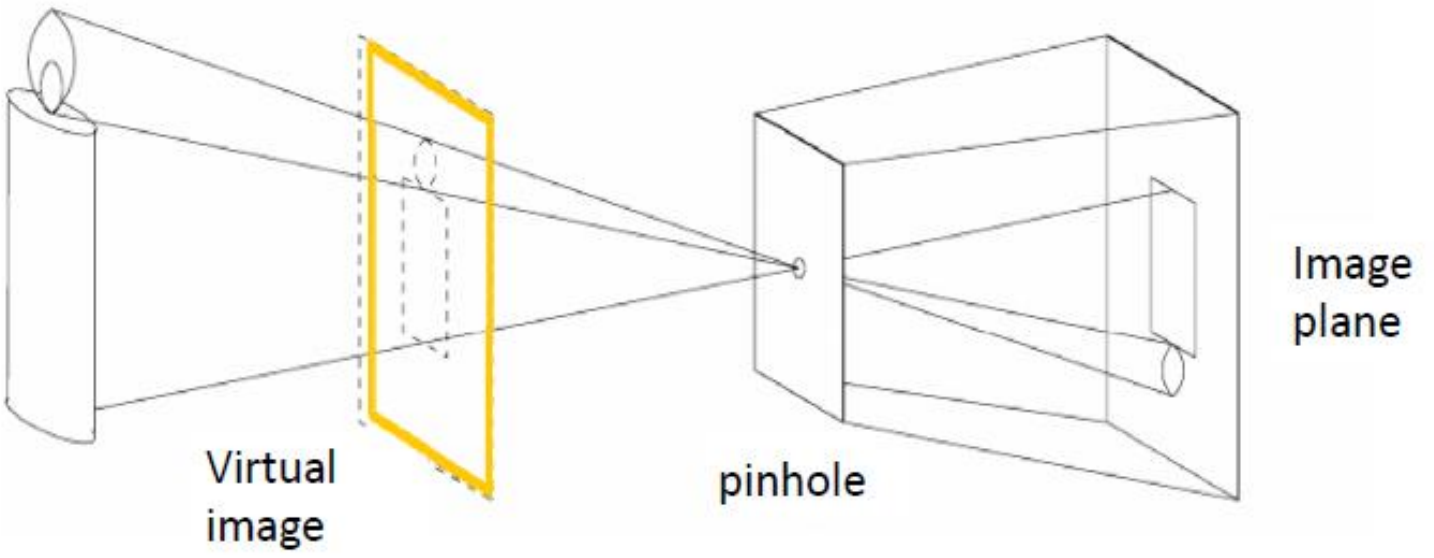


Image Formation
Perspective effects

- **Far away objects appear smaller**
- **Parallel lines in the scene intersect in the image**
- **Converge in image on horizon line**

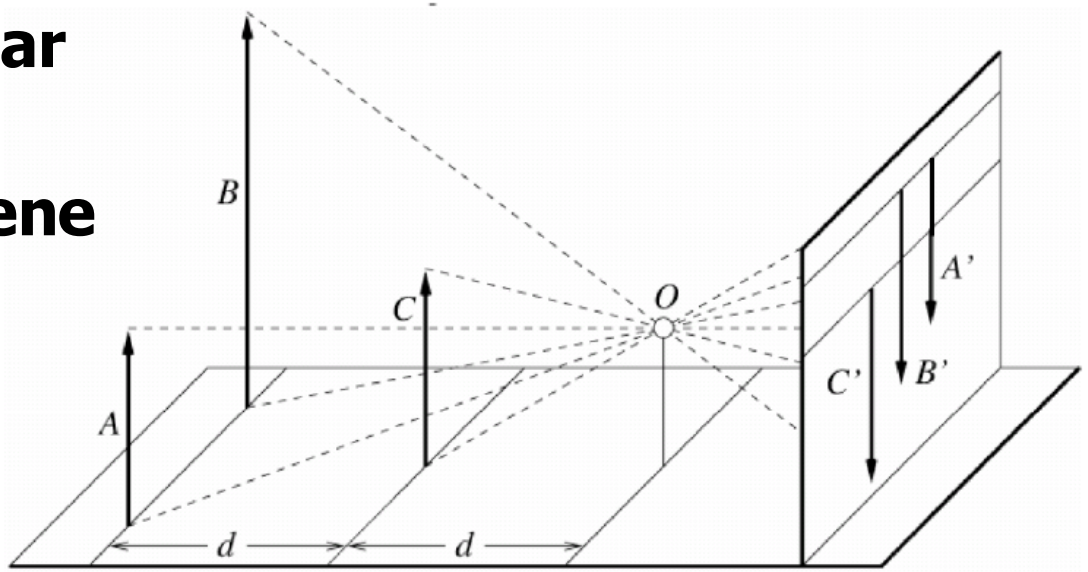


Image Formation
Perspective

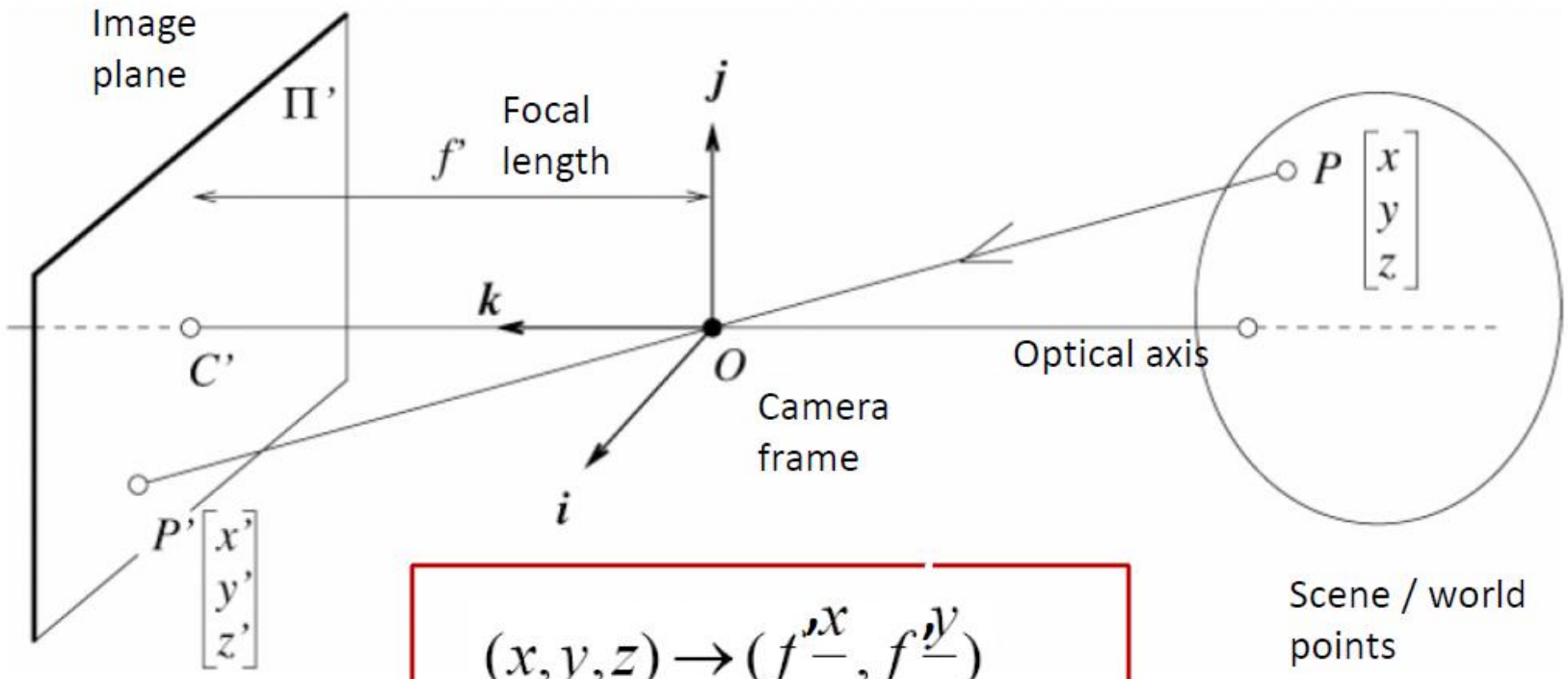
- **Use of correct perspective projection indicated in 1st century B.C. frescoes**
- **Skill resurfaces in Renaissance: Artists develop systematic methods to determine perspective projection (around 1480-1515)**

Raphael

Durer, 1525

Image Formation
Perspective projection equations

- **3D world mapped to 2D projection in image plane**



$$(x, y, z) \rightarrow \left(f' \frac{x}{z}, f' \frac{y}{z}\right)$$

Scene point $\rightarrow$ Image coordinates

Image Formation
Perspective(Homogeneous coordinates)

- **Is this a linear transformation?**
  - **No - division by z is nonlinear**
- **Trick: add one more coordinate:**

$$(x, y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

homogeneous image coordinates

$$(x, y, z) \Rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

homogeneous scene coordinates

- **Converting *from* homogeneous coordinates**

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow (x/w, y/w)$$

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \Rightarrow (x/w, y/w, z/w)$$

Image Formation
Perspective Projection Matrix

- **Projection is a matrix multiplication using homogeneous coordinates:**

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/f' & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z/f' \end{bmatrix} \Rightarrow (f'\frac{x}{z}, f'\frac{y}{z})$$

divide by the third coordinate to convert back to non-homogeneous coordinates

- **For complete mapping from world points to image pixel positions we should consider the situation of camera frame (camera calibration)**
  - Camera physical parameters: the focal length of the lens, the size of the pixels, the position of the principal point, and the position and orientation of the camera.

Image Formation
Perspective projection & calibration

- **Perspective equations so far in terms of camera's reference frame….**
- **Camera's *intrinsic* and *extrinsic* parameters needed to calibrate geometry.**
  - Intrinsic: Image coordinates relative to camera ↔Pixel coordinates
  - Extrinsic: Camera frame ↔ World frame

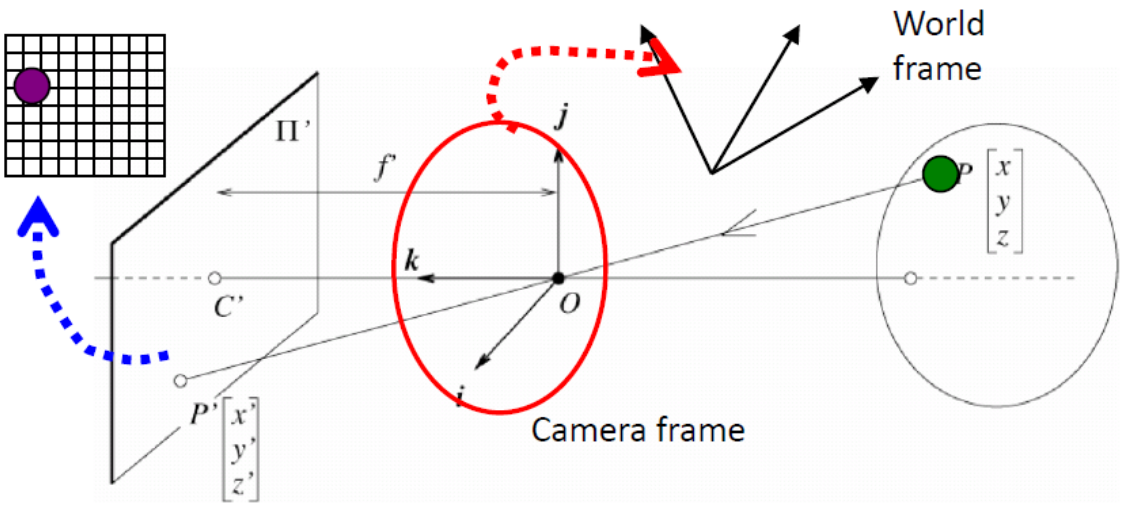


Image Formation
Perspective projection & calibration

- **Intrinsic parameters: from idealized world coordinates to pixel values**
- **Perspective projection for "pixels" in some arbitrary spatial square units (α):**

$$\begin{cases} u = \alpha \dfrac{x}{z} \\ v = \alpha \dfrac{y}{z} \end{cases}$$

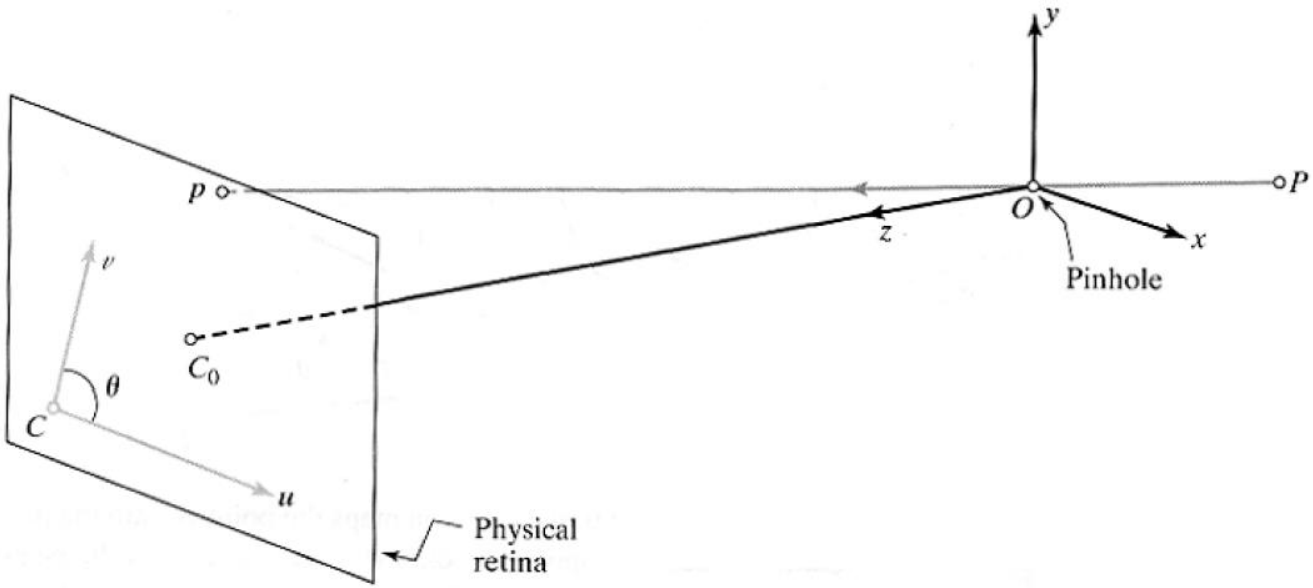
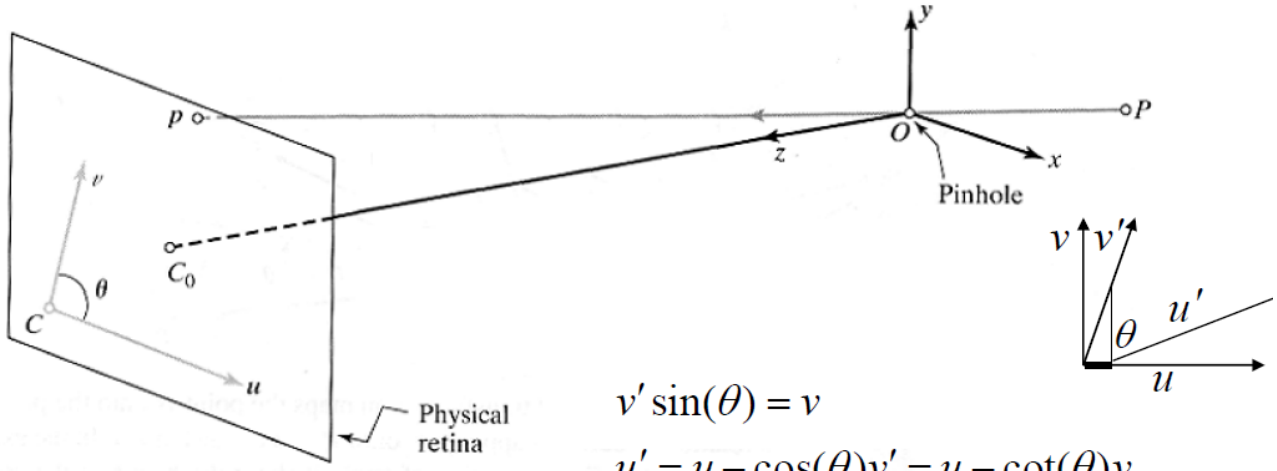


Image Formation
Perspective projection & calibration

- **Intrinsic parameters**
  - Maybe pixels are not square
  - We don't know the origin of our camera pixel coordinates
  - May be skew between camera pixel axes

$$\begin{cases} u = \alpha \dfrac{x}{z} \\ v = \alpha \dfrac{y}{z} \end{cases} \Rightarrow \begin{cases} u = \alpha \dfrac{x}{z} \\ v = \beta \dfrac{y}{z} \end{cases} \Rightarrow \begin{cases} u = \alpha \dfrac{x}{z} + u_0 \\ v = \beta \dfrac{y}{z} + v_0 \end{cases} \Rightarrow \begin{cases} u = \alpha \dfrac{x}{z} - \alpha \cot(\theta) \dfrac{y}{z} + u_0 \\ v = \dfrac{\beta}{\sin(\theta)} \dfrac{y}{z} + v_0 \end{cases}$$



$$v' \sin(\theta) = v$$

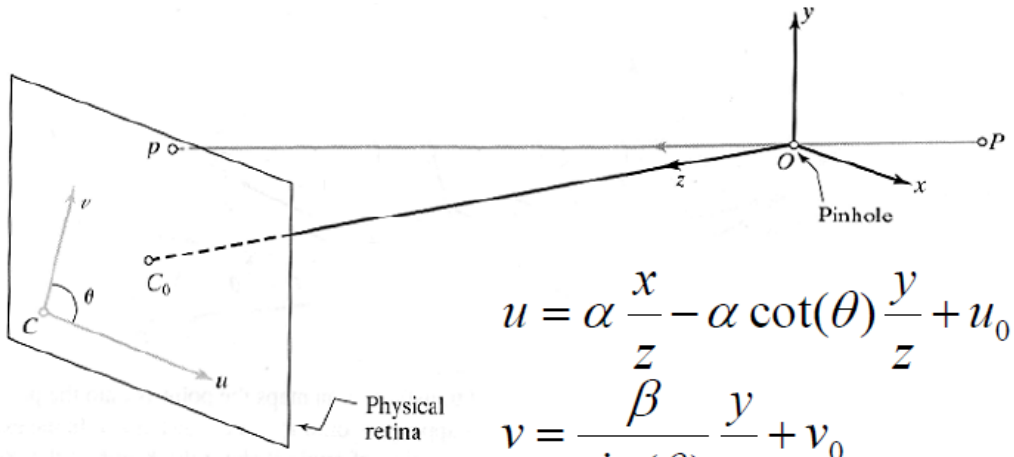
$$u' = u - \cos(\theta) v' = u - \cot(\theta) v$$

Image Formation
Perspective projection & calibration

Razi University

- **Using homogenous coordinates, intrinsic parameters can be written as:**

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha & -\alpha\cot(\theta) & u_0 & 0 \\ 0 & \dfrac{\beta}{\sin(\theta)} & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$



$$u = \alpha \frac{x}{z} - \alpha \cot(\theta) \frac{y}{z} + u_0$$

$$v = \frac{\beta}{\sin(\theta)} \frac{y}{z} + v_0$$

In pixels → $\vec{p} = \mathrm{K}\ \vec{p}_C$ ← In camera-based coords

Image Formation
Perspective projection & calibration

- **Extrinsic parameters: translation and rotation of camera frame**
  - $\vec{p}_C$ *is point coordinate in camera*
  - $\vec{p}_w$ *is point coordinate in world*
  - R_C^W *is rotation of camera frame corresponding to world frame*
  - $\vec{t}_C^W$ *is translation of camera frame corresponding to world frame*

$$\vec{p}_C = R_C^W \vec{p}_W + \vec{t}_C^W$$

Non-homogeneous coordinates

$$\begin{pmatrix} \vec{p}_C \end{pmatrix} = \begin{pmatrix} \boxed{\begin{matrix} - & - & - \\ - & R_C^W & - \\ - & - & - \end{matrix}} & \boxed{\begin{matrix} | \\ \vec{t}_C^W \\ | \end{matrix}} \\ 0 \quad 0 \quad 0 & 1 \end{pmatrix} \begin{pmatrix} \vec{p}_W \end{pmatrix}$$

Homogeneous coordinates

$$\vec{p}_C = \mathbf{E}\ \vec{p}_W$$

Image Formation
Perspective projection & calibration

- **Combining extrinsic and intrinsic calibration parameters, in homogeneous coordinates**

$$\vec{p} = \mathbf{K}_{3x4} \quad \vec{p}_C$$

$$\vec{p} = \mathbf{K}_{3x4}\,\mathbf{E}_{4x4} \quad \vec{p}_W$$

$$\vec{p} = \begin{pmatrix} \alpha & -\alpha\cot(\theta) & u_0 & 0 \\ 0 & \dfrac{\beta}{\sin(\theta)} & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} - & - & - & \vec{t}_C^{\,W} \\ & R_C^W & & \\ - & - & - & \\ 0 & 0 & 0 & 1 \end{pmatrix} \vec{p}_W$$

$$\vec{p} = \mathbf{M}_{3x4}\,\vec{p}_W$$

Image Formation
Perspective projection & calibration

- **From before, we had these equations relating image positions, (u, v), to points at 3-d positions $\vec{p}_w$ (in homogeneous coordinates):**

$$\vec{p} = \mathbf{M}_{3x4}\,\vec{p}_W$$

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} m_1^{\mathrm{T}} \\ m_2^{\mathrm{T}} \\ m_3^{\mathrm{T}} \end{pmatrix} \begin{pmatrix} p_{Wx} \\ p_{Wy} \\ p_{Wz} \\ 1 \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{pmatrix} \begin{pmatrix} p_{Wx} \\ p_{Wy} \\ p_{Wz} \\ 1 \end{pmatrix} \Rightarrow \begin{cases} u = \dfrac{m_1^{\mathrm{T}} \cdot \vec{p}_W}{m_3^{\mathrm{T}} \cdot \vec{p}_W} \\ v = \dfrac{m_2^{\mathrm{T}} \cdot \vec{p}_W}{m_3^{\mathrm{T}} \cdot \vec{p}_W} \end{cases}$$

- **So for each feature point, i, we have:**

$$(m_1^{\mathrm{T}} - u_i m_3^{\mathrm{T}}) \cdot \vec{P}_i = 0$$

$$(m_2^{\mathrm{T}} - v_i m_3^{\mathrm{T}}) \cdot \vec{P}_i = 0$$

- **Stack all these measurements of i=1…n points into a big matrix:**

$$\begin{pmatrix} P_1^T & 0^T & -u_1 P_1^T \\ 0^T & P_1^T & -v_1 P_1^T \\ \cdots & \cdots & \cdots \\ P_n^T & 0^T & -u_n P_n^T \\ 0^T & P_n^T & -v_n P_n^T \end{pmatrix} \begin{pmatrix} m_1 \\ m_2 \\ m_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}$$

Image Formation
Camera calibration

- **Showing all the elements:**
  - **We can use straight lines as calibration points (P1 to Pn)**

$$\begin{pmatrix} P_{1x} & P_{1y} & P_{1z} & 1 & 0 & 0 & 0 & 0 & -u_1P_{1x} & -u_1P_{1y} & -u_1P_{1z} & -u_1 \\ 0 & 0 & 0 & 0 & P_{1x} & P_{1y} & P_{1z} & 1 & -v_1P_{1x} & -v_1P_{1y} & -v_1P_{1z} & -v_1 \\ & & & & & & \cdots & & \cdots & \cdots & \cdots & \\ P_{nx} & P_{ny} & P_{nz} & 1 & 0 & 0 & 0 & 0 & -u_nP_{nx} & -u_nP_{ny} & -u_nP_{nz} & -u_n \\ 0 & 0 & 0 & 0 & P_{nx} & P_{ny} & P_{nz} & 1 & -v_nP_{nx} & -v_nP_{ny} & -v_nP_{nz} & -v_n \end{pmatrix} \begin{pmatrix} m_{11} \\ m_{12} \\ m_{13} \\ m_{14} \\ m_{21} \\ m_{22} \\ m_{23} \\ m_{24} \\ m_{31} \\ m_{32} \\ m_{33} \\ m_{34} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}$$

$$\mathbf{Q}\,\mathbf{M} = \mathbf{0}$$

Image Formation
Camera calibration

- **We want to solve (Q M =0) for the unit vector m (the stacked one) that minimizes $|QM|^2$**
- **The minimum eigenvector of the matrix Q^TQ gives us that, because it is the unit vector x that minimizesX^TQ^TQX (see Forsyth&Ponce, 5.3) .**
- **Once you have the M matrix, can recover the intrinsic and extrinsic parameters as:**

$$\mathbf{M} = \begin{pmatrix} \alpha \boldsymbol{r}_1^T - \alpha \cot\theta \boldsymbol{r}_2^T + u_0 \boldsymbol{r}_3^T & \alpha t_x - \alpha \cot\theta t_y + u_0 t_z \\ \frac{\beta}{\sin\theta} \boldsymbol{r}_2^T + v_0 \boldsymbol{r}_3^T & \frac{\beta}{\sin\theta} t_y + v_0 t_z \\ \boldsymbol{r}_3^T & t_z \end{pmatrix}$$

- where $\boldsymbol{r}_1^T$, $\boldsymbol{r}_2^T$ and $\boldsymbol{r}_3^T$ denote the three rows of the matrix rotation $\boldsymbol{R}$ and t_x, t_y and t_z are the coordinates of the translation vector $\boldsymbol{t}$

Image Formation
Camera calibration (Rotation Matrix)

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}$$

$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}$$

$$R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Express 3d rotation as series of rotations around coordinate axes by angles α, β, γ

Overall rotation is product of these elementary rotations:

$$\mathbf{R} = \mathbf{R}_x \mathbf{R}_y \mathbf{R}_z$$

Image Formation
Projection properties

- **Many-to-one**
  - All points along same ray are mapped to same point in image
- **Points → ?**
  - points
- **Lines → ?**
  - lines (collinearity preserved)
- **Distances and angles are / are not preserved?**
  - are not
- **Degenerate cases:**
  - Line through focal point projects to a point.
  - Plane through focal point projects to line
  - Plane perpendicular to image plane projects to part of the image.

Image Formation
Basic Planar Transformation

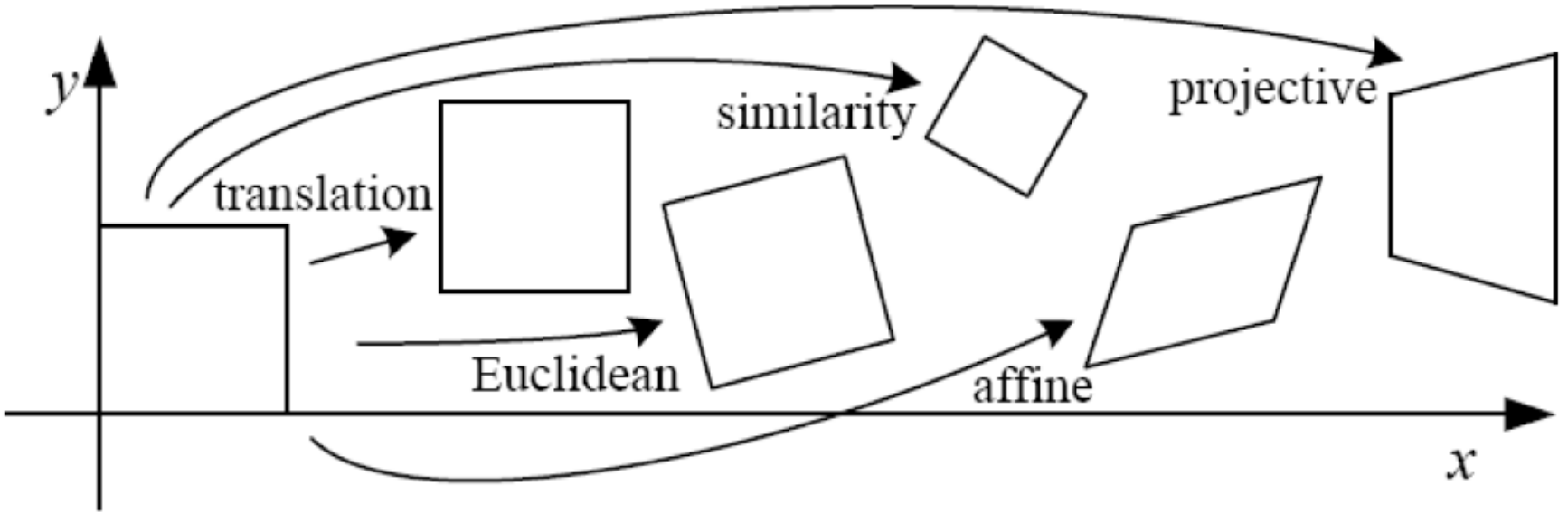


Image Formation

Properties of Basic Planar Transformation in 2D

Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\left[\begin{array}{c\|c} \boldsymbol{I} & \boldsymbol{t} \end{array}\right]_{2\times3}$	2	orientation $+\cdots$	
rigid (Euclidean)	$\left[\begin{array}{c\|c} \boldsymbol{R} & \boldsymbol{t} \end{array}\right]_{2\times3}$	3	lengths $+\cdots$	
similarity	$\left[\begin{array}{c\|c} s\boldsymbol{R} & \boldsymbol{t} \end{array}\right]_{2\times3}$	4	angles $+\cdots$	
affine	$\left[\begin{array}{c} \boldsymbol{A} \end{array}\right]_{2\times3}$	6	parallelism $+\cdots$	
projective	$\left[\begin{array}{c} \tilde{\boldsymbol{H}} \end{array}\right]_{3\times3}$	8	straight lines	

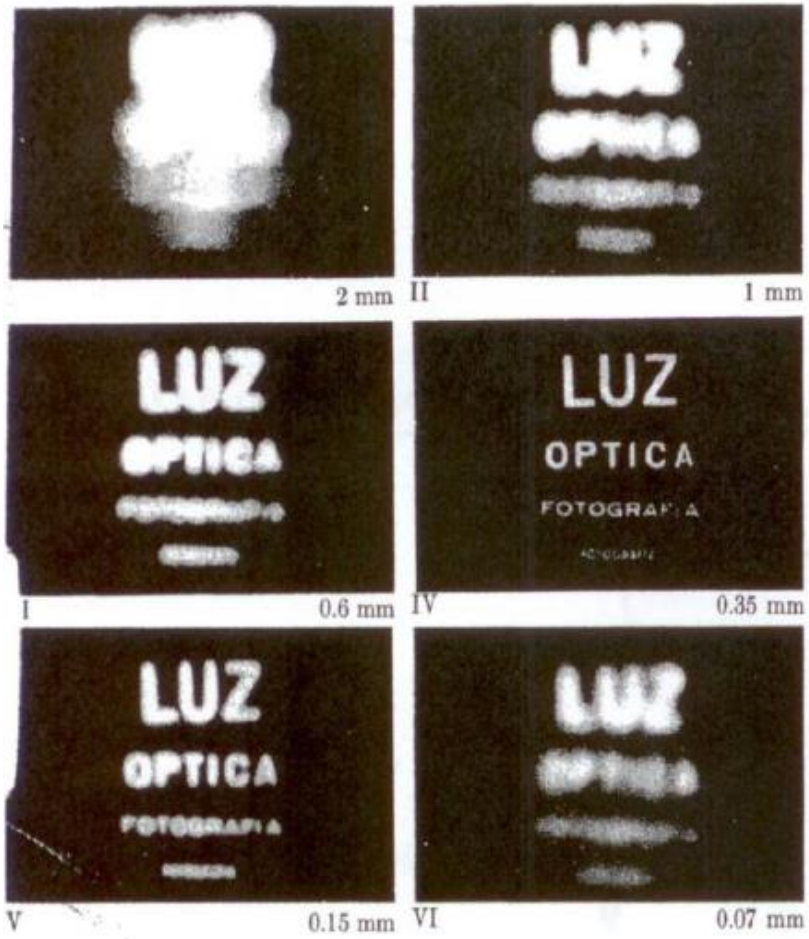
Image Formation

Properties of Basic Planar Transformation in 3D

Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\left[\begin{array}{c\|c} \boldsymbol{I} & \boldsymbol{t} \end{array}\right]_{3\times4}$	3	orientation + ···	
rigid (Euclidean)	$\left[\begin{array}{c\|c} \boldsymbol{R} & \boldsymbol{t} \end{array}\right]_{3\times4}$	6	lengths + ···	
similarity	$\left[\begin{array}{c\|c} s\boldsymbol{R} & \boldsymbol{t} \end{array}\right]_{3\times4}$	7	angles + ···	
affine	$\left[\begin{array}{c} \boldsymbol{A} \end{array}\right]_{3\times4}$	12	parallelism + ···	
projective	$\left[\begin{array}{c} \tilde{\boldsymbol{H}} \end{array}\right]_{4\times4}$	15	straight lines	

Image Formation
Pinhole size / aperture

- How does the size of the aperture affect the image we'd get?



Larger

Smaller

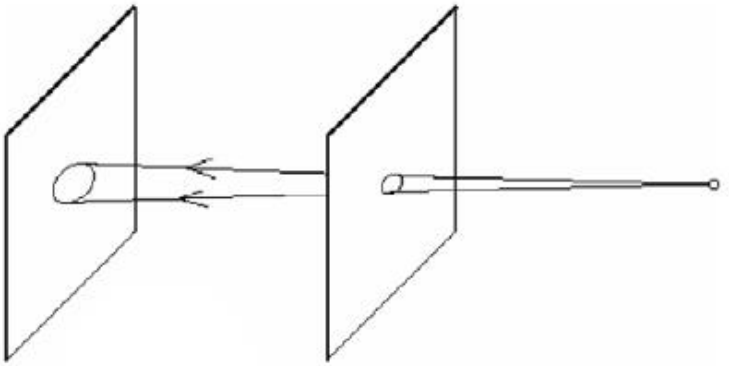
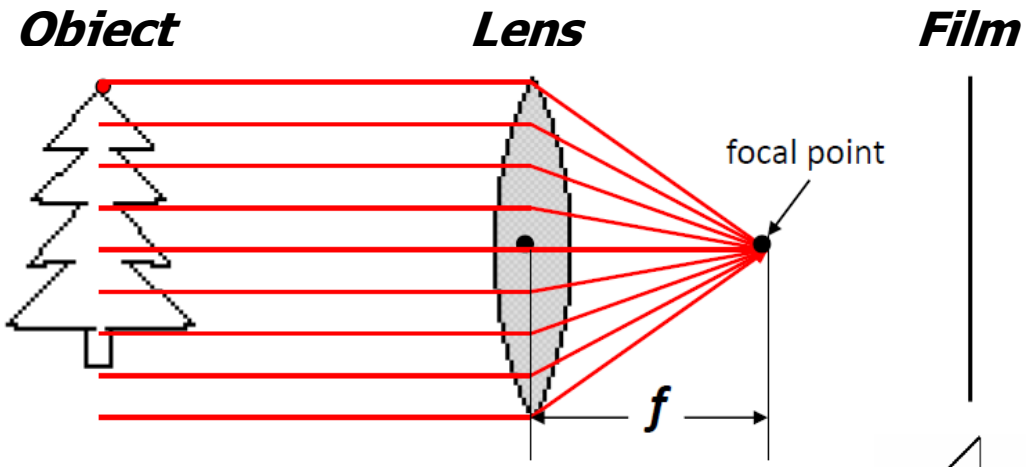
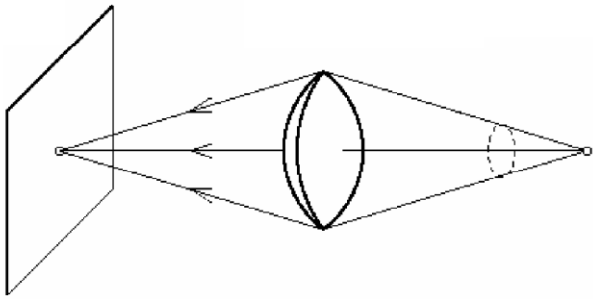


Image Formation
Adding a lens

- **A lens focuses light onto the film**
  - **Rays passing through the center are not deviated**
  - **All parallel rays p y converge to one point on a plane located at the *focal length* f**





lens vs. Pinhole

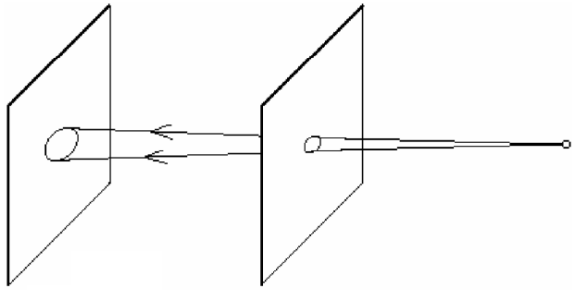


Image Formation
Cameras with lenses

- **A lens focuses parallel rays onto a single focal point**
- **Gather more light, while keeping focus; make pinhole perspective projection practical**

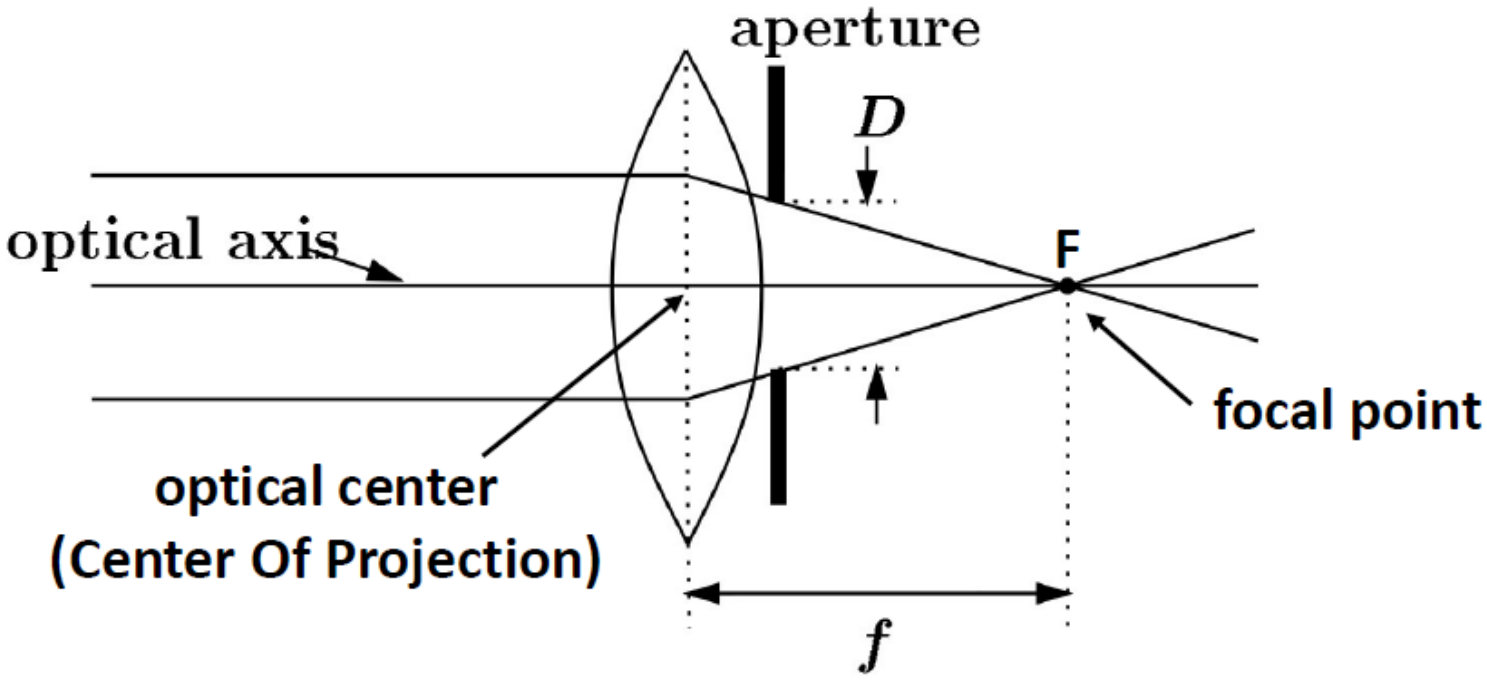


Image Formation
Cameras with Thin lenses

- **Rays entering parallel on one side go through focus on other, and vice versa.**
- **In ideal – all rays from P imaged at P’.**

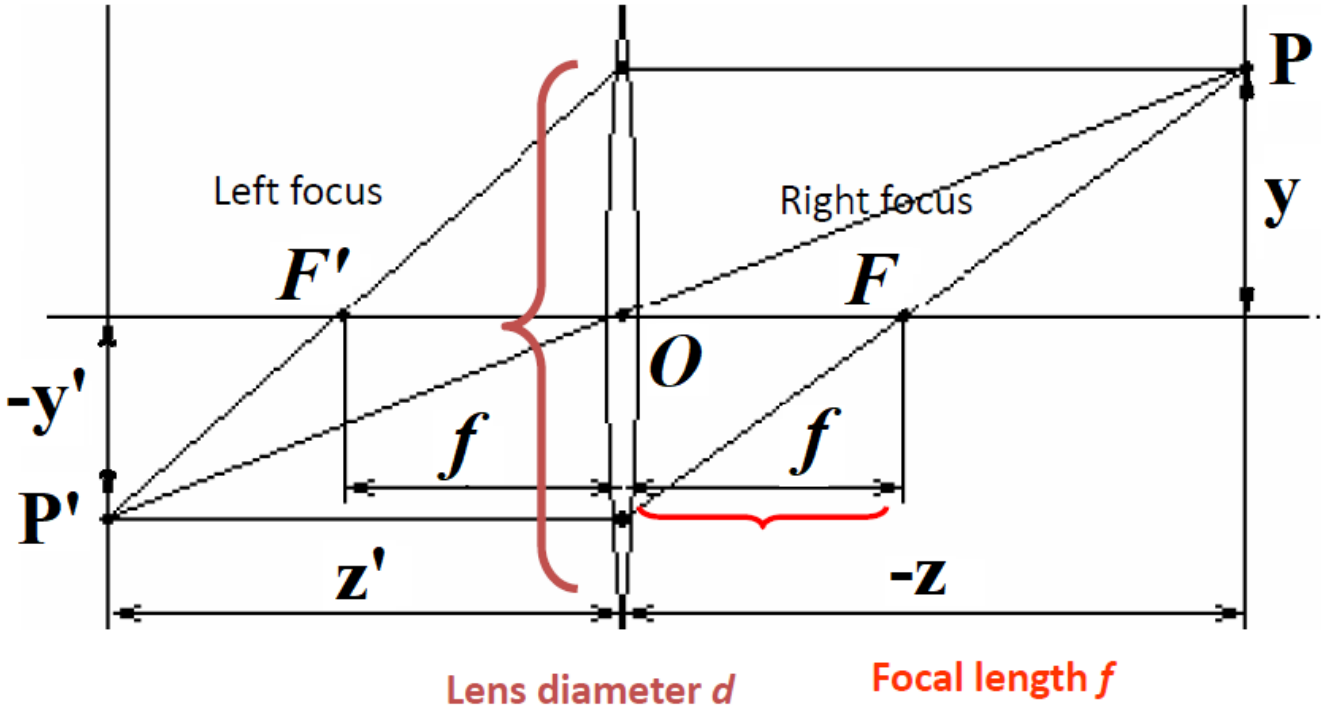


Image Formation
Thin lens equation

- **Any object point satisfying this equation is in focus**

$$\frac{1}{f} = \frac{1}{u} + \frac{1}{v}$$

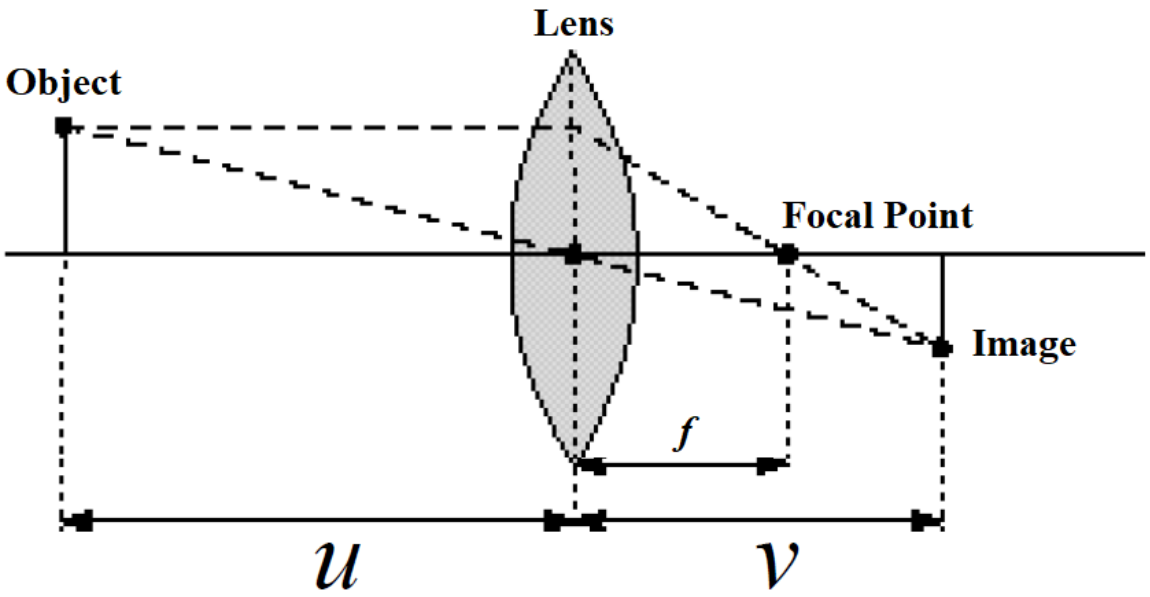


Image Formation
Focus and depth of field

- **Depth of field:** distance between image planes where blur is tolerable
- **Thin lens:** scene points at distinct depths come in focus at different image planes.
  - (Real camera lens systems have greater depth of field.)

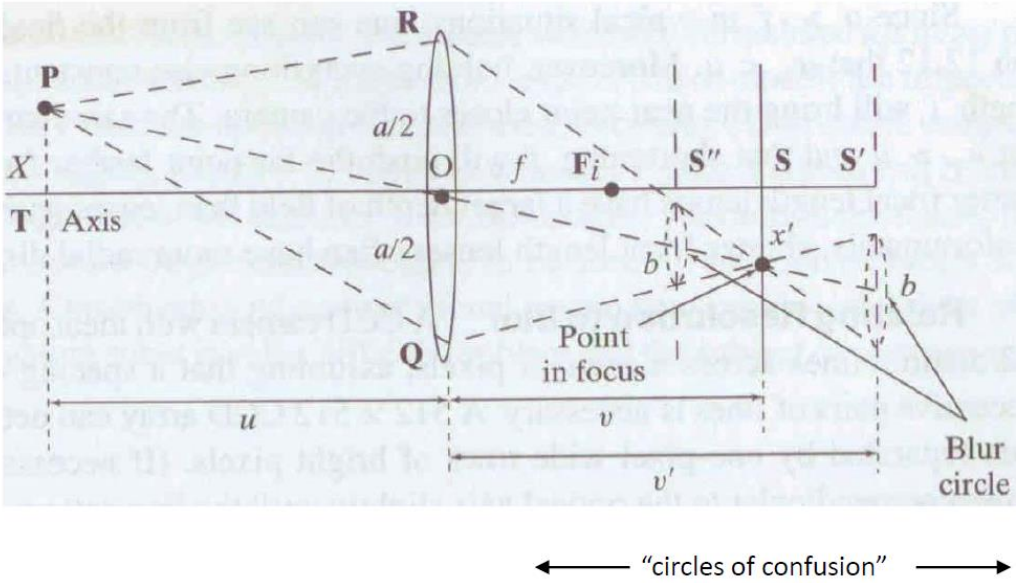


Image Formation
Focus and depth of field

- **How does the aperture affect the depth of field?**
  - **A smaller aperture increases the range in which the object is approximately in focus**

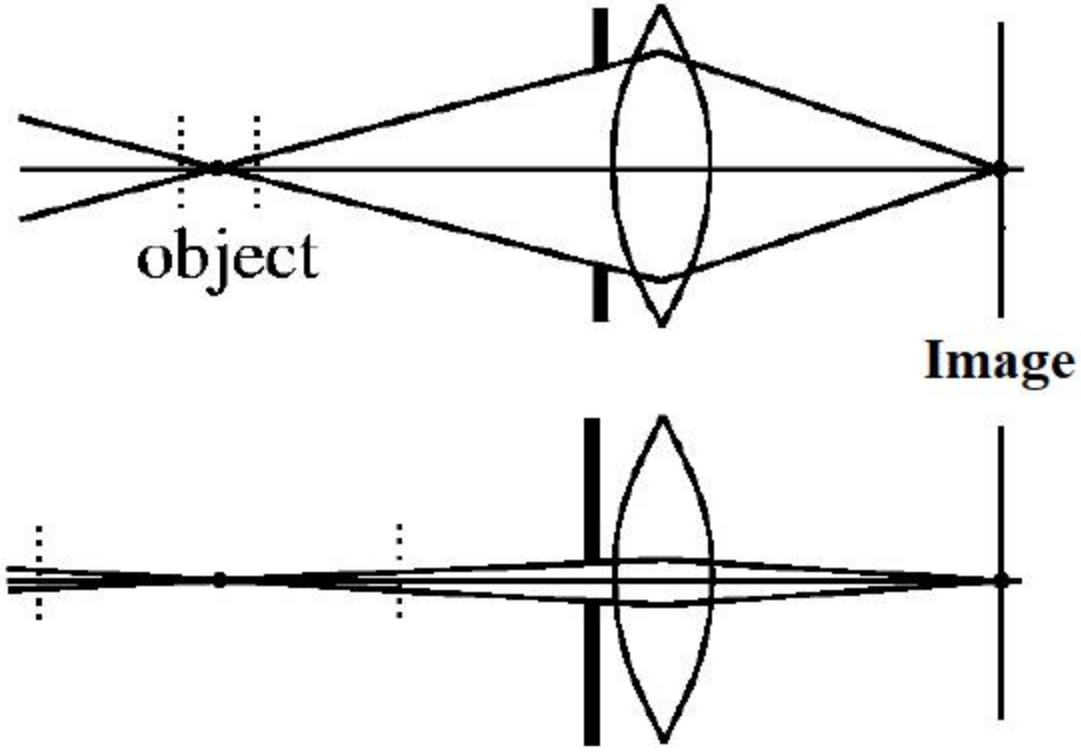
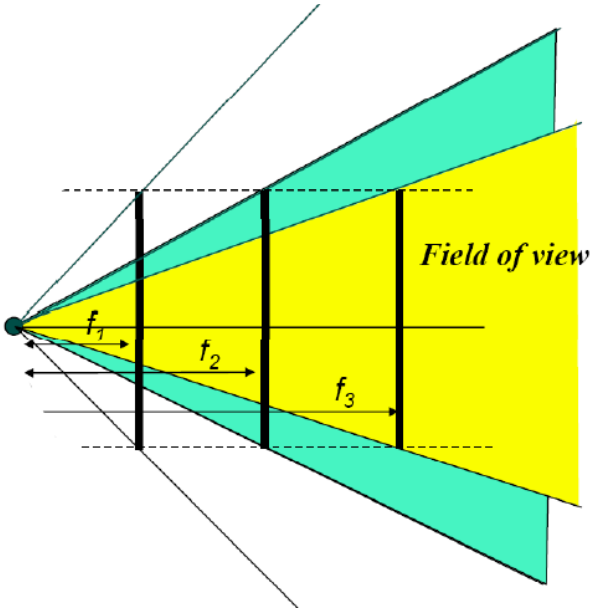


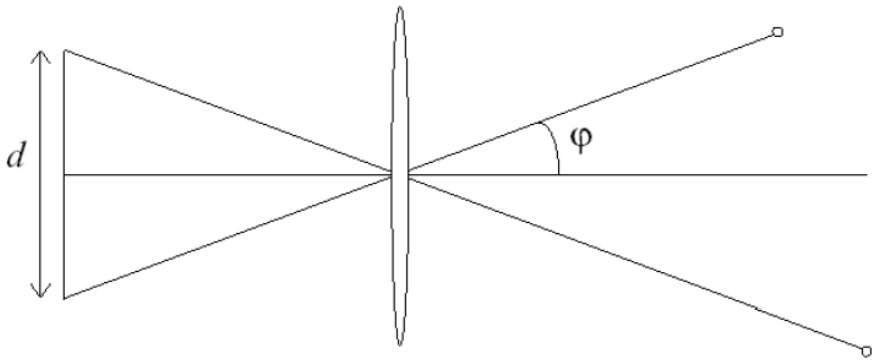
Image Formation
Field of view depends on focal length

- As f gets smaller, image becomes more *wide angle*
  - more world points project onto the finite image plane
- As f gets larger, image becomes more *telescopic*
  - smaller part of the world projects onto the finite image plane



$$\varphi = \tan^{-1}\left(\frac{d}{2f}\right)$$

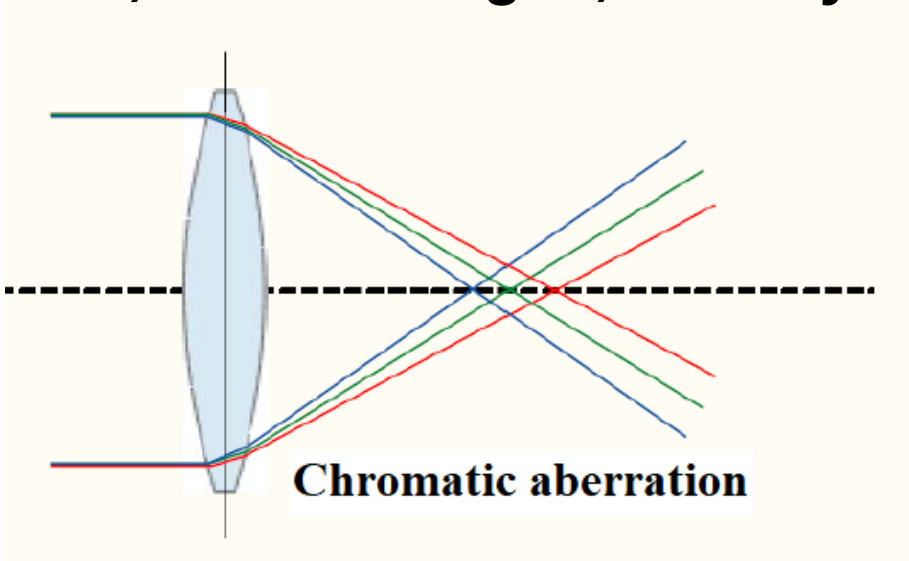
Smaller FOV = larger Focal Length



Size of field of view governed by size of the camera retina:

Image Formation
Chromatic aberration

Lenses and other optical systems have defects which lead to blur, color changes, usually called aberrations in optics.



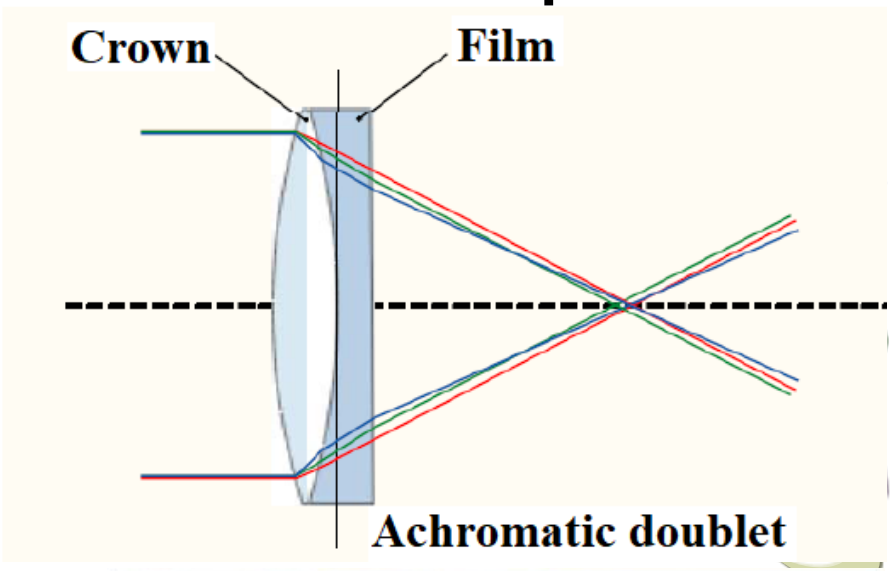


Image Formation
Digital cameras

- **Film → sensor array**
- **Often an array of charge coupled devices**
- **Each CCD is light sensitive diode that converts photons (light energy) to electrons**

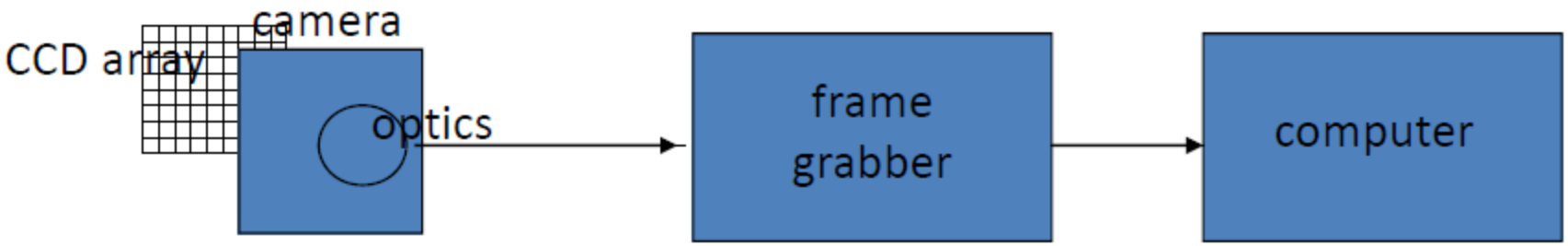


Image Formation
Digital cameras

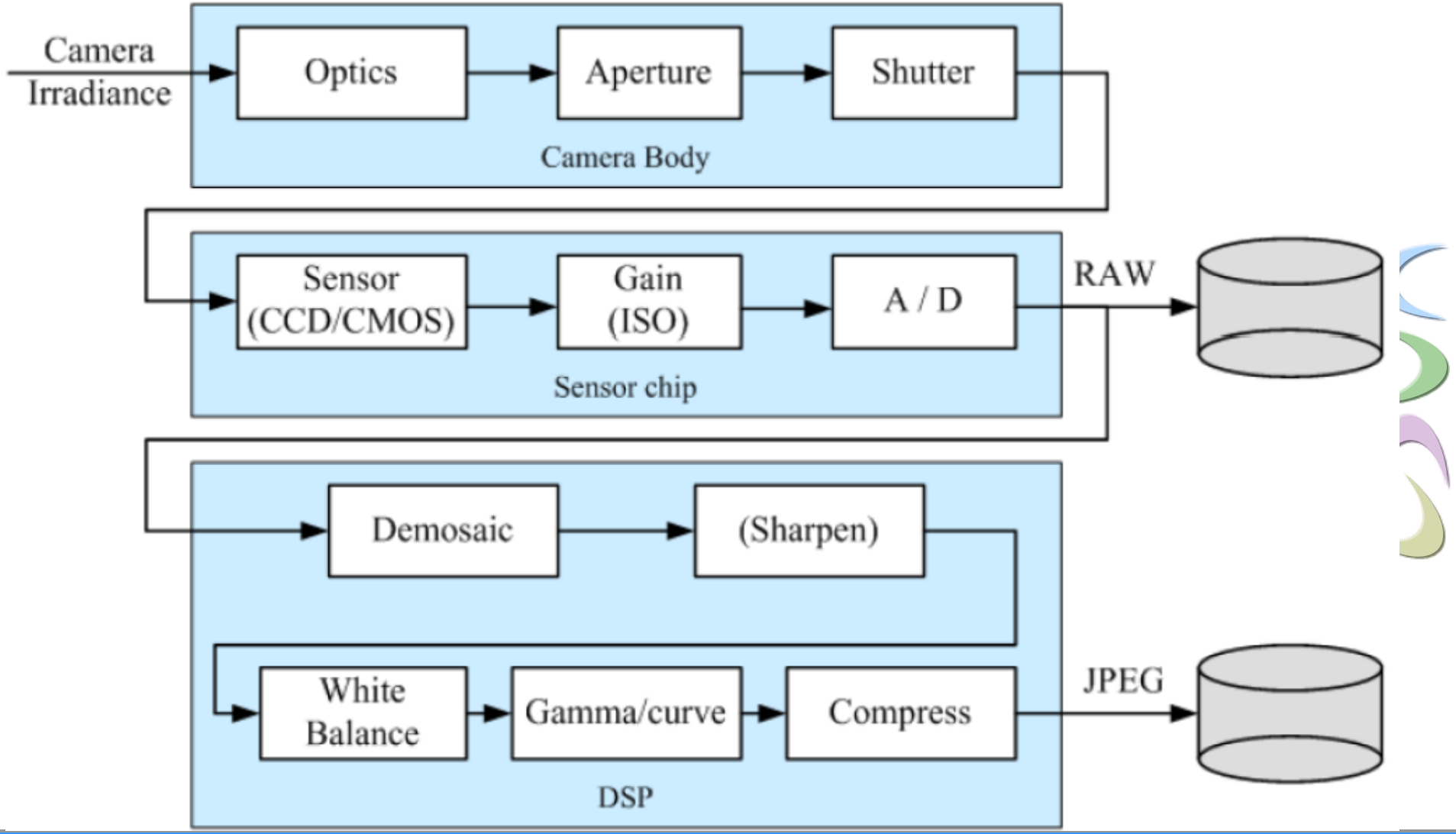


Image Formation
Resolution

- **Sensor Resolution: size of real world scene element a that images to a single pixel**
- **Image Resolution: number of pixels**

- **Think of images as matrices taken from CCD array.**

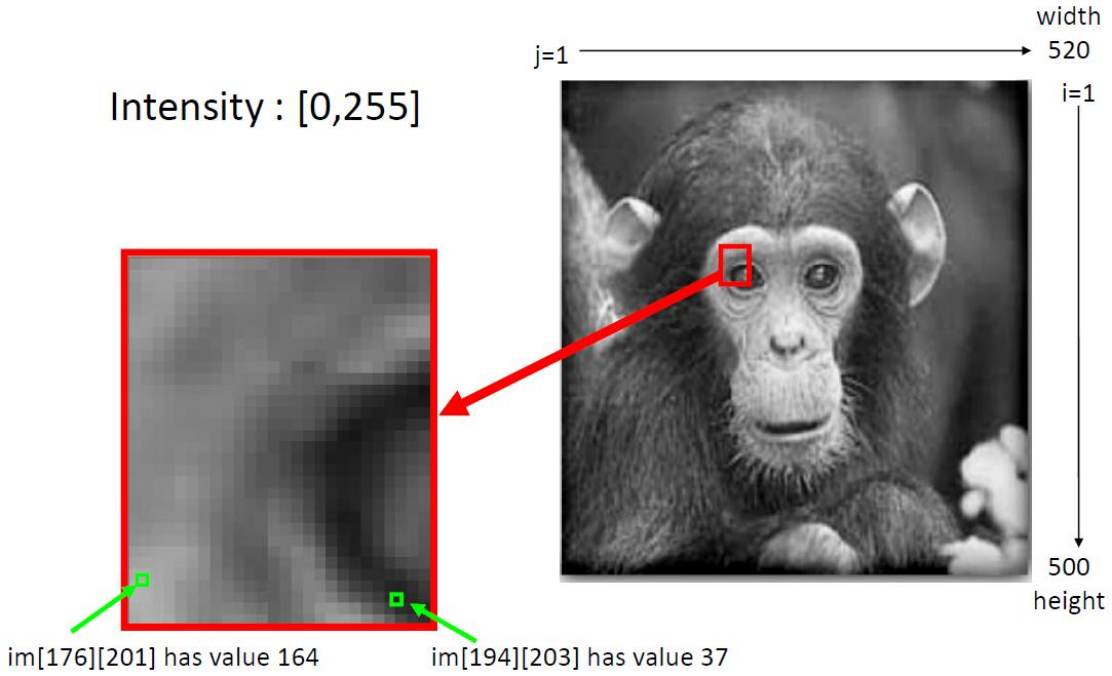
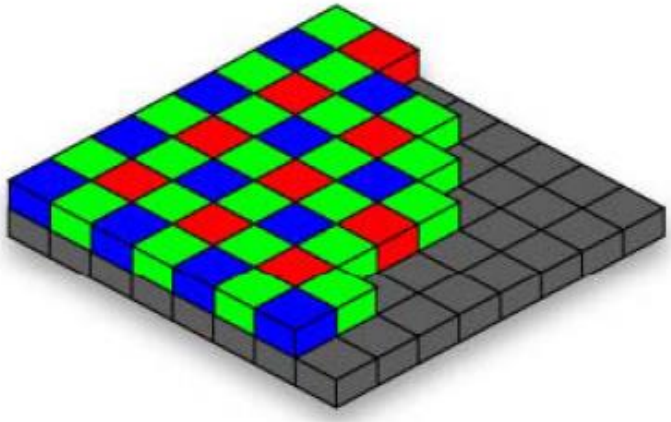
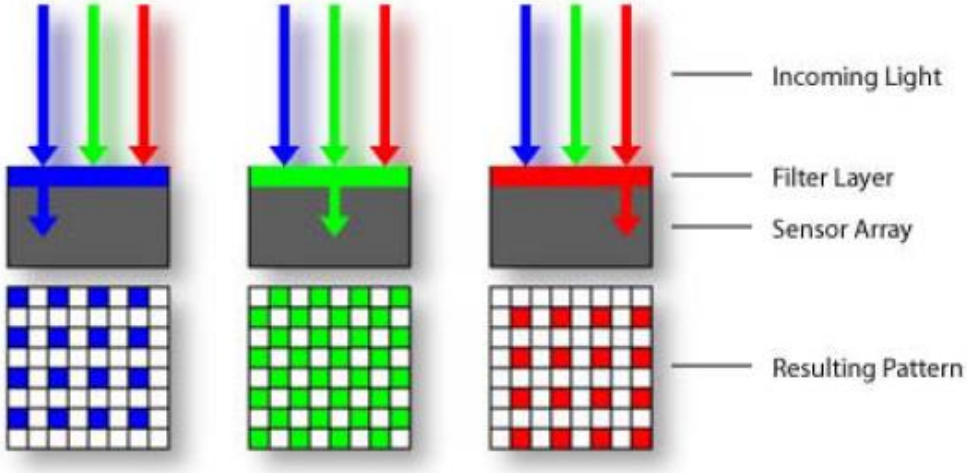


Image Formation
Color sensing in digital cameras

Bayer grid



Estimate missing components from neighboring values (demosaicing)



Digital Image Terminology

0	0	0	0	1	0	0
0	0	1	1	1	0	0
0	1	95	96	94	93	92
0	0	92	93	93	92	92
0	0	93	93	94	92	93
0	1	92	93	93	93	93
0	0	94	95	95	96	95

pixel (with value 94)

its 3x3 neighborhood

region of medium intensity

resolution (7x7)

- binary image
- gray-scale (or gray-tone) image
- color image
- multi-spectral image
- range image
- labeled image

Image Formation
Summary

- **Image formation affected by geometry, photometry, and optics.**
- **Projection equations express how world points mapped to 2d image.**
- **Homogenous coordinates allow linear system for projection equations.**
- **Lenses make pinhole model practical**
- **Digital imagers, Bayer demosaicing**

Parameters (focal length, aperture, lens diameter, sensor sampling…) strongly affect image obtained.

Light and Color

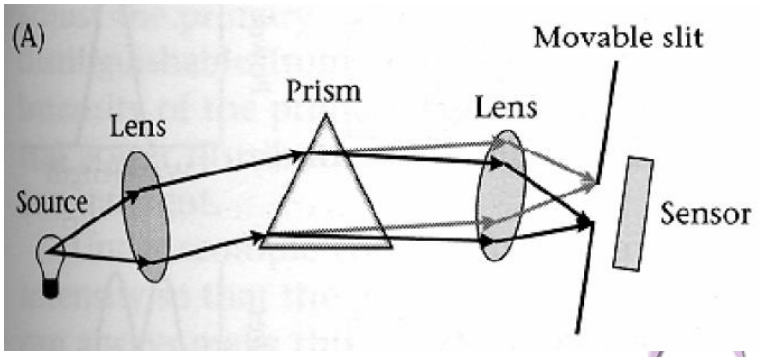
Contents

- **Measuring color**
  - **Spectral power distributions**
  - **Color mixing**
  - **Color matching experiments**
  - **Color spaces**
    - **Uniform color spaces**
- **Perception of color**
  - **Human photoreceptors**
  - **Environmental effects, adaptation**
- **Using color in machine vision systems**

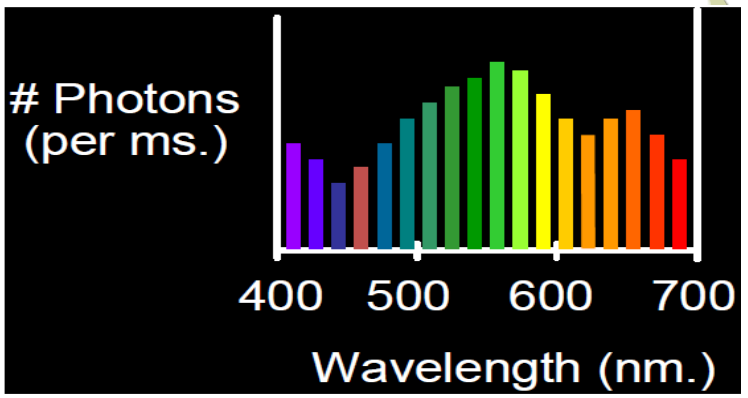
Light and Color

- **Color of light arriving at camera depends on**
  - Spectral reflectance of the surface light is leaving
  - Spectral radiance of light falling on that patch

- **Spectroradiometer**: separate input light into its different wavelengths, and measure the energy at each.



- **Spectral power distribution** is the power per unit area at each wavelength of a radiant object



Light in Space

- The measurement of light is a field in itself, known as radiometry.
- We need a series of units that describe how energy is transferred from light sources to surface patches, and what happens to the energy when it arrives at a surface.
  - The first matter to study is the behavior of light in space.
  - At each point on a piece of surface is a hemisphere of directions, along which light can arrive or leave .
  - Two sources that generate the same pattern on this input hemisphere must have the same effect on the surface at this point (because an observer at the surface can't tell them apart).
  - This applies to sources, too; two surfaces that generate the same pattern on a source's output hemisphere must receive the same amount of energy from the source.

Light in Space
Surface reflectance

- **In many applications, pixel gray-level is constructed as an estimate of image irradiance as a result of light reflection from scene objects.**
  - **The radiance of an opaque object that does not emit its own energy depends on irradiance caused by other energy sources.**
- **The illumination that the viewer perceives depends on the:**
  - **strength,**
  - **position,**
  - **orientation,**
  - **type (point or diffuse) of the light sources,**
  - **ability of the object surface to reflect energy,**
  - **and the local surface orientation (given by its normal vector).**

Light in Space
Radiance & Irradiance

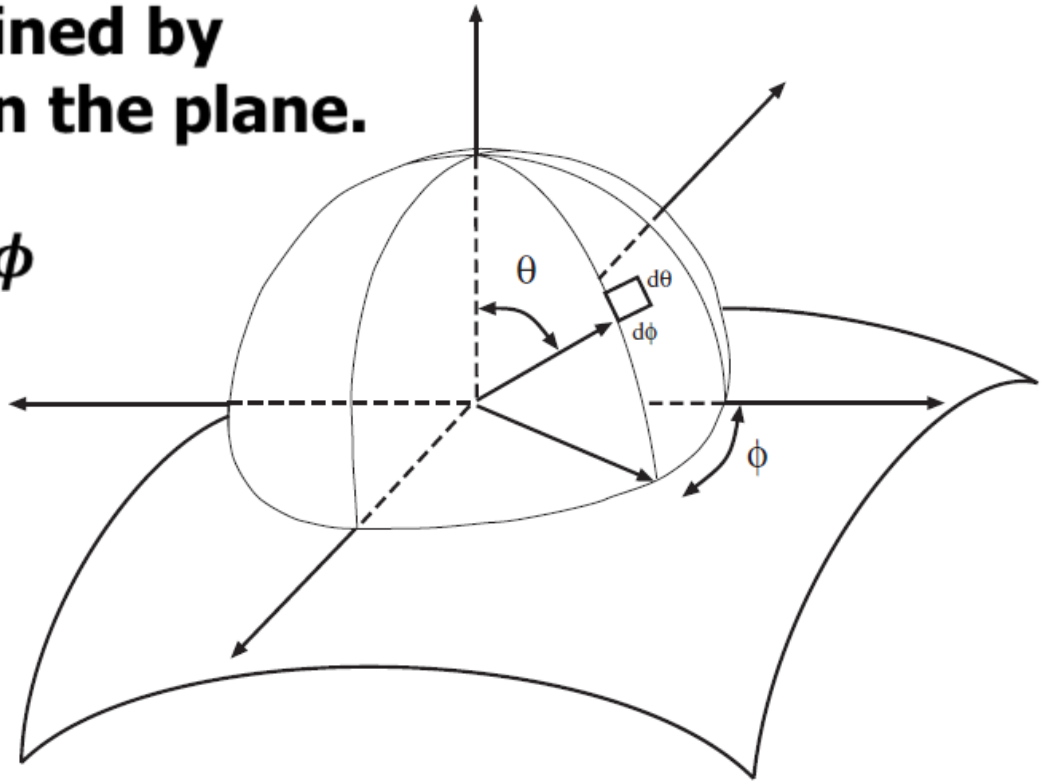
- **The appropriate unit for measuring the distribution of light in space is radiance, which is defined as:**
  - **The amount of energy travelling at some point in a specified direction, per unit time, per unit area perpendicular to the direction of travel, per unit solid angle**
  - **Radiance is a function of position and direction**
  - **Radiance is Constant Along a Straight Line**
- **The appropriate unit for representing incoming power which is irradiance, defined as:**
  - **incident power per unit area not foreshortened.**
  - **A surface illuminated by radiance $L_i(x, \theta_i, \phi_i)$coming in from a differential region of solid angle $d\omega$ at angles (θ_i, ϕ_i) receives irradiance**

$$L_i(x, \theta_i, \phi_i) \cos(\theta_i)\, d\omega$$

Light in Space
Solid Angle

- **The pattern a source generates on an input hemisphere can be described by the solid angle that the source subtends.**
  - **Solid angle (ω) is defined by analogy with angle on the plane.**

$$d\omega = \sin(\theta)\, d\theta d\phi$$



Light at Surfaces
Bidirectional Reflectance Distribution Function

- When light strikes a surface, it may be absorbed, transmitted, or scattered; usually, a combination of these effects occur.
- The picture is complicated further by the willingness of some surfaces to absorb light at one wavelength, and then radiate light at a different wavelength as a result.
  - This effect, known as fluorescence, is fairly common: scorpions fluoresce visible light under x-ray illumination;
- The most general model of local reflection is the bidirectional reflectance distribution function, usually abbreviated BRDF, defined as
  - The ratio of the radiance in the outgoing direction to the incident irradiance

Light at Surfaces
Bidirectional Reflectance Distribution Function

- $$BRDF = f(\theta_i, \phi_i, \theta_e, \phi_e) = \frac{\partial L(\theta_i, \phi_i)}{\partial E(\theta_e, \phi_e)}$$
  - **The unite of BRDF is sr^{-1} $(invers\ stradian)$**

- **For Lambertian surface (also ideally opaque, with ideal diffusion) reflects light energy in all directions, and thus the radiance is constant in all directions. The BRDF $f_{Lambert}$ is constant:**

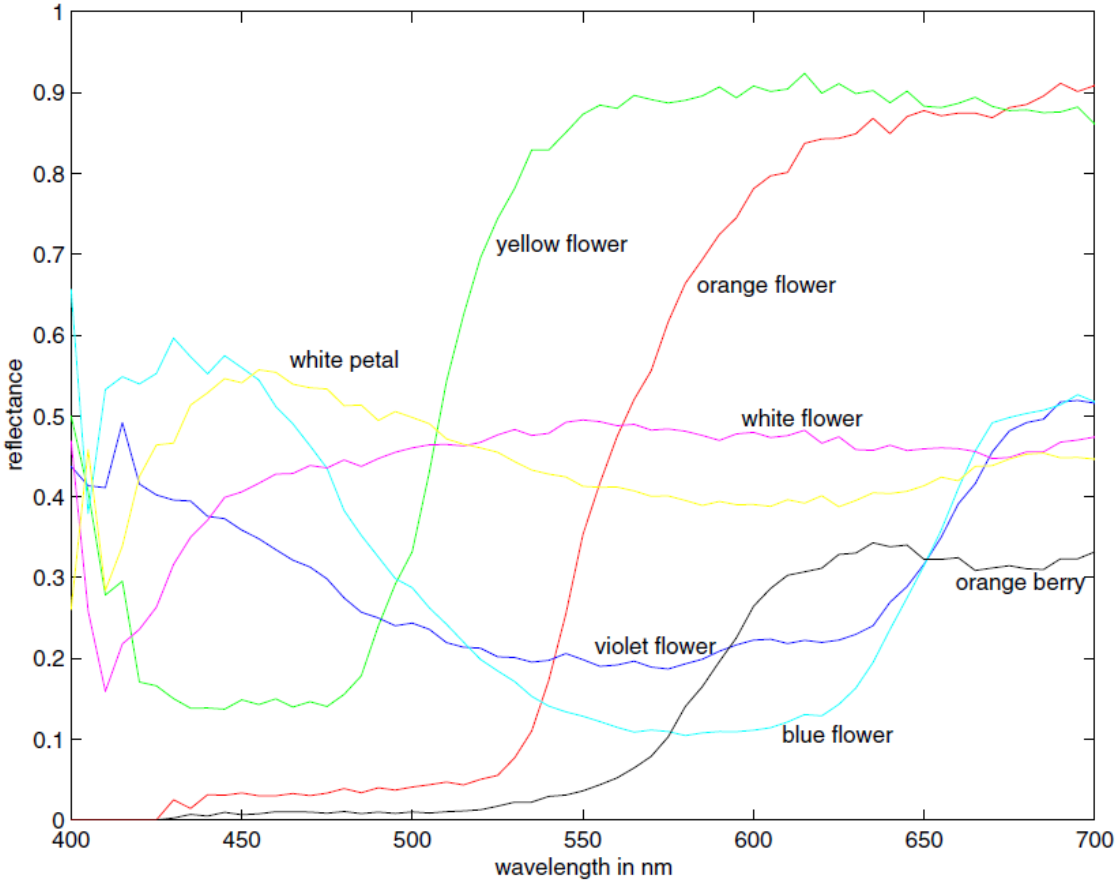
$$f_{Lambert}(\theta_i, \phi_i, \theta_e, \phi_e) = \frac{\rho(\lambda)}{\pi}$$

- Where $\rho(\lambda)$ is reflectance coefficient or albedo that defined as the proportion of incident energy reflected back to the half-space.

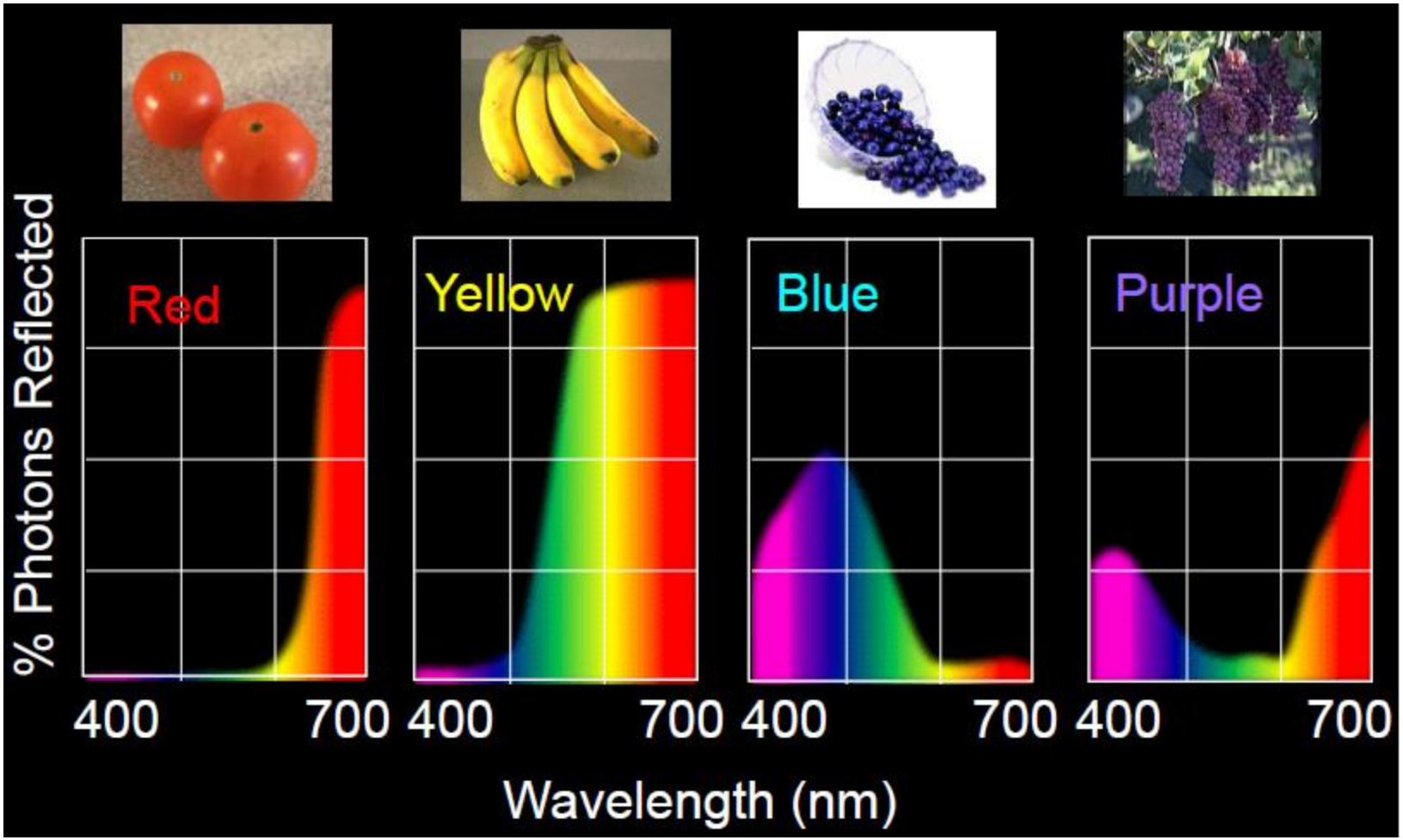
Light and Color

- **The color viewed is also affected by the surface's spectral reflectance properties.**

- **Spectral reflectance** for some natural objects is how much of each wavelength is reflected for that surface.

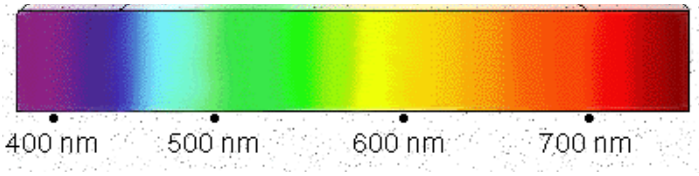


Color and light
Examples of reflectance spectra of surfaces



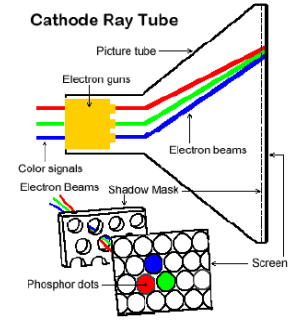
Color

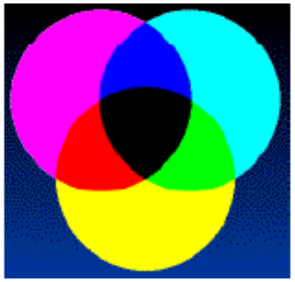
- Used heavily in human vision
- Color is a pixel property, making some recognition problems easy
- Visible spectrum for humans is 400 nm (blue) to 700 nm (red)
- Machines can "see" much more; ex. X-rays, infrared, radio waves

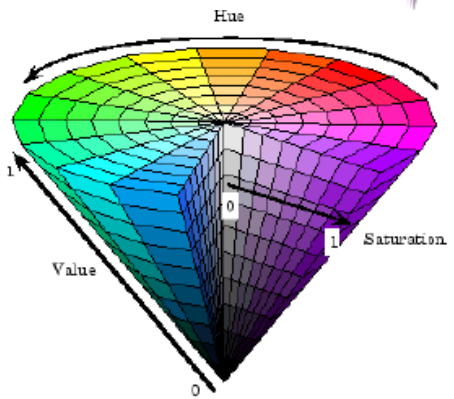


Color
Coding methods for humans

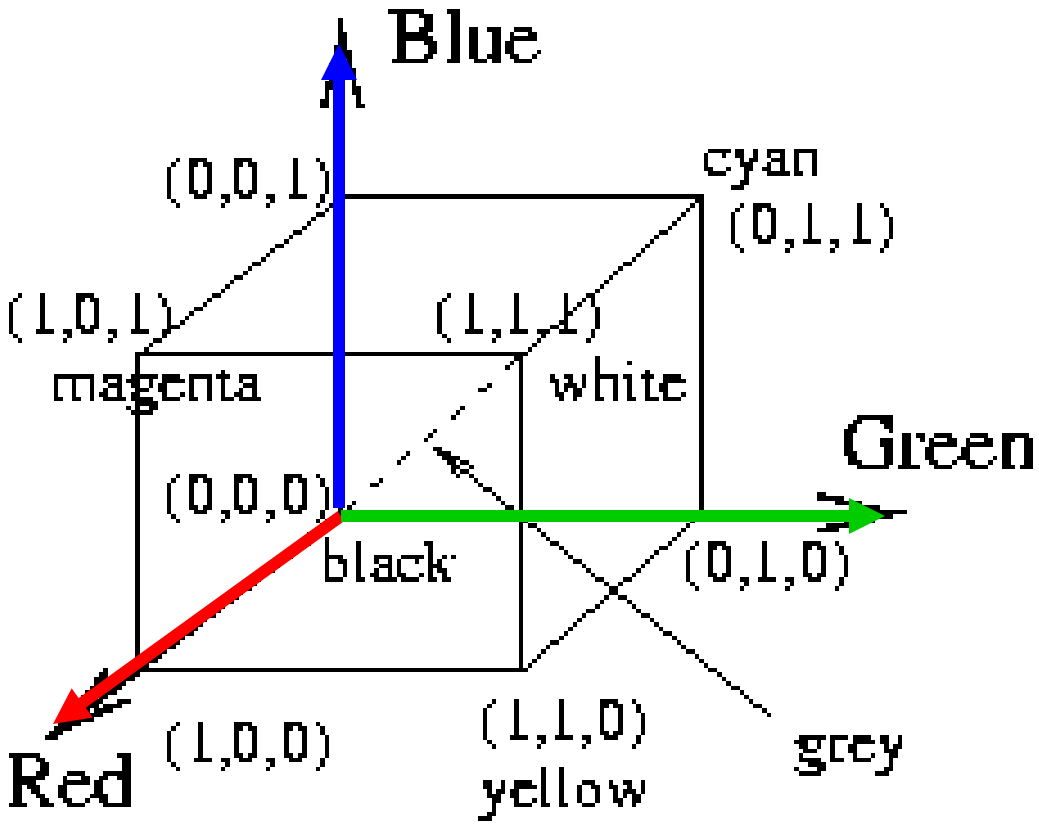
- **RGB** is an additive system (add colors to black) used for displays.
- **CMY** is a subtractive system for printing.
- **HSI** is a good perceptual space for art, psychology, and recognition.
- **YIQ** used for TV is good for compression.



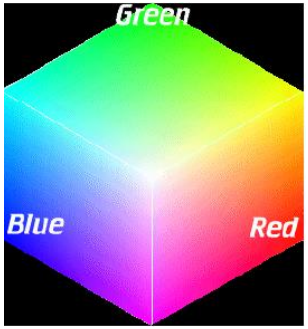




RGB color cube

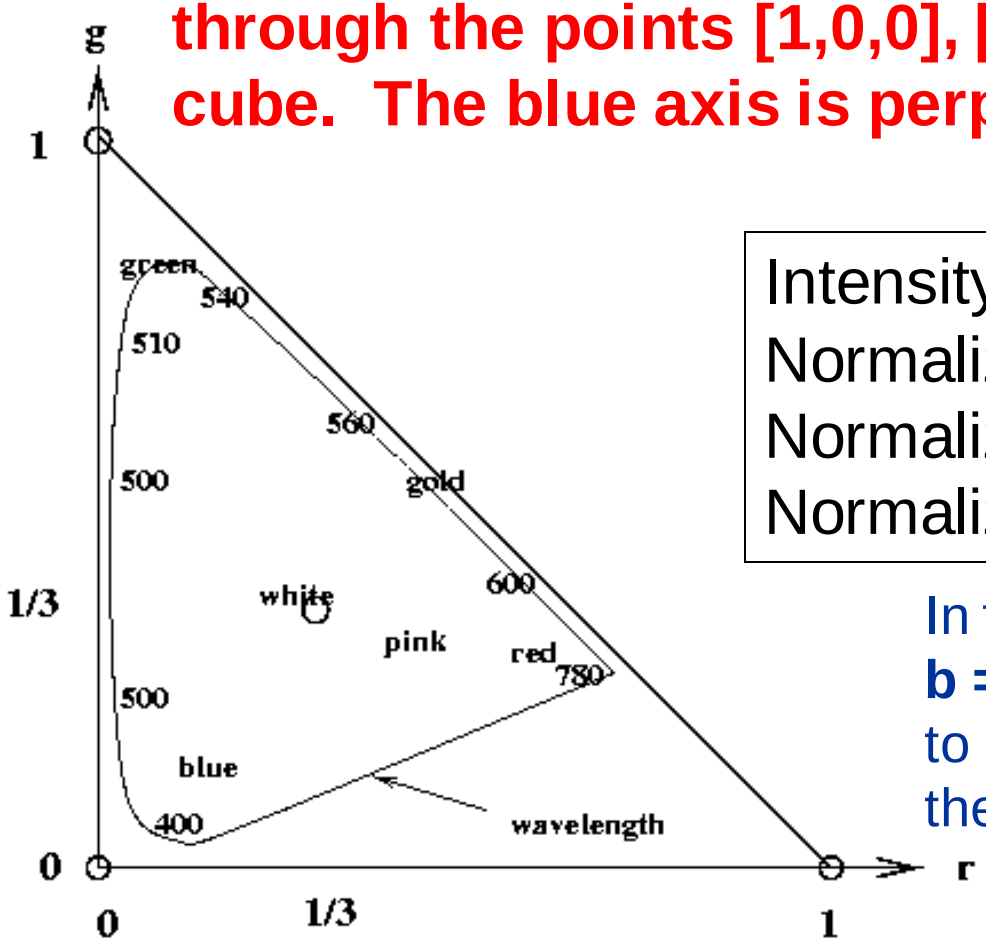


- R, G, B values normalized to (0, 1) interval
- Human perceives gray for triples on the diagonal
- "Pure colors" on corners



Color palette and normalized RGB

Color triangle for normalized RGB coordinates is a slice through the points [1,0,0], [0,1,0], and [0,0,1] of the RGB cube. The blue axis is perpendicular to the page.



Intensity	**I = (R+G+B) / 3**
Normalized red	**r = R/(R+G+B)**
Normalized green	**g = G/(R+G+B)**
Normalized blue	**b = B/(R+G+B)**

In this normalized representation, **b = 1 – r –g**, so we only need to look at r and g to characterize the color.

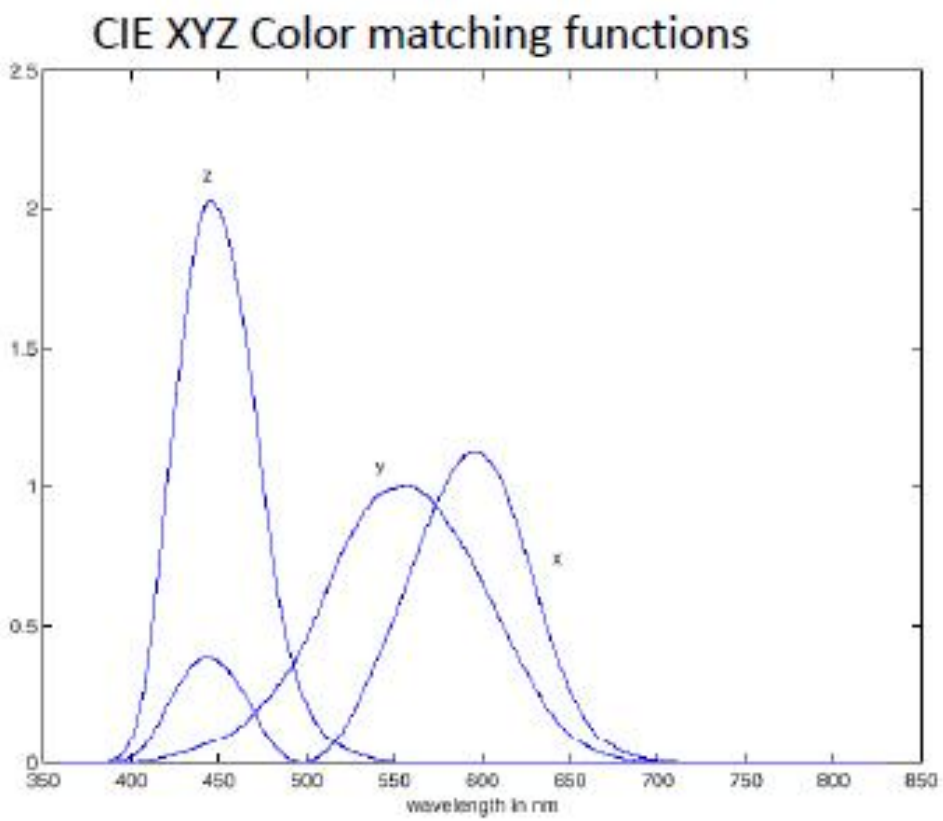
CIE color system

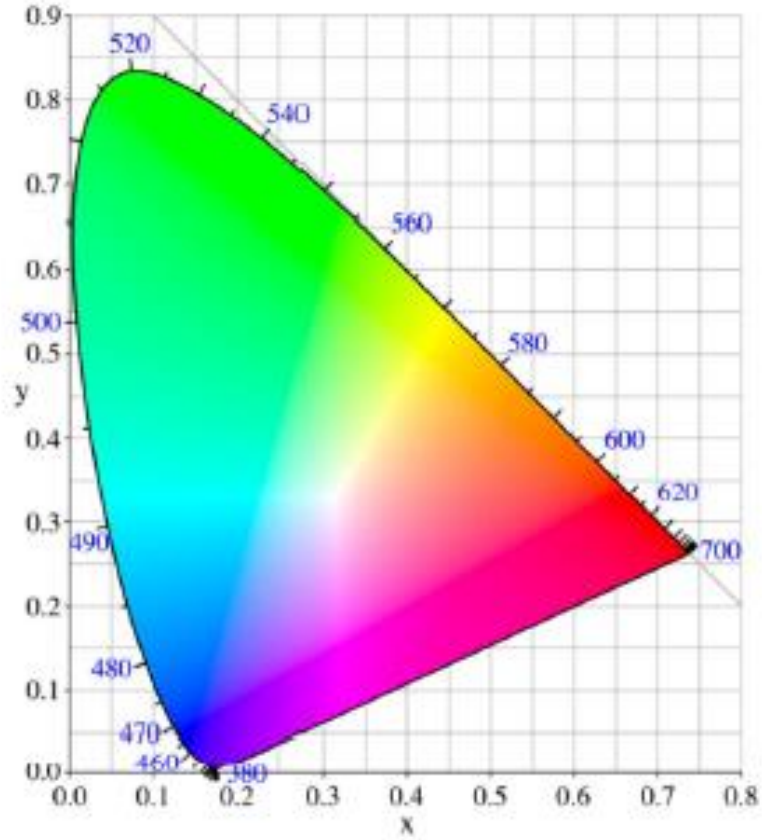
- **Commission Internationale de l'Eclairage (CIE)- this commission determines standards for color and lighting.**
- **It developed the Norm Color system (called CIEXYZ Color System) and the Lab Color System (also called the CIELAB Color System).**
- **CIE, the color system we've been using in recent object recognition work**

CIE color system

- **Usually projected to display:**

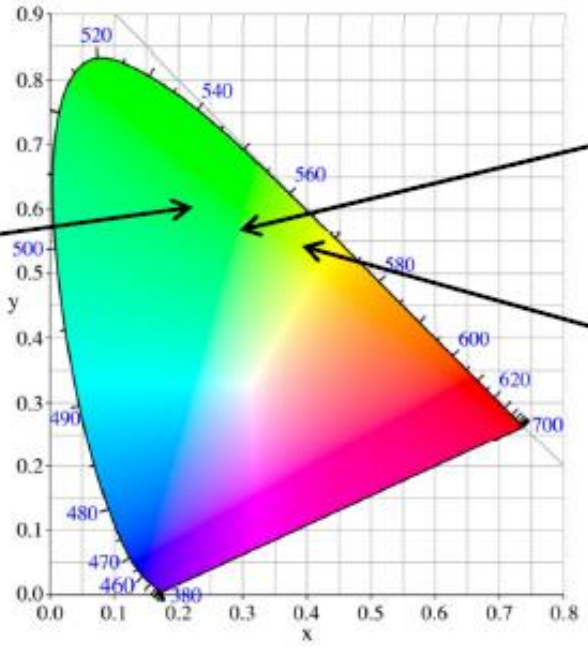
$$(x,y) = (X/(X+Y+Z), Y/(X+Y+Z))$$





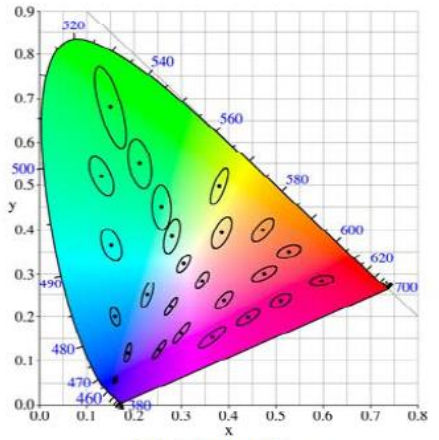
Distances in CIE color space

- **Are distances between points in a color space perceptually meaningful?**
  - **Not necessarily: CIE XYZ is not a *uniform* color space, so magnitude of differences in coordinates are poor indicator of color "distance".**

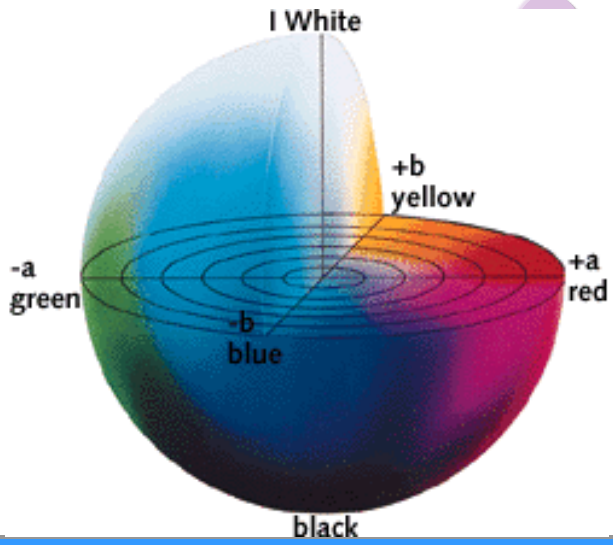


Uniform color spaces
CIELAB, Lab, L*a*b

- **CIELab Attempts to correct this limitation by remapping color space so that just noticeable differences (contained by circles distances) are more perceptually meaningful.**
- **One luminance channel (L) and two color channels (a and b).**
- **In this model, the color differences which you perceive correspond to Euclidian distances in CIELab.**
- **The a axis extends from green (-a) to red (+a) and the b axis from blue (-b) to yellow (+b). The brightness (L) increases from the bottom to the top of the three-dimensional model.**

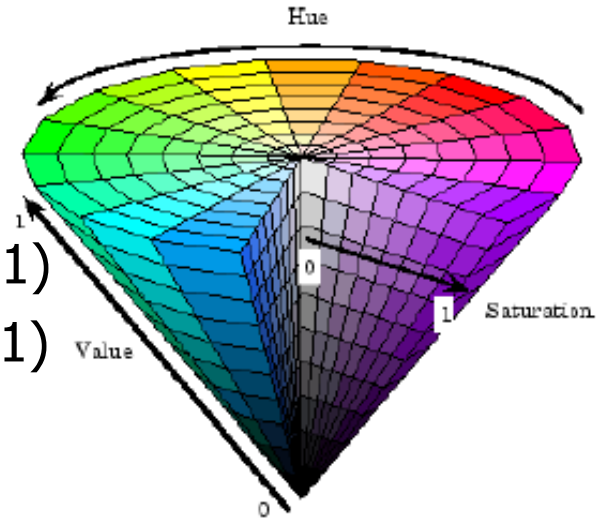


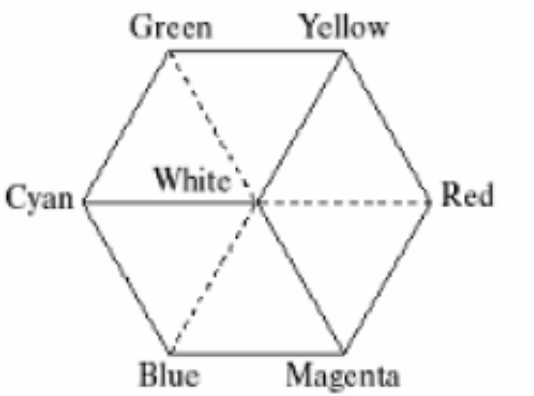
McAdam ellipses:
Just noticeable differences in color

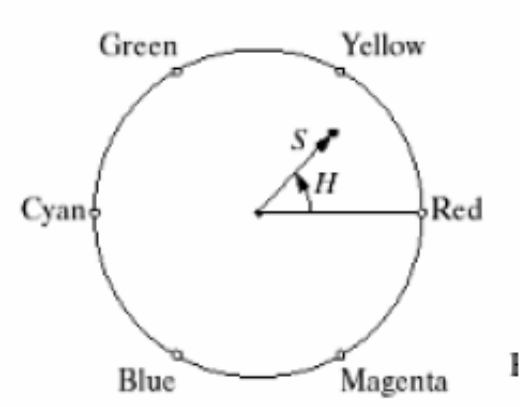


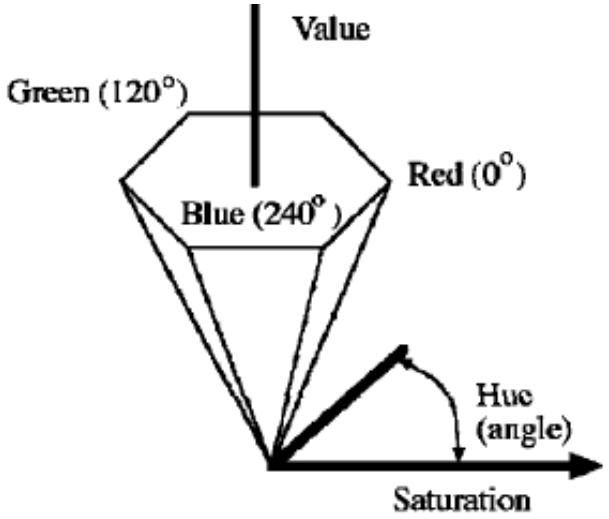
HSI (HSV) Color Space

- **H**ue, **S**aturation, **I**ntensity (**V**alue)
  - **Nonlinear** – reflects topology of colors
  - Hue is encoded as an angle (0 to 2π).
  - Saturation is the distance to the vertical axis (0 to 1)
  - Intensity is the height along the vertical axis (0 to 1)
- Matlab code: hsv2rgb, rgb2hsv.









HSI Color Space
Editing saturation of colors



(Left) Image of food originating from a digital camera; (center) saturation value of each pixel decreased 20%; (right) saturation value of each pixel increased 40%.

YIQ and YUV for TV signals

- It also known as YCbCr color space and have better compression properties
- Luminance Y encoded using more bits than chrominance values I and Q;
  - humans more sensitive to Y than I, Q
- Luminance used by black/white TVs
  - All 3 values used by color TVs
- YUV encoding used in some digital video and JPEG and MPEG compression

Conversion from RGB to YIQ

An approximate linear transformation from RGB to YIQ:

$$
\begin{aligned}
luminance \quad Y &= 0.30R + 0.59G + 0.11B \\
R - cyan \quad I &= 0.60R - 0.28G - 0.32B \\
magenta - green \quad Q &= 0.21R - 0.52G + 0.31B
\end{aligned}
$$

We often use this for color to gray-tone conversion.

- Matlab Code: ycbcr2rgb, rgb2ycbcr

Colors Clustering
for image segmentation

Razi University

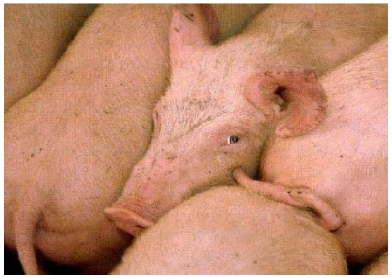
- **Can cluster on color values and pixel locations**
- **Can use connected components and an approximate color criteria to find regions**
- **Can train an algorithm to look for certain colored regions – for example, skin color**

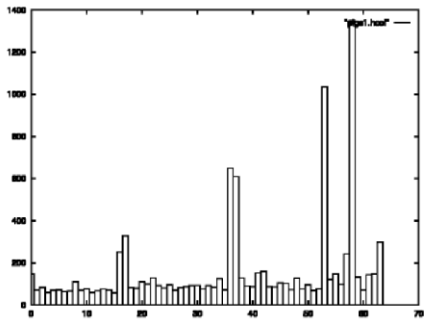
Form K-means clusters from a set of n-dimensional vectors

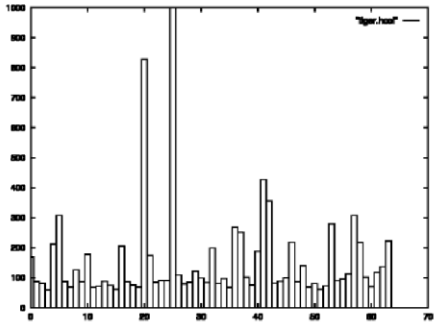
1. Set ic (iteration count) to 1
2. Choose randomly a set of K means m1(1), …, mK(1).
3. For each vector xi, compute D(xi,mk(ic)), k=1,…K
   and assign xi to the cluster Cj with nearest mean.
4. Increment ic by1,update the means to get m1(ic),…,mK(ic).
5. Repeat steps 3 and 4 until Ck(ic) = Ck(ic+1) for all k.

Color histograms can represent an image

- **Histogram is fast and easy to compute.**
- **Size can easily be normalized so that different image histograms can be compared.**
- **Can match color histograms for database query or classification.**







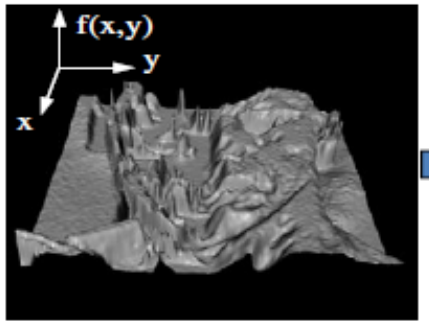
Finding a face in video frame

- (left) input video frame
- (center) pixels classified according to RGB space
- (right) largest connected component with aspect similar to a face (all work contributed by Vera Bakic)

Image Filtering

Images Representation

- In computer vision we operate on **digital** (**discrete**) images:
  - **Sample** the 2D space on a regular grid
  - **Quantize** each sample (round to nearest integer)
- Image thus represented as a **matrix** of integer values.

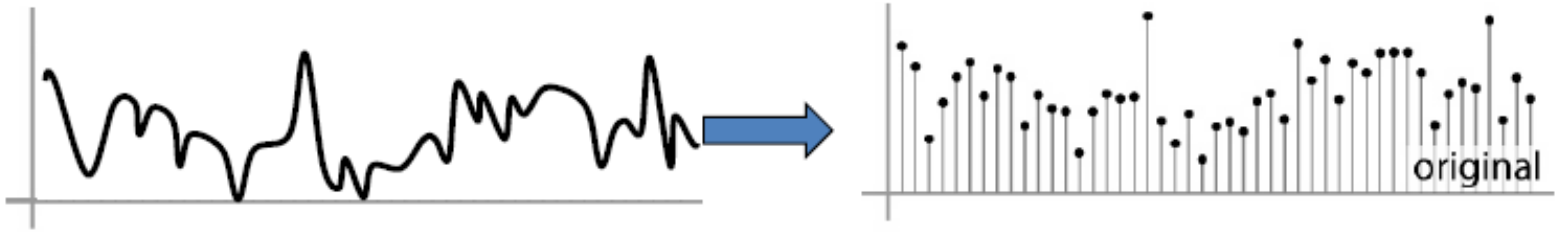


j

i

82	79	23	119	120	105	4	0
10	10	9	62	12	78	34	0
10	58	197	46	46	0	0	48
176	135	5	188	191	68	0	49
2	1	1	29	26	37	0	77
0	89	144	147	187	102	62	208
255	252	0	166	123	62	0	31
166	63	127	17	1	0	99	30

2D

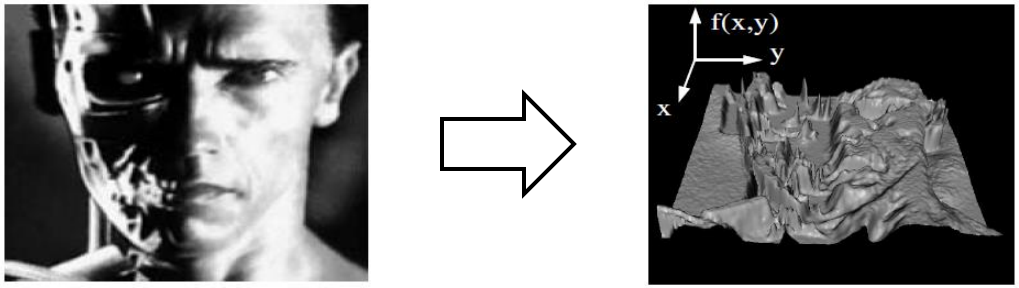


1D

Images Representation

- **We can think of an image as a function, f, from R^2 to R:**
  - **$f(x, y)$ gives the intensity at position (x, y)**
  - **Realistically, we expect the image only to be defined over a rectangle, with a finite range:**

$$f: [a, b] \times [c, d] \rightarrow [0, 1]$$



- **A color image is just three functions pasted together. We can write this as a "vector-valued" function:**

$$f(x, y) = \begin{bmatrix} r(x, y) \\ g(x, y) \\ b(x, y) \end{bmatrix}$$

Image Filtering

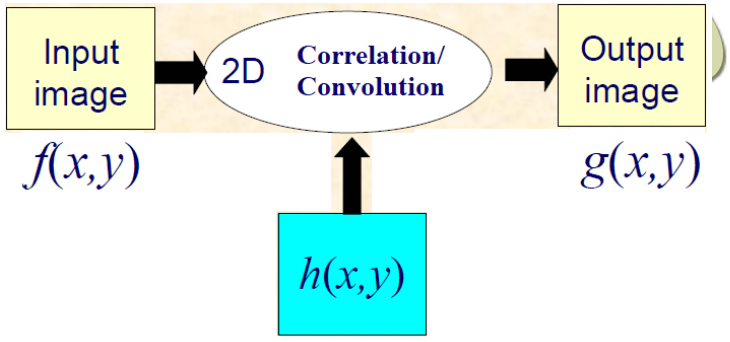
- **Filters can be thought of as a way for extracting (highlighting) or removing some characteristics from image function f .**

- **Usually the filtering can be done in two domains:**
  - **Time (Spatial) domain**
  - **Frequency domain**

- **In each domain corresponding kernel/mask is prepared and applied on original/transformed version of images**
  - **Filtering can be performed in the spatial domain by convolution with specifically designed kernels (filter array), or in the frequency (Fourier) domain by masking specific frequency regions.**

Spatial domain Image Filtering

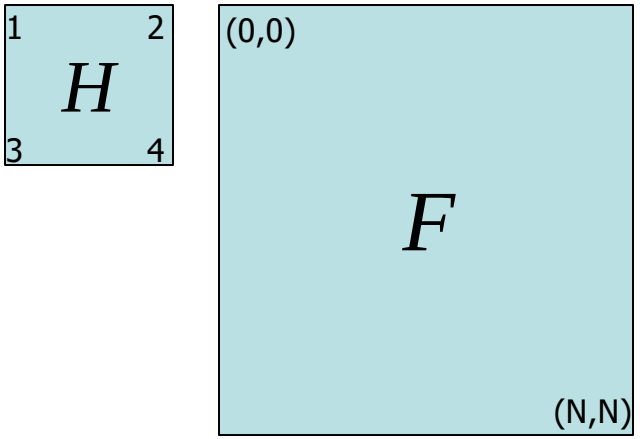
- In the Spatial domain, filters allow local image neighborhood to influence our description and features
  - Averaging for smoothing/bluring to reduce noise
  - Derivatives to locate contrast, gradient, edge
  - Template matching to locate requested patterns locations.
- Filters have highest response on neighborhoods that "look like" it; can be thought of as template matching.
- Applying the filter kernel, in space domain, is performed by two ways:
  - **Correlation** between kernel and image pixels' neighborhoods.
  - **Convolution** of kernel and image pixels' neighborhoods.



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

Spatial domain Image Filtering
Correlation Filtering

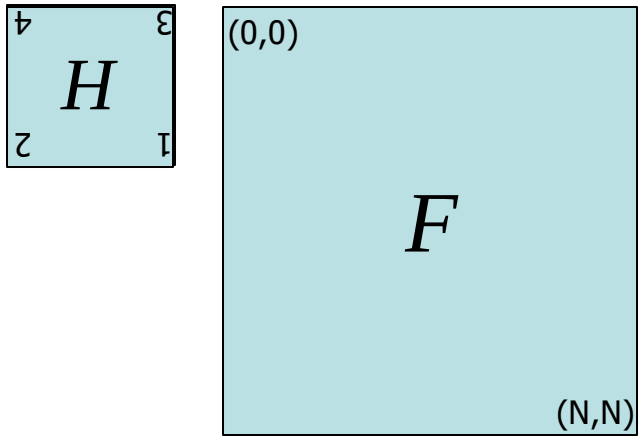
- **Correlation:** $$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u, j+v]$$
- **This is called cross-correlation, denoted** $G = H \otimes F$
- **Filtering an image**
  - **Replace each pixel by a weighted combination of its neighbors.**
  - **The filter "kernel" or "mask" is the prescription for the weights in the linear combination.**



Spatial domain Image Filtering
Convolution Filtering

- **Convolution:** $$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u,j-v]$$
  - **Flip the filter in both dimensions (bottom to top, right to left)**
  - **Then apply cross-correlation**

$$G = H \star F$$



Spatial domain Image Filtering
Filter Properties

- **Shift invariant:**
  - **Operator behaves the same everywhere, *i.e.* the value of the output depends on the pattern in the image neighborhood, not the position of the neighborhood.**

- **Linear:**
  - **Superposition:** $h * (f1 + f2) = (h * f1) + (h * f2)$
  - **Scaling:** $h * (k f) = k (h * f)$

- **Separability**: In some cases, filter is separable, and we can factor into two steps:
  - **Convolve all rows**
  - **Convolve all columns**

Spatial domain Image Filtering
Filter Properties

- **Averaging**
  - Values positive
  - Sum to 1 → constant regions same as input
  - Amount of smoothing proportional to mask size
  - Remove "high-frequency" components; "low-pass" filter
- **Derivatives**
  - Opposite signs used to get high response in regions of high contrast
  - Sum to 0 → no response in constant regions
  - High absolute value at points of high contrast
- **Filters act as templates**
  - Highest response for regions that "look the most like the filter"

Spatial domain Image Filtering
Properties of Convolution

- **Linear & shift invariant**
- **Commutative:**

$$f * g = g * f$$

- **Associative**

$$(f * g) * h = f * (g * h)$$

- **Identity:**

  unit impulse e = [..., 0, 0, 1, 0, 0, ...]. f * e = f

- **Differentiation:**

$$\frac{\partial}{\partial x}(f * g) = \frac{\partial f}{\partial x} * g$$

Spatial domain Image Filtering
Convolution Calculation C++ Code

$$\text{result}(i,j) = \sum_m \sum_n \text{kernel}(m,n) \cdot \text{image}(i-m, j-n)$$

```
int  i, j, m, n, sum, image[iend][jend],
     kernel[mend][nend], result [iend][jend];

for (i = mend; i < iend; i++) {
    for (j = nend; j < jend; j++) {
        sum = 0;
        for ( m = 0; m < mend; m++) {
            for ( n = 0; n < nend; n++ ) {
                sum += kernel[m][n] * image[i-m][j-n];
            }
        }
        result[i][j] = sum/(mend*nend);
    }
}
```

Spatial domain Image Filtering
Convolution vs. Correlation

- **Correlation**

  Matlab Code:

  `filter2`

  `imfilter`

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u, j+v]$$

$$G = H \otimes F$$

Note the difference!

- **Convolution**

  Matlab Code:

  `conv2`

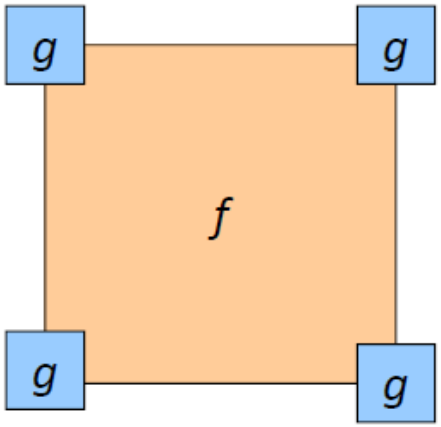
$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u, j-v]$$

$$G = H \star F$$

- **Note**
  - If $H[-u,-v] = H[u,v]$, then **correlation ≡ convolution.**

Spatial domain Image Filtering
Boundary Issues

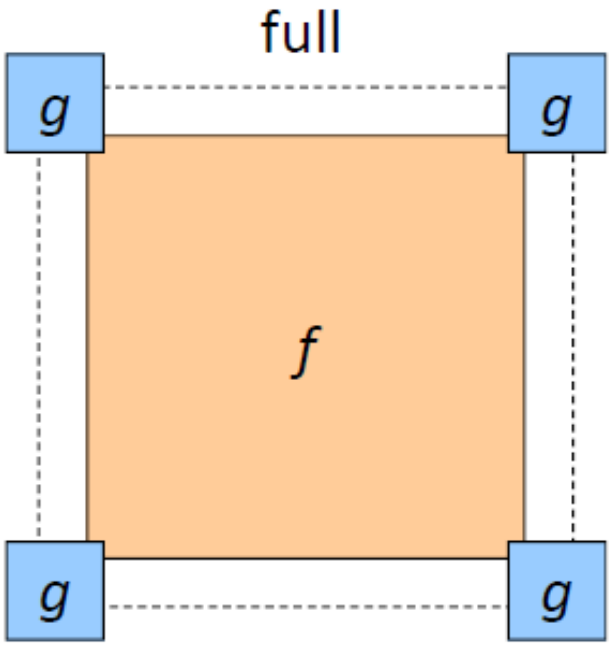
- **What about near the edge?**
- **The filter window falls off the edge of the image**
- **Need to extrapolate**

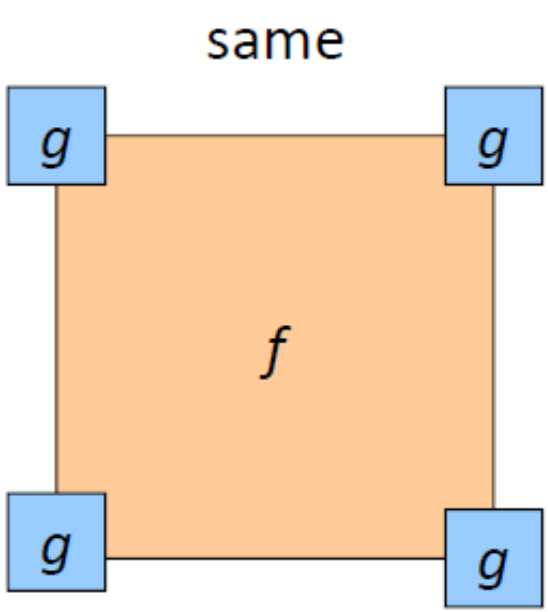


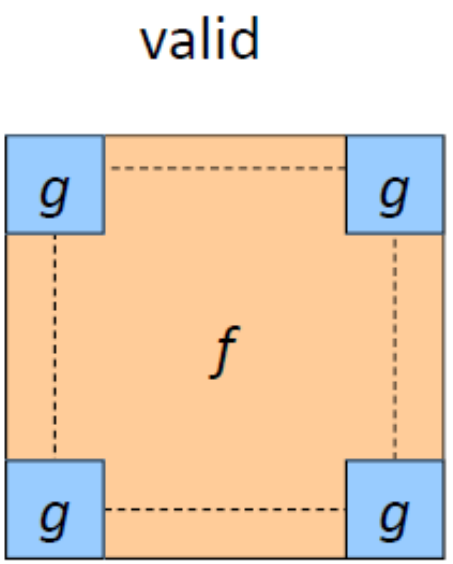
- **Methods (MATLAB):**
  - **clip filter (black):** **imfilter(f, g, 0)**
  - **wrap around:** **imfilter(f, g, 'circular')**
  - **copy edge:** **imfilter(f, g, 'replicate')**
  - **reflect across edge:** **imfilter(f, g, 'symmetric')**

Spatial domain Image Filtering
Boundary Issues

- **What is the size of the output?**
- **MATLAB: filter2(g, f, *shape*)**
  - ***shape*** **= 'full': output size is sum of sizes of f and g**
  - ***shape*** **= 'same': output size is same as f**
  - ***shape*** **= 'valid': output size is difference of sizes of f and g**

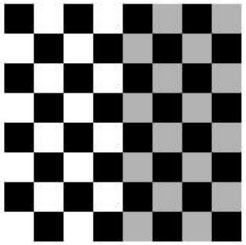
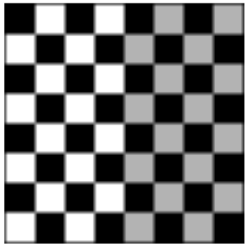
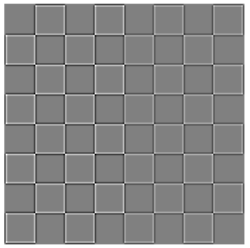




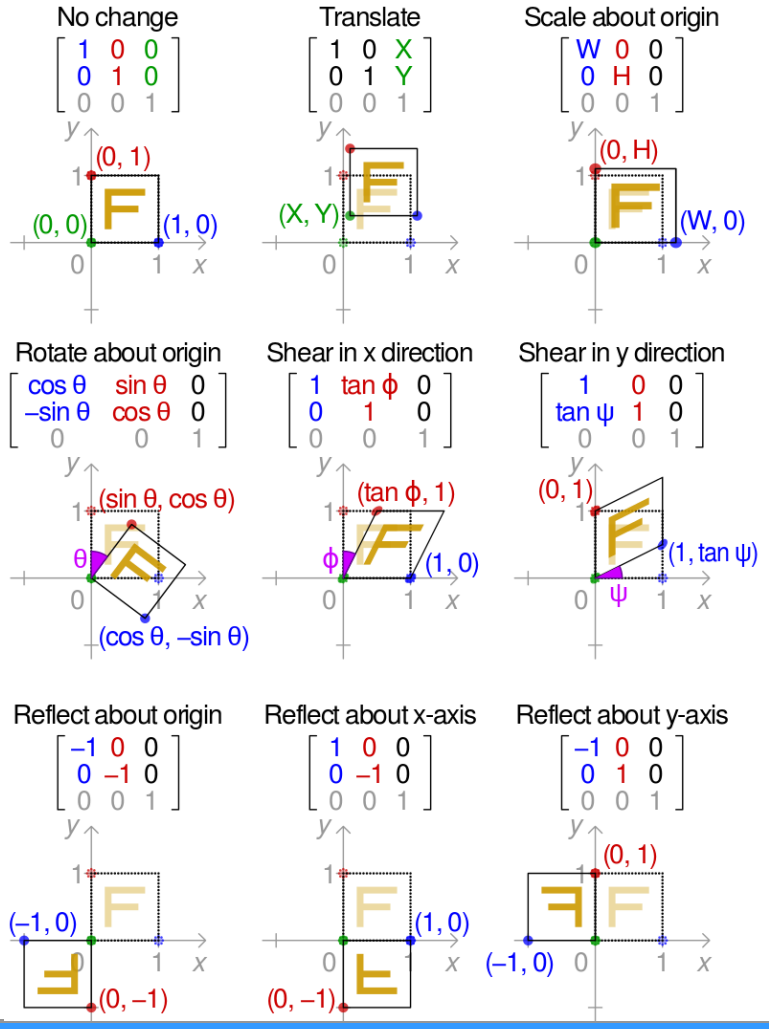


Spatial domain Image Filtering
Examples

Value based Transforms

Filter type	Kernel or mask	Example
Original Image	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	Identity (Original) 
Spatial Lowpass	$\frac{1}{9} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	3 × 3 Mean Blur 
Spatial Highpass	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	Laplacian Edge Detection 

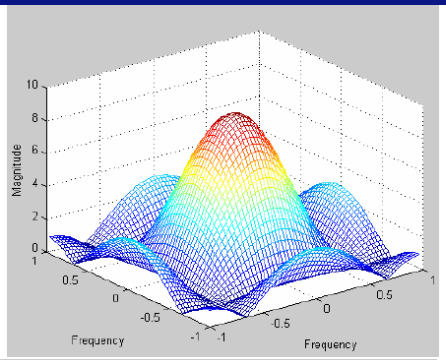
Position based Transforms



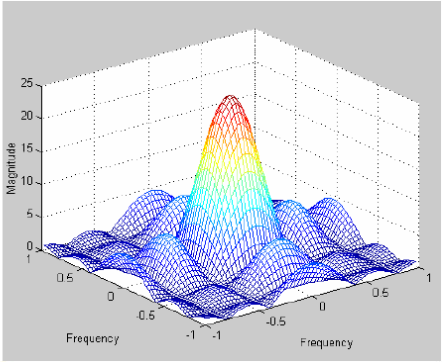
Spatial domain Image Filtering
Low Pass Filter Examples

Razi University

$$h_1 = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

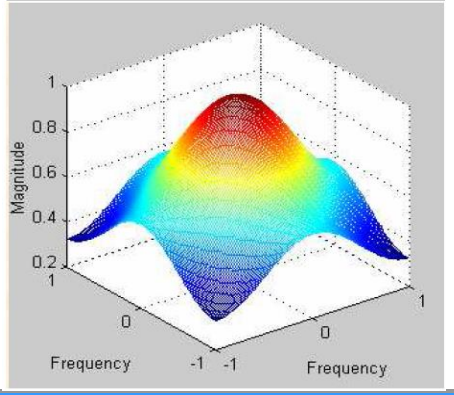


$$h_2 = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$



$$h_3 = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Gaussian



Source image

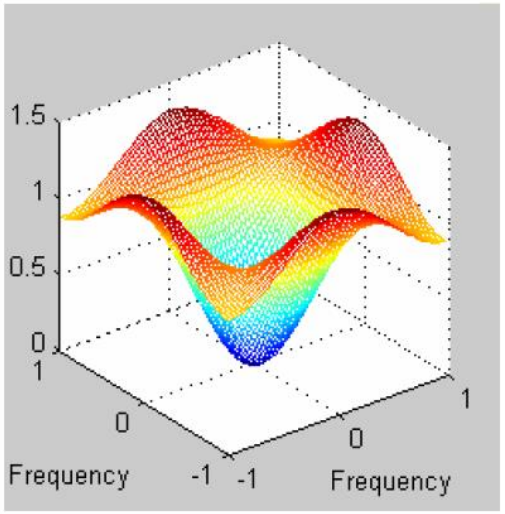
3x3 mask

Gaussian mask

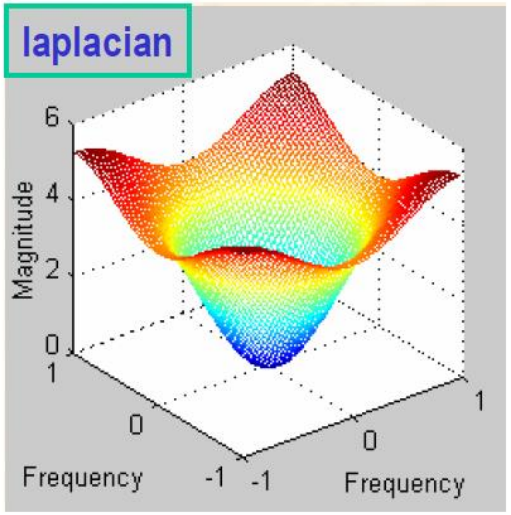
5x5 mask

Spatial domain Image Filtering
High Pass Filter Examples

$$h_1 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$



$$h_2 = \begin{bmatrix} 0.17 & 0.67 & 0.17 \\ 0.67 & -3.33 & 0.67 \\ 0.17 & 0.67 & 0.17 \end{bmatrix}$$



Spatial domain Image Filtering
High Boost Filter

$$f(x,y) = f_L(x,y) + f_H(x,y)$$

$$f_{HB}(x,y) = Af(x,y) - f_L(x,y) =$$
$$= (A-1)f(x,y) + f(x,y) - f_L(x,y) =$$
$$= (A-1)f(x,y) + f_H(x,y), \qquad A \geq 1$$

$$h_{HB} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9A-1 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

A=?

Laplacian Filter

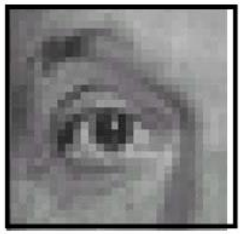
HB with A=1.1

HB with A=1.5

Spatial domain Image Filtering
Examples

Original

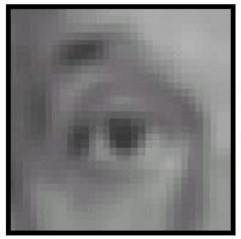
$$\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{matrix}$$



Shifted left by 1 pixel with correlation

Original

$$\frac{1}{9}\begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$



Blur (with a box filter)

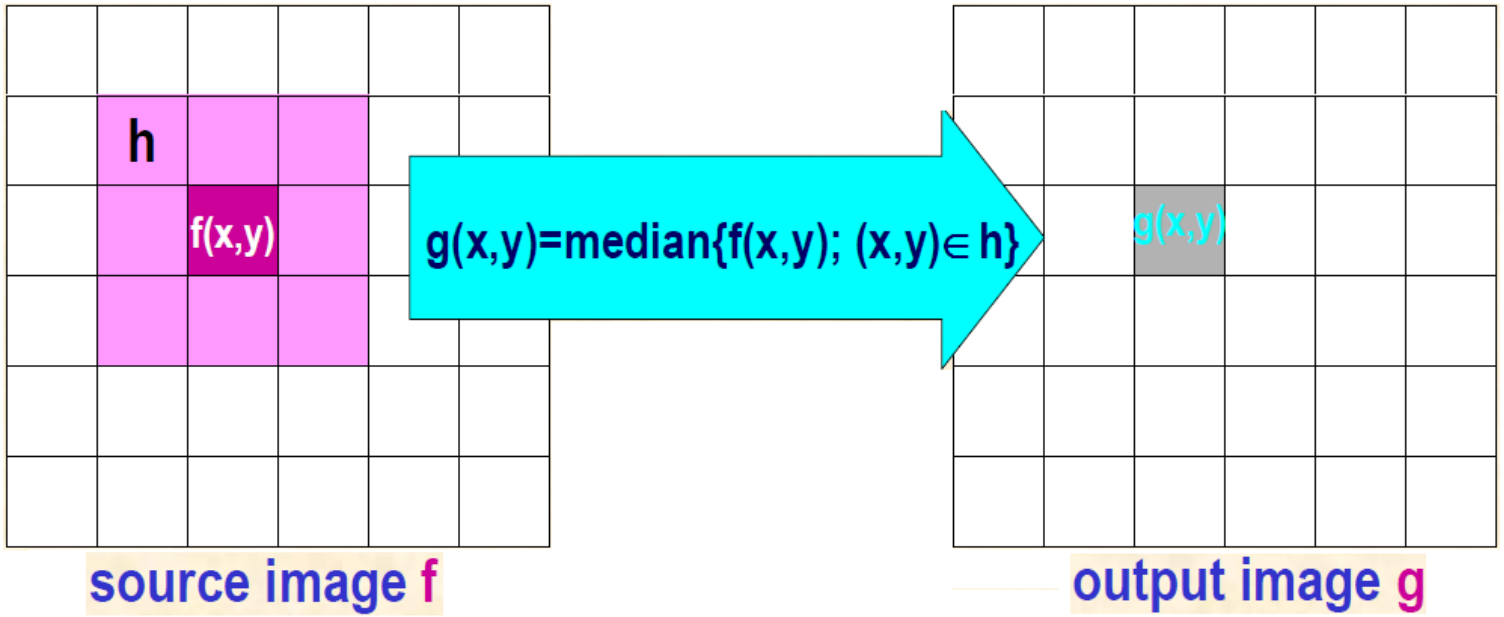
Original

$$\begin{matrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{matrix} - \frac{1}{9}\begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

Sharpening filter
- Accentuates differences with local average

Spatial Domain Image Filtering
Median Filter

The median m of a set of values (e.g. image pixels in the filtering mask) is such that half the elements in the set are less than m and other half are grater than m .



$x(n)=\{1, 5, -7, 101, -25, 3, 0, 11, 7\}$

median

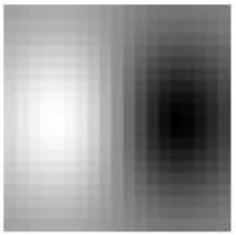
Sorted sequence of elements: $x_s(n)=\{-25, -7, 0, 1, \mathbf{3}, 5, 7, 11, 101\}$

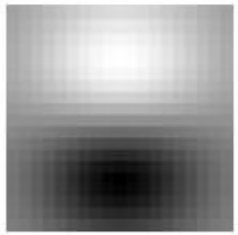
Spatial Domain Image Filtering
Median Filter

- **Excellent in reducing impulsive noise** (odd size smaller than half size of the filtering mask)
- **Keeps sharpness of image edges** (as opposed to linear smoothing filters)
- **Values of the output image are equal or smaller than the values of the input image** (no rescaling)
- **Large computing cost involved**

Spatial Domain Image Filtering
Filters as Features (Template matching)

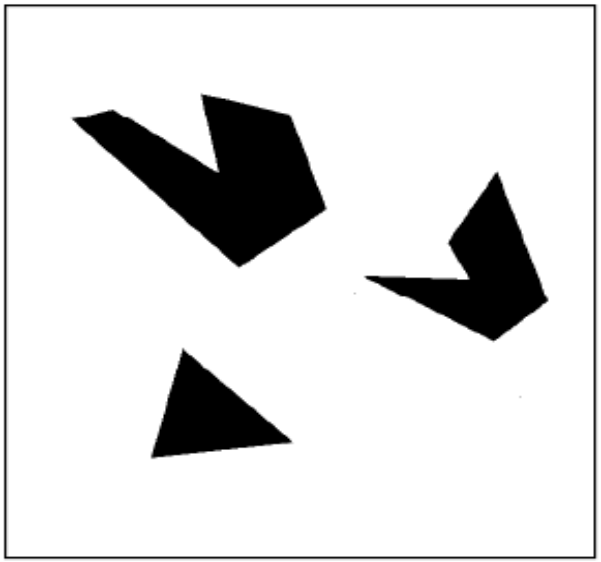
- Previously, thinking of filtering as a way to remove or reduce **noise**
- Now, consider how filters will allow us to abstract higher-level "**features**".
  - Map raw pixels to an intermediate representation that will be used for subsequent processing
  - Goal: reduce amount of data, discard redundancy, preserve what's useful
- **Filters as templates**: Filters look like the effects they are intended to find - "**matched filters**"



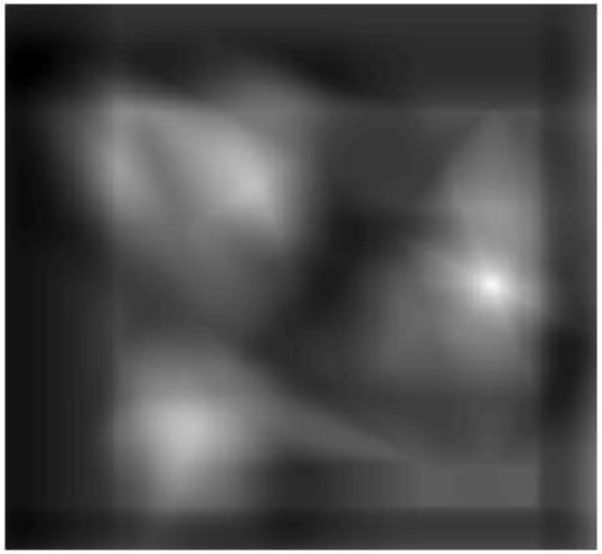


Spatial Domain Image Filtering
Template matching

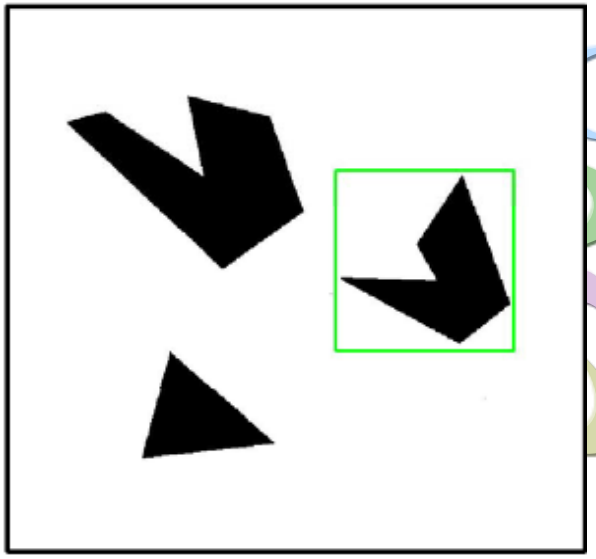
- **Use normalized cross-correlation score to find a given pattern (template) in the image.**
  - **Normalization needed to control for relative brightnesses**



Scene



Correlation map

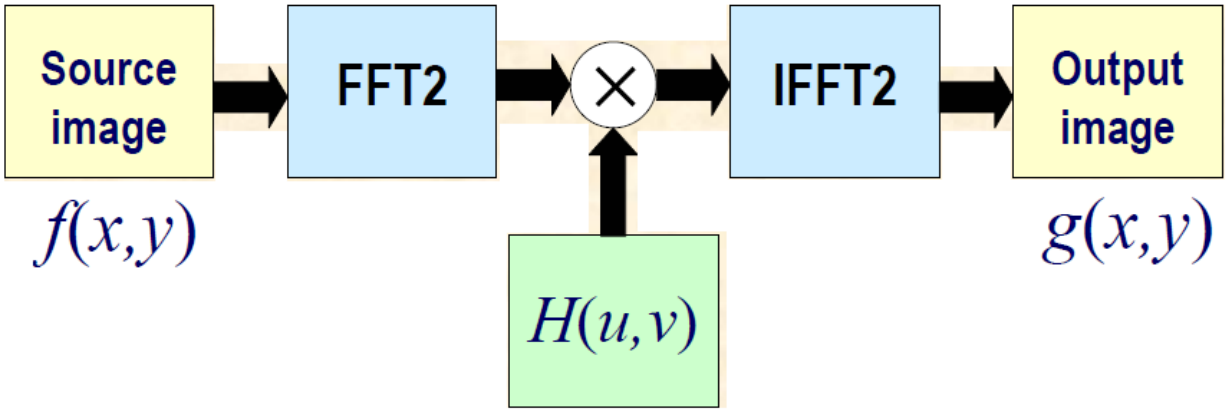


Detected template

Template (mask)

Frequency Domain Filtering

- **In the frequency domain filtering, at first,**
  - **The input image (f) is transferred to frequency domain by employing Fourier transform**
  - **Then extract/remove corresponding frequency component from it by multiplying with proper mask (H).**
  - **Finally, by applying inverse transform, the filtered image is obtained.**



$$g(x,y) = IF\{\ H(u,v) \times F\{f(x,y)\}\ \}$$

Frequency Domain Filtering
The Discrete Fourier Transform

- **The key point of the frequency domain filtering is Fourier transform which computed as below for 2D data such as images:**

- **Fourier Transform:**

$$F[u,v] = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f[x,y]e^{-i\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$$

- **Inverse Fourier Transform (reconstruction):**

$$f[x,y] = \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F[u,v]e^{+i\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}$$

The Discrete Fourier Transform

Representation in terms of magnitude and phase

$$F[u,v] = R[u,v] + iJ[u,v] = \left|F(u,v)\right| e^{i\Phi(u,v)}$$

Magnitude spectrum

$$\left|F(u,v)\right| = \sqrt{R(u,v)^2 + I(u,v)^2} = \sqrt{F(u,v)F^*(u,v)}$$

Power spectrum

$$F(u,v)F^*(u,v)$$

Phase spectrum

$$\Phi(u,v) = \tan^{-1}\left(\frac{I(u,v)}{R(u,v)}\right)$$

The Discrete Fourier Transform

The Fourier Transform of a real-valued image is conjugate-symmetric

$$F[u,v] = F^*(-u,-v)$$

Therefore the magnitude spectrum is even symmetric

$$|F(u,v)| = |F(-u,-v)|$$

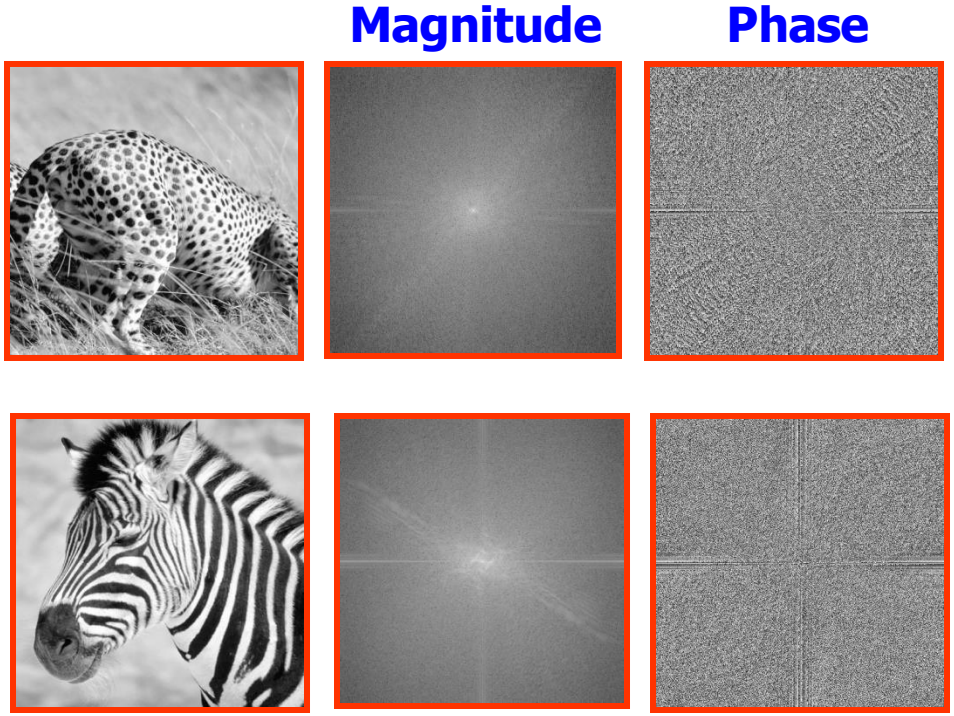
and the phase spectrum is odd symmetric

$$\Phi(u,v) = -\Phi(-u,-v)$$

Note that the Fourier spectrum is often re-arranged for display such that the zero-frequency component is in the center.

The Discrete Fourier Transform

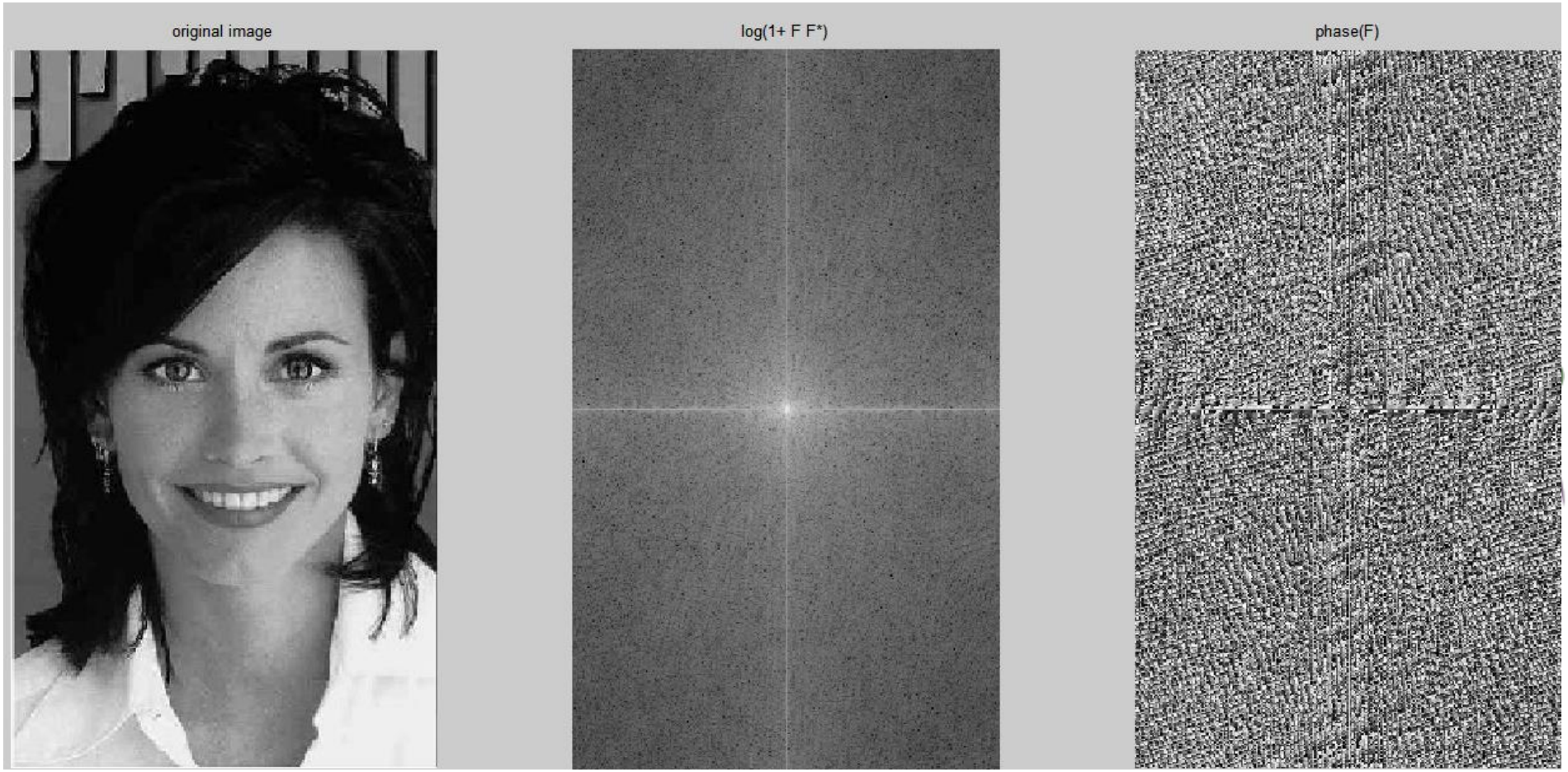
- Fourier transform of a real function is complex
  - difficult to plot and visualize
  - we can think of the phase and magnitude of the transform



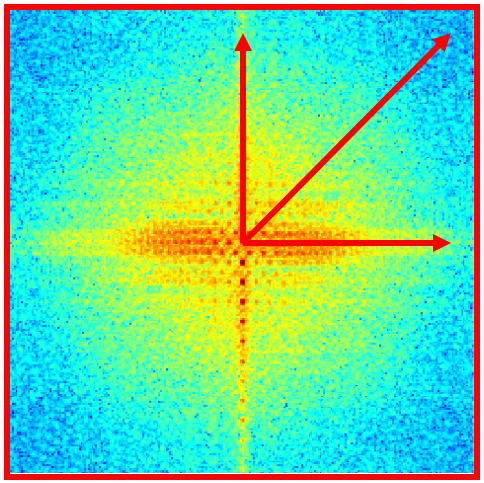
The Discrete Fourier Transform
Matlab Code

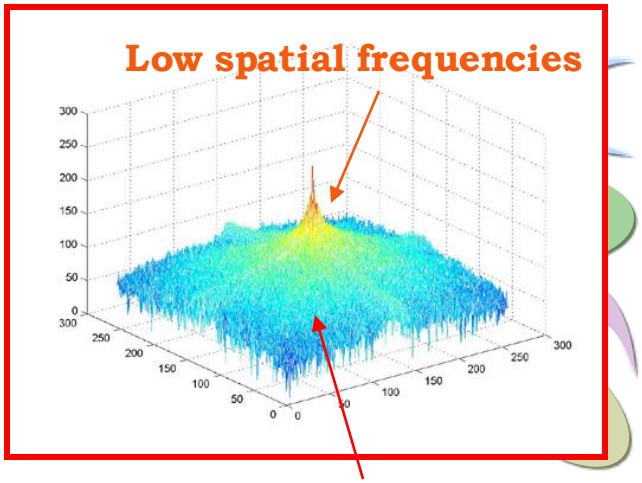

```
im = imread('waves.bmp');

F = fft2(im);
%fft2(X) returns the two-dimensional discrete Fourier transform (DFT) of X,
%computed with a fast Fourier transform (FFT) algorithm. The result Y is the same
%size as X.

F = fftshift(F);
%fftshift(X) rearranges the outputs of fft, fft2, and fftn by moving the zero-
%frequency component to the center of the array. It is useful for visualizing a
%Fourier transform with the zero-frequency component in the middle of the spectrum

%angle(X): Phase angle
figure;
subplot(1,3,1),imagesc(im); colormap gray, axis image, axis off,
        title('original image');
subplot(1,3,2),imshow(log(1+F.*conj(F)), []); title('log(1+ F F*)');
subplot(1,3,3),imshow(angle(F), []); title('phase(F)');

imspect(im);
%IMSPECT - Plots image amplitude spectrum averaged over all orientations.
showsurf(real(F));  %SHOWSURF - shows parametric surface in a convenient way
```

The Discrete Fourier Transform
Matlab Code Output



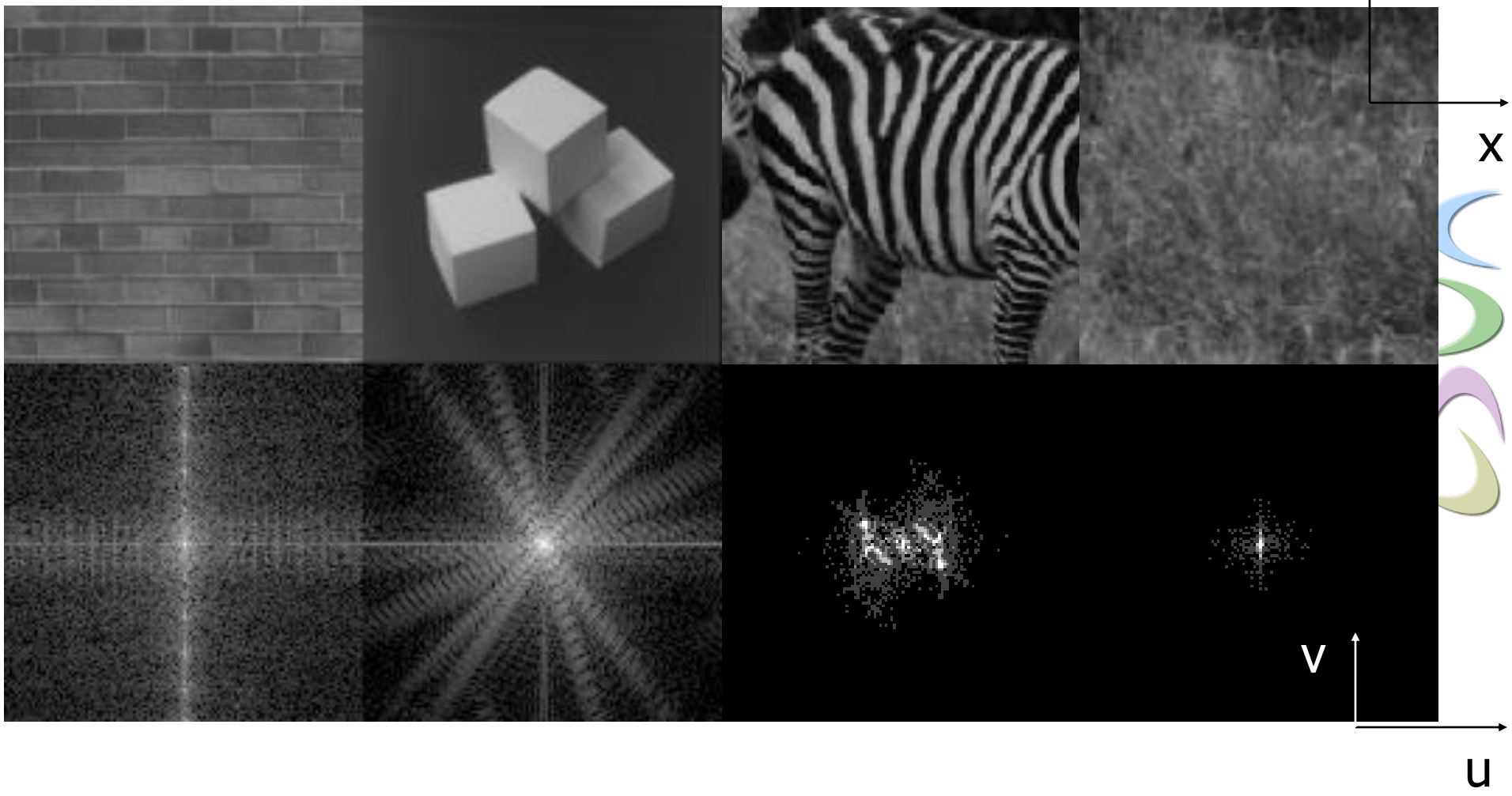
How to interpret a Fourier Spectrum



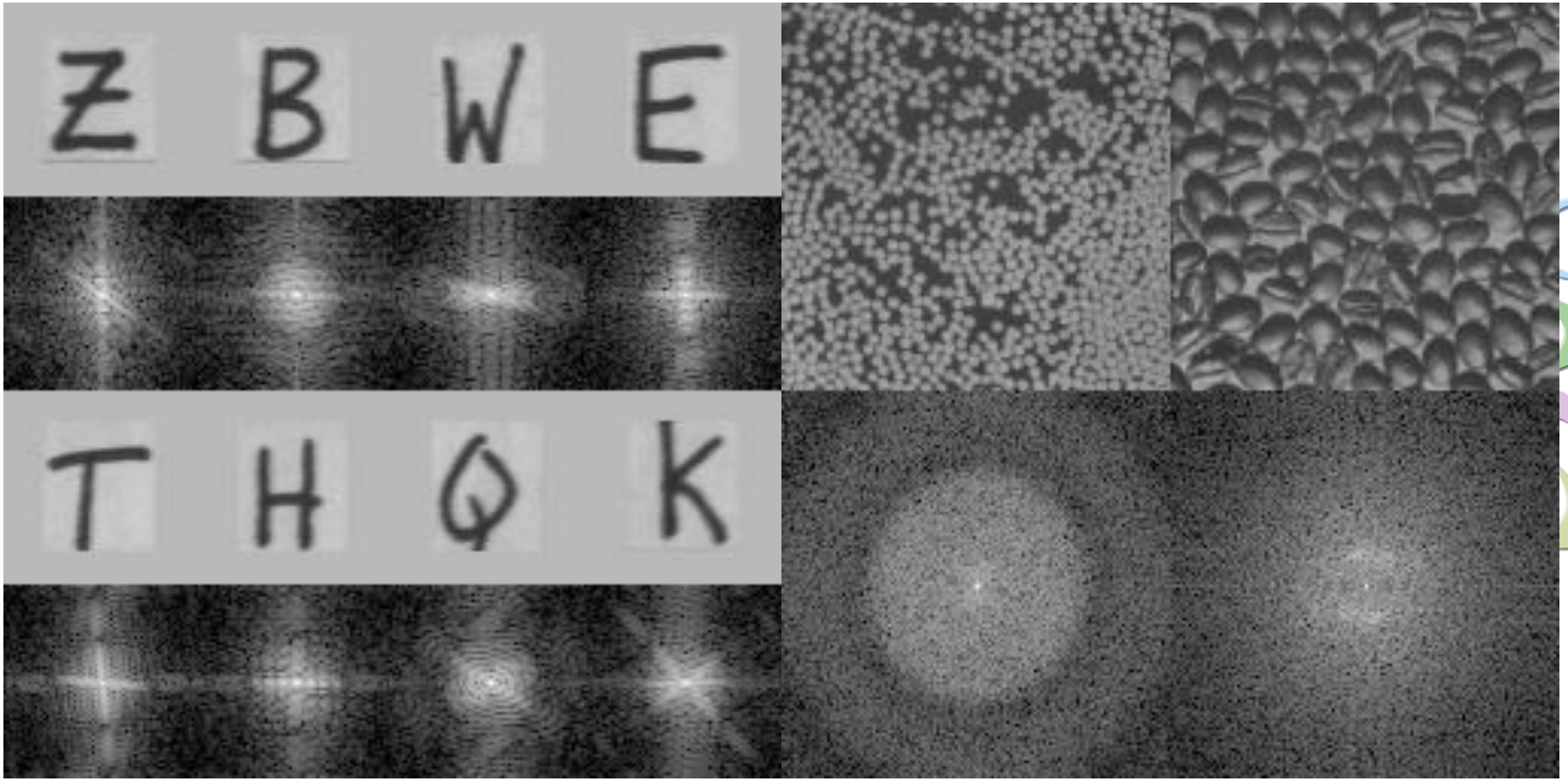


Log power spectrum

The Discrete Fourier Transform
Matlab Code Output

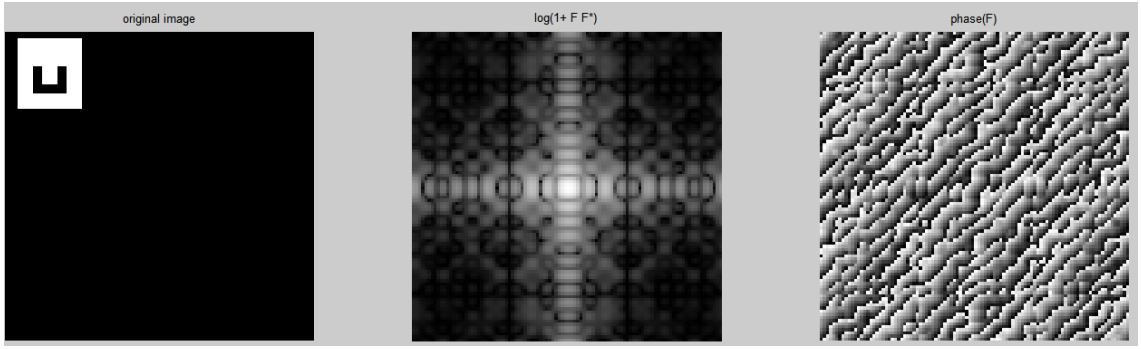


The Discrete Fourier Transform
Matlab Code Output



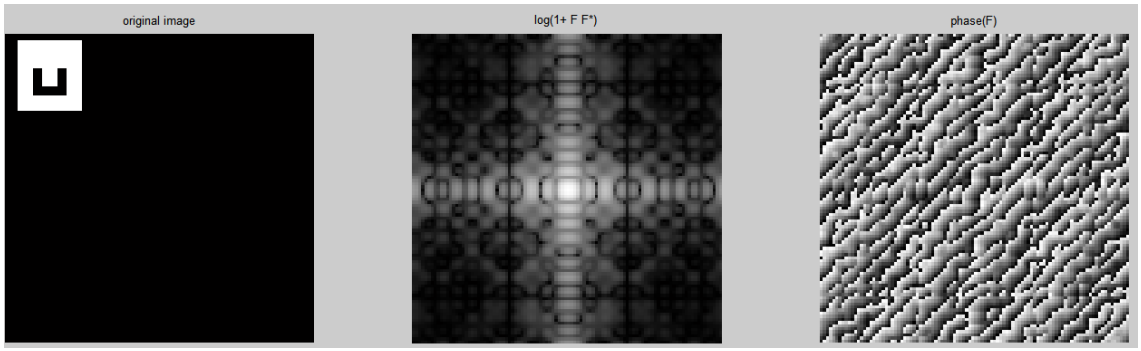
The Discrete Fourier Transform
Properties

Shift Theorem: Shifting the original pattern in (x, y) by some 2D displacement (α, β) merely multiplies its 2DFT by $\exp(-i(\alpha\mu + \beta\nu))$. Thus the 2DFT of the shifted pattern $f(x-\alpha, y-\beta)$ is: $F(\mu, \nu)\exp(-i(\alpha\mu+\beta\nu))$.



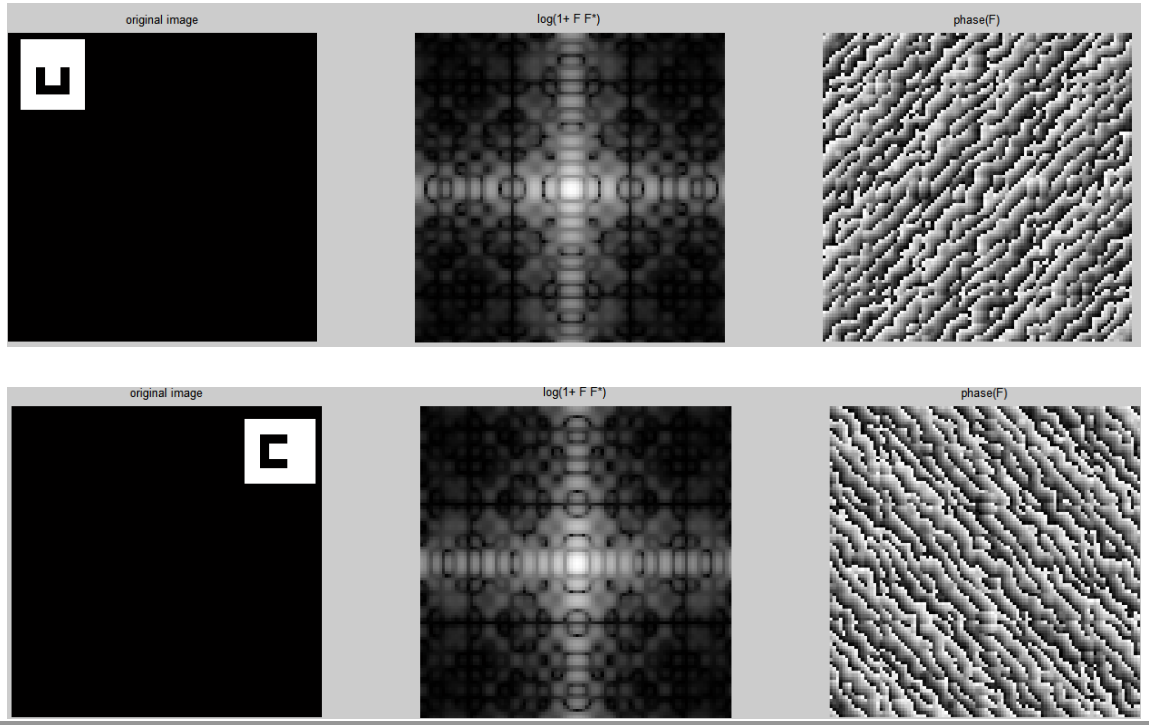
The Discrete Fourier Transform
Properties

Similarity Theorem: If the size of the original pattern $f(x, y)$ changes (shrinks/expands), say by a factor α in the x-direction, and by a factor β in the y-direction, becoming $f(\alpha x, \beta y)$, then the 2DFT of the pattern, $F(\mu, \nu)$, also changes (expands/shrinks) by the reciprocal of those factors and with similarly scaled amplitude. It becomes: $\frac{1}{|\alpha\beta|} F(\frac{\mu}{\alpha}, \frac{\nu}{\beta})$.

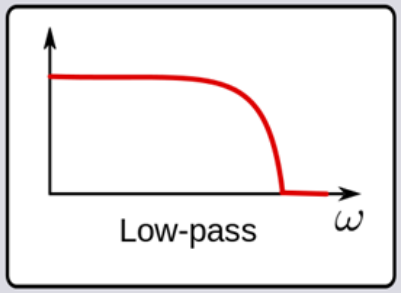


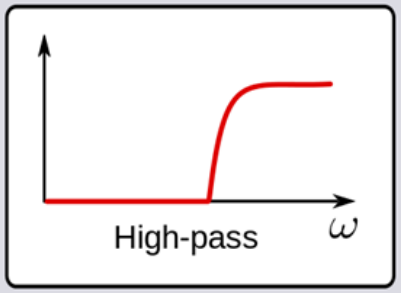
The Discrete Fourier Transform
Properties

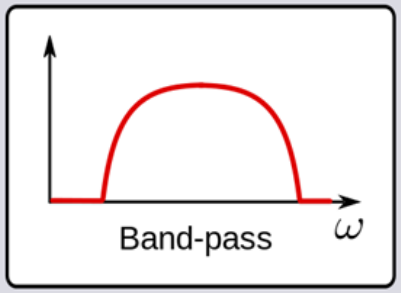
Rotation Theorem: If the original pattern $f(x, y)$ rotates through some angle θ, becoming $f(x\cos(\theta)+y\sin(\theta), -x\sin(\theta)+y\cos(\theta))$, then its 2DFT $F(\mu, \nu)$ also just rotates through the same angle. It becomes: $F(\mu\cos(\theta)+\nu\sin(\theta), -\mu\sin(\theta)+\nu\cos(\theta))$.

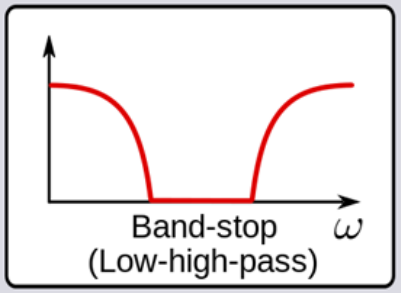


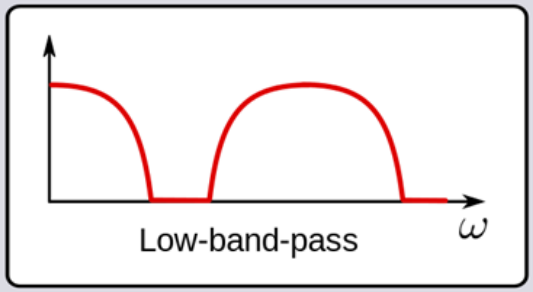
Frequency domain Filtering
1D Examples

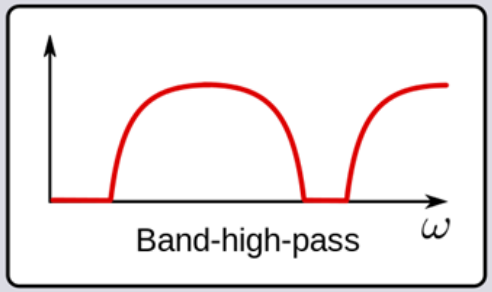


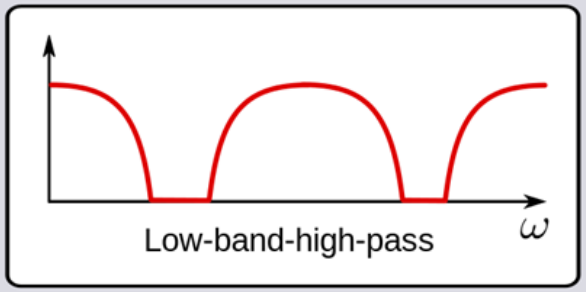






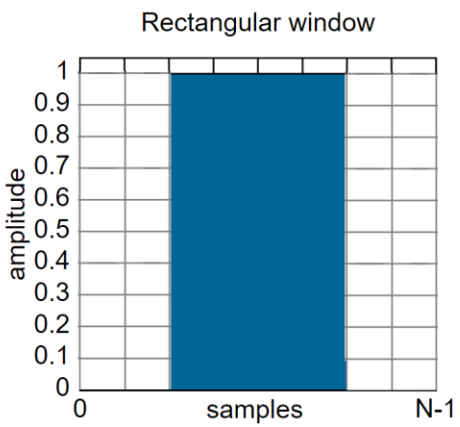




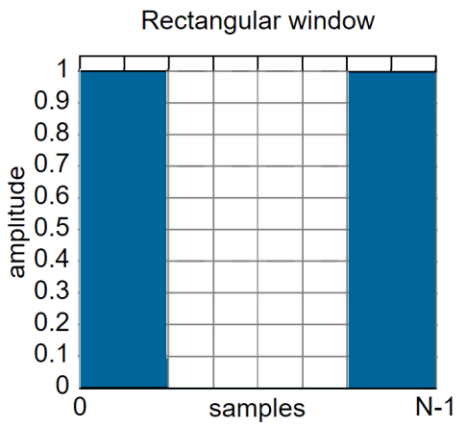


Frequency domain Filtering
Examples

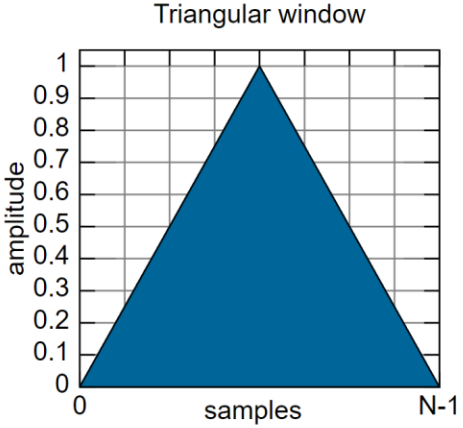
Razi University



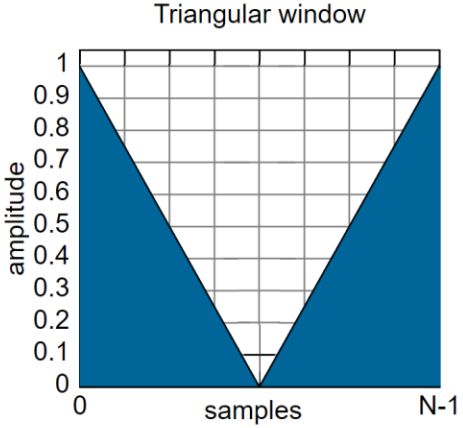
Low Pass Filter



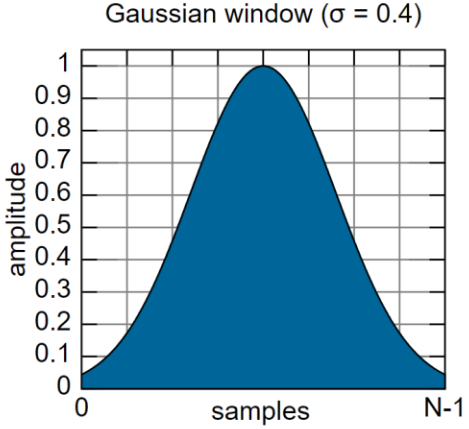
High Pass Filter



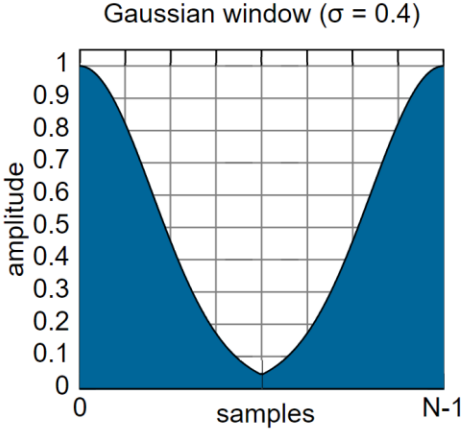
Low Pass Filter



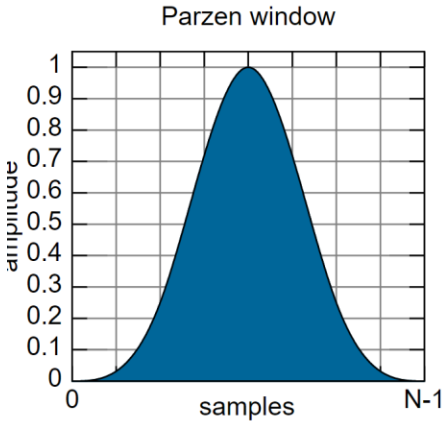
High Pass Filter



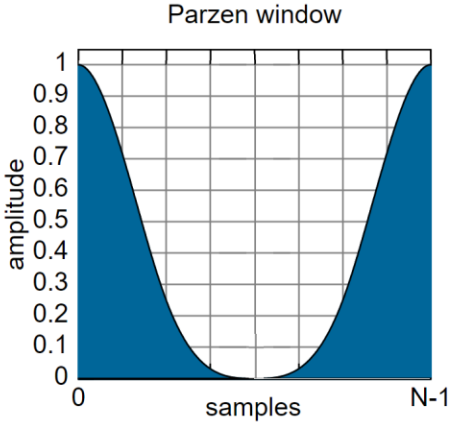
Low Pass Filter



High Pass Filter

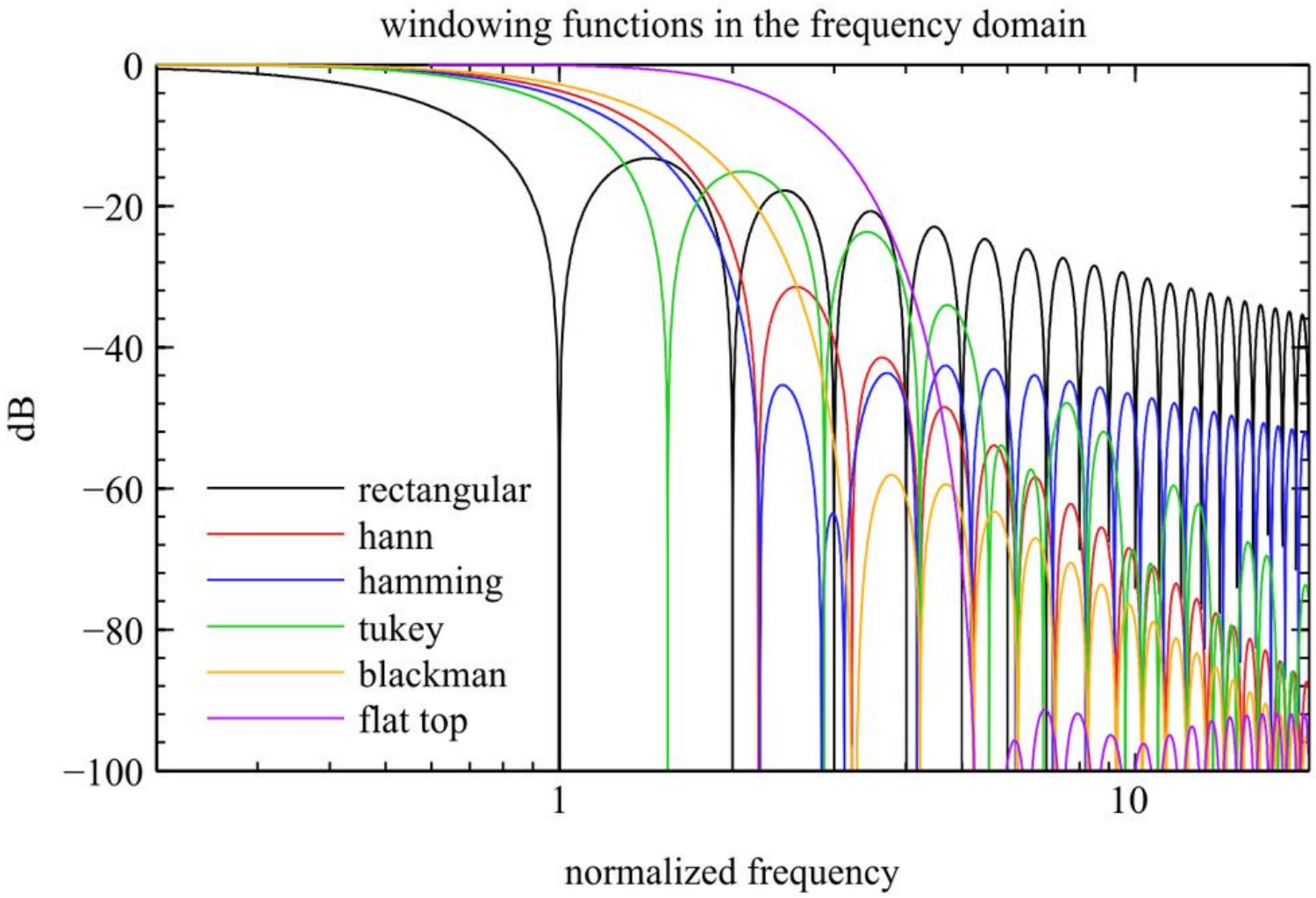


Low Pass Filter



High Pass Filter

Frequency domain Filtering
Examples

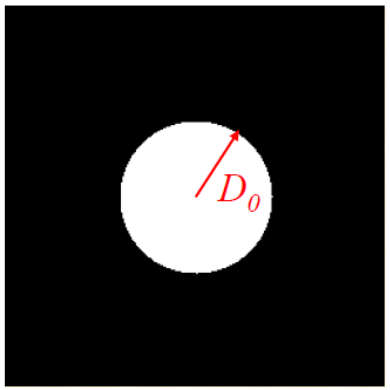


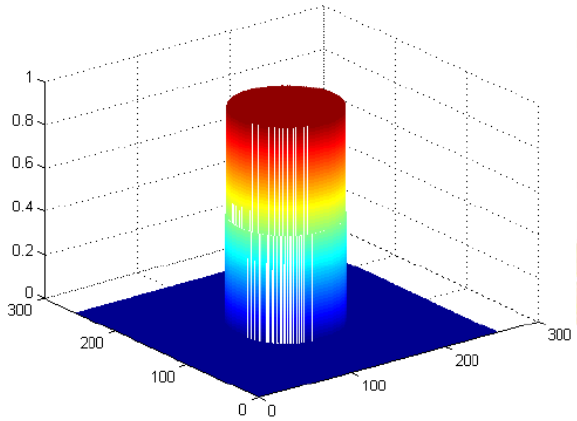
Frequency domain Filtering
2D Examples (Low Pass Filter)

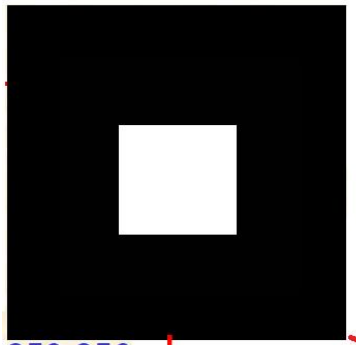
Ideal low-pass filter

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

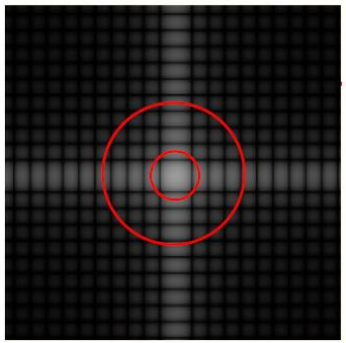
$$D(u,v)=\sqrt{u^2+v^2}$$

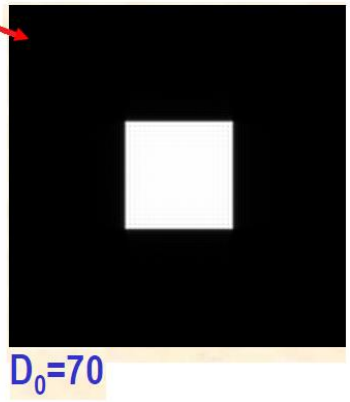






FFT





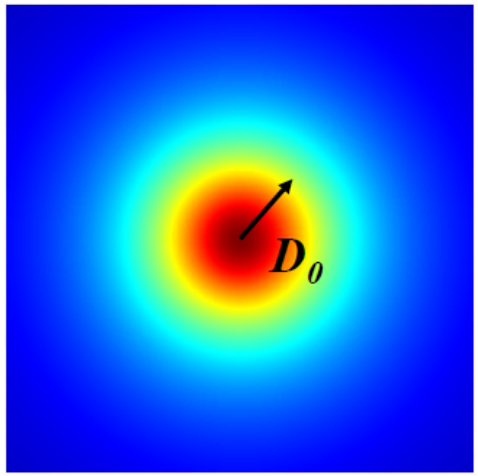
Frequency domain Filtering
2D Examples (Low Pass Filter)

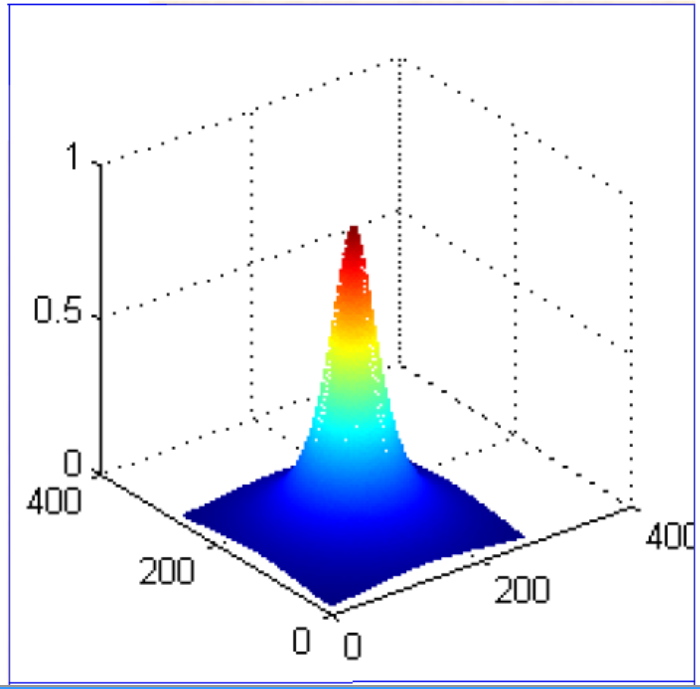
Butterworth low-pass filter

$$H(u,v)=\frac{1}{1+(\sqrt{2}-1)[D(u,v)/D_0]^{2n}}$$

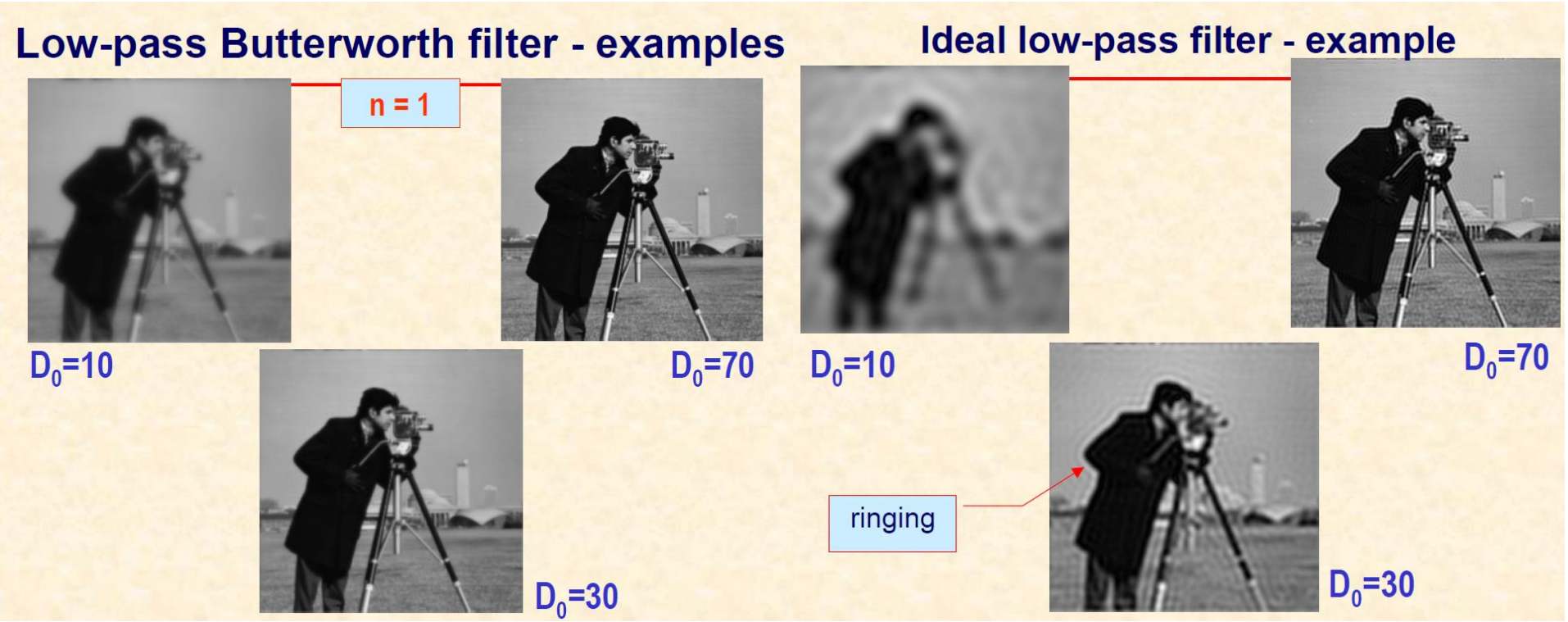
$$D(u,v)=\sqrt{u^2+v^2}, \quad n=1,2,\ldots$$

n- filter order





Frequency domain Filtering
2D Examples (Low Pass Filter)

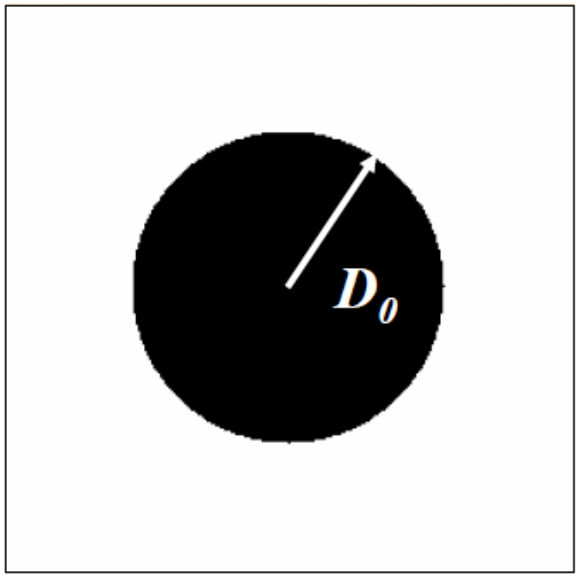


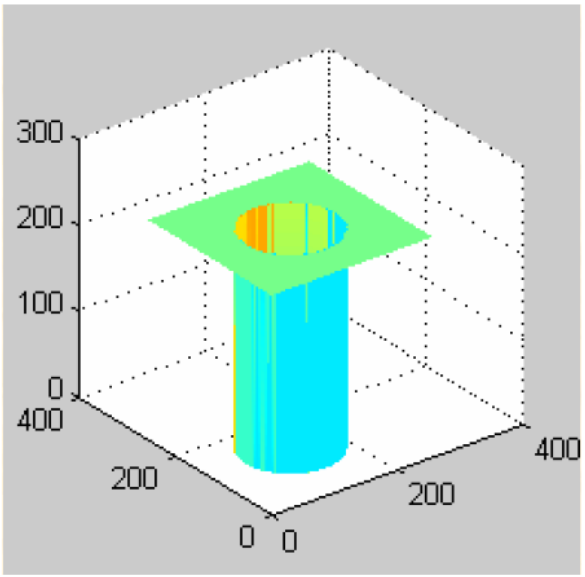
Frequency domain Filtering
2D Examples (High Pass Filter)

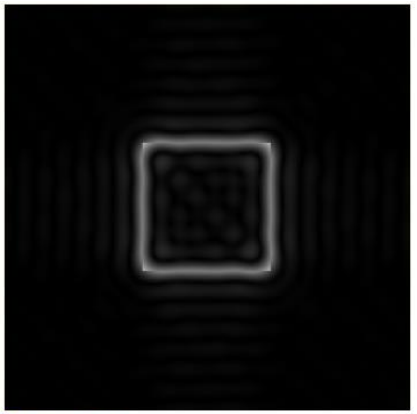
Ideal high-pass filter

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

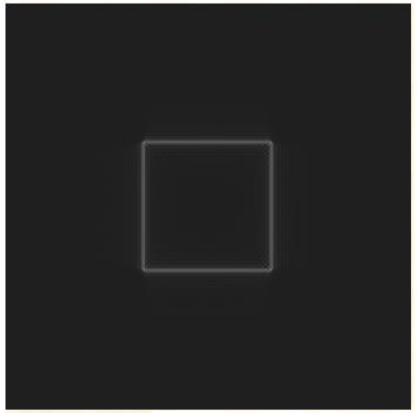
$$D(u,v)=\sqrt{u^2+v^2}$$







D_0=10



D_0=70

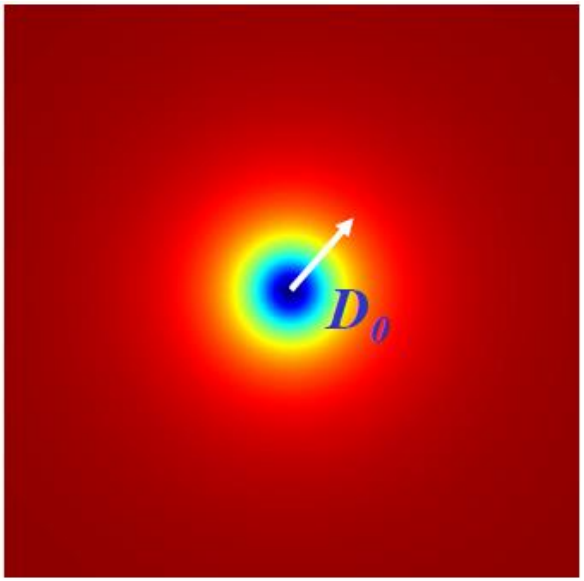
Frequency domain Filtering
2D Examples (High Pass Filter)

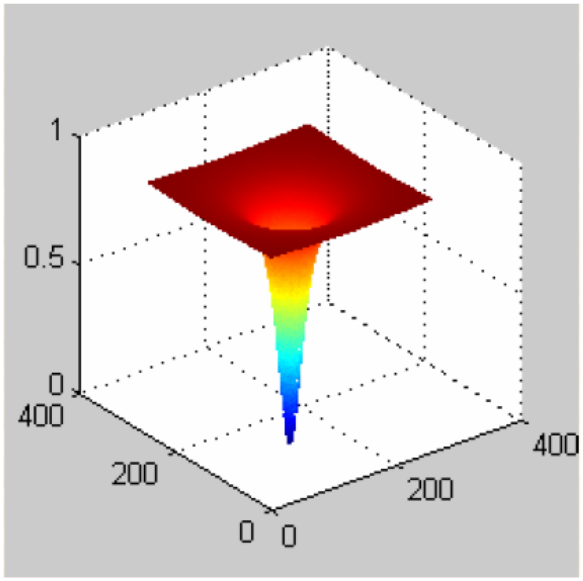
Butterworth high-pass filter

$$H(u,v)=\frac{1}{1+(\sqrt{2}-1)[D_0/D(u,v)]^{2n}}$$

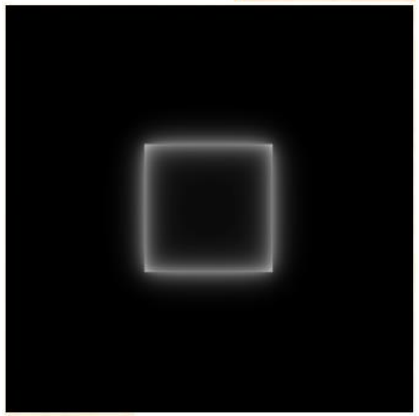
$$D(u,v)=\sqrt{u^2+v^2}, \quad n=1,2,\ldots$$

n- filter order

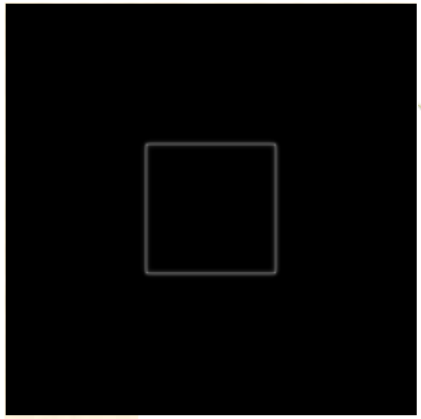




n = 1



D_0=10



D_0=70

Frequency domain Filtering
Effect of employing Different Components

Razi University

Selecting Fourier components in descending order of magnitude.

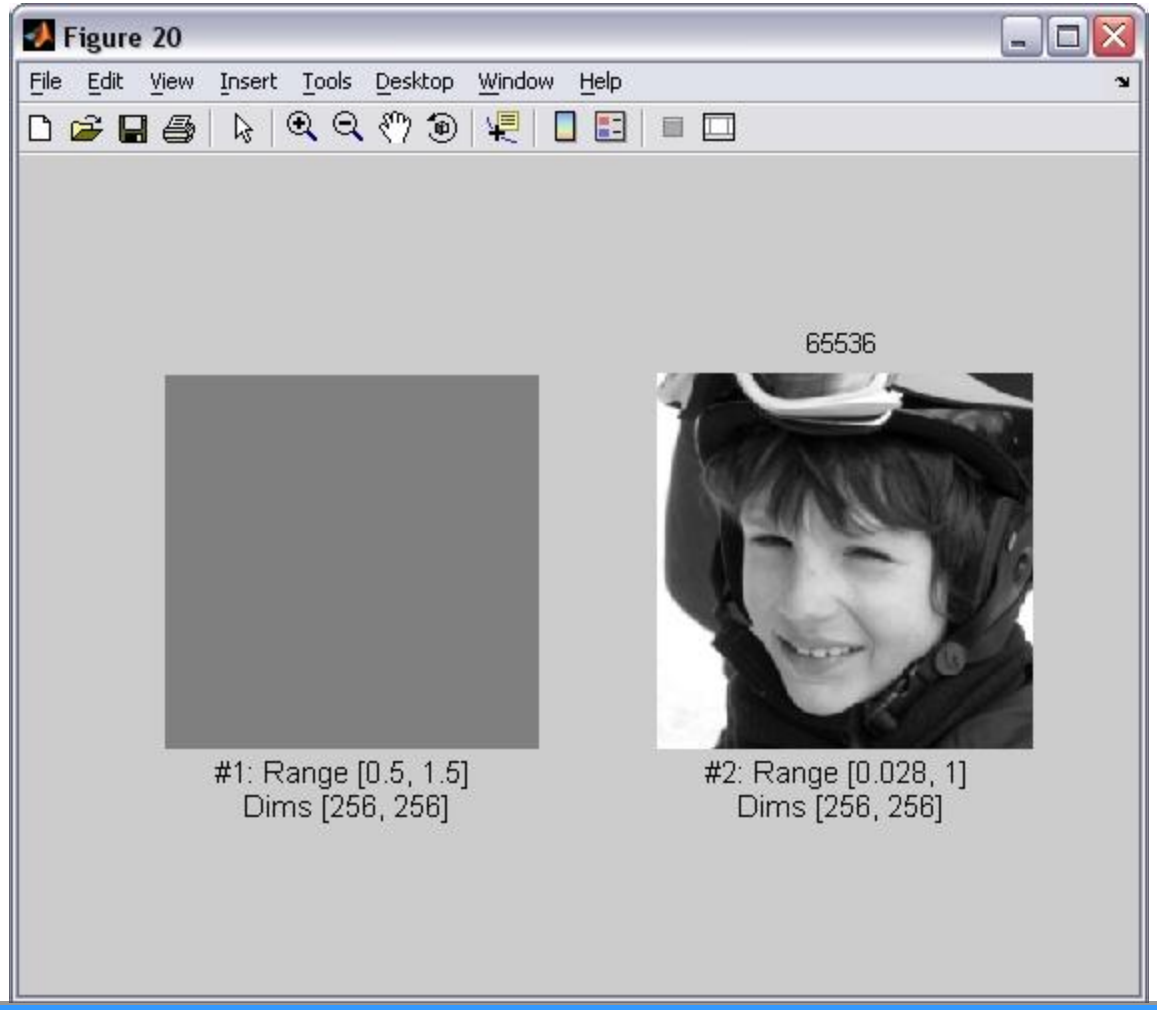
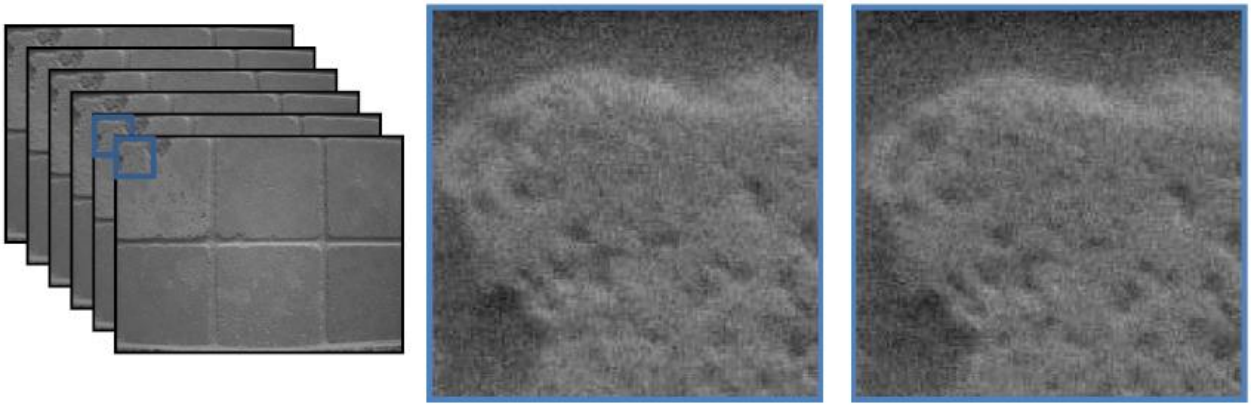


Image Filtering
Noise Reduction

- We can measure **noise** in multiple images of the same static scene.
- How could we reduce the noise, i.e., give an estimate of the true intensities?



- **First attempt at a solution**
  - Let's replace each pixel with an average of all the values in its neighborhood
  - Assumptions:
    - Expect pixels to be like their neighbors
    - Expect noise processes to be independent from pixel to pixel

Image Filtering
Common types of noise

- **Salt and pepper noise**:
  - random occurrences of black and white pixels
- **Impulse noise:**
  - Random occurrences of white pixels
- **Gaussian noise**:
  - Variations in intensity drawn from a Gaussian normal distribution

Original

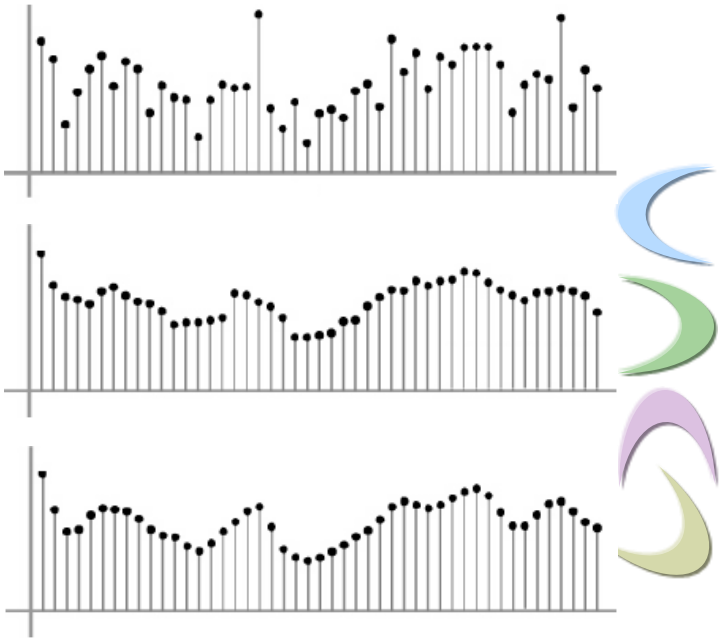
Salt and pepper noise

Impulse noise

Gaussian noise

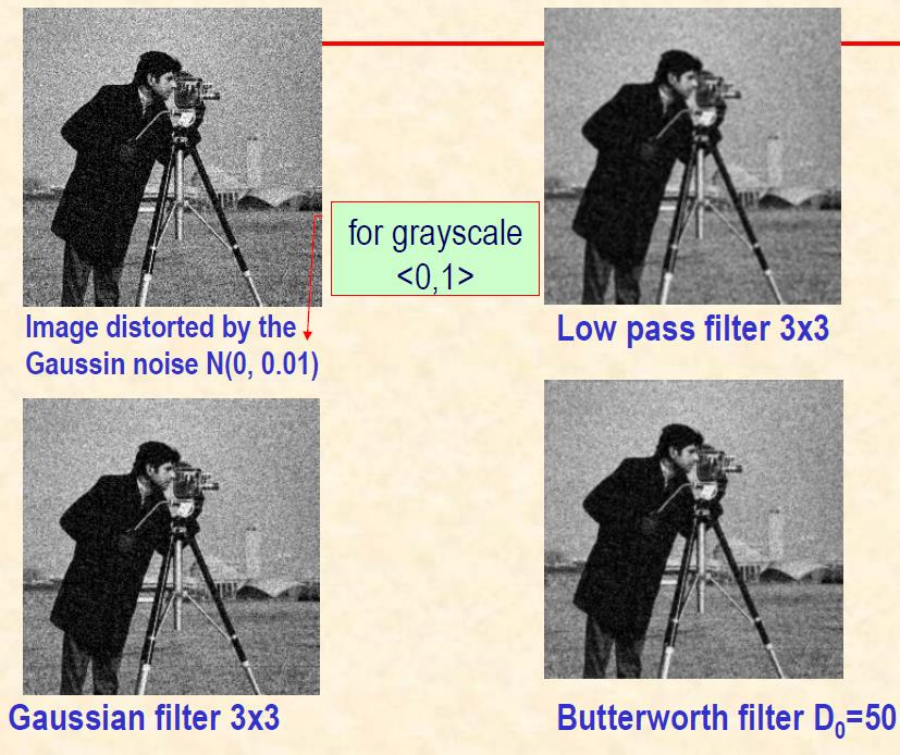
Image Filtering
Noise Reduction

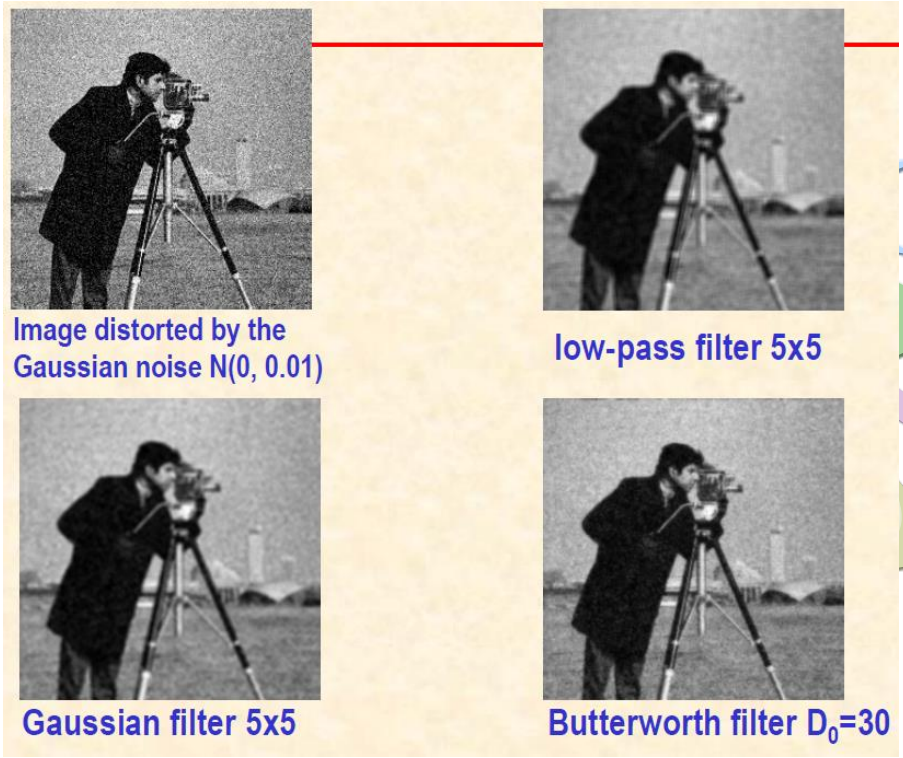
- **Noise reduction by Moving Average:** Let's replace each pixel with an average of all the values in its neighborhood
  - Can add weights to our moving average
    - Uniform weights [1, 1, 1, 1, 1] / 5
    - Non-uniform weights [1, 4, 6, 4, 1] / 16



- **Noise reduction by Low pass filter:** noise can be seen as high changes in the data or high frequency components on data.

Spatial & Frequency domain Filtering
2D Examples (Low Pass Filter)





Razi University

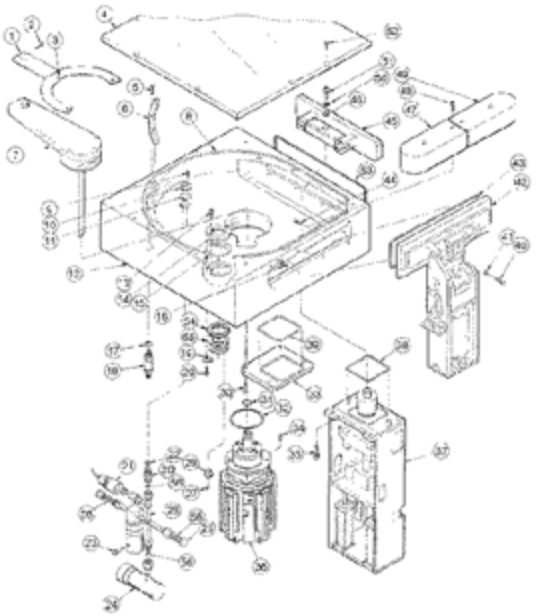
Local Features Analysis

Local Feature Analysis
Lines and Arcs Extraction

In some image sets, lines, curves, and circular arcs are more useful than regions or helpful in addition to regions.

Lines and arcs are often used in

- **object recognition**
- **stereo matching**
- **document analysis**



To detect lines, curves and other boundaries, at first, we need to detect edge points.

Local Feature Analysis
Edge Detection

Basic idea: look for a neighborhood with strong signs of change.

Problems:

- neighborhood size
- how to detect change

81	82	26	24
82	33	25	25
81	82	26	24

Solution:

- attempt to approximate the gradient at a pixel via masks
- threshold the gradient to select the edge pixels

Local Feature Analysis
Gradient Mask

Let assume:

- S_x **is a gradient mask in x direction**
- S_y **is a gradient mask in y direction**

On a pixel of the image I
- let g_x be the response to S_x
- let g_y be the response to S_y

⇒ Then the gradient is $\nabla I = [g_x \quad gy]^T$

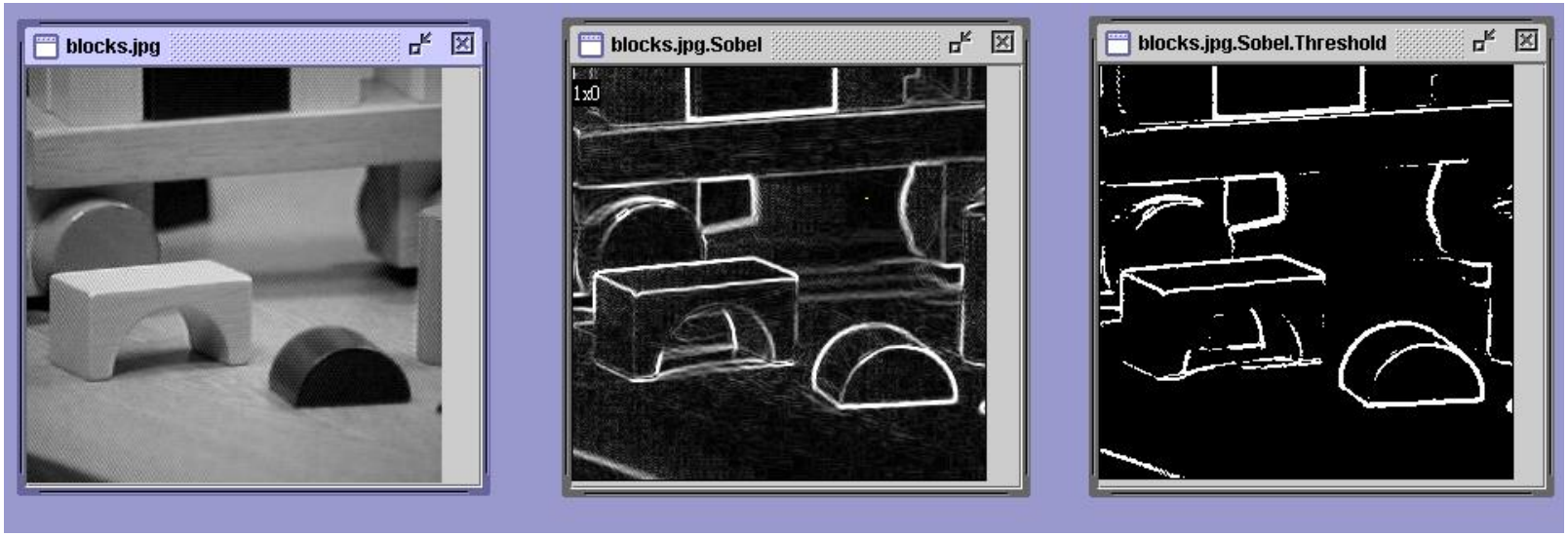
And $g = (g_x^2 + g_y^2)^{1/2}$ is the gradient magnitude.

$\theta = \text{atan}(g_y / g_x)$ is the gradient direction.

Local Feature Analysis
Common Masks for Computing Gradient

	Sx	Sy
Sobel:	$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$
Prewitt:	$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$
Roberts	$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

Local Feature Analysis
Example of Sobel Operator



original image

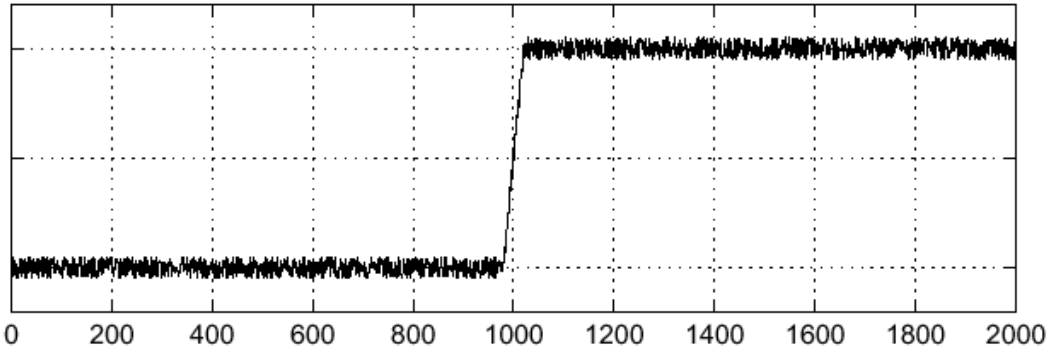
gradient magnitude using Sobel Operator

thresholded gradient magnitude

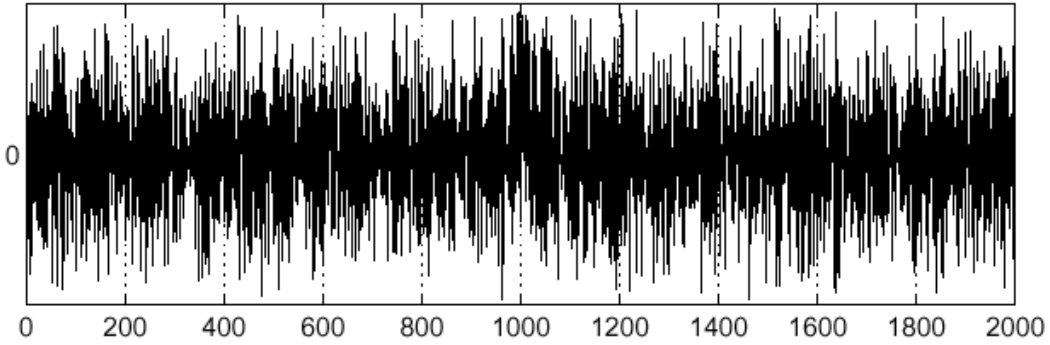
Local Feature Analysis
Effect of Noise

- **Consider a single row or column of the image**
  - **Plotting intensity as a function of position gives a signal**

$f(x)$



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



Where is the edge?

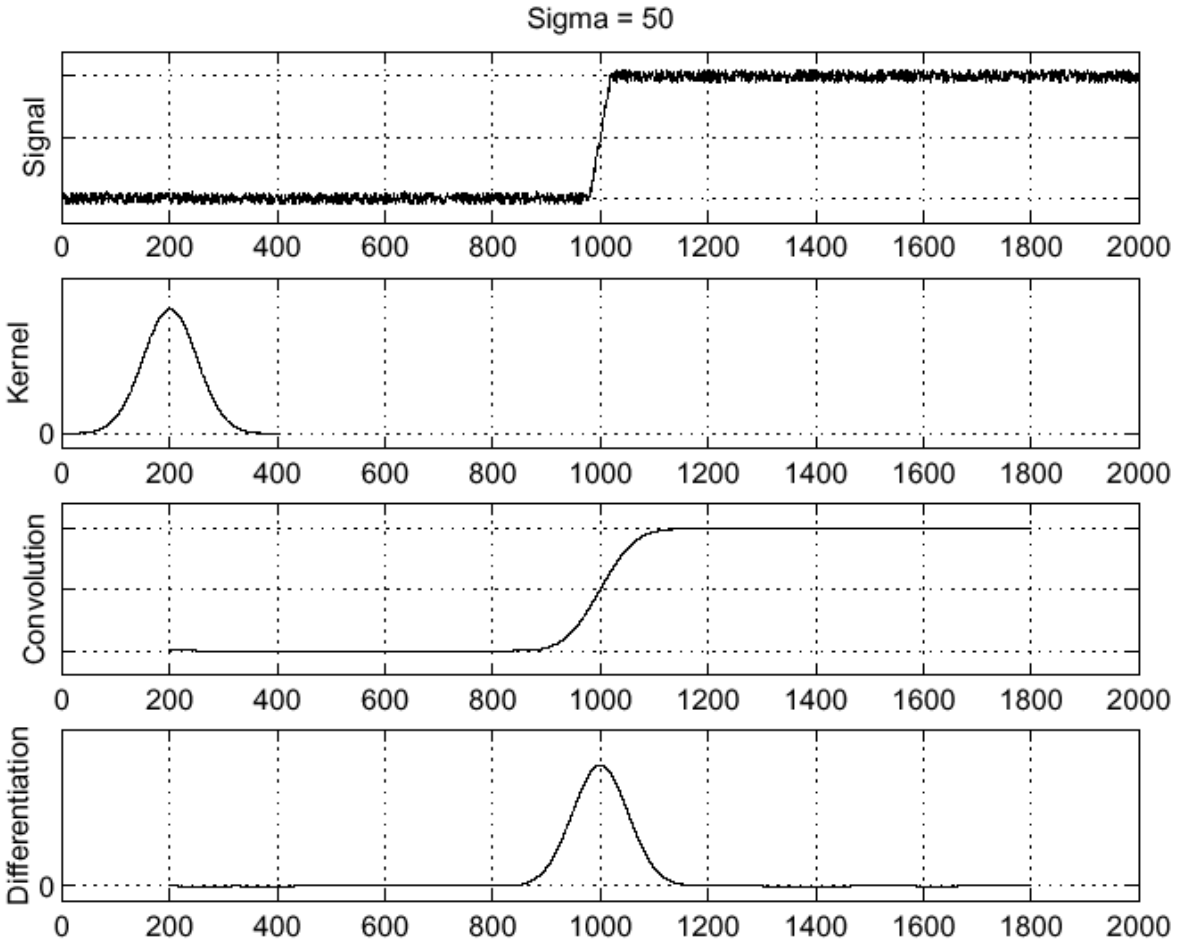
Local Feature Analysis
Solution: Smooth First

f

h

$h \star f$

$\frac{\partial}{\partial x}(h \star f)$



Where is the edge? Look for peaks in $\frac{\partial}{\partial x}(h \star f)$

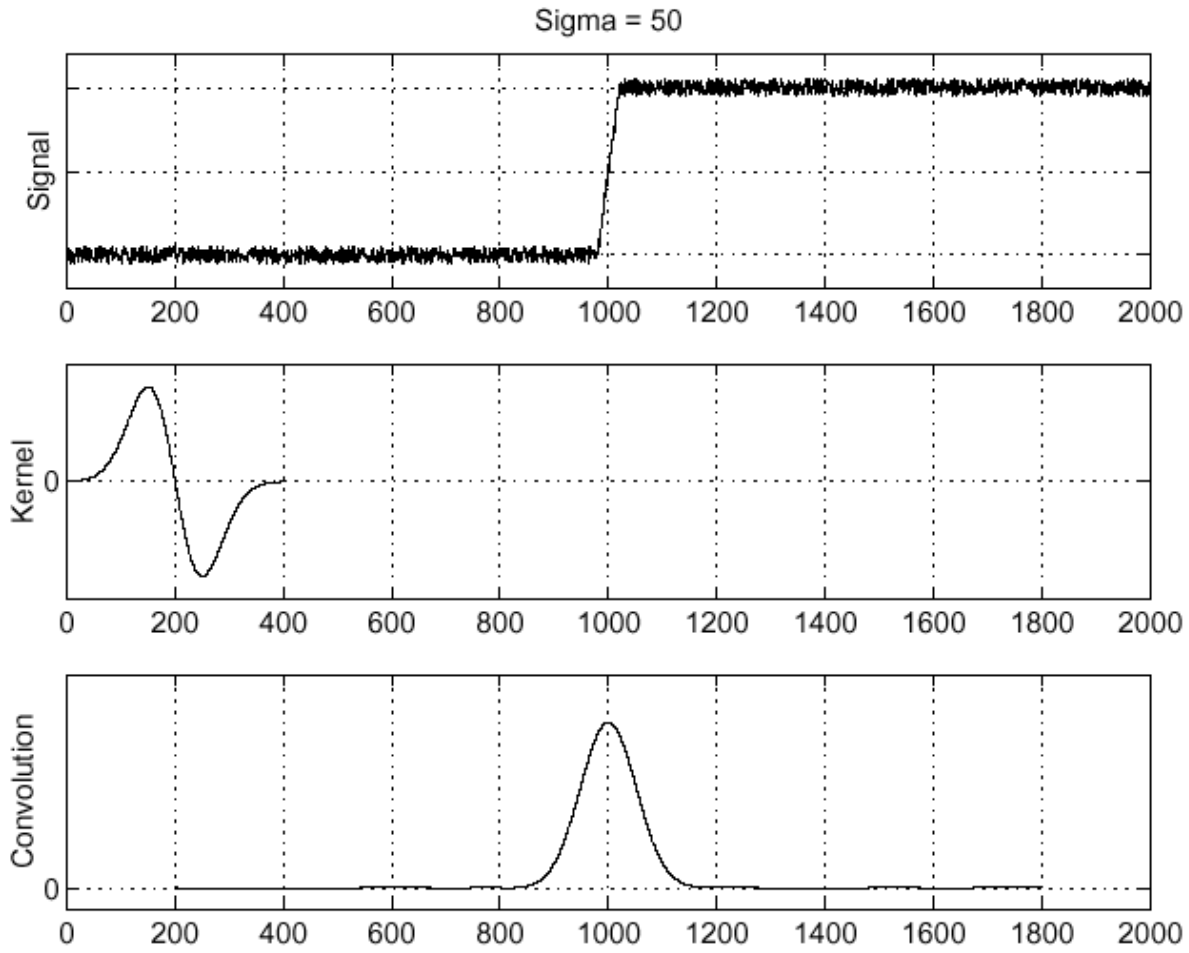
Local Feature Analysis
Derivative Theorem of Convolution

We can Use: $\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$

f

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

$(\frac{\partial}{\partial x}h) \star f$



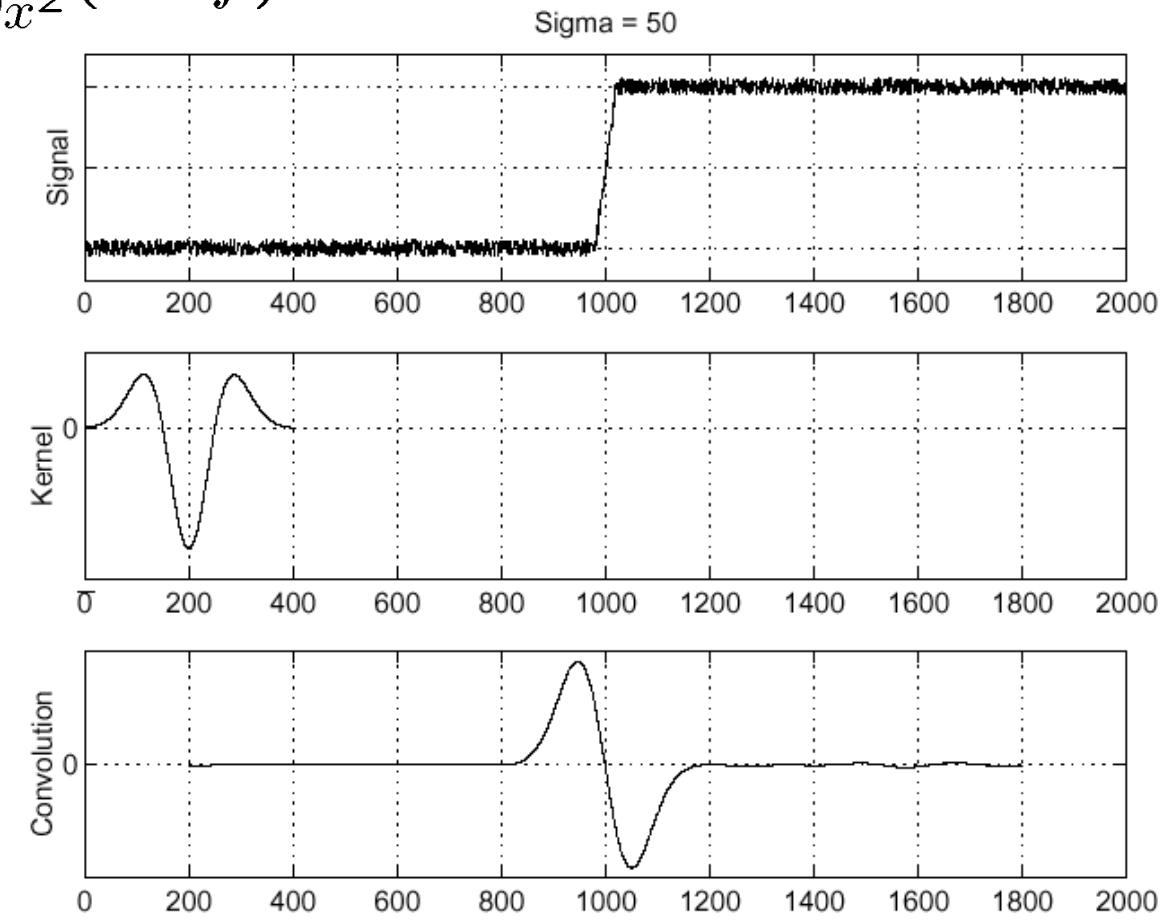
Differential Operators: Laplacian of Gaussian (LoG)

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

f

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

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

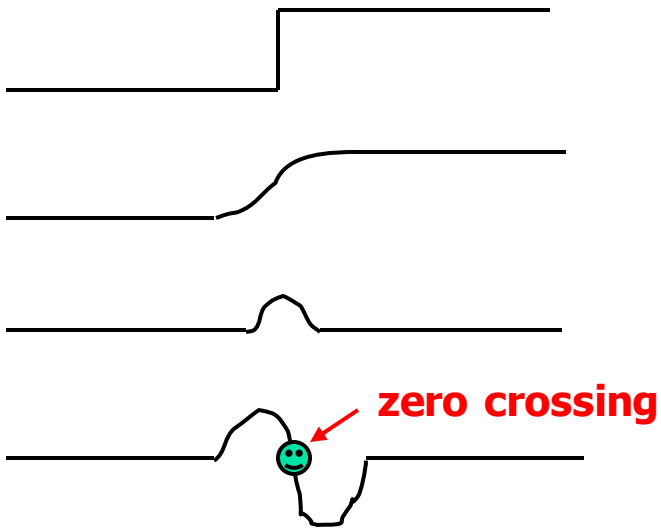


Where is the edge? **Zero-crossings of bottom graph**

Local Feature Analysis
Differential Operators: Zero Crossing

Motivation:

The zero crossings of the second derivative of the image function are more precise than the peaks of the first derivative.



step edge

smoothed

1st derivative

2nd derivative

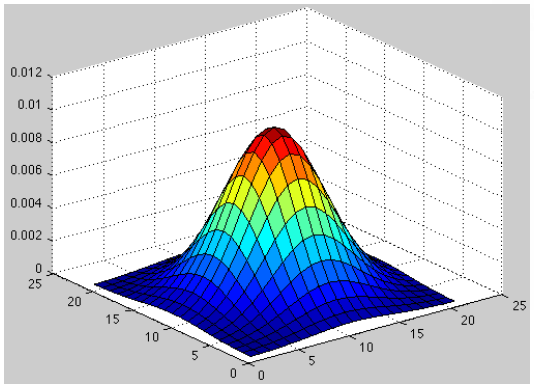
Local Feature Analysis
Second Derivative

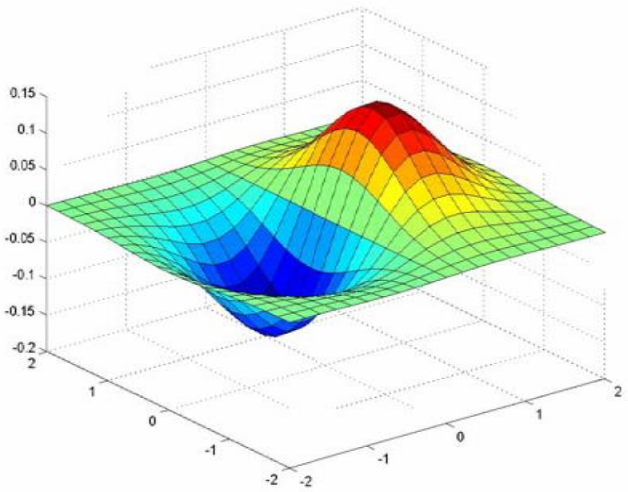
- **How do we estimate the Second Derivative?**
  - **Laplacian Filter:** $\nabla^2 f = \partial f^2/\partial^2 x + \partial f^2/\partial^2 y$

0	1	0
1	-4	1
0	1	0

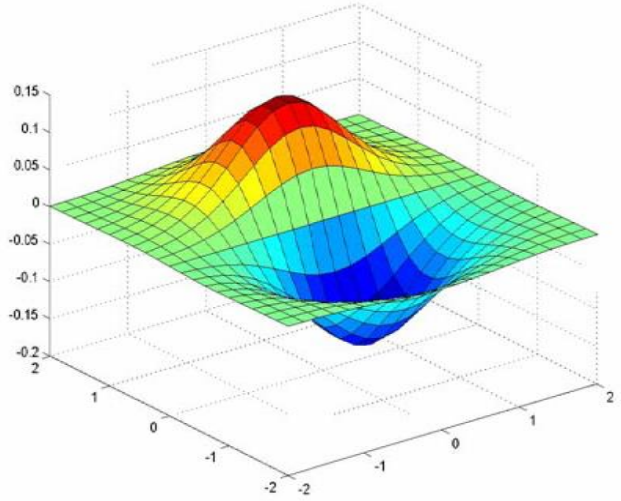
- Standard mask implementation
- Derivation: In 1D, the first derivative can be computed with mask [-1 0 1]
- The 1D second derivative is [1 -2 1]
- The Laplacian mask estimates the 2D second derivative.

Derivative of Gaussian Filters



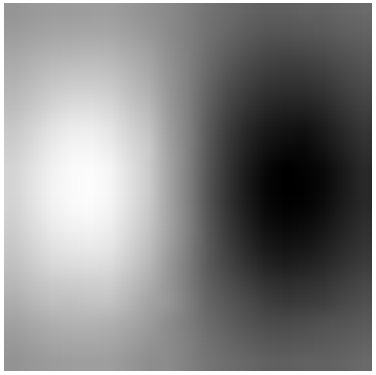


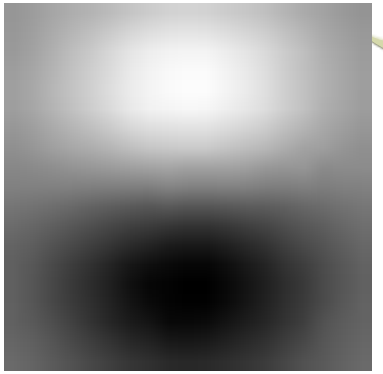
x-direction



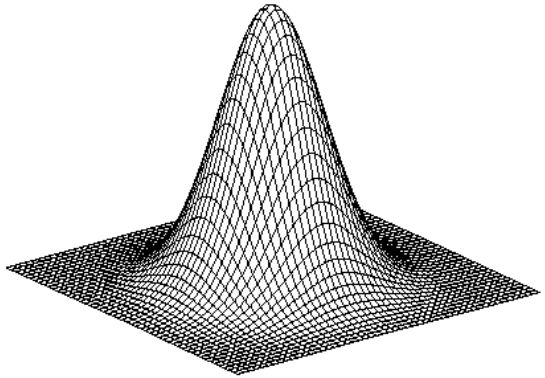
y-direction

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

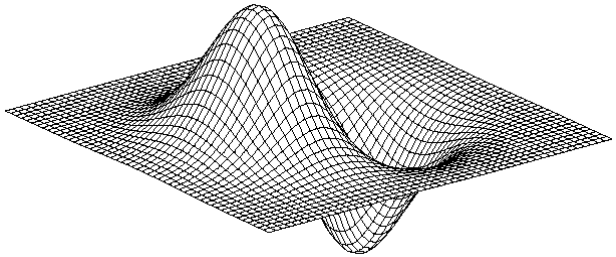




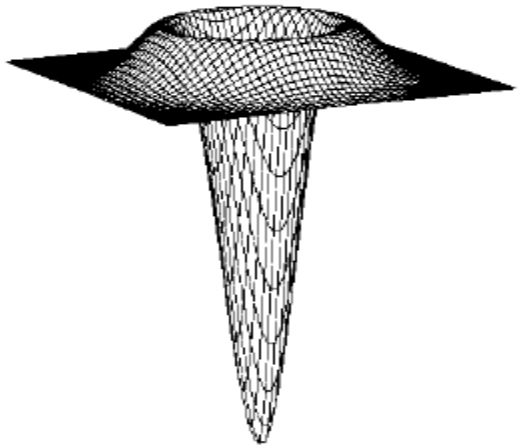
2D Edge Detection Filters



Gaussian



Derivative of Gaussian



Laplacian of Gaussian

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

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

$$\nabla^2 h_\sigma(u,v)$$

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

Local Feature Analysis
Properties of Derivative Masks

- **Coordinates of derivative masks have opposite signs in order to obtain a high response in regions of high contrast.**
- **The sum of coordinates of derivative masks is zero, so that a zero response is obtained on constant regions.**
- **First derivative masks produce high absolute values at points of high contrast.**
- **Second derivative masks produce zero-crossings at points of high contrast.**

Local Feature Analysis
Second Derivative: Marr/Hildreth Operator

- First smooth the image via a Gaussian convolution.
- Apply a Laplacian filter (estimate 2nd derivative).
- Find zero crossings of the Laplacian of the Gaussian.

This can be done at multiple resolutions.

Local Feature Analysis
Second Derivative: Haralick Operator

- Fit the gray-tone intensity surface to a piecewise cubic polynomial approximation.
- Use the approximation to find zero crossings of the second directional derivative in the direction that maximizes the first directional derivative.

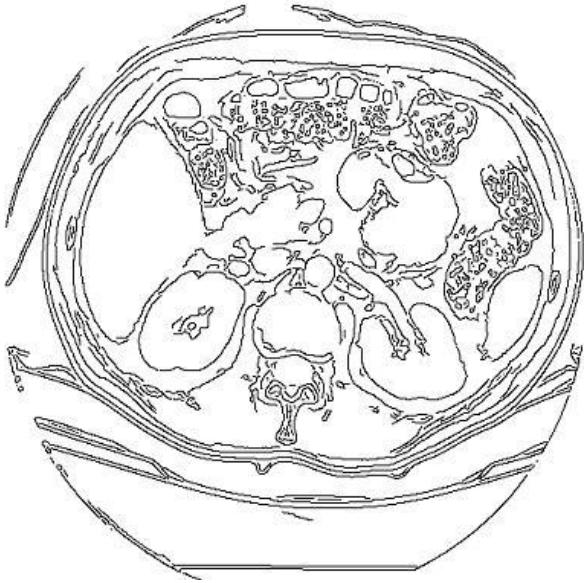
The derivatives here are calculated from direct mathematical expressions wrt the cubic polynomial.

Local Feature Analysis
Second Derivative: Canny Edge Detector

- Smooth the image with a Gaussian filter with spread σ.
- Compute gradient magnitude and direction at each pixel of the smoothed image.
- Zero out any pixel response $\leq$ the two neighboring pixels on either side of it, along the direction of the gradient.
- Track high-magnitude contours.
- Keep only pixels along these contours, so weak little segments go away.

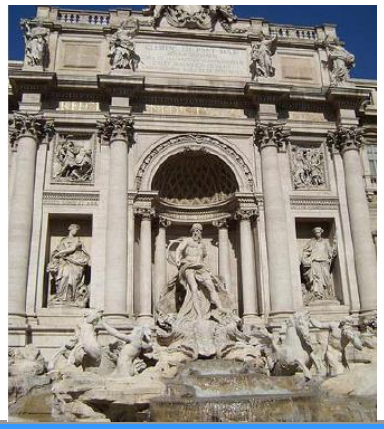
Local Feature Analysis
Second Derivative: Canny Characteristics

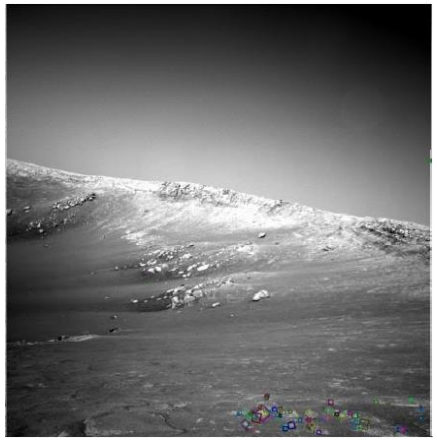
- **The Canny operator gives single-pixel-wide images with good continuation between adjacent pixels**
- **It is the most widely used edge operator today; no one has done better since it came out in the late 80s. Many implementations are available.**
- **It is very sensitive to its parameters, which need to be adjusted for different application domains.**



Local Features Analysis
Image Alignment

- **We need to match (align) images**
  - Global methods sensitive to occlusion, lighting, parallax effects. So look for local features that match well.
  - How would you do it by eye?





Local Features Analysis
Image Alignment

- **Local features and alignment**
  - Detect feature points in both images
  - Find corresponding pairs
  - Use these pairs to align images

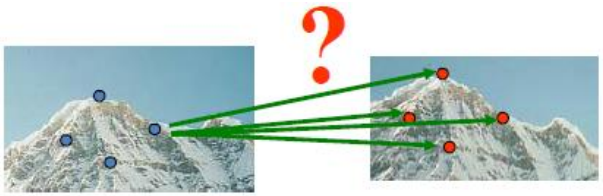
Local Features Analysis
Image Alignment

- **Problem 1:**
  - Detect the **same point independently** in both images

no chance to match!

We need a repeatable detector

- **Problem 2:**
  - For each point **correctly recognize** the corresponding one

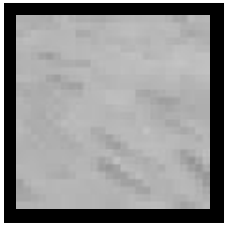


We need a reliable and distinctive descriptor

Local Features Analysis
Feature Detection and Matching

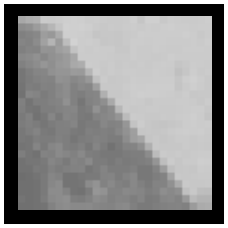
- **To this end, we should design a group of descript to extract subset of local feature types that invariant to common geometric and photometric transformations.**
  - **Illumination invariant**
  - **Rotation invariant**
  - **Scale invariant**
  - **Shift invariant**
  - **…**
- **Basic steps:**
  - **1) Detection of distinctive interest points**
  - **2) Extraction of invariant local descriptors**
  - **3) Determining correspondences**
  - **4) Selection of similar images**

Basic Steps
Interest Points



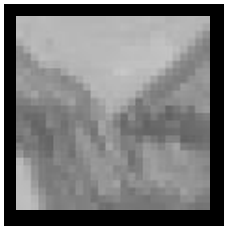
0D structure: single points

➡ not useful for matching



1D structure: lines

➡ edge, can be localised in 1D, subject to the aperture problem



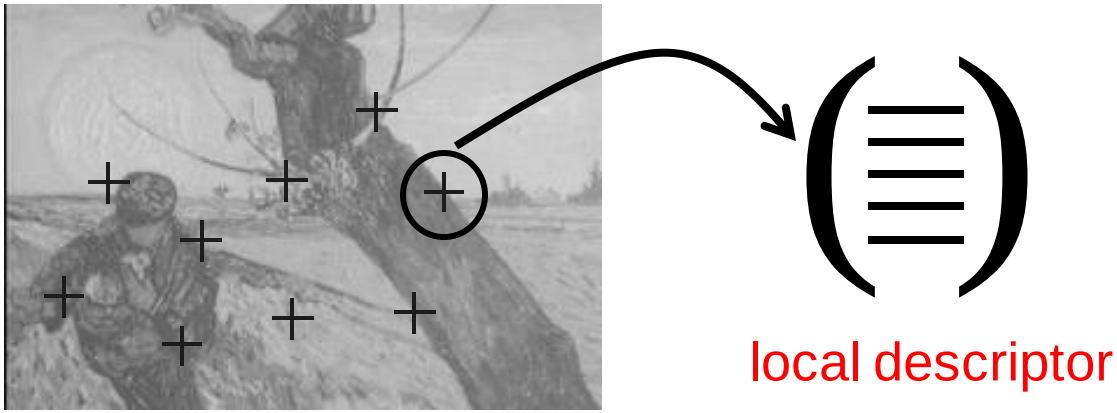
2D structure: corners

➡ corner, or **interest point**, can be localised in 2D, good for matching

Interest Points have **2D** structure.

Basic Steps
Local invariant photometric descriptors

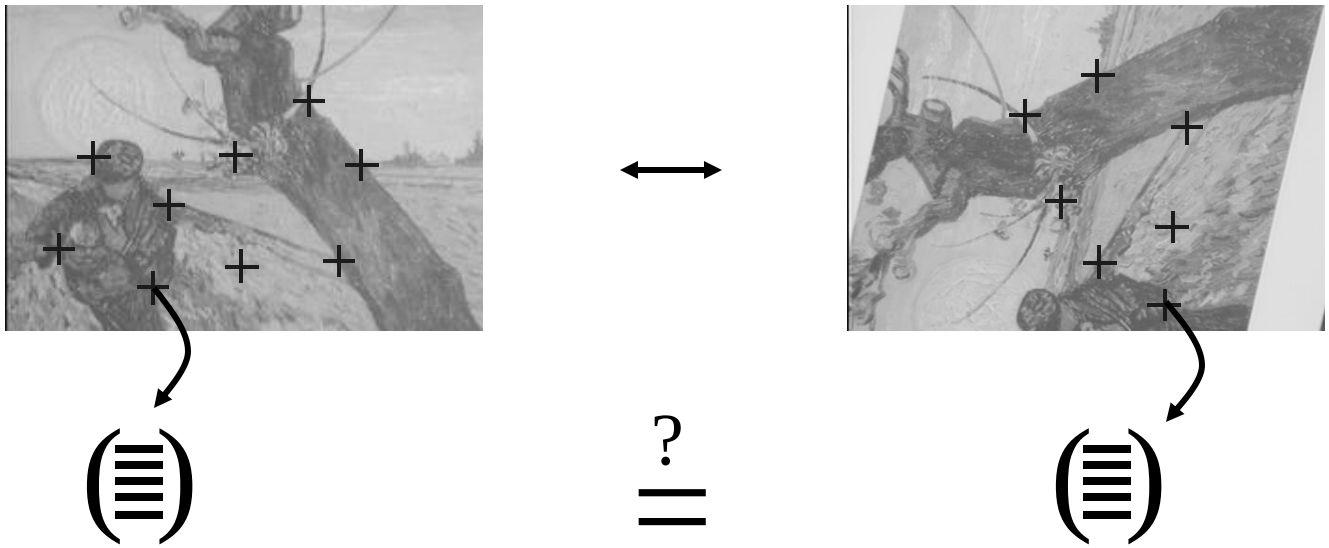
- **State-of-the-Art Local Descriptors:**
  - **Harris Corner Detector**
  - **SIFT interest point detector**
  - **HOG descriptor**
  - **LBP descriptor**
  - **…**



Local **: robust to occlusion/clutter + no segmentation**
Photometric **: (use pixel values) distinctive descriptions**
Invariant **: to image transformations + illumination changes**

Basic Steps
Determining correspondences

Razi University



- Vector comparison using a distance measure
- **What are some suitable distance measures?**
  - We can use the sum-square difference of the values of the pixels in a square neighborhood about the points being compared. **This is the simplest measure.**

$$SSD = \sum\sum(W1_{i,j} - W2_{i,j})^2$$

Basic Steps
Matching

1. **Matching based on correlation alone**
2. **Matching based on geometric primitives**
   - **e.g. line segments**

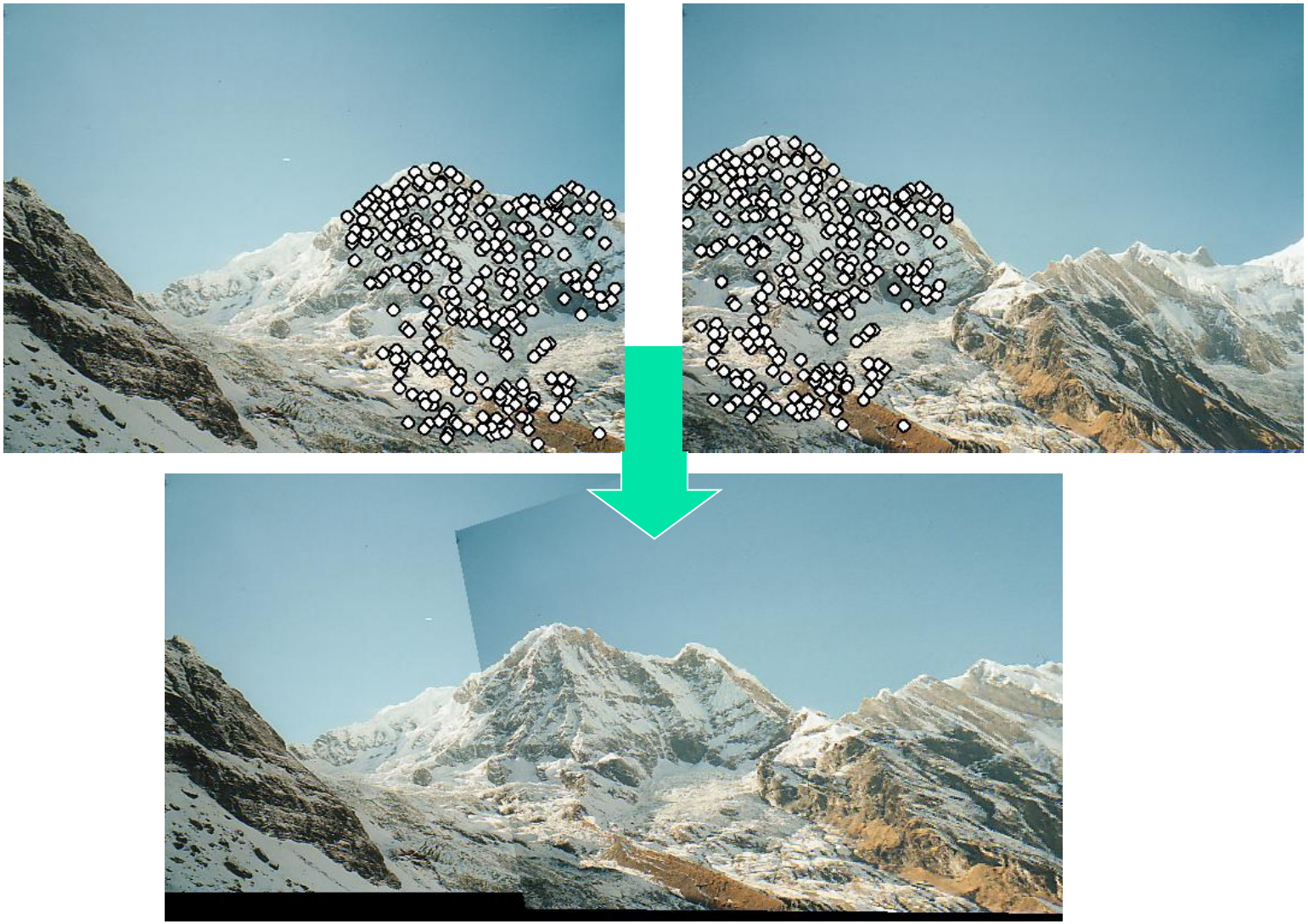
⇒ **Not very discriminating (why?)**

⇒ **Solution : matching with interest points & correlation**

- **Extraction of interest points with the Harris detector**
- **Comparison of points with cross-correlation**
- **Verification with the fundamental matrix**

The **fundamental matrix** maps points from the first image to corresponding points in the second matrix using a homography that is determined through the solution of a set of equations that usually minimizes a least square error.

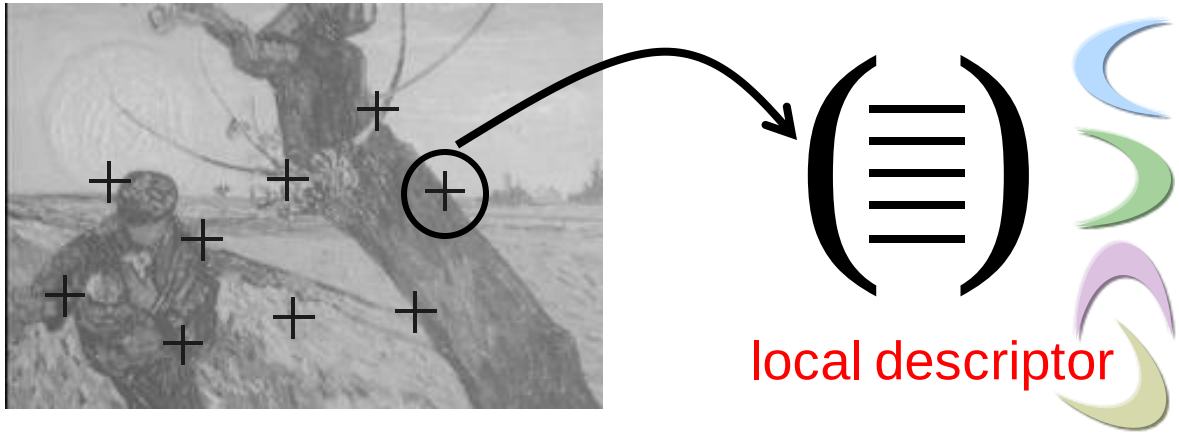
Local Features Analysis
Matching Results



Local Features Extraction (Review)

Local Feature descriptors

- **State-of-the-Art Local Descriptors:**
  - **Harris Corner Detector**
  - **SIFT interest point detector**
  - **HOG descriptor**
  - **LBP descriptor**
  - **LTP descriptor**
  - **LPQ descriptor**
  - **…**



Local **: robust to occlusion/clutter + no segmentation**
Photometric **: (use pixel values) distinctive descriptions**
Invariant **: to image transformations + illumination changes**

Harris Detector Background:
Moravec Corner Detector

- take a window w in the image
- shift it in four directions (1,0), (0,1), (1,1), (-1,1)
- compute a difference for each
- compute the min difference at each pixel
- local maxima in the min image are the corners

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

Harris corner detector is based on the idea of auto-correlation

Important difference in all directions => interest point

Harris Detector Background: Shortcomings of Moravec Operator

- **Only tries 4 shifts. We'd like to consider "all" shifts.**
- **Uses a discrete rectangular window. We'd like to use a smooth circular (or later elliptical) window.**
- **Uses a simple min function. We'd like to characterize variation with respect to direction.**

Result: Harris Operator

Harris Detector

Auto-correlation fn (SSD) for a point (x, y) and a shift $(\Delta x, \Delta y)$

$$f(x,y) = \sum_{(x_k, y_k) \in W} (I(x_k, y_k) - I(x_k + \Delta x, y_k + \Delta y))^2$$

SSD means summed square difference

Discrete shifts can be avoided with the auto-correlation matrix

what is this?

with $$I(x_k + \Delta x, y_k + \Delta y) = I(x_k, y_k) + \begin{pmatrix} I_x(x_k, y_k) & I_y(x_k, y_k) \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}$$

$$f(x,y) = \sum_{(x_k, y_k) \in W} \left(\begin{pmatrix} I_x(x_k, y_k) & I_y(x_k, y_k) \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} \right)^2$$

Harris Detector

Rewrite as inner (dot) product

$$f(x,y) = \sum_{(x_k,y_k)\in W} \left(\begin{bmatrix} I_x(x_k,y_k) & I_y(x_k,y_k) \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \right)^2$$

$$= \sum_{(x_k,y_k)\in W} \begin{bmatrix} \Delta x & \Delta y \end{bmatrix} \begin{bmatrix} I_x(x_k,y_k) \\ I_y(x_k,y_k) \end{bmatrix} \begin{bmatrix} I_x(x_k,y_k) & I_y(x_k,y_k) \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$

$$= \sum_{W} \begin{bmatrix} \Delta x & \Delta y \end{bmatrix} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$

$$= \begin{bmatrix} \Delta x & \Delta y \end{bmatrix} \sum_{W} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}$$

$$= \begin{pmatrix} \Delta x & \Delta y \end{pmatrix} \begin{bmatrix} \sum_{(x_k,y_k)\in W} (I_x(x_k,y_k))^2 & \sum_{(x_k,y_k)\in W} I_x(x_k,y_k) I_y(x_k,y_k) \\ \sum_{(x_k,y_k)\in W} I_x(x_k,y_k) I_y(x_k,y_k) & \sum_{(x_k,y_k)\in W} (I_y(x_k,y_k))^2 \end{bmatrix} \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}$$

The center portion is a 2x2 Auto-correlation matrix M

Harris Detection

- **Auto-correlation matrix**
  - captures the structure of the local neighborhood
  - measure based on eigenvalues of M
    - 2 strong eigenvalues => interest point
    - 1 strong eigenvalue => contour
    - 0 eigenvalue => uniform region

- **Interest point detection**
  - threshold on the eigenvalues
  - local maximum for localization

Harris Detection:
Some Details from the Harris Paper

- **Corner strength $R = Det(M) - k\,Tr(M)^2$**
- **Let α and β be the two eigenvalues. We don't have to calculate them! Instead, use trace and determinant:**
  - **$Tr(M) = \alpha + \beta$**
  - **$Det(M) = \alpha\beta$**

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \qquad \det(\mathbf{A}) = a_{11}a_{22} - a_{12}a_{21}$$

$$\operatorname{tr}(\mathbf{A}) = a_{11} + a_{22}$$

- **R is positive for corners, negative for edges, and small for flat regions**

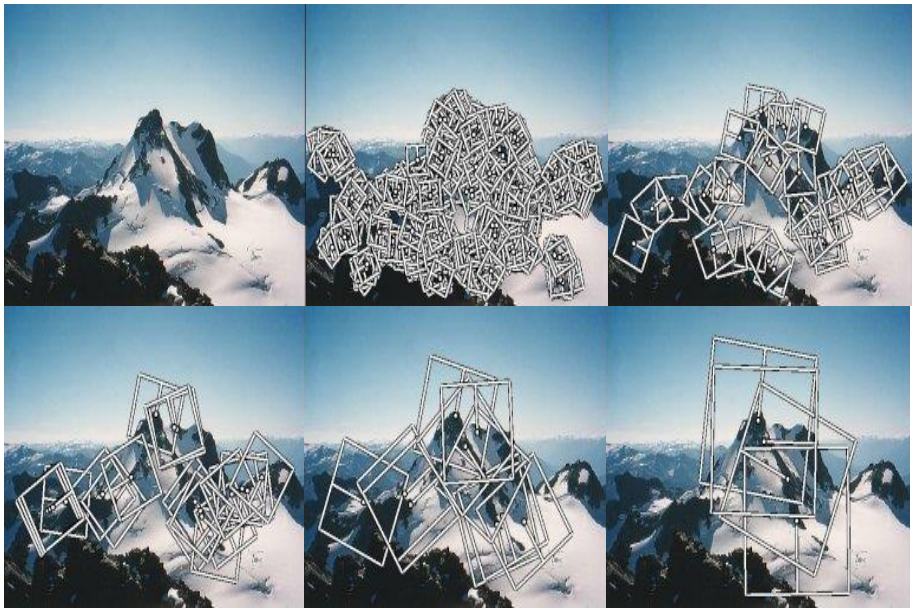
- **Select corner pixels that are 8-way local maxima**

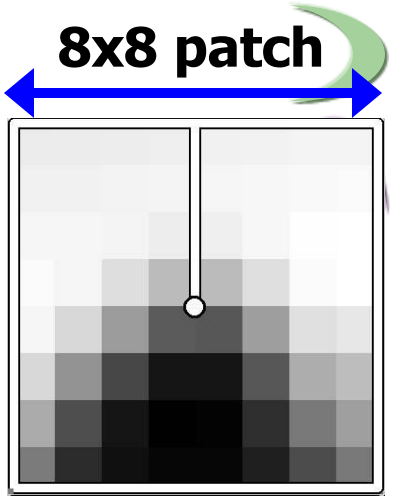
Harris Detection:
Summary of the approach

- **Basic feature matching = Harris Corners & Correlation**
- **Very good results in the presence of occlusion and clutter**
  - **local information**
  - **discriminant gray value information**
  - **invariance to image rotation and illumination**
  - **Not invariance to scale and affine changes**
- **Solution for more general view point changes**
  - **local invariant descriptors to scale and rotation**
  - **extraction of invariant points and regions**
- **Extract oriented patches at multiple scales using dominant orientation**

Multi-Scale Oriented Patches

- **Sample scaled, oriented patch**
  - 8x8 patch, sampled at 5 x scale
- **Bias/gain normalized** (subtract the mean of a patch and divide by the variance to normalize)
  - $I' = (I - \mu)/\sigma$





Matching Interest Points: Summary

- **Harris corners / correlation**
  - **Extract and match repeatable image features**
  - **Robust to clutter and occlusion**
  - **BUT not invariant to scale and rotation**

- **Multi-Scale Oriented Patches**
  - **Corners detected at multiple scales**
  - **Descriptors oriented using local gradient**
    - **Also, sample a blurred image patch**
  - **Invariant to scale and rotation**

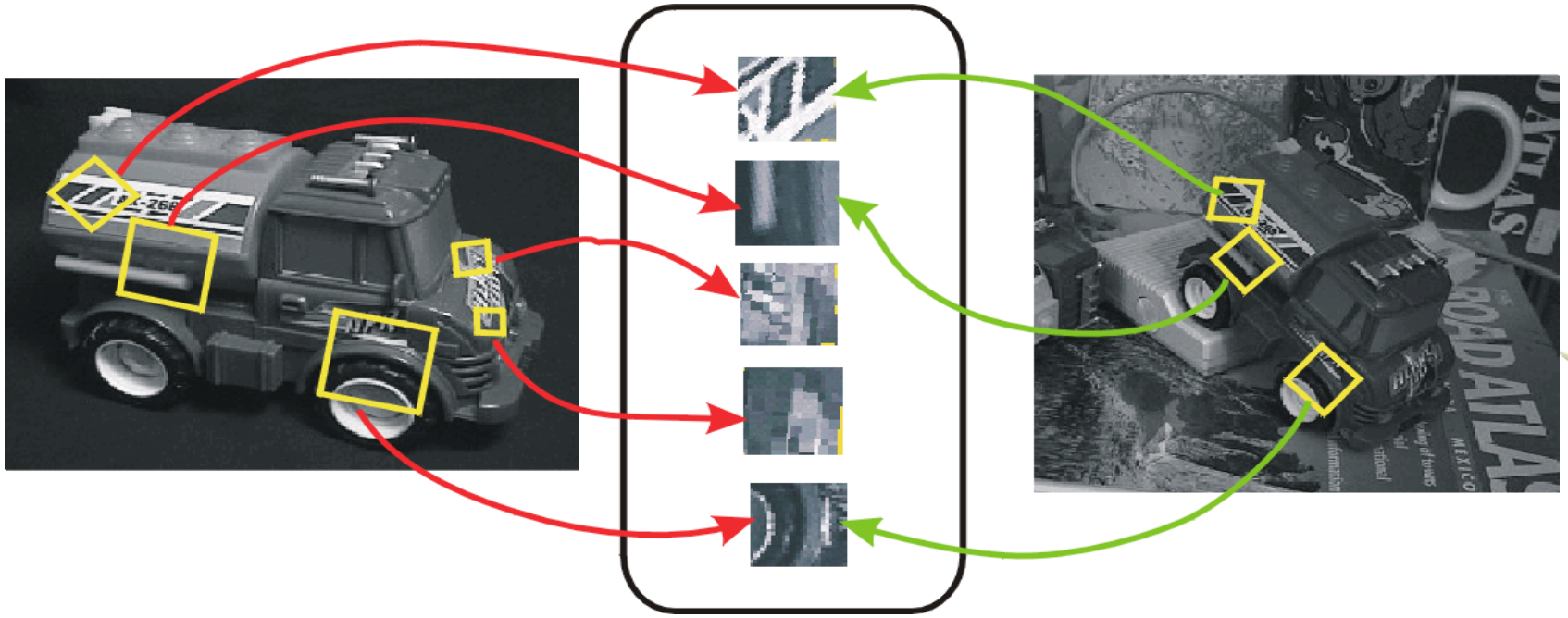
Leads to: **SIFT** – state of the art image features

SIFT Descriptor

- **Motivation**
  - **The Harris operator is not invariant to scale and correlation is not invariant to rotation.**
  - **For better image matching, Lowe's goal was to develop an interest operator that is invariant to scale and rotation.**
  - **Also, Lowe aimed to create a descriptor that was robust to the variations corresponding to typical viewing conditions. The descriptor is the most-used part of SIFT.**
    - **But Schmid and Mohr developed a rotation invariant descriptor for it in 1997.**

SIFT Descriptor
Idea of SIFT

- **Image content is transformed into local feature coordinates that are invariant to translation, rotation, scale, and other imaging parameters**



SIFT Features

SIFT Descriptor
Claimed Advantages of SIFT

- **Locality**: features are local, so robust to occlusion and clutter (no prior segmentation)
- **Distinctiveness**: individual features can be matched to a large database of objects
- **Quantity**: many features can be generated for even small objects
- **Efficiency**: close to real-time performance
- **Extensibility**: can easily be extended to wide range of differing feature types, with each adding robustness

SIFT Descriptor
Overall Procedure at a High Level

1. **Scale-space extrema detection**
   Search over multiple scales and image locations.

2. **Keypoint localization**
   Fit a model to determine location and scale.
   Select keypoints based on a measure of stability.

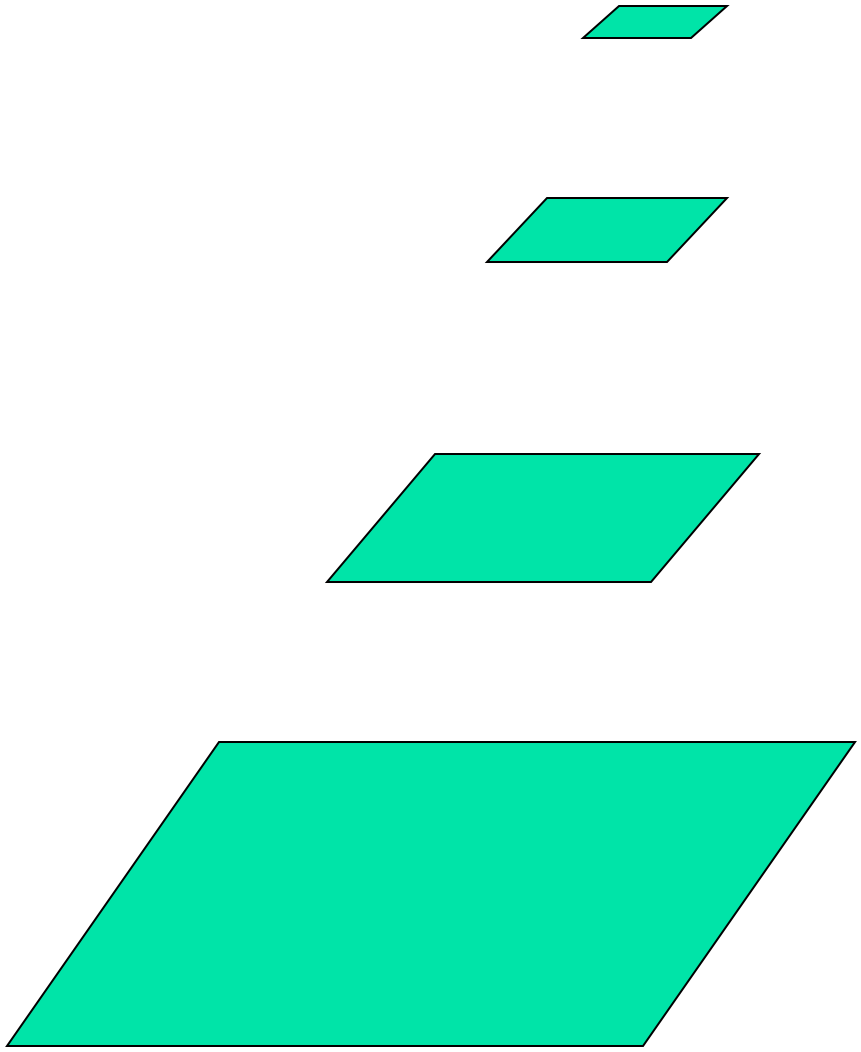
3. **Orientation assignment**
   Compute best orientation(s) for each keypoint region.

4. **Keypoint description**
   Use local image gradients at selected scale and rotation to describe each keypoint region.

SIFT Descriptor
1. Scale-space extrema detection

- **Goal:** Identify locations and scales that can be repeatably assigned under different views of the same scene or object.
- **Method:** search for stable features across multiple scales using a continuous function of scale.
- **Prior work** has shown that under a variety of assumptions, the best function is a **Gaussian function.**
- **The scale space of an image is a function** $L(x,y,\sigma)$ **that is produced from the convolution of a Gaussian kernel (at different scales) with the input image.**

Image Pyramids



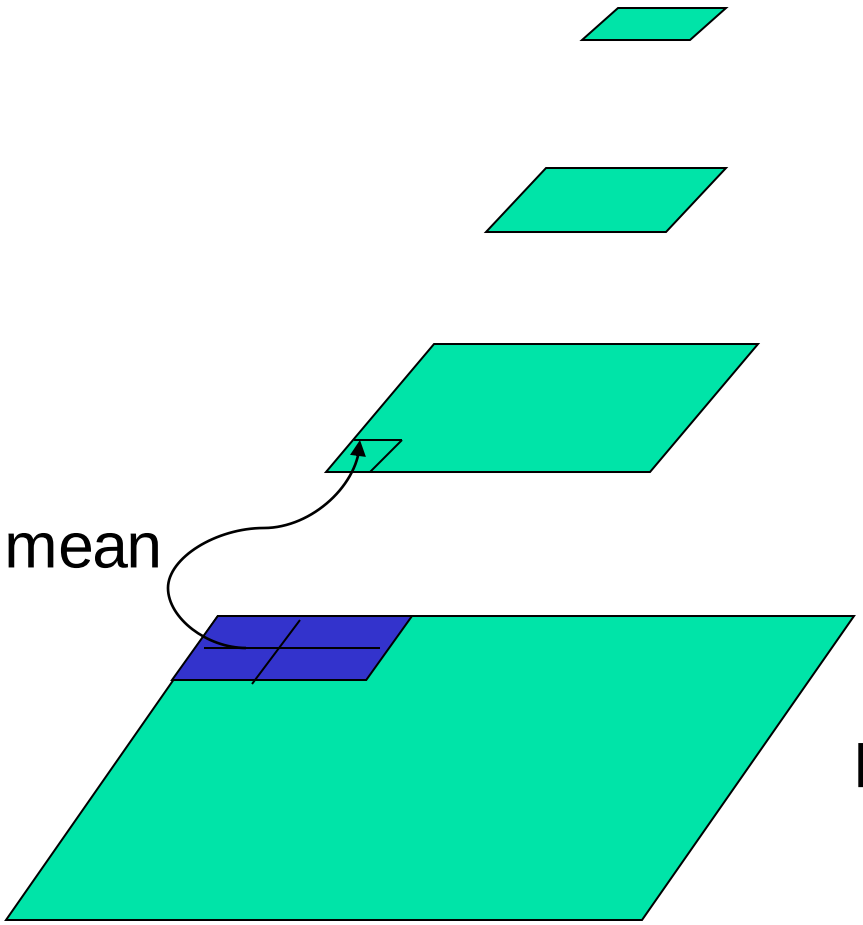
And so on.

3rd level is derived from the 2nd level according to the same funtion

2nd level is derived from the original image according to some function

Bottom level is the original image.

Mean Pyramid



And so on.

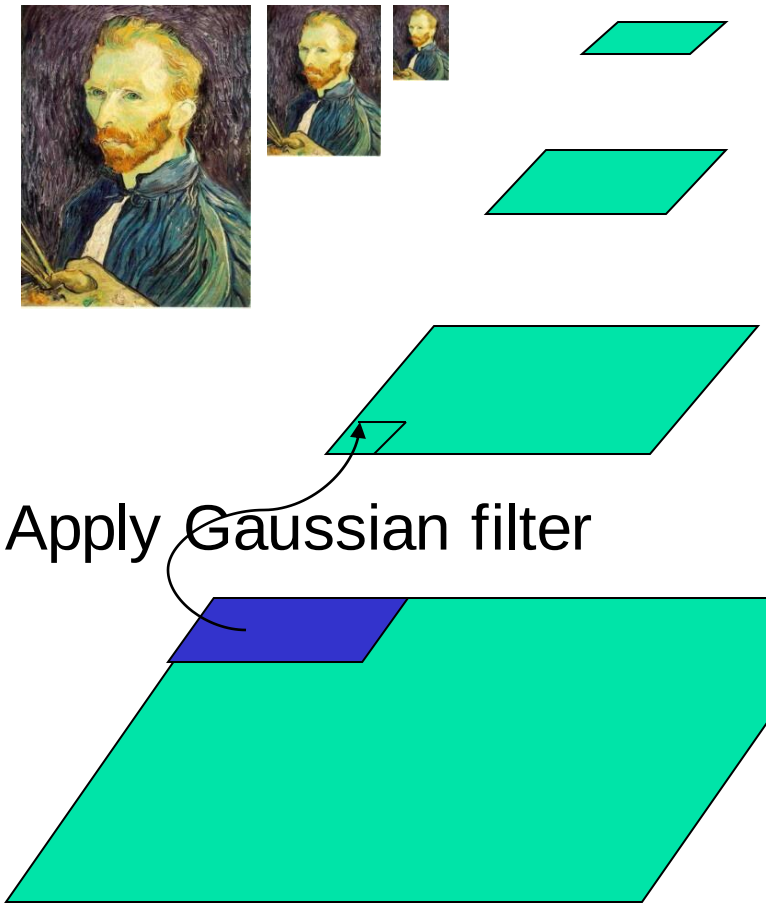
At 3rd level, each pixel is the mean of 4 pixels in the 2nd level.

At 2nd level, each pixel is the mean of 4 pixels in the original image.

Bottom level is the original image.

Gaussian Pyramid

At each level, image is smoothed and reduced in size.

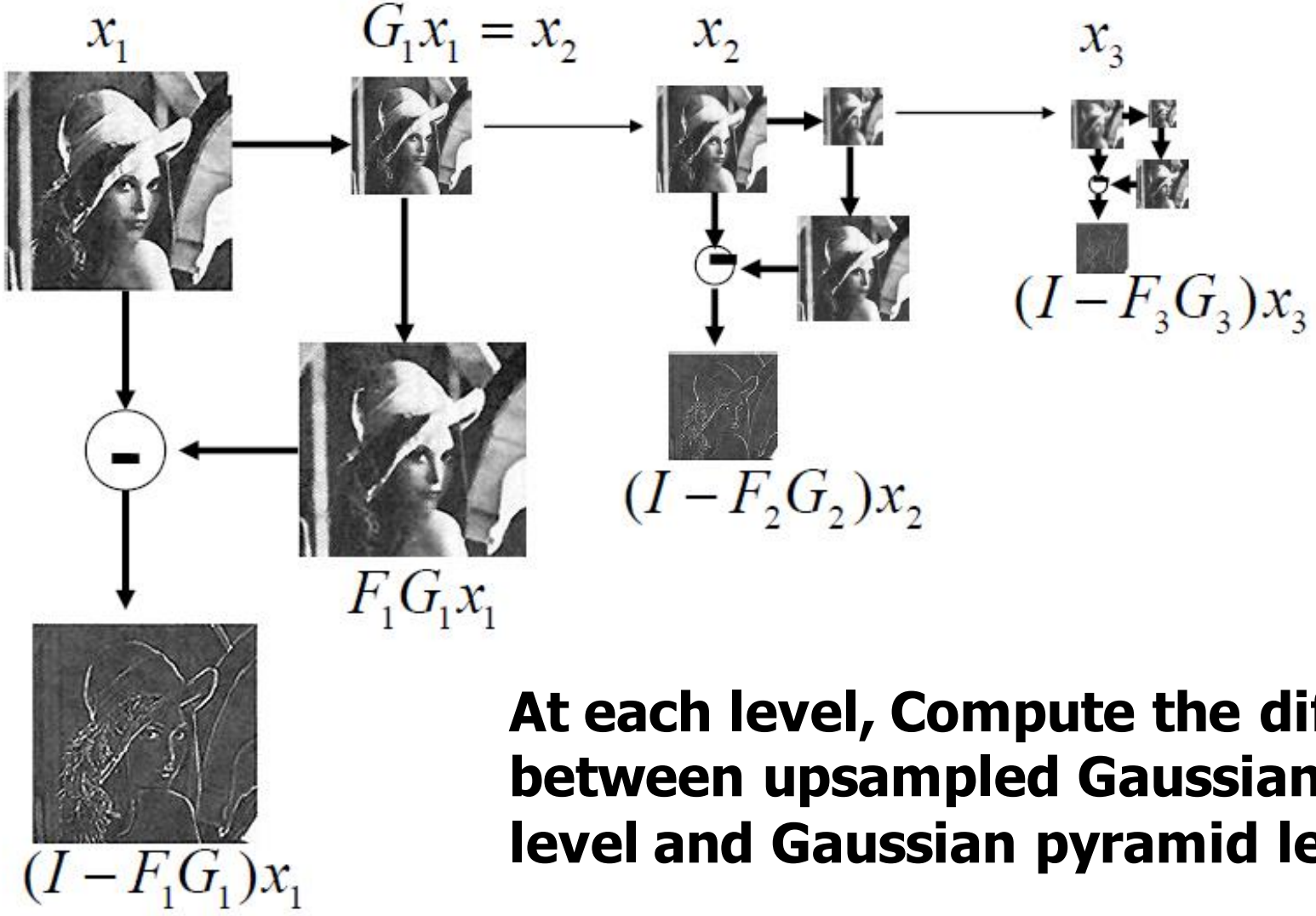


And so on.

At 2nd level, each pixel is the result of applying a Gaussian mask to the first level and then subsampling to reduce the size.

Bottom level is the original image.

Laplacian Pyramid



At each level, Compute the difference between upsampled Gaussian pyramid level and Gaussian pyramid level.

SIFT Descriptor
Lowe's Pyramid Scheme

- **Scale space is separated into octaves:**
  - **Octave 1 uses scale σ**
  - **Octave 2 uses scale 2σ**
  - **etc.**

- **In each octave, the initial image is repeatedly convolved with Gaussians to produce a set of scale space images.**

- **Adjacent Gaussians are subtracted to produce the DOG**

- **After each octave, the Gaussian image is down-sampled by a factor of 2 to produce an image ¼ the size to start the next level.**

SIFT Descriptor
Lowe's Pyramid Scheme

s+2 filters

$\sigma_{s+1}=2^{(s+1)/s}\sigma_0$

.

.

$\sigma_i=2^{i/s}\sigma_0$

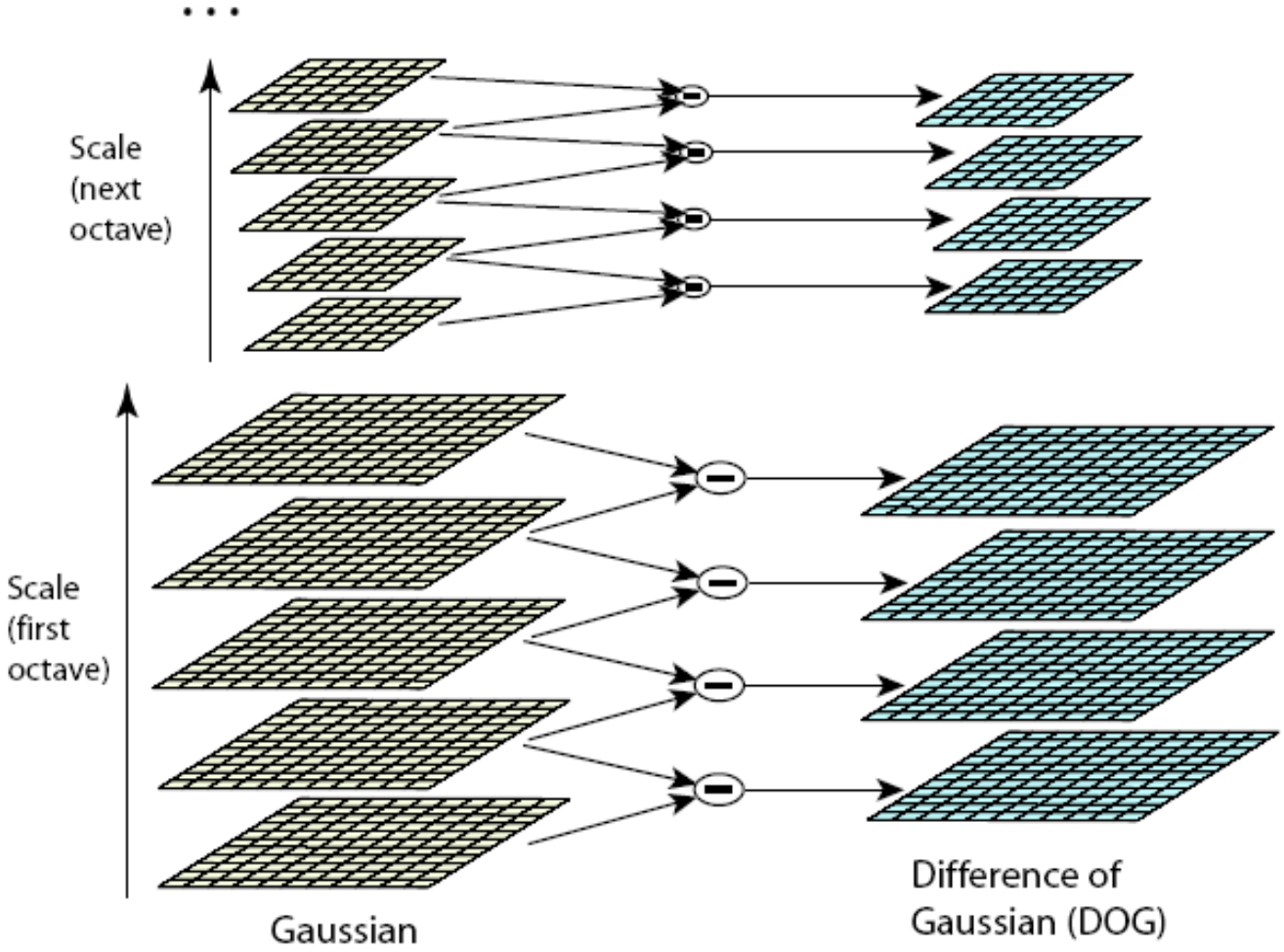
.

.

$\sigma_2=2^{2/s}\sigma_0$

$\sigma_1=2^{1/s}\sigma_0$

σ_0



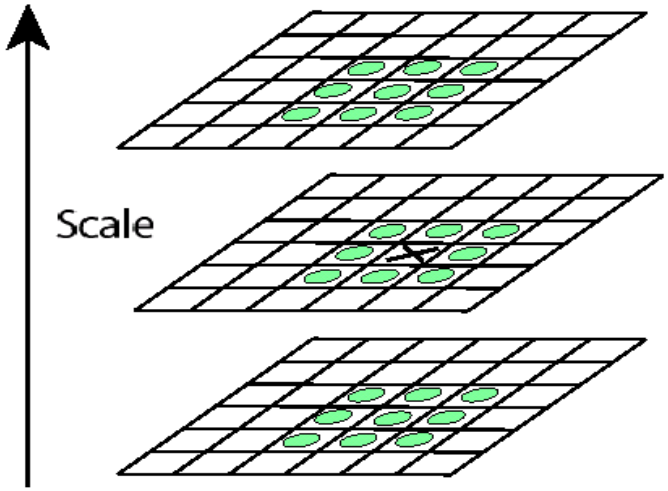
The parameter s determines the number of images per octave.

SIFT Descriptor
2. Key point localization

- **Detect maxima and minima of Difference-of-Gaussian in scale space**

s+2 difference images.
top and bottom ignored.
s planes searched.

- **Each point is compared to its 8 neighbors in the current image and 9 neighbors each in the scales above and below and determine:**



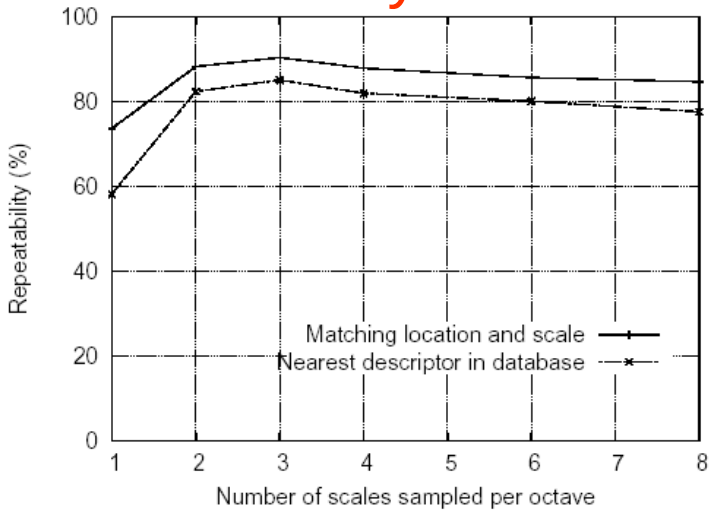
is it maximum or minimum between all of them?

For each **max or min** found, output is the **location** and the **scale** of keypoint condidates.

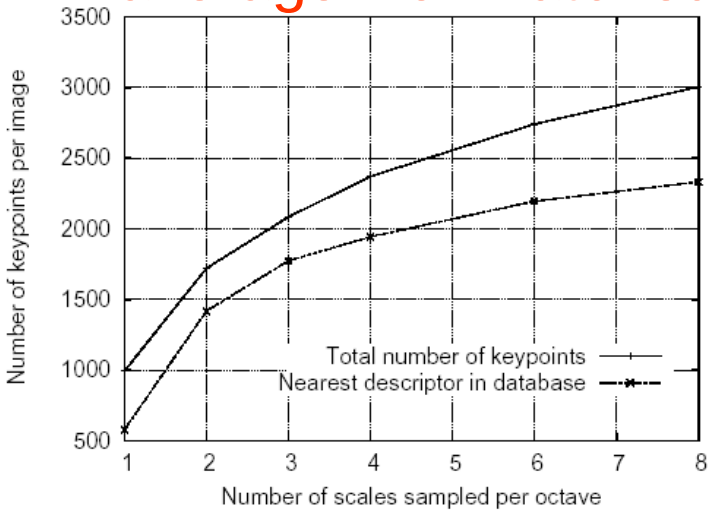
Scale-space extrema detection: experimental results over 32 images that were synthetically transformed and noise added.

- **Sampling in scale for efficiency**
  - **How many scales should be used per octave? S=?**
    - **More scales evaluated, more keypoints found**
    - **S < 3, stable keypoints increased too**
    - **S > 3, stable keypoints decreased**
    - **S = 3, maximum stable keypoints found**

% correctly matched



average no. matched



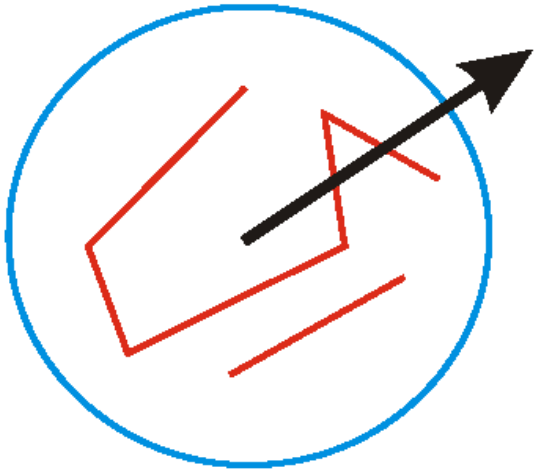
SIFT Descriptor
2. Key point localization

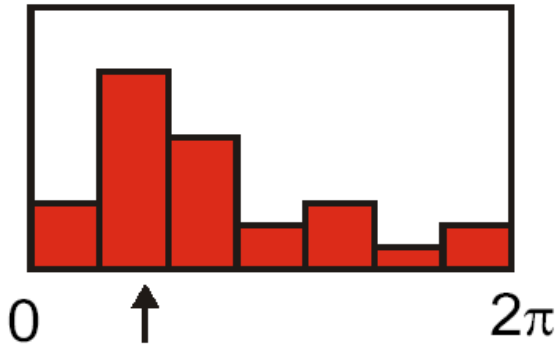
- **Once a keypoint candidate is found, perform a detailed fit to nearby data to determine**
  - **location, scale, and ratio of principal curvatures**
- **In initial work keypoints were found at location and scale of a central sample point.**
- **In newer work, they fit a 3D quadratic function to improve interpolation accuracy.**
  - **The Hessian matrix was used to eliminate edge responses.**
    - **Reject flats: $|D(x)| < 0.03$**
    - **Reject edges: $r = \alpha/\beta < 10$**

Let α be the eigenvalue with larger magnitude and β the smaller of Hessian matrix.

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

SIFT Descriptor
3. Orientation assignment





- **Create histogram of local gradient directions at selected scale**
- **Assign canonical orientation at peak of smoothed histogram**
- **Each key specifies stable 2D coordinates:**

  (x, y, scale, orientation)

If 2 major orientations, use both.

SIFT Descriptor
Keypoint localization with orientation

Razi University

233x189

Original Image

(a)

(b)

832

initial keypoints

729

keypoints after gradient threshold

(c)

(d)

536

keypoints after ratio threshold

SIFT Descriptor
4. Keypoint Descriptors

- **At this point, each keypoint has**
  - **location**
  - **scale**
  - **Orientation**

- **Next is to compute a descriptor for the local image region about each keypoint that is**
  - **highly distinctive**
  - **invariant as possible to variations such as changes in viewpoint and illumination**
  - **Rotate the window to standard orientation**
  - **Scale the window size based on the scale at which the point was found.**

SIFT Descriptor

Lowe's 2X2 Keypoint Descriptor(over 8X8)

gradient magnitude and orientation at each point weighted by a Gaussian

orientation histograms: sum of gradient magnitude at each direction

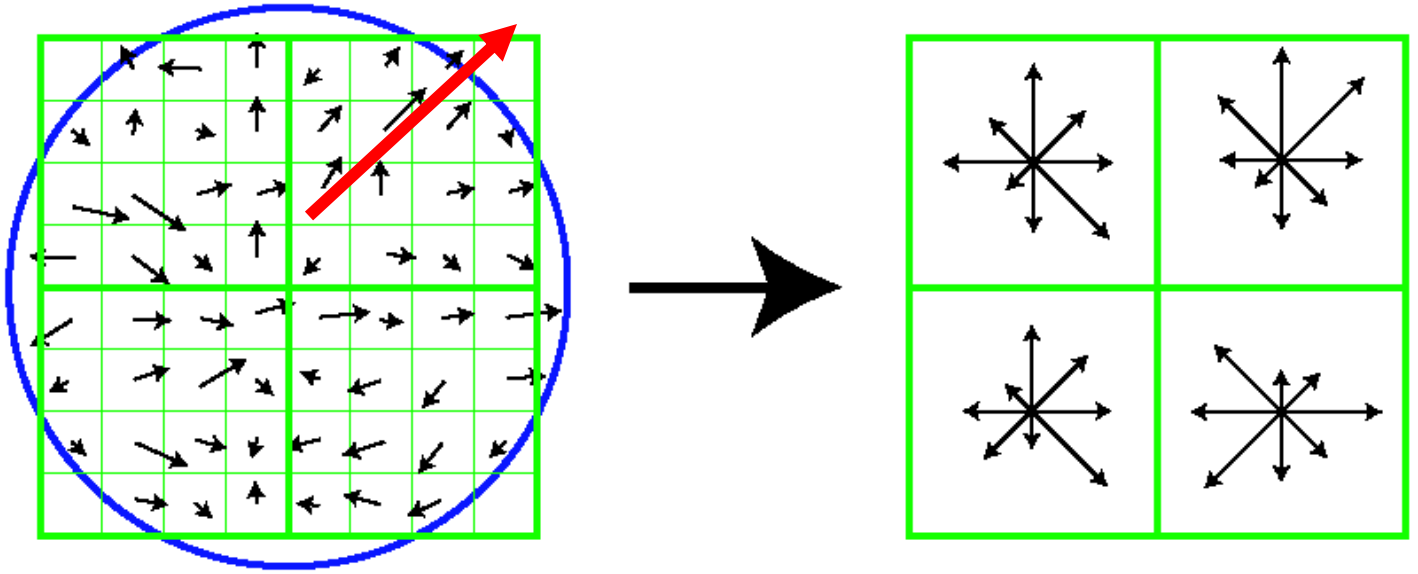


Image gradients

Keypoint descriptor

In experiments, 4x4 arrays of 8 bin histogram is used, a total of 128 features for one keypoint

SIFT Descriptor
Lowe's Keypoint Descriptor

- Use the normalized region about the keypoint
- Compute gradient magnitude and orientation at each point in the region
- Weight them by a Gaussian window overlaid on the circle
- Create an orientation histogram over the 4 X 4 subregions of the window 4 X 4 descriptors over 16 X 16 sample array were used in practice. 4 X 4 times 8 directions gives a vector of 128 values.

SIFT Descriptor

Using SIFT for Matching “Objects”

SIFT Descriptor
Applications and Benefits

- **SIFT Feature point Descriptor is used also for:**
  - **Image alignment (homography, fundamental matrix)**
  - **3D reconstruction (e.g. Photo Tourism)**
  - **Motion tracking**
  - **Object recognition**
  - **Indexing and database retrieval**
  - **Robot navigation**
  - **… many others**
- **SIFT Descriptor Benefits are:**
  - **General feature: not tied to any specific object**
  - **Can be used to detect rather complex objects that are not all one color**
  - **Location invariant, rotation invariant**
  - **Selects relevant scale, so scale invariant**

HOG Descriptor

- **Histograms of Oriented Gradients is a local descriptor proposed by Navneet Dalal and Bill Triggs (CVPR 2005) (first used for Human Detection)**

- **HoG algorithm is:**
  - **1. Partition the input image to cells**
  - **2. Compute gradients in the region to be described**
  - **2. Put them in 9 bins according to orientation**
  - **3. Group the obtained histograms of cells (by concatenation) into large blocks (typically a cell is of 8x8 pixels and a block is of 2x2 cells)**
  - **4. Normalize the histogram of each block**
  - **5. Concatenate the histogram of all blocks and construct HOG feature vector**

HOG Descriptor Flowchart

$G_x = I \otimes [-1\ 0\ 1]$

$G_y = I \otimes [-1\ 0\ 1]^T$

Apply the 1-D centered, point discrete derivative mask

Group the cells into connected blocks to normalize the gradient strength

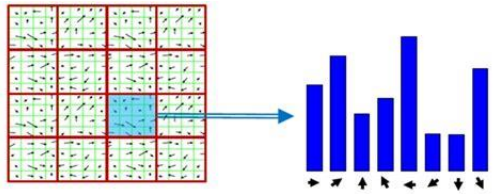
Gradient Computation → Orientation Binning → Descriptor Blocks

Magnitude

$$G = \sqrt{G_x^2 + G_y^2}$$

Angle

$$\alpha = ac\tan\left(\frac{G_y}{G_x}\right)$$



From each pixel within the cell is created an orientation-based histogram

The HOG Descriptor is the vector generated from the normalized cell histograms from all block regions

HOG Descriptor Details

Gradients

- [-1 0 1] and $[-1\ 0\ 1]^T$ were good enough.

Cell Histograms

- Each pixel within the cell casts a weighted vote for an orientation-based histogram channel based on the values found in the gradient computation. (9 channels worked)

Blocks

- Group the cells together into larger blocks, either R-HOG blocks (rectangular) or C-HOG blocks (circular).
  - R-HOG blocks appear quite similar to the SIFT descriptors.
  - But, R-HOG blocks are computed in dense grids at some single scale without orientation alignment.
  - SIFT descriptors are computed at sparse, scale-invariant key image points and are rotated to align orientation.

HOG Descriptor Details

- **Block Normalization**

They tried 4 different kinds of normalization. Let v be the block to be normalized and e be a small constant.

L2-norm: $$f = \frac{v}{\sqrt{\|v\|_2^2 + e^2}}$$

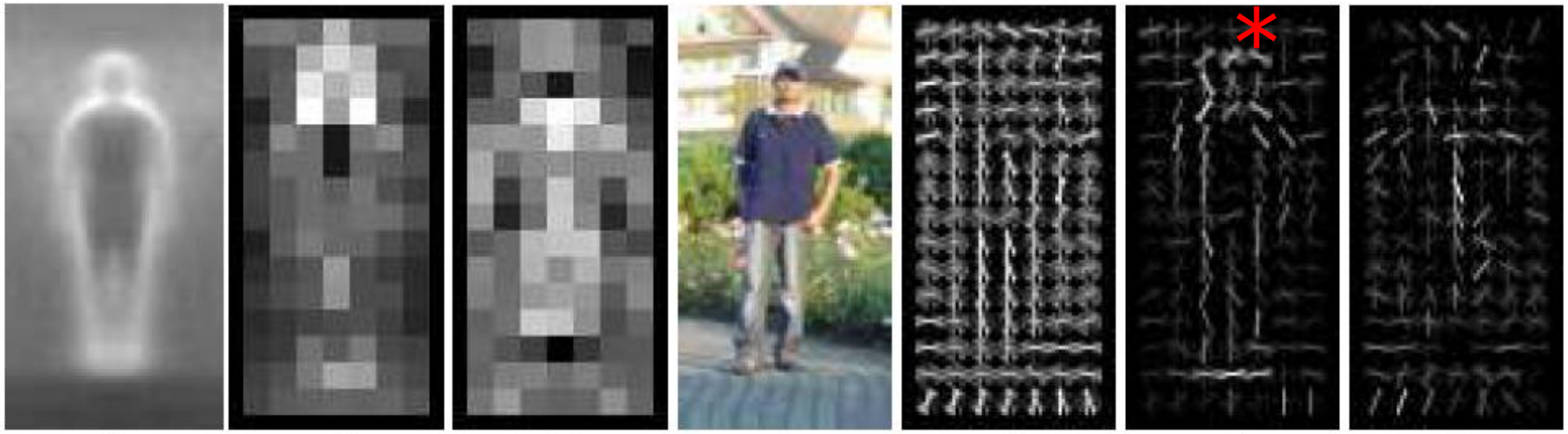
L2-hys: L2-norm followed by clipping (limiting the maximum values of v to 0.2) and renormalizing,

L1-norm: $$f = \frac{v}{(\|v\|_1 + e)}$$

L1-sqrt: $$f = \sqrt{\frac{v}{(\|v\|_1 + e)}}$$

HOG Descriptor
Pictorial Example

Razi University



(a)average gradient image over training examples
(b)each "pixel" shows max positive SVM weight in the block centered on that pixel
(c)same as (b) for negative SVM weights
(d)test image
(e)its R-HOG descriptor
(f) R-HOG descriptor weighted by positive SVM weights
(g)R-HOG descriptor weighted by negative SVM weights

Local Binary Pattern (LBP)

For each PIXEL of an image, a BINARY CODE is produced
→ to make a new matrix with the new value (binary to decimal value).

$$LBP_{p,r}(N_c) = \sum_{p=0}^{P-1} g\left(N_p - N_c\right)2^p$$

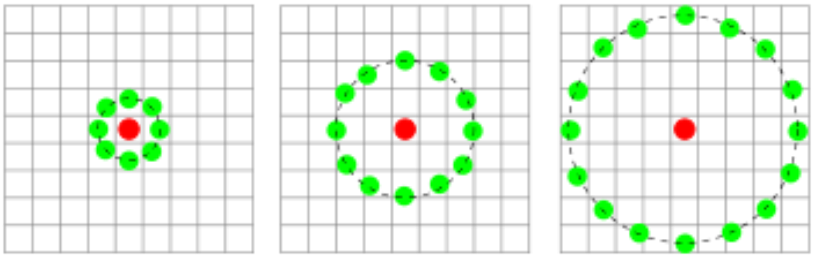
where, neighborhood pixels (N_p) in each block is thresholded by its center pixel value (N_c)

p→ sampling points (e.g., $p = 0, 1, \ldots, 7$ for a 3x3 cell, where P = 8)

r → radius (for 3x3 cell, it is 1).

Binary threshold function $g(x)$ is,

$$g(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$$



Computation of LBP

Binary code for > N_c

0	1	0
1	N_c	0
0	0	1

	3	7	2
	8	**4**	1
	2	3	5

Component-wise multiplication

1	2	4
128		8
64	32	16

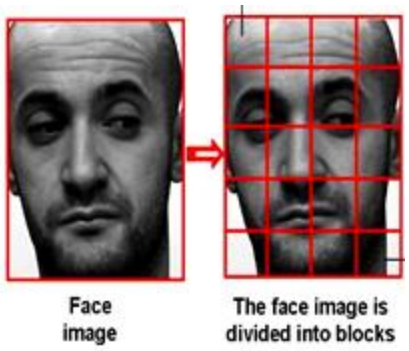
0	2	0
128	**?**	0
0	0	16

Representation

$\sum$ **Sum**

LBP

146



Face image

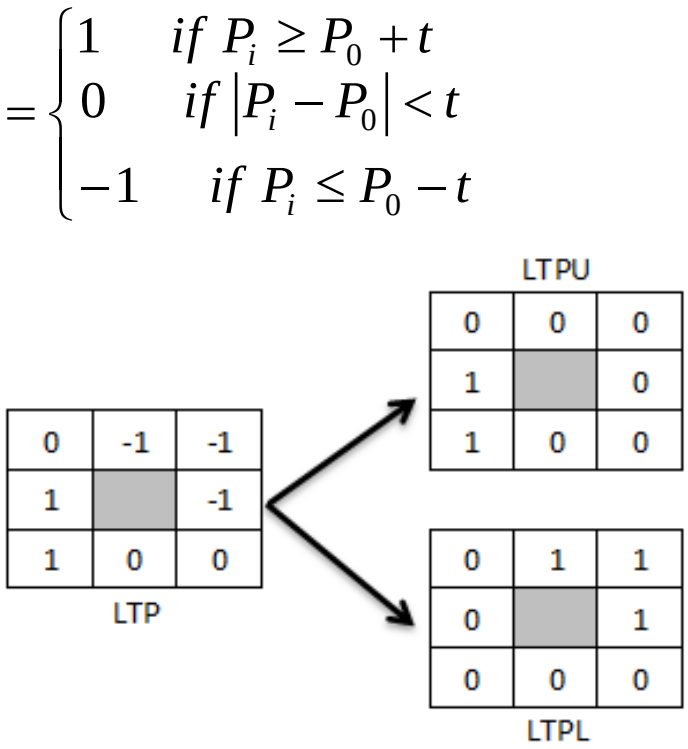
The face image is divided into blocks

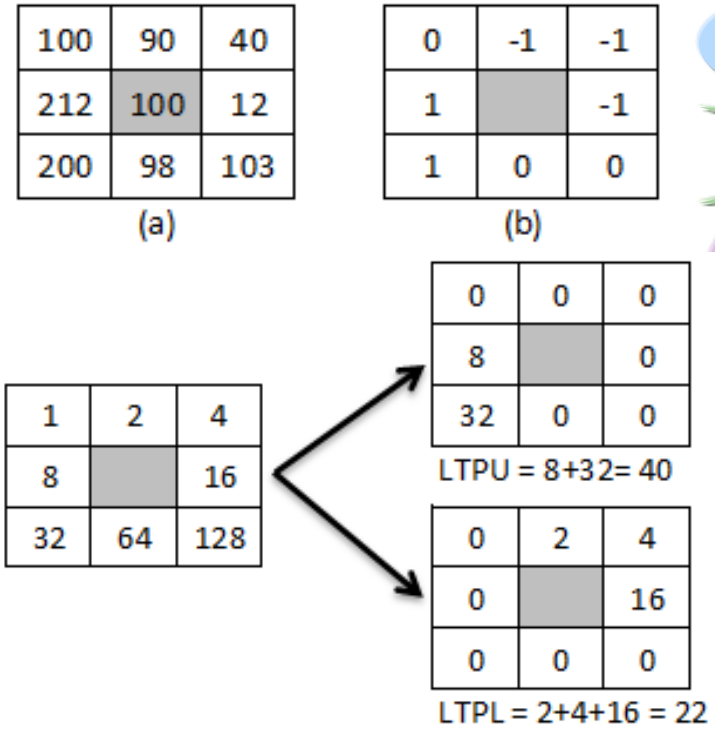
- **Optionally normalize the histogram.**
- **Concatenate normalized histograms of all cells. This gives the feature vector for the input image.**

Computation of LTP

- **LTP is similar to LBP, but :**
  - LBP is sensitive to noise in near uniform regions.
  - LTP solves this later problem.
  - LTP extends LBP to 3-valued (1, 0, -1) codes, in which local pixels are compared to a user defined thresholds –t and +t.

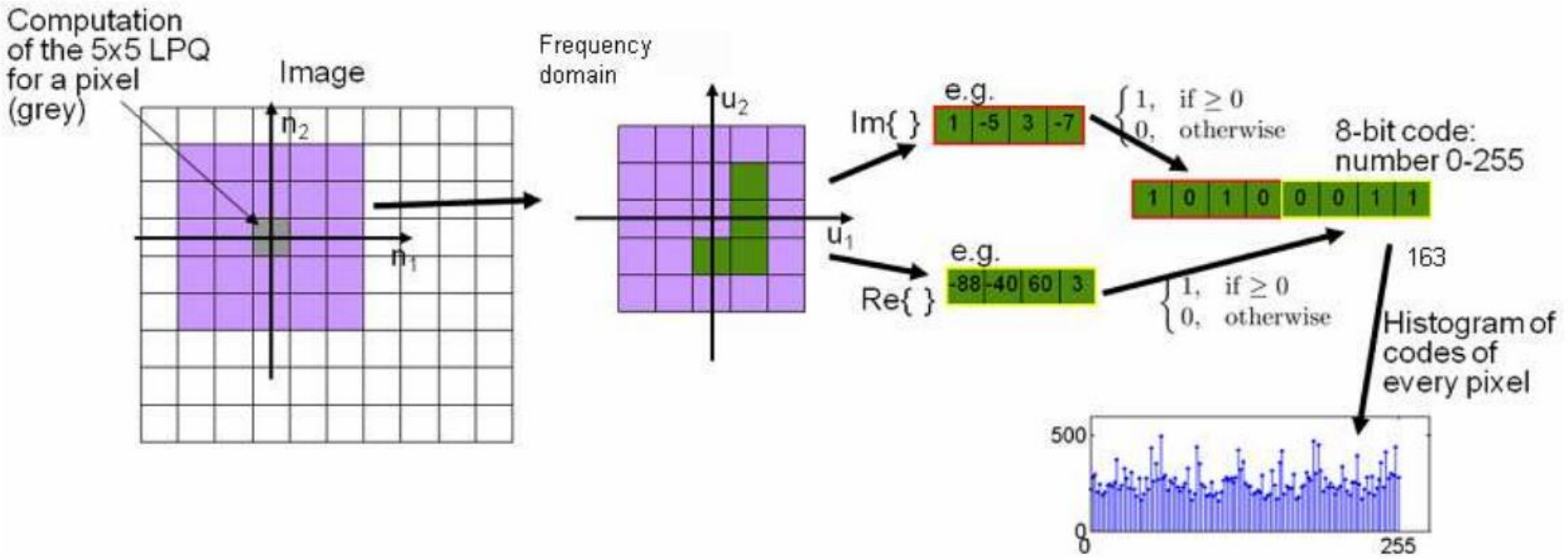
$$P_i' = \begin{cases} 1 & if\ P_i \geq P_0 + t \\ 0 & if\ |P_i - P_0| < t \\ -1 & if\ P_i \leq P_0 - t \end{cases}$$





Computation of LPQ

- **LPQ computation steps are:**
  - Compute Fourier Transform
  - For each pixel Real and Imaginary parts of four neighbors are binarized and perform LPQ code as below:



Texture

Texture

- **Texture is a description of the spatial arrangement of color or intensities in an image or a selected region of an image**

- **What defines a texture?**
  - Includes: more regular patterns
  - Includes: more random patterns

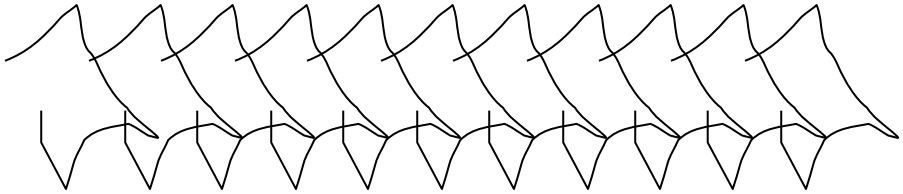
- **Often the same thing in the world can occur as texture or an object, depending on the scale we are considering.**

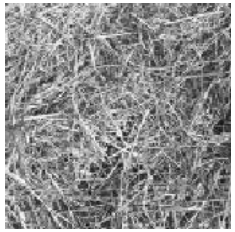
Texture-related tasks

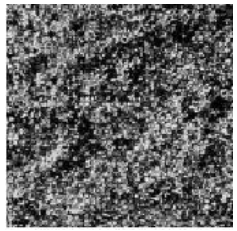
- **Shape from texture**
  - Estimate surface orientation or shape from image texture
- **Segmentation/classification from texture cues**
  - Analyze, represent texture
  - Group image regions with consistent texture
- **Synthesis**
  - Generate new texture patches/images given some examples

Texture Description Types

- **Structural approach:**
  - **Uses a set of textons/texels in some regular or repeated pattern**
  - **Problem**
    - **How do you decide what is a texton(especially in natural images)?**







- **Statistical approach:**
  - **Detecting textons is difficult or impossible in real images. Therefore, numeric quantities or statistics that describe a texture can be computed from the gray tones (or colors) alone.**
  - **This approach is less intuitive, but is computationally efficient.**

Statistical Texture Description

Textures are made up of repeated local patterns, so:

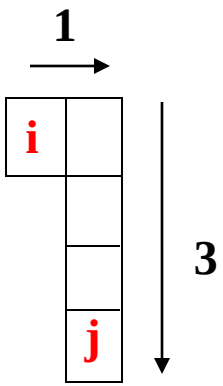
- **Find the patterns**
  - Use filters that look like patterns (spots, bars, raw, patches…)
  - Consider magnitude of response
- **Describe their statistics within each local window**
  - Mean, standard deviation
  - Histogram
  - Histogram of "prototypical" feature occurrences

Statistical Texture Description
Co-occurrence Matrix Features

- A **co-occurrence matrix** is a 2D array C in which both the rows and columns represent a set of possible image values.
  - $C_d(i,j)$ indicates how many times value i co-occurs with value j in a particular spatial relationship d.
  - The spatial relationship is specified by a vector $d = (dr, dc)$.
  - From C_d we can compute N_d, the normalized co-occurrence matrix, where each value is divided by the sum of all values.

1	1	0	0
1	1	0	0
0	0	2	2
0	0	2	2
0	0	2	2
0	0	2	2

gray-tone image



d = (3,1)

	0	1	2
0	1	0	3
1	2	0	2
2	0	0	1

C_d

co-occurrence matrix

	0	1	2
0	0.11	0.0	0.33
1	0.22	0.0	0.22
2	0.0	0.0	0.11

N_d

Normalized co-occurrence matrix

Statistical Texture Description
Co-occurrence Matrix Features

To obtain Texture feature, usually calculate Haralic functions:

$$Energy = \sum_i \sum_j N_d^2(i,j)$$

$$Entropy = -\sum_i \sum_j N_d(i,j) log_2 N_d(i,j)$$

$$Contrast = \sum_i \sum_j (i-j)^2 N_d(i,j)$$

$$Homogeneity = \sum_i \sum_j \frac{N_d(i,j)}{1+|i-j|}$$

$$Correlation = \frac{\sum_i \sum_j (i-\mu_i)(j-\mu_j) N_d(i,j)}{\sigma_i \sigma_j}$$

where μ_i, μ_j are the means and σ_i, σ_j are the standard deviations of the row and column

Energy measures uniformity of the normalized matrix.

Statistical Texture Description
Co-occurrence Matrix Features

How do you choose d?

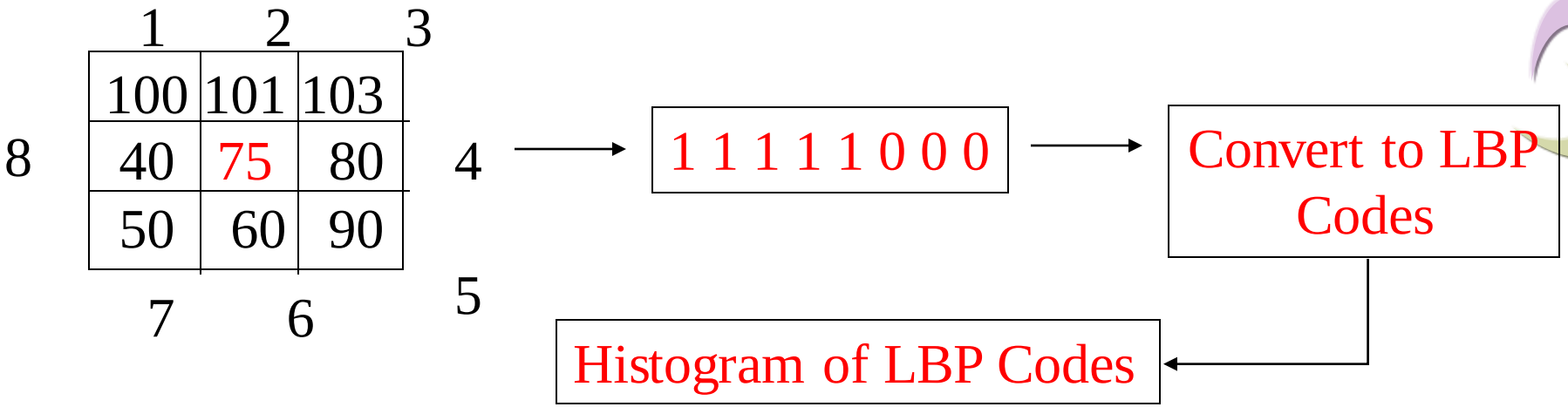
- This is actually a critical question with **all** the statistical texture methods.
- Are the "textons" tiny, medium, large, all three …?
- Not really a solved problem.

Zucker and Terzopoulos suggested using a χ^2 statistical test to select the value(s) of d that have the most structure for a given class of images.

Texture Description
Local Binary Pattern

- For each pixel p, create an 8-bit number $b_1\ b_2\ b_3\ b_4\ b_5\ b_6\ b_7\ b_8$, where $b_i = 0$ if neighbor i has value less than p's value and 1 otherwise.

- Represent the texture in the image (or a region) by the histogram of these numbers.



Texture Measure
Autocorrelation function

- **Autocorrelation function compares the dot product (energy) of non shifted image with a shifted image:**

$$\rho(dr, dc) = \frac{\sum_{r=0}^{N}\sum_{c=0}^{N} I[r,c]I(r+dr,c+dc]}{\sum_{r=0}^{N}\sum_{c=0}^{N} I^2[r,c]} = \frac{I[r,c] \circ I_d[r,c]}{I[r,c] \circ I[r,c]}$$

- **Autocorrelation function can detect repetitive patterns of textons**
- **Also defines fineness/coarseness of the texture**
  - **Coarse texture → function drops off slowly**
  - **Fine texture → function drops off rapidly**
  - **Can drop differently for r and c**
  - **Regular textures → function will have peaks and valleys; peaks can repeat far away from [0, 0]**
  - **Random textures → only peak at [0, 0]; breadth of peak gives the size of the texture**

Texture Description
Filter Bank

- **Signal-processing-based algorithms use texture filters applied to the image to create filtered images from which texture features are computed.**

- **We can generalize to apply a collection of multiple (d) filters: a "filter bank"**
  - Then our feature vectors will be d-dimensional.

- **What filters to put in the bank?**
  - Typically we want a combination of scales and orientations, different types of patterns.
  - Some of famous filter banks are **Laws**, **Gabor**, **Wavelet** and **Pyramid** filter banks

Texture Description
Laws' Texture Energy Features

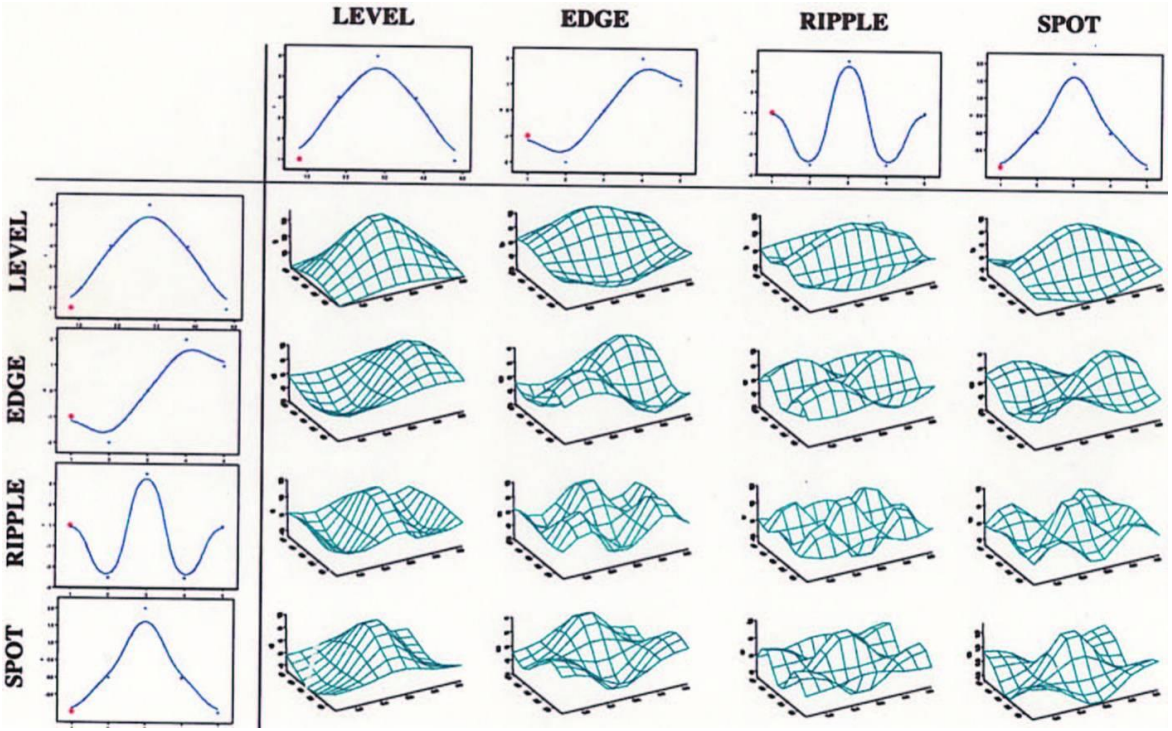
- **The Laws Algorithm uses texture filter banks applied to the image to create filtered images from which texture features are computed as the energy of it.**

- **The Laws Algorithm**
  - Creation of 2D Masks by multiplying corresponding 1D masks
  - Filter the input image using k^{th} texture mask to obtain filtered images F_k
  - Compute texture energy by summing the absolute value of filtered images in local neighborhoods around each pixel.

$$E_k[r,c] = \sum |F_k[i,j]|$$

  - Combine features to achieve rotational invariance.

Texture Description
Laws' Texture Masks

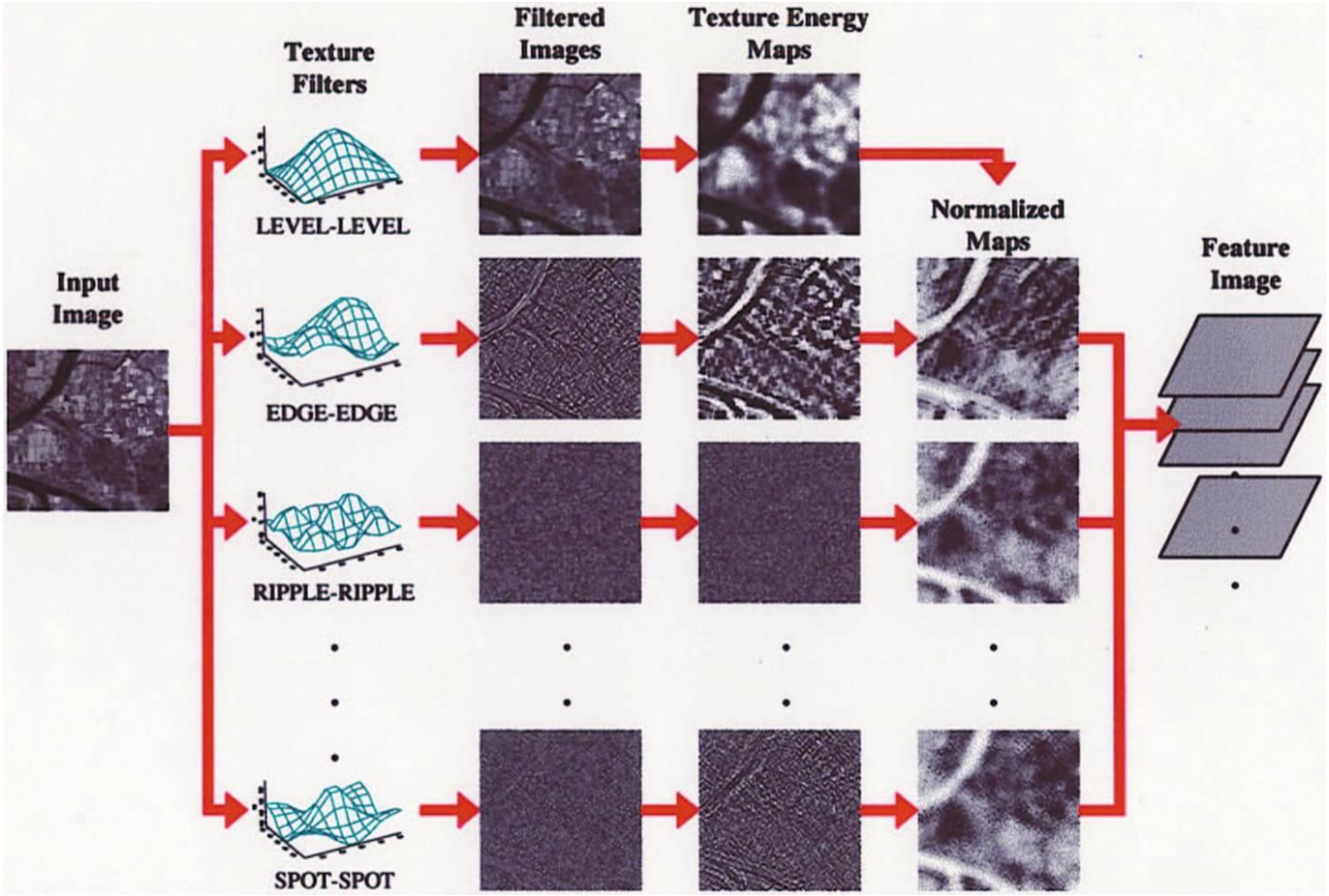


L5 (Level) = [1 4 6 4 1]
E5 (Edge) = [-1 -2 0 2 1]
S5 (Spot) = [-1 0 2 0 -1]
R5 (Ripple) = [1 -4 6 -4 1]

$$\begin{bmatrix} -1 \\ -2 \\ 0 \\ 2 \\ 1 \end{bmatrix} \times \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix} = \begin{bmatrix} -1 & -4 & -6 & -4 & -1 \\ -2 & -8 & -12 & -8 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 8 & 12 & 8 & 2 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

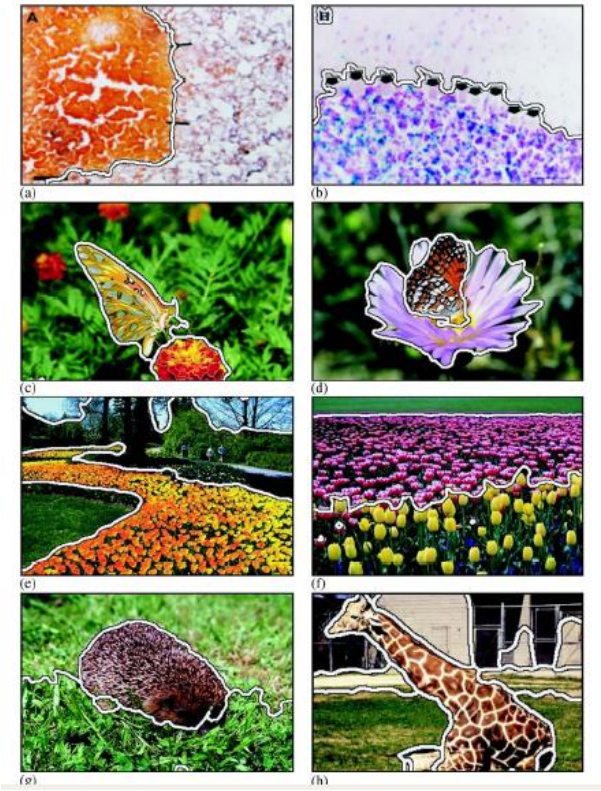
E5 L5 E5L5

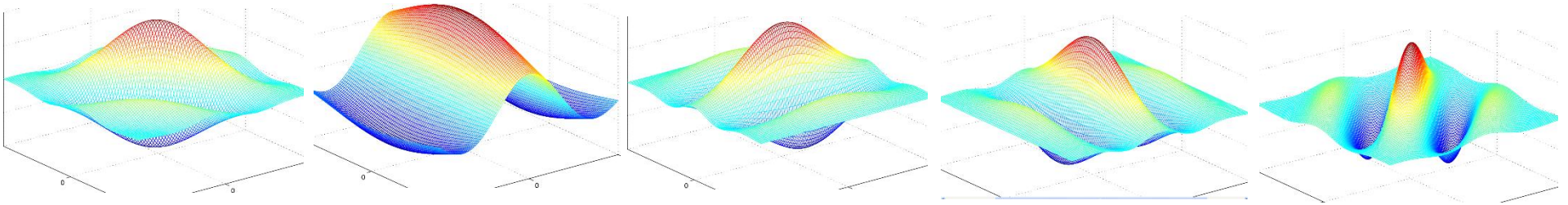
Texture Description
Laws' Texture Process



Texture Description
Gabor/Wavelet Filters

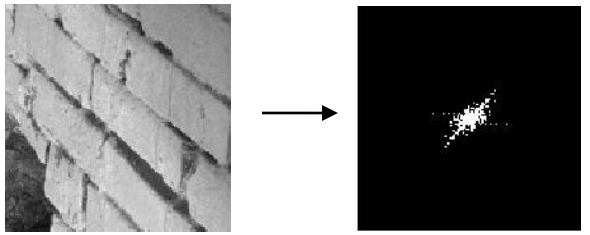
- **Similar approach to Laws**
- **Gabor/Wavelets filter bank (at different frequencies and different orientations) are applied on image or blocks of image**
- **Energy of filtered image are calculated as texture feature.**



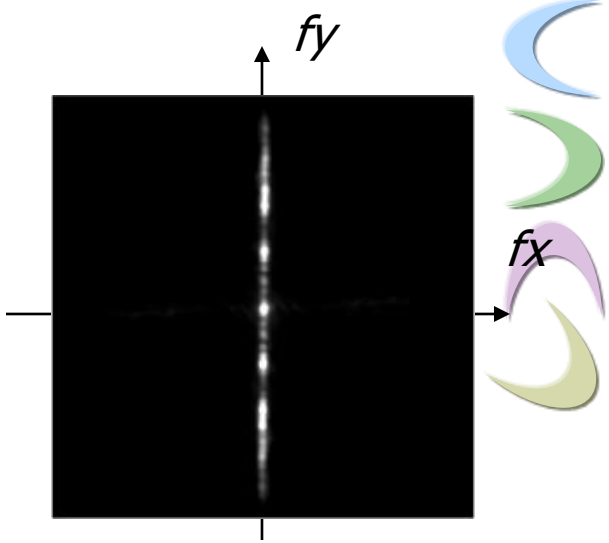


Texture Measure
Fourier power spectrum

- **High frequency power → fine texture**
- **Concentrated power → regularity**
- **Directionality → directional texture**



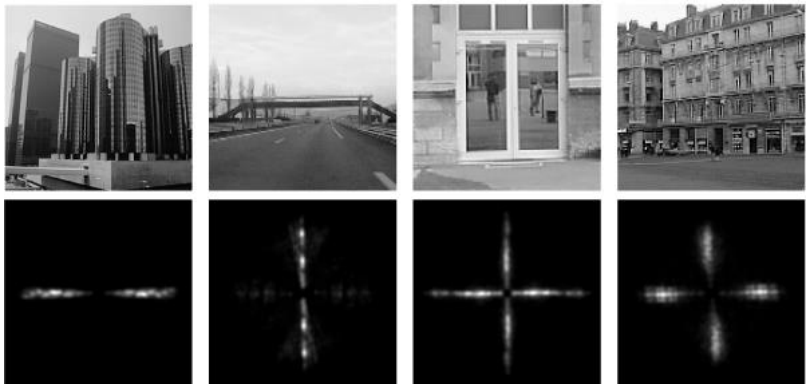
$$A(f_x, f_y) = \left| \sum_{x,y} i(x,y) e^{-2\pi j (f_x x + f_y y)} \right|$$



Magnitude of the Fourier Transform encodes unlocalised information about dominant orientations and scales in the image.

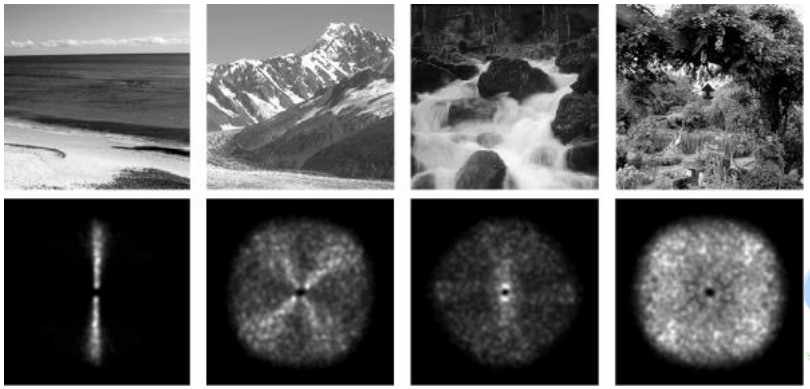
Texture Measure
Statistics of Scene Categories

Man-made environments

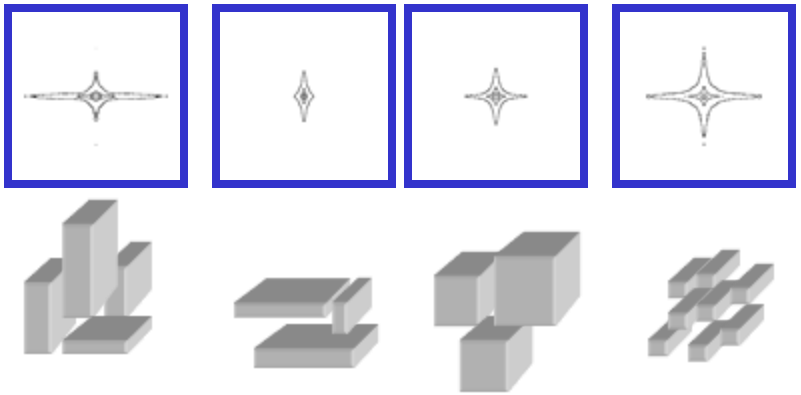


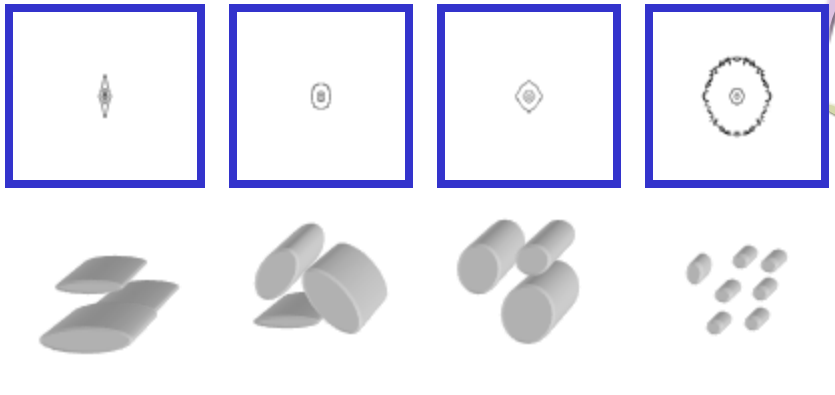
Spectral signature of man-made environments

Natural environments



Spectral signature of natural environments





Texture Measure
Proper Scale Selection

- **Texture is a local neighborhood property.**
- **Texture features computed at a wrong scale can lead to confusion.**
- **Texture features should be computed at a scale which is appropriate to the local structure being described.**

The white rectangles show some sample texture scales from the image.

Texture Measure
Scale Selection Terminology

- **Gradient of the L^*component (assuming that the image is in the $L^*a^*b^*$color space)** $:\nabla I = \begin{bmatrix} I_x \\ I_y \end{bmatrix}$

- **Symmetric Gaussian :** $G_\sigma(x,y) = G_\sigma(x) * G_\sigma(y)$

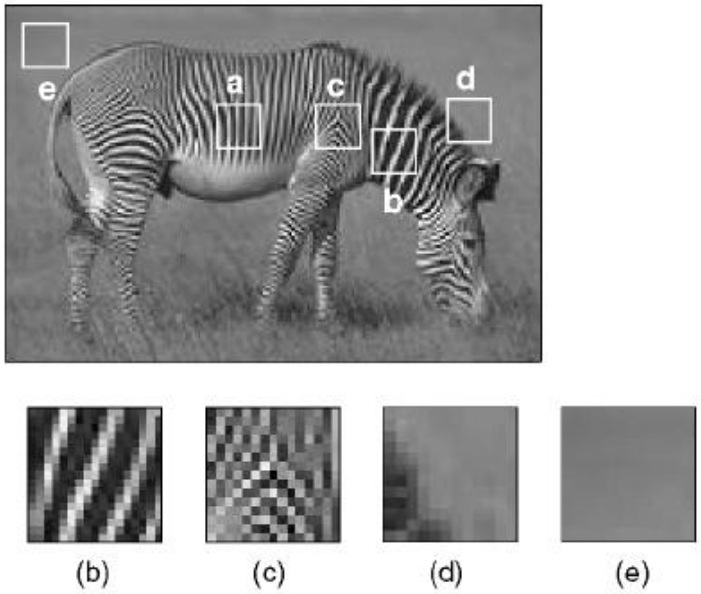
- **Second moment matrix:** $M_\sigma(x,y) = G_\sigma(x,y) * (\nabla I)(\nabla I)^T$

Notes:

- $G_\sigma(x,y)$ is a separable approximation to a Gaussian.
- σ is the standard deviation of the Gaussian $[0,.5,\ldots\ 3.5]$.
- σ controls the size of the window around each pixel $[1,2,5,10,17,26,37,50]$.
- $M_\sigma(x,y)$ is a 2×2 matrix and is computed at different scales defined by σ.

Texture Measure
Scale Selection Terminology

- **Make use of polarity (a measure of the extent to which the gradient vectors in a certain neighborhood all point in the same direction) to select the scale at which M_σ is computed**



Edge: polarity is close to 1 for all scales σ
Texture: polarity varies with σ
Uniform: polarity takes on arbitrary values

Texture Measure
Scale Selection Terminology

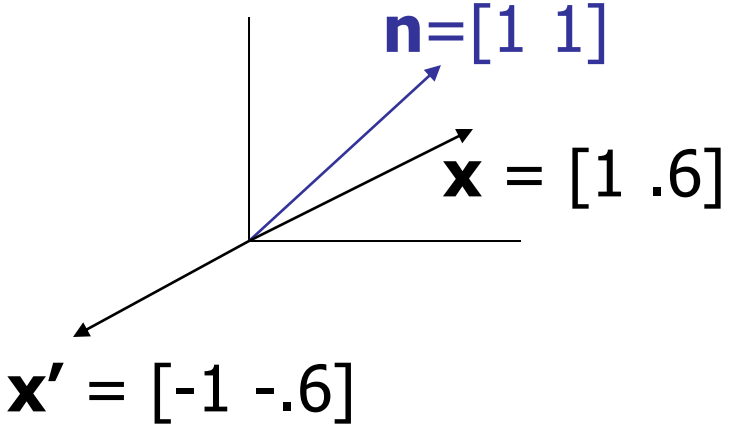
polarity p_σ

$$p_\sigma = \frac{|E_+ - E_-|}{E_+ + E_-}$$

$$E_+ = \sum_{x,y} G_\sigma(x,y)[\nabla I \cdot \hat{n}]_+$$

$$E_- = \sum_{x,y} G_\sigma(x,y)[\nabla I \cdot \hat{n}]_-$$

Example:



- **n** is a unit vector perpendicular to the dominant orientation.
- The notation [x]+ means x if x > 0 else 0

  The notation [x]- means x if x < 0 else 0
- We can think of E^+ and E^- as measures of how many gradient vectors in the window are on the positive side and how many are on the negative side of the dominant orientation in the window.

Texture Measure
Scale Selection Terminology

- **Texture scale selection is based on the derivative of the polarity with respect to scale σ.**

1. Compute polarity at every pixel in the image for $\sigma_k = k/2$, $(k = 0, 1 \ldots 7)$.

2. Convolve each polarity image with a Gaussian with standard deviation $2k$ to obtain a smoothed polarity image.

3. For each pixel, the selected scale is the first value of σ for which the difference between values of polarity at successive scales is less than 2%.

Texture Measure
Texture Features Extraction

- **Extract the texture features at the selected scale**
  - **Polarity (polarity at the selected scale) :** $p = p_{\sigma*}$
  - **Anisotropy :** $a = 1 - \lambda_2 / \lambda_1$
    - λ_1 and λ_2 denote the eigenvalues of M_σ
    - λ_2/λ_1 measures the degree of orientation: when λ_1 is large compared to λ_2 the local neighborhood possesses a dominant orientation.
    - When they are close, we have no dominant orientation.
    - When they are small, the local neighborhood is constant.

  - **Local Contrast:** $C = 2(\lambda_1 + \lambda_2)^{3/2}$

A pixel is considered homogeneous if $\lambda 1 + \lambda 2 <$ a local threshold

Segmentation and Clustering

Image Segmentation

- Image segmentation is the operation of partitioning an image into a collection of connected sets of pixels.
- Separating "content" from background
- Separating image into parts corresponding to "real" objects
- Complete segmentation
  - Each part corresponds to a real object
  - No overlapping parts
- Partial segmentation
  - Regions of homogeneous brightness, texture, color, etc.
  - Overlapping parts, needs further processing

Image Segmentation

- **Image segmentation can be categorized as:**

1. Global knowledge based

- Thresholding based on histogram

2. Regions, which usually cover the image using

- Region Growing /Watershed
- Split and Merge
- Clustering

2. Linear structures, such as

- Line segments
- Curve segments

3. 2D shapes, such as

- Circles
- Ellipses
- Ribbons (long, symmetric regions)
- Active Contour Models

Regions

Lines and Circular Arcs

Image Segmentation
Basic Formulation and Criteria

- **Let R represent the entire image region. Segmentation partitions R into n subregions, R_1, R_2, ..., R_n, such that:**

a) **Every pixel must be in a region** $\bigcup_{i=1}^{n} R_i = R$

b) **Points in a region must be connected.** R_i is a connected region, $i = 1, 2, \ldots, n$.

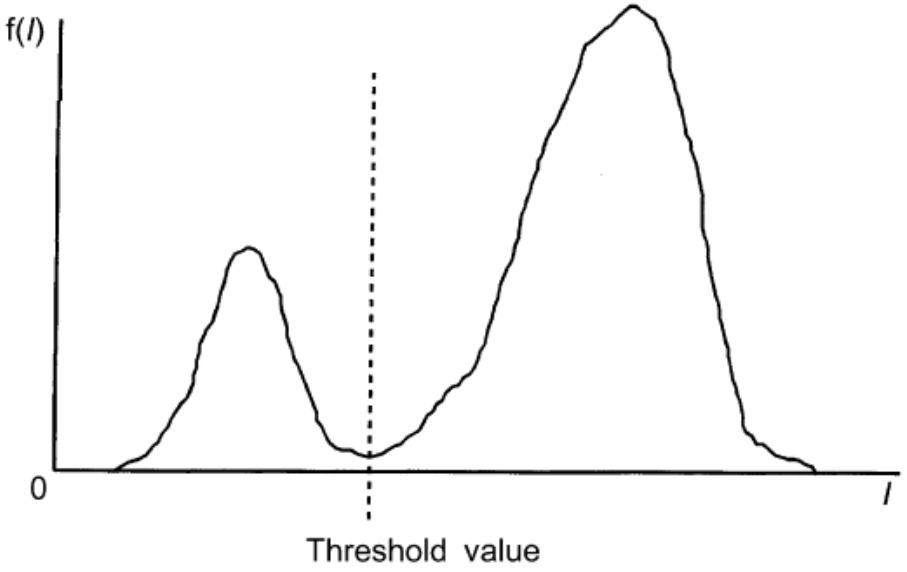
c) **Regions must be disjoint.** $R_i \cap R_j = \phi$ for all i and $j, i \neq j$

d) **All pixels in a region satisfy specific properties.** $P(R_i) = TRUE$ for $i = 1,2,\ldots,n$.

e) **Different regions have different properties.** $P(R_i \cup R_j) = FALSE$ for $i \neq j$.

Segmentation by Thresholding

- **Simple Global Thresholding**
  - **Simply select a threshold value between peak values from a histogram plot and set zero below than threshold and set 255 greater than threshold.**



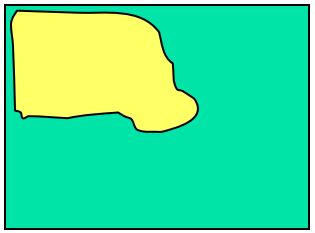
  - **Disadvantage: lost a lot of data information and there is no guarantee of grouped (well-separated) histogram.**

Region Based Segmentation

- **Goal: find regions that are "homogeneous" by some criterion**
- **Region Growing**
  - **Start with tiny regions, combine neighboring regions that are sufficiently similar to each other**
  - **Need: initialization, similarity criterion**
- **Region Splitting**
  - **Start with one big region, separate regions that are not sufficiently homogeneous**
  - **Need: homogeneity criterion, split rule**

Region Growing

Region growing techniques start with one pixel of a potential region and try to grow it by adding adjacent pixels till the pixels being compared are too dissimilar.



- **The first pixel selected can be just the first unlabeled pixel in the image or a set of seed pixels can be chosen from the image.**
- **Usually a statistical test is used to decide which pixels can be added to a region.**

Region Growing Example (Haralick & Shapiro)

- **Assume: region is connected component with the same population mean and variance**
- **Initialization: R is a single pixel (e.g. top-left)**
- **Similarity Criterion: region R to neighboring pixel y**
  - **Compute the mean intensity value of R: ($\bar{x}$)**
  - **Compute the scatter (variance) of R: (s^2)**
  - **Compute a T-value –**

    measures difference between y and the "typical pixel" in R which has N pixels.

    $$T = \sqrt{\frac{N(N-1)}{N+1}(y-\bar{x})^2/s^2}$$
  - **If T is small enough, add y to R and recompute statistics**
  - **Otherwise, y cannot be added to R -- either find another compatible neighboring region, or start a new one**

Region Based Segmentation
Watershed

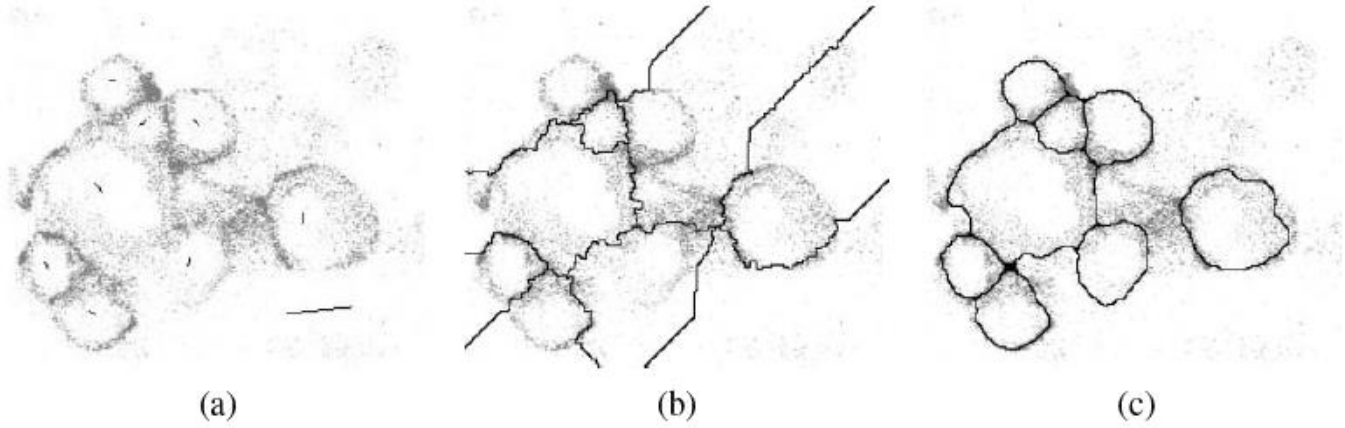
- **Smooth the image**
- **Run an edge detector (get gradient magnitude = edge strength)**
- **Create a 'seed' at each local min i.e. where there is no edge.**
- **Essentially, the idea is to 'flood' the image, marking each pixel as it goes 'underwater'**

5	5	6	1	1
1	3	5	2	1
1	2	4	5	3
0	1	1	2	2
1	1	1	1	1

```
val = 0
While (unmarked pixels exist){
    for(pixel in image)
        if (pixel.val = val)
            if 2 neighbors are different
                        pixel is border
            else if no neighbors are marked
                        pixel is seed
            else        mark pixel = neighbor
    val++
}
```

Region Based Segmentation
Improving Watershed

- **Instead of using *all* minima, user selects minima directly.**
  - **"pixel is seed" test removed from algorithm**
- **Use morphology to clean up local boundaries, smooth edges (e.g. closing)**



(a) (b) (c)

Region Based Segmentation
Region Splitting Example (Ohlander)

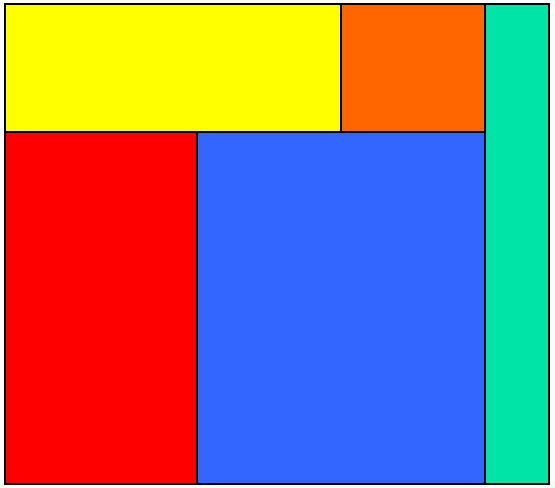
- Push a mask consisting of the entire image
- While the stack is not empty
  - Pop a mask from the stack
  - Compute Histogram of the current mask
  - Divide the histogram into clusters (find valleys between peaks)
  - If there are multiple peaks
    - For each peak, compute connected components and push a mask for each
  - Else, label all pixels in mask as one region
- Result: Set of regions (each a connected component)
  - Homogeneity criterion
    - Single-peak histogram
  - Split rule
    - Split according to histogram peak/valley analysis
    - Consider separate connected components as separate regions

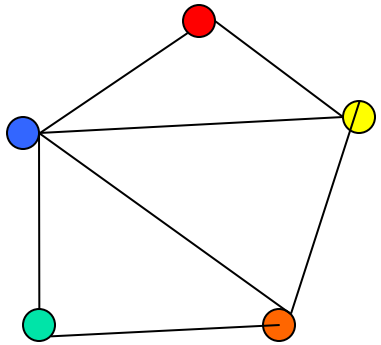
Region Based Segmentation
Growing and Splitting

- **Combining splitting and merging may yield best regions**
  - **Better than splitting alone, because splitting is constrained to specific "split" boundaries**
  - **Better than merging alone, because merging two whole regions might be overkill**
- **Use hierarchical data structure: split moves down, merge moves up**

Region Based Segmentation
Graph Based Segmentation

- **Image graph:**
  - **Each vertex is a region**
  - **Each edge is a boundary between two regions**
  - **Each edge has a weight corresponding to the dis-similarity between regions (e.g. intensity difference)**





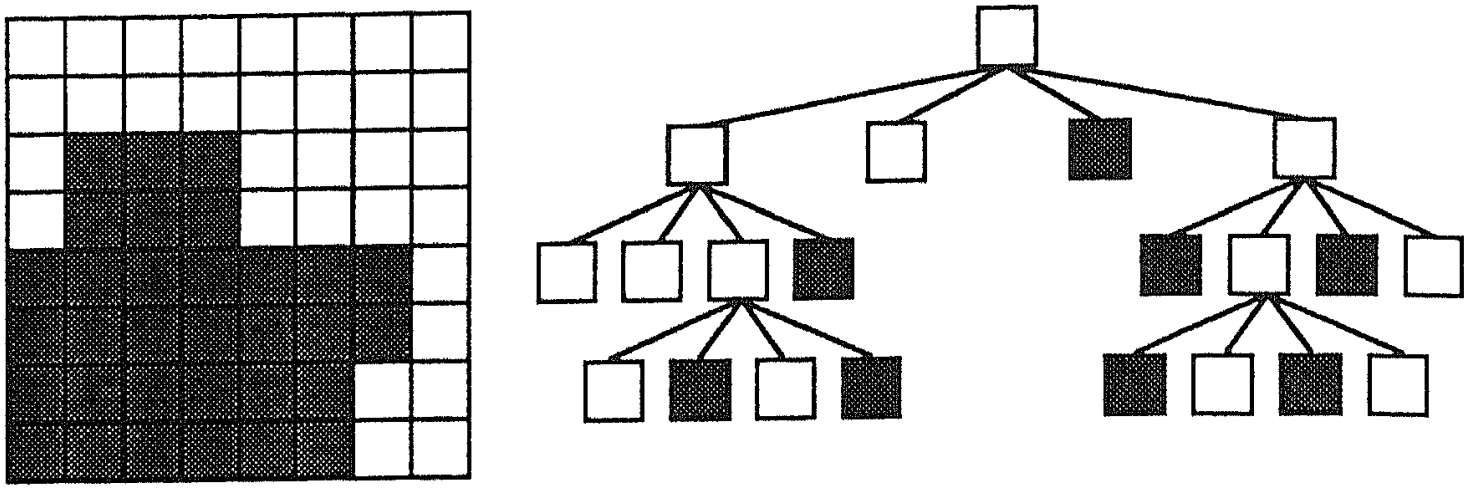
Region Based Segmentation
Graph Based Segmentation

- Initialize the graph as one region per pixel
- For each region R, internal difference = largest edge weight in minimum spanning tree [Int(R)]
- Difference between R1 and R2 is minimum edge weight connecting the two regions (i.e. any pixel from R1 to any pixel from R2) [Diff(R1, R2)]
- If Diff(R1,R2) < min(Int(R1)+p(R1), Int(R2)+p(R2)) merge R1 and R2
  - p(R) is a region penalty they set to k / (region area)
- Merge regions in *decreasing order* of the edges separating them, i.e. most dissimilar first

Region Based Segmentation
Quadtree

- **Quadtree is a data structure that captures multiple scales, where each tree represents a square subimage.**

- **Each subimage can have 4 children (upper-left quadrant, upper-right, lower-left, and lower-right)**
  - **If region is homogeneous, it is a leaf (no matter how big)**
  - **Smallest regions are single pixels (and they are always leaves)**



Region Based Segmentation
Split & Merge with Quadtrees

- **Initialize segmentation**
  - **arbitrarily or using prior knowledge**
- **For each region R, if it is not homogeneous, split it.**
  - **This builds a quadtree.**
- **If R1 and R2 are neighbors, and they can be merged into a homogeneous region, do it.**
  - **This might destroy the quadtree structure, because you might get a 1/4 region connected to a 1/16 region**
- **If any regions are "too small", merge them with the "best" neighbor.**

Clustering based Segmentation

- **Goal: threshold a multi-dimensional histogram to find clusters**
  - **In theory, 5-dimensional histogram (3 color + 2 location)**
  - **In reality, just pick 2 or 3 dimensions (e.g. L*a*b*)**

- There are K clusters $C_1, \ldots, C_K$ with means $m_1, \ldots, m_K$.

- The **least-squares error** is defined as

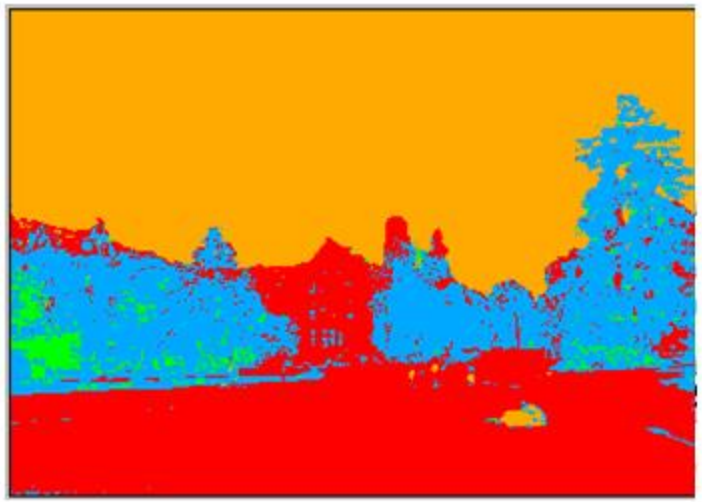
$$D = \sum_{k=1}^{K} \sum_{x_i \in C_k} \| x_i - m_k \|^2.$$

- Out of all possible partitions into K clusters, choose the one that minimizes D.

Clustering based Segmentation
K-means Segmentation

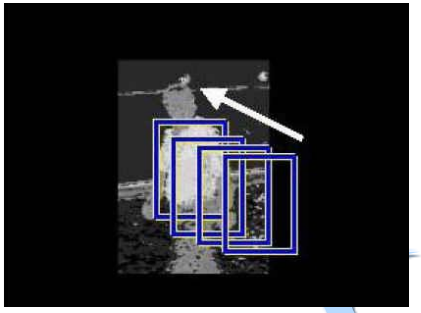
For K-means clusters from a set of n-dimensional vectors

1. Set ic (iteration count) to 1
2. Choose randomly a set of K means $m_1(1)$, …, $m_K(1)$.
3. For each vector x_i compute $D(x_i , m_k(ic))$, k=1,…K
   and assign x_i to the cluster C_j with nearest mean.
4. Increment ic by 1, update the means to get $m_1(ic)$,…,$m_K(ic)$.
5. Repeat steps 3 and 4 until $C_k(ic) = C_k(ic+1)$ for all k.



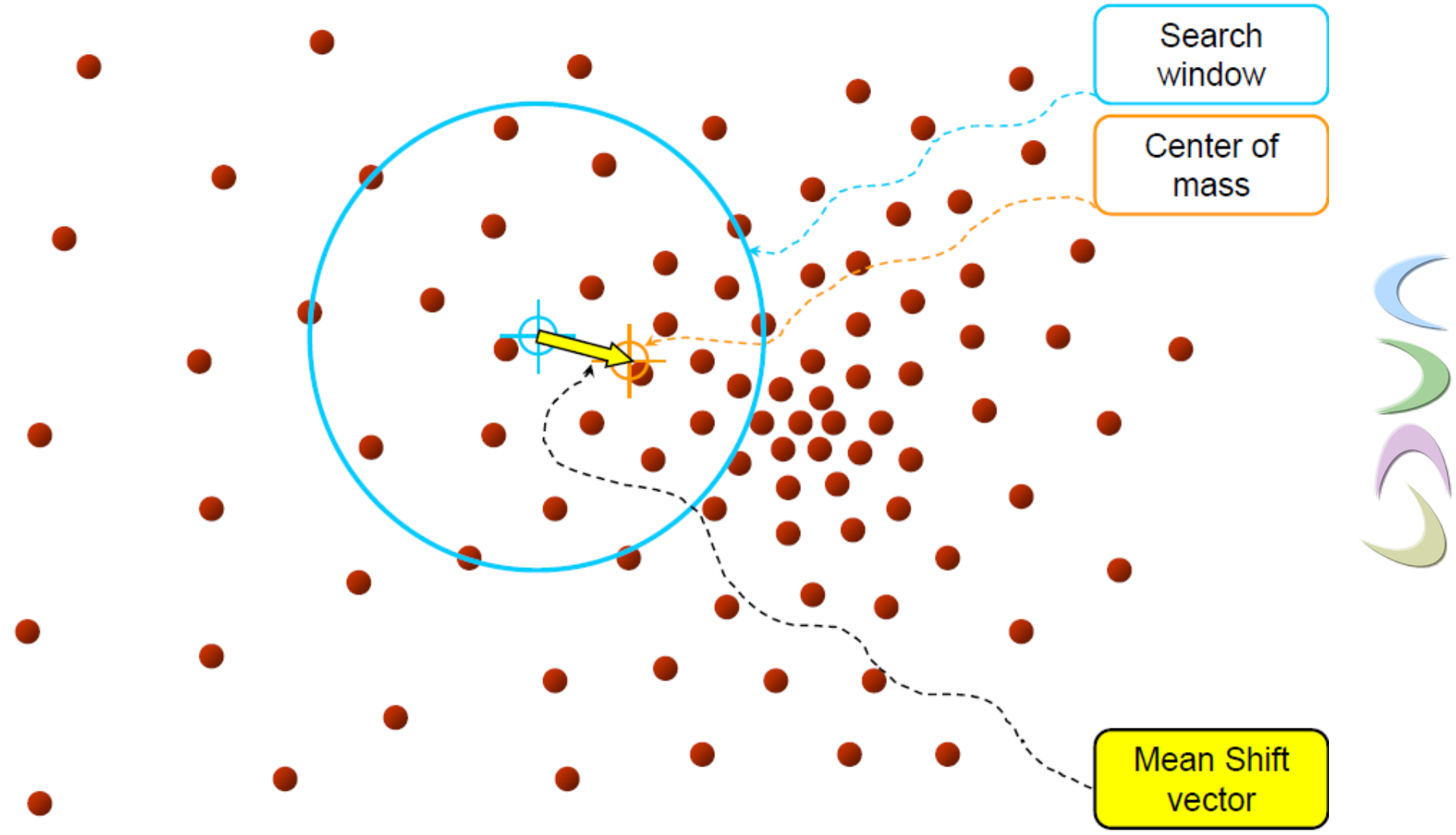
Clustering based Segmentation
Mean shift Clustering

- **The mean shift algorithm seeks *modes* or local maxima of density in the feature space.**
  - **Simple, like K-means**
  - **But you don't have to select K**
  - **Statistical method**
  - **Guaranteed to converge to a fixed number of clusters.**

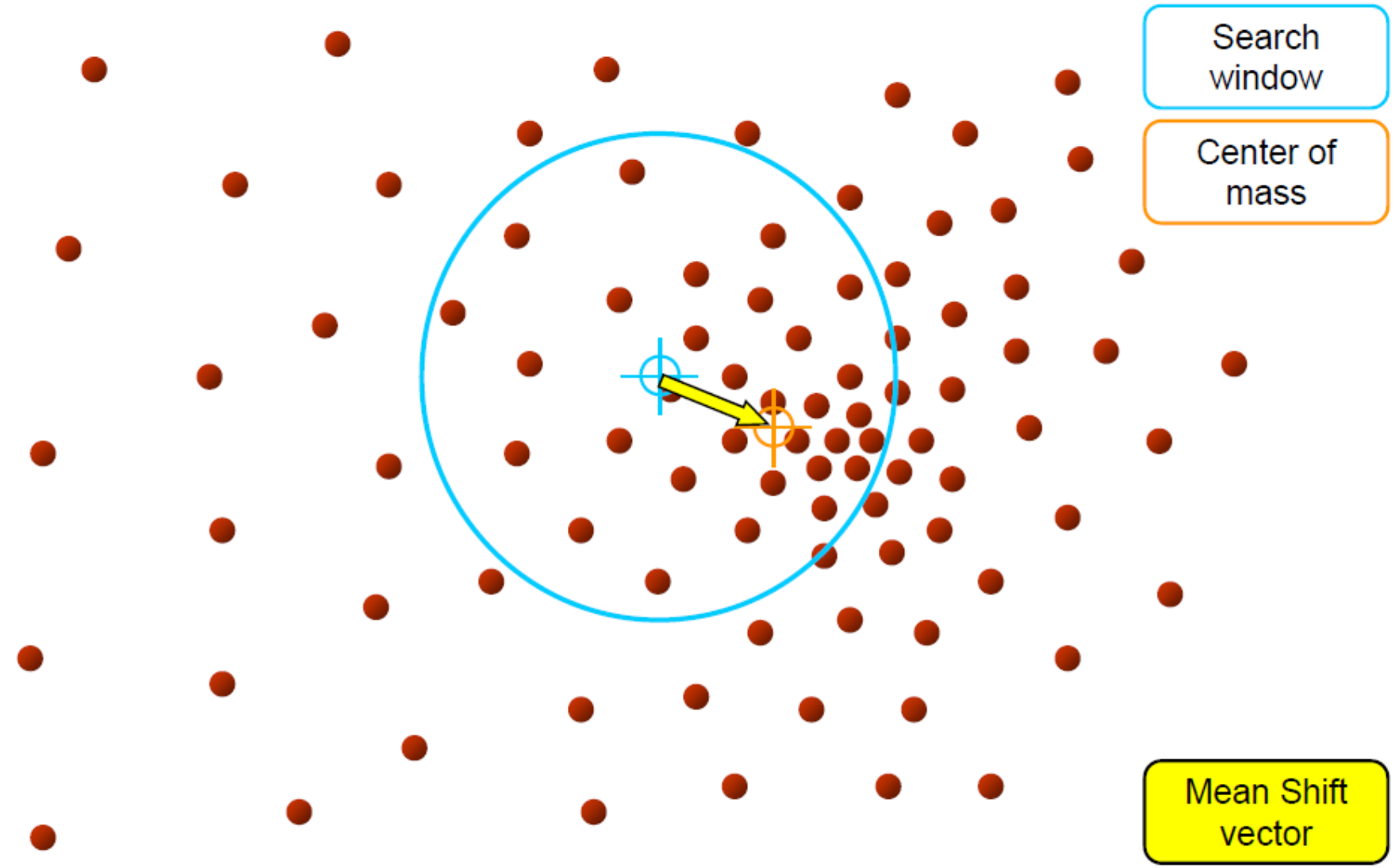


- **Mean shift algorithm climbs the gradient of a probability distribution to find the nearest domain mode (peak)**
  - **Instead of exhaustive search in the window, the gradient information provided by the mean shift is used to reduce the time cost**
  - **Much better than moving the search window pixel by pixel and scanning row by row**

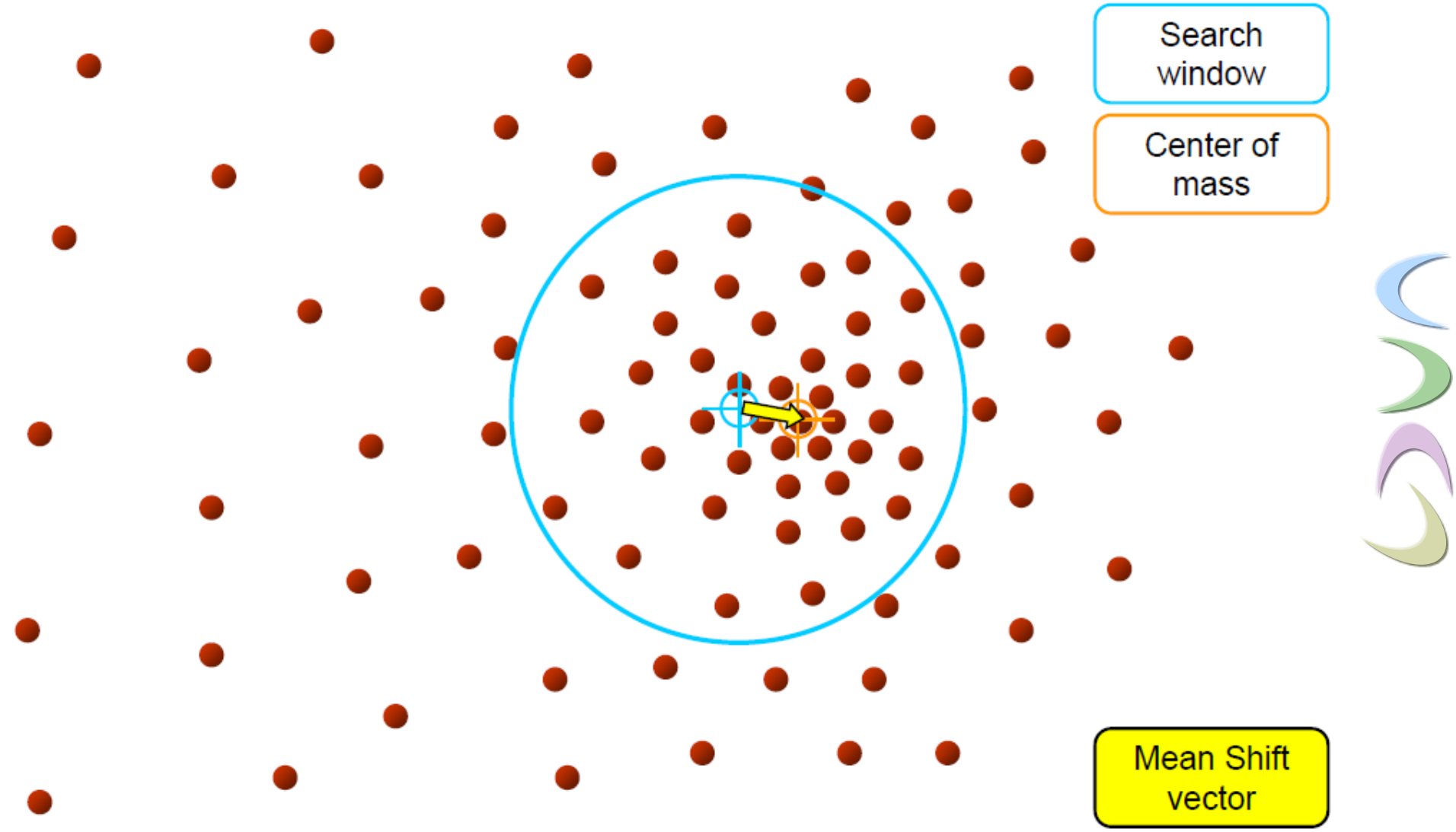
Clustering based Segmentation
Mean shift Clustering



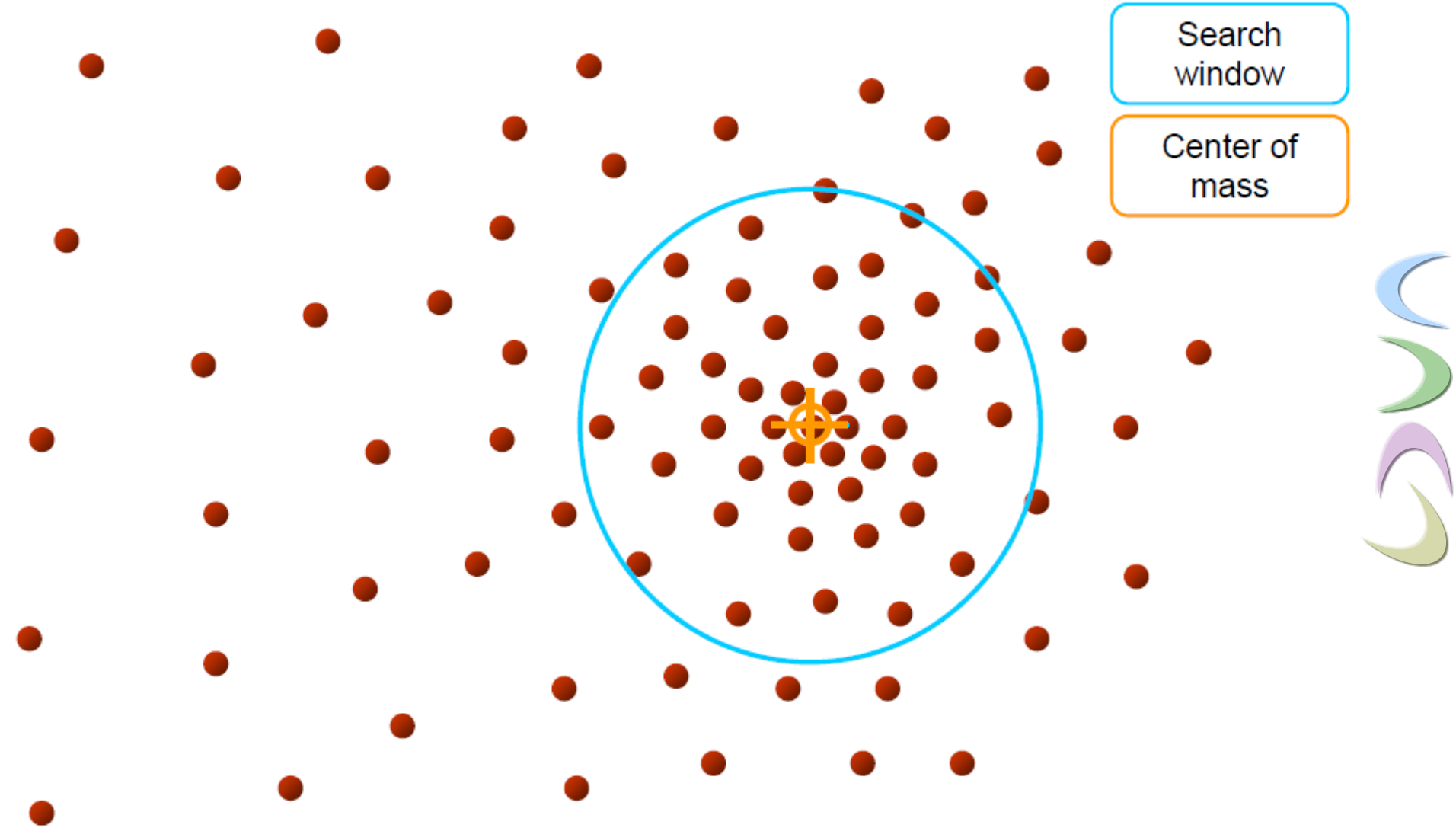
Clustering based Segmentation
Mean shift Clustering



Clustering based Segmentation
Mean shift Clustering

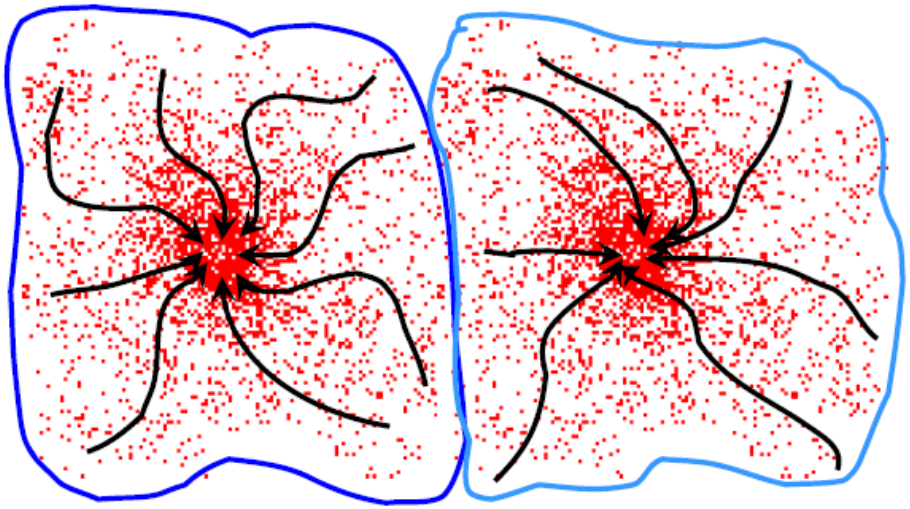


Clustering based Segmentation
Mean shift Clustering



Clustering based Segmentation
Mean shift Clustering

- **Initialize windows at each data point**
- **Perform mean shift for each window until convergence**
- **Merge windows that end up near the same "peak" or mode**
  - **Cluster: all data points in the attraction basin of a mode**
  - **Attraction basin: the region for which all trajectories lead to the same mode**

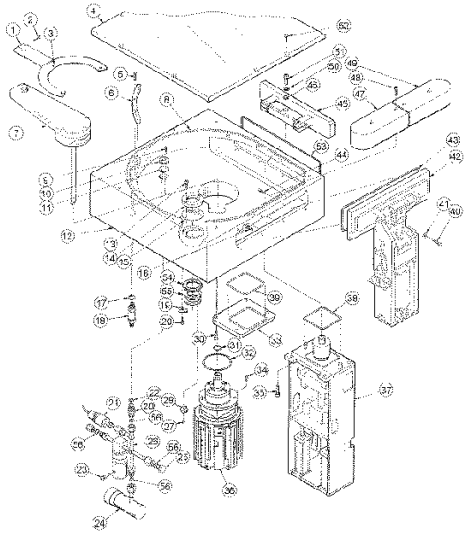


Lines and Arcs Segmentation

In some image sets, lines, curves, and circular arcs are more useful than regions or helpful in addition to regions.

Lines and arcs are often used in

- **object recognition**
- **stereo matching**
- **document analysis**



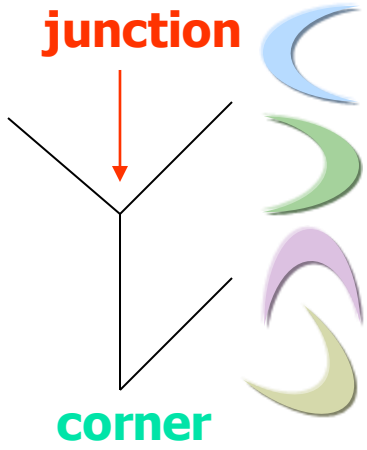
To detect lines, curves, and circular arcs in images, we should first extract the edge map of it using different edge detector operators like Sobel, Canny and etc.

Finding Line and Curve Segments from Edge Images

Given an edge image, how do we find line and arc segments?

Answer: By Using Tracking Technique

Use masks to identify the following events:



1. **start** of a new segment
2. **interior point** continuing a segment
3. **end** of a segment
4. **junction** between multiple segments
5. **corner** that breaks a segment into two

Finding Line and Curve Segments from Edge Images

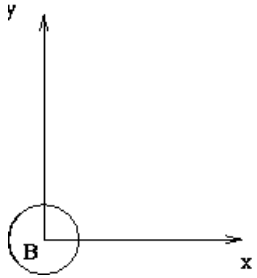
- **for each edge pixel P, classify its type using masks and perform appropriate task:**
  1. **isolated point :**
     - **ignore it**
  2. **start point :**
     - **make a new segment**
  3. **interior point :**
     - **add to current segment**
  4. **end point :**
     - **add to current segment and finish it**
  5. **junction or corner :**
     - **add to incoming segment**
     - **finish incoming segment**
     - **make new outgoing segment(s)**

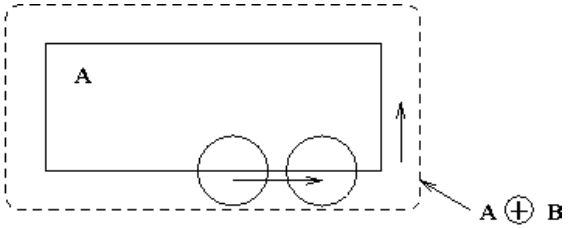
Image Segmentation
Morphological Filters

- **Morphological Filters Provide mathematical descriptions of geometric structures.**
  - **Based on *sets*.**
    - **Groups of pixels which define an image region or structure.**
- **What is this used for?**
  - **Binary images.**
  - **Can be used for post-processing segmentation results!**
- **Core techniques**
  - **Erosion, Dilation.**
  - **Open, Close.**

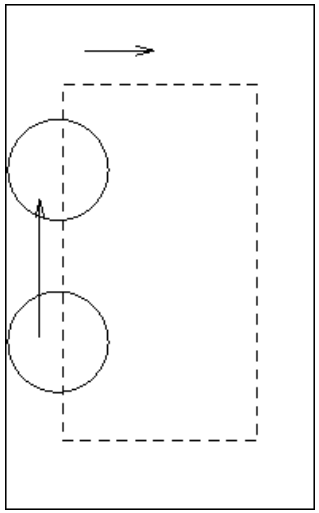
Morphological Filters
Dilation, Erosion

- **Two sets:**
  - **Image**
  - **Morphological *kernel.***





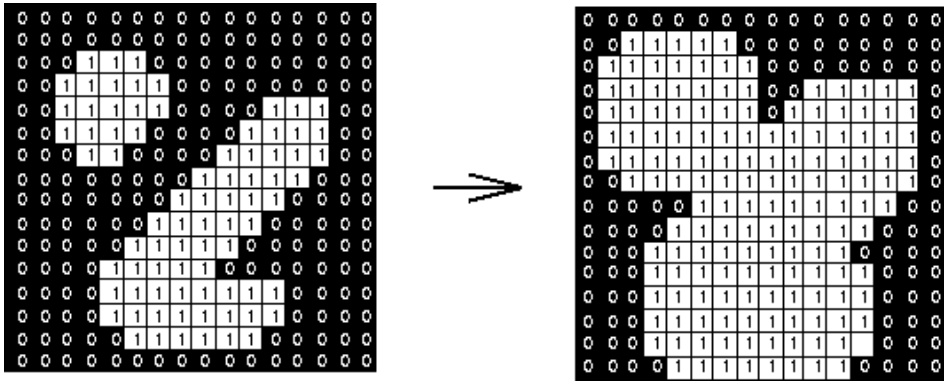
- **Dilation (D)** $D(A,B) = A \oplus B = \bigcup_{\beta \in B} (A + \beta)$
  - **Union of the kernel with the image set.**
  - **Increases resulting area.**
- **Erosion (E)** $E(A,B) = A \ominus (-B) = \bigcap_{\beta \in B} (A - \beta)$
  - **Intersection.**
  - **Decreases resulting area.**



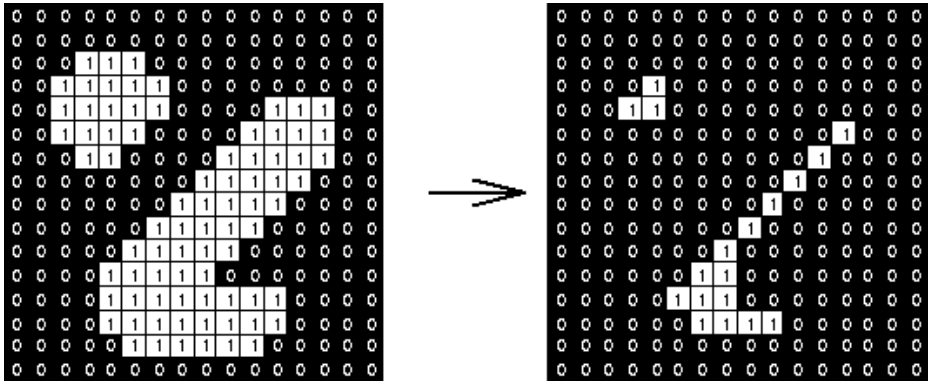
Morphological Filters
Dilation, Erosion

- **Example using a 3x3 morphological kernel**

- Dilation

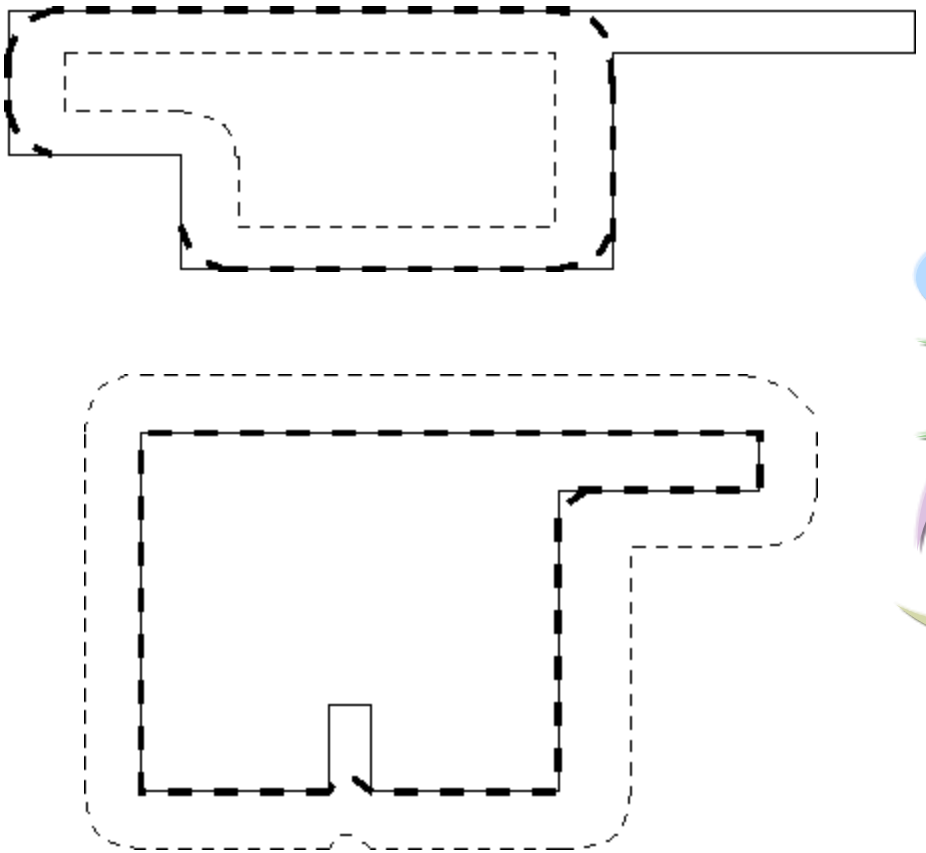


- Erosion



Morphological Filters
Opening, Closing

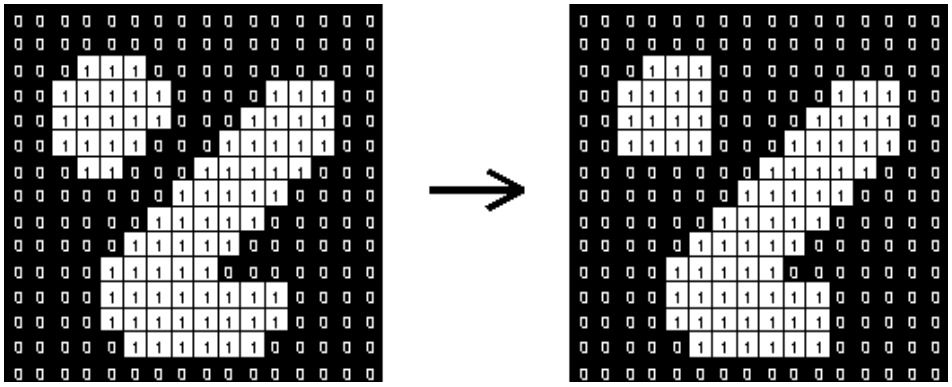
- **Opening**
  - **Erosion, followed by dilation.**
  - **Less destructive than an erosion.**
  - **Adapts image shape to kernel shape.**
- **Closing**
  - **Dilation, followed by erosion.**
  - **Less destructive than a dilation.**
  - **Tends to close shape irregularities.**



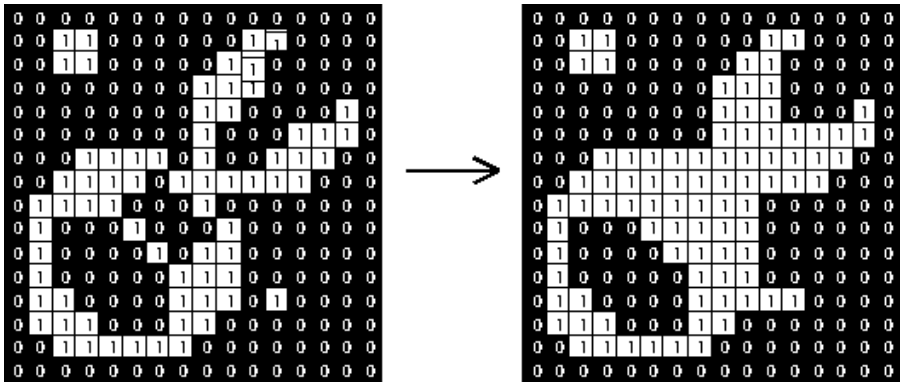
Morphological Filters
Opening, Closing

- **Example using a 3x3 morphological kernel**

- Opening



- Closing



Morphological Filters
Core morphological operators

Dilation

Erosion

Closing

Opening

Image Segmentation
Example: Morphological Opening

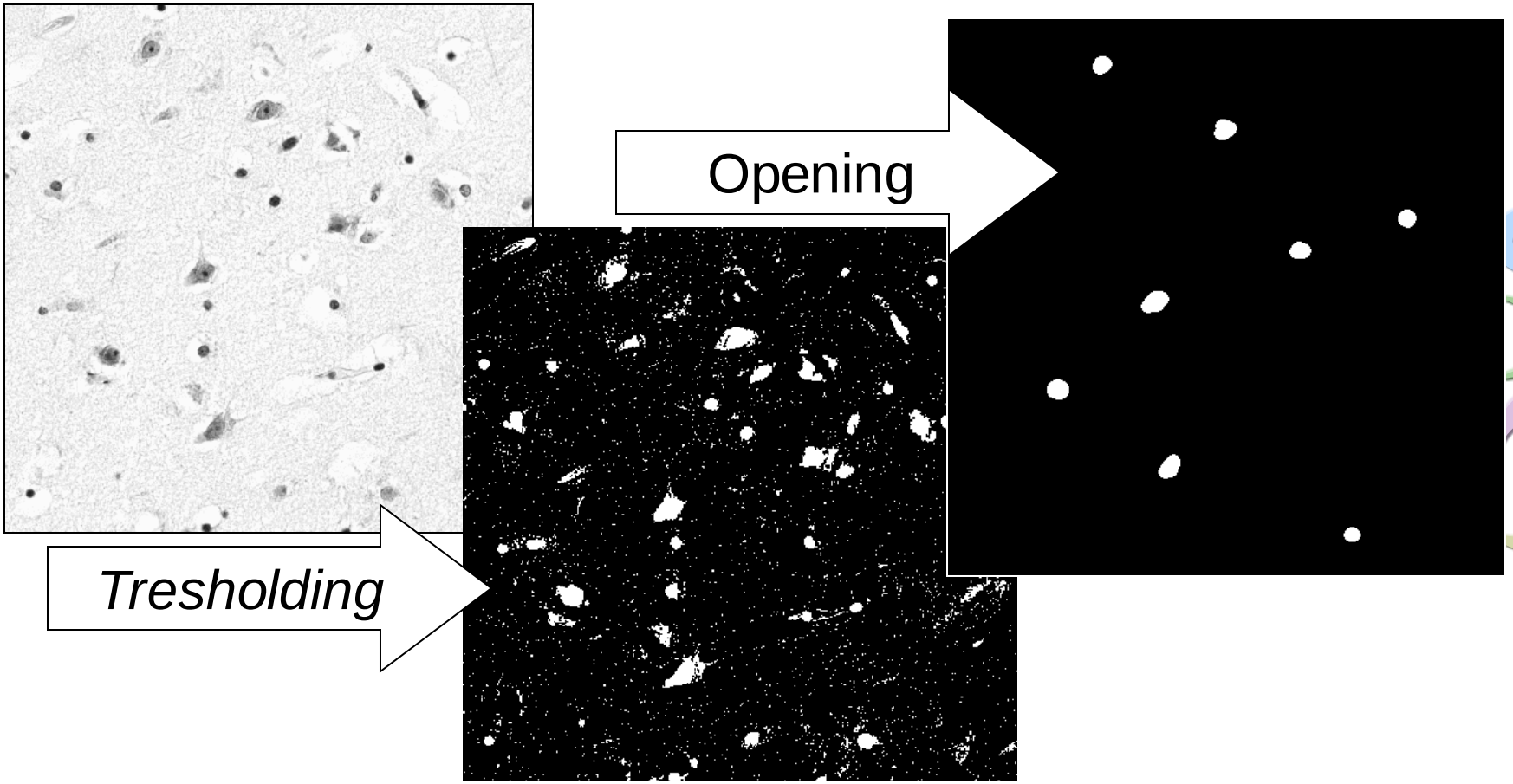


Image Segmentation
Example: Morphological Closing

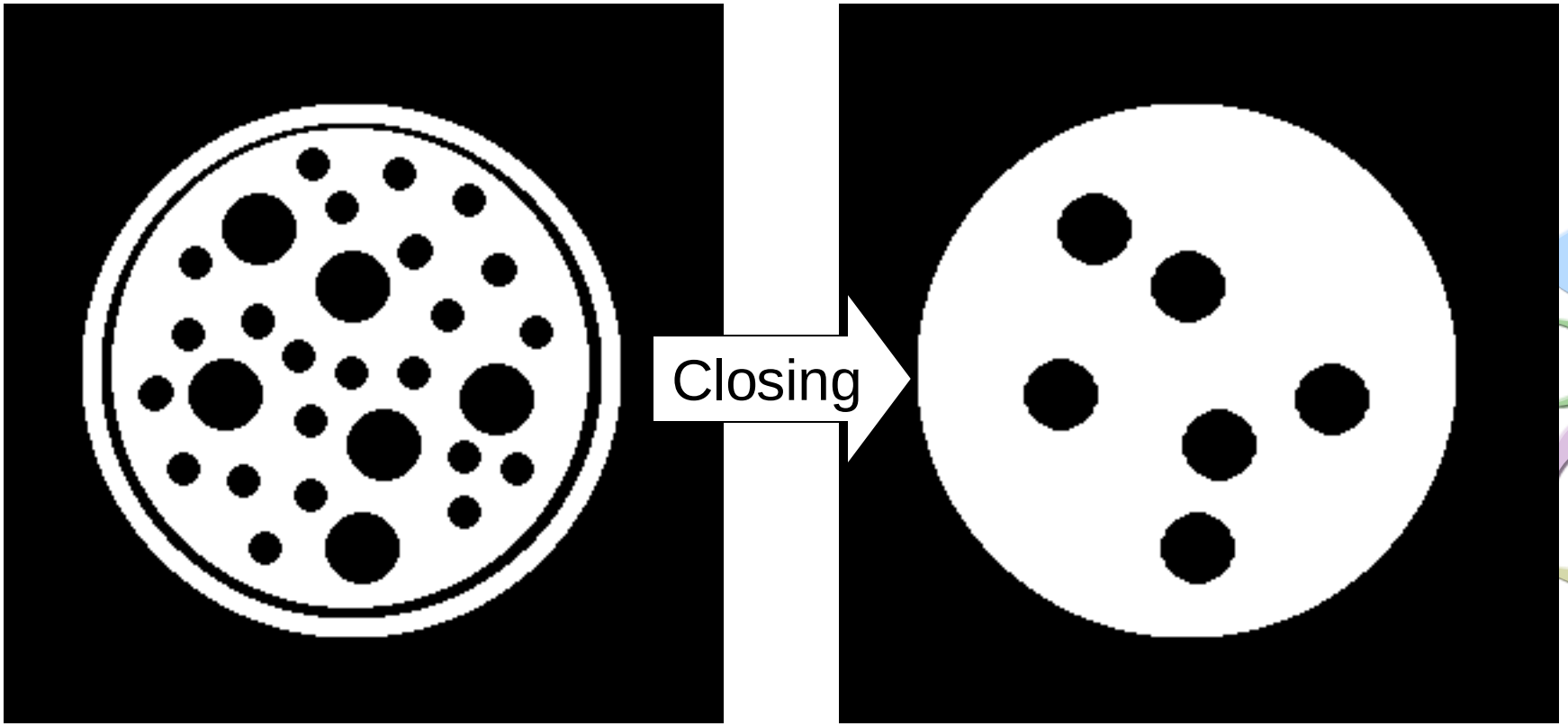
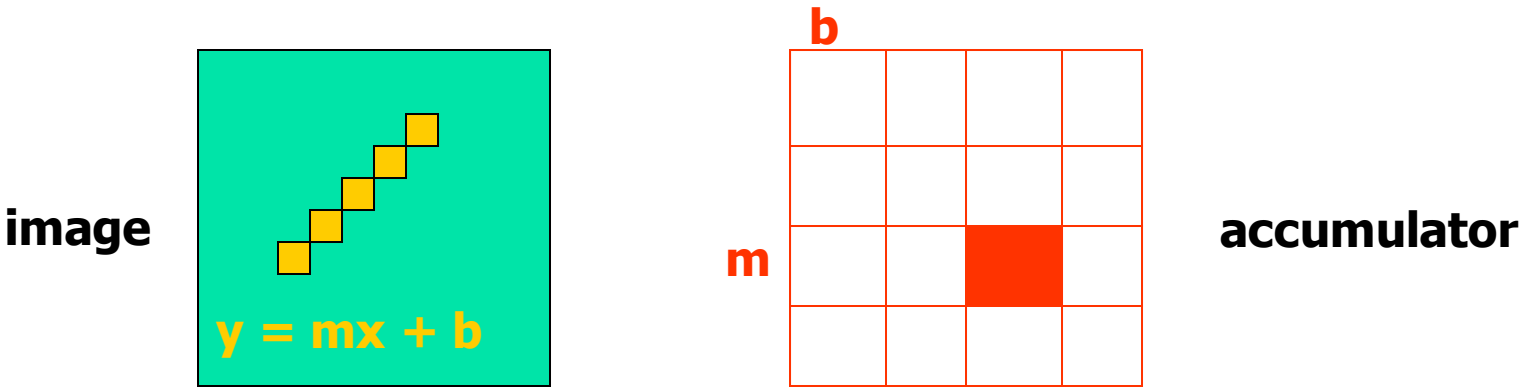


Image Segmentation
Hough Transform

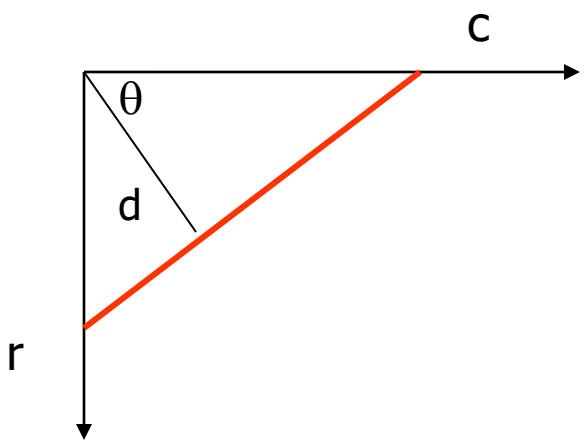
- The **Hough transform** is a method for detecting lines, curves or 2D shapes specified by a **parametric function**.

- If the parameters are **p1, p2, … pn**, then the Hough procedure uses an **n-dimensional accumulator** array in which it accumulates votes for the correct parameters of the lines or curves found on the image.



Hough Transform
Finding Straight Line Segments

- **y = mx + b is not suitable**
- **The equation generally used is: d = r sin θ + c cos θ**



d is the distance from the line to origin

θ is the angle the perpendicular makes with the column axis

Hough Transform
Procedure to Accumulate Lines

- Set accumulator array A to all zero.
  Set point list array PTLIST to all NIL.

- For each pixel (R,C) in the image {
  - compute gradient magnitude GMAG
  - if GMAG > gradient_threshold {
    - compute quantized tangent angle THETAQ
    - compute quantized distance to origin DQ
    - increment A(DQ,THETAQ)
    - update PTLIST(DQ,THETAQ) } }

Hough Transform
Example (Finding Straight Line Segments)

gray-tone image

0	0	0	100	100
0	0	0	100	100
0	0	0	100	100
100	100	100	100	100
100	100	100	100	100

DQ

-	-	3	3	-
-	-	3	3	-
3	3	3	3	-
3	3	3	3	-
-	-	-	-	-

THETAQ

-	-	0	0	-
-	-	0	0	-
90	90	40	20	-
90	90	90	40	-
-	-	-	-	-

Accumulator A

distance \ angle	0	10	20	30	40	...	90
360	-	-	-	-	-	...	-
.	-	-	-	-	-	...	-
6	-	-	-	-	-	...	-
3	4	-	1	-	2	...	5
0	-	-	-	-	-	...	-

PTLIST

360	-	-	-	-	-	-	-
.	-	-	-	-	-	-	-
6	-	-	-	-	-	-	-
3	*	-	*	-	*	-	*
0	-	-	-	-	-	-	-

(3,1)
(3,2)
(4,1)
(4,2)
(4,3)

(1,3)(1,4)(2,3)(2,4)

Hough Transform
Example (Finding Straight Line Segments)

How do you extract the line segments from the accumulators?

```
pick the bin of A with highest value V
while V > value_threshold {
   order the corresponding pointlist from PTLIST
   merge in high gradient neighbors within 10 degrees
   create line segment from final point list
   zero out that bin of A
   pick the bin of A with highest value V
}
```

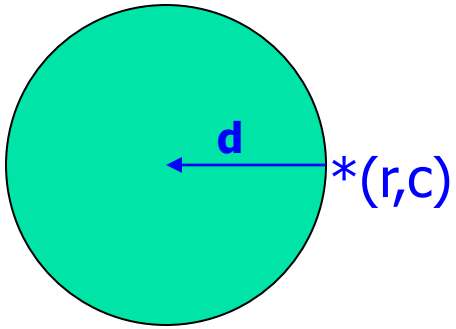
Hough Transform
Finding Circles

Equations:

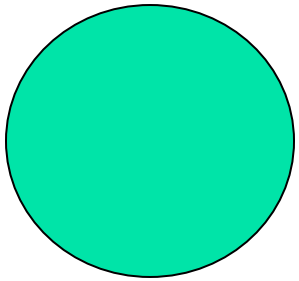
$$r = r0 + d \sin \theta$$
$$c = c0 + d \cos \theta$$

r, c, d are parameters

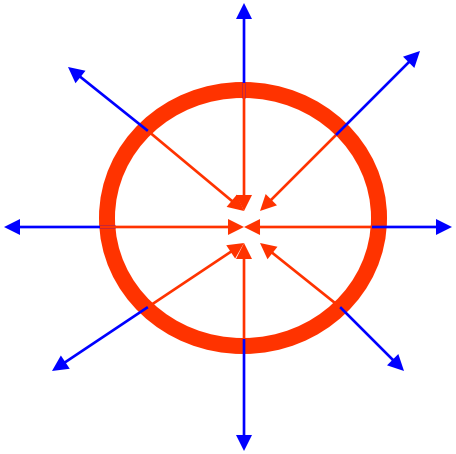
Main idea: The gradient vector at an edge pixel points to the center of the circle.



Hough Transform
Finding Circles (Why it works?)



Filled Circle:
Outer points of circle have gradient direction pointing to center.



Circular Ring:
Outer points gradient towards center.
Inner points gradient away from center.

The points in the away direction don't accumulate in one bin!

Hough Transform
Procedure to Accumulate Circles

- Set accumulator array A to all zero.
  Set point list array PTLIST to all NIL.

- For each pixel (R,C) in the image {
  For each possible value of D {
  - compute gradient magnitude GMAG
  - if GMAG > gradient_threshold {
  . Compute THETA(R,C,D)
  . R0 := R - D*cos(THETA)
  . C0 := C - D* sin(THETA)
  . increment A(R0,C0,D)
  . update PTLIST(R0,C0,D) }}

Image Segmentation
Active Contour Models (Snakes)

- **The contour is defined in the (*x*, *y*) plane of an image as a parametric curve**

$$v(s)=(x(s), y(s))$$

- **Contour is said to possess an energy (E_{snake}) which is defined as the sum of the three energy terms.**

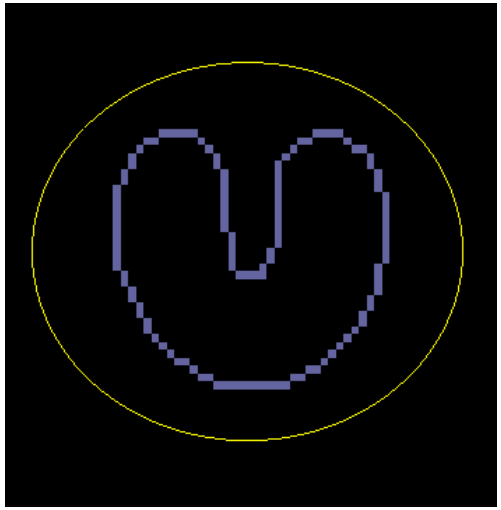
$$E_{snake} = E_{\text{int}ernal} + E_{external} + E_{constra\,\text{int}}$$

- **The energy terms are defined in a way such that the final position of the contour will have minimum energy (E_{min})**

- **Therefore our problem of detecting objects reduces to an energy minimisation problem.**

Image Segmentation
Active Contour Models (Snakes)

- **A higher level process or a user initializes any curve *close to the object boundary.***
- **The snake then starts *deforming* and moving towards the desired object boundary.**
- **In the end it completely "shrink-wraps" around the object.**



Object Detection and Tracking

Object Detection and Tracking

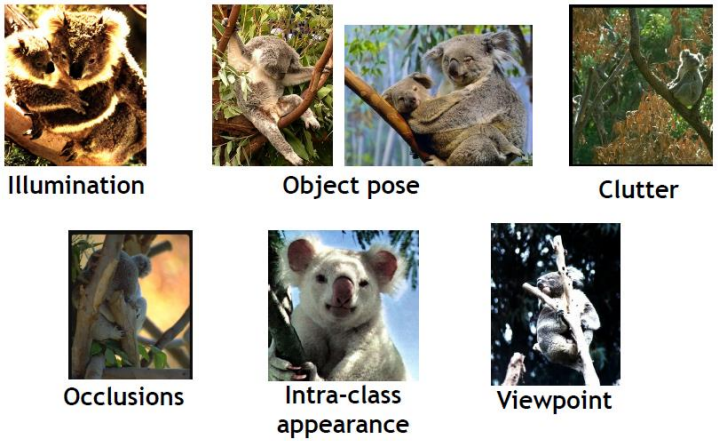
- **Object Detection**
  - **Detection via learning and classification**
- **Tracking scenarios**
  - **Follow a point**
  - **Follow a template**
  - **Follow a changing template**
  - **Follow all the elements of a moving object, fit a model to it.**
- **Motion estimation**
  - **Simple gradient based motion estimation**
  - **Patch-based motion (optic flow)**

Object Detection

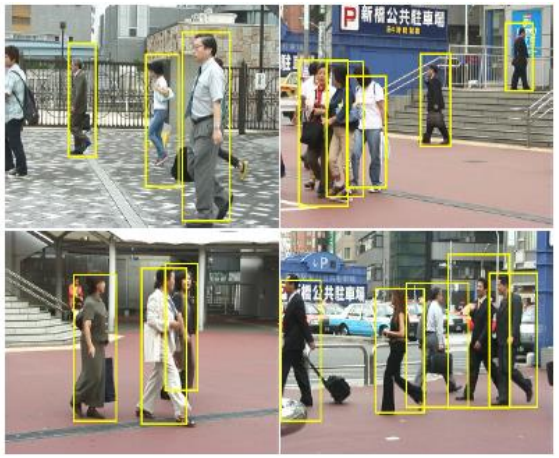
- **What is the goal of object detection?**
  - **Say yes/no as to whether an object present in image**
  - **And/or:**
    - **Determine pose of an object, e.g. for robot to grasp**
    - **Categorize all objects**
    - **Forced choice from pool of categories**
    - **Bounding box on object**
    - **Full segmentation**
    - **Build a model of an object category**

Object Detection

- **Object Detection Challenges:**
  - **Robustness in different conditions such as:**
    - **Illumination**
    - **Object pose**
    - **Clutter**
    - **Occlusions**
    - **Intra-class appearance**
    - **Viewpoint**



  - **Detection in Crowded Scenes**

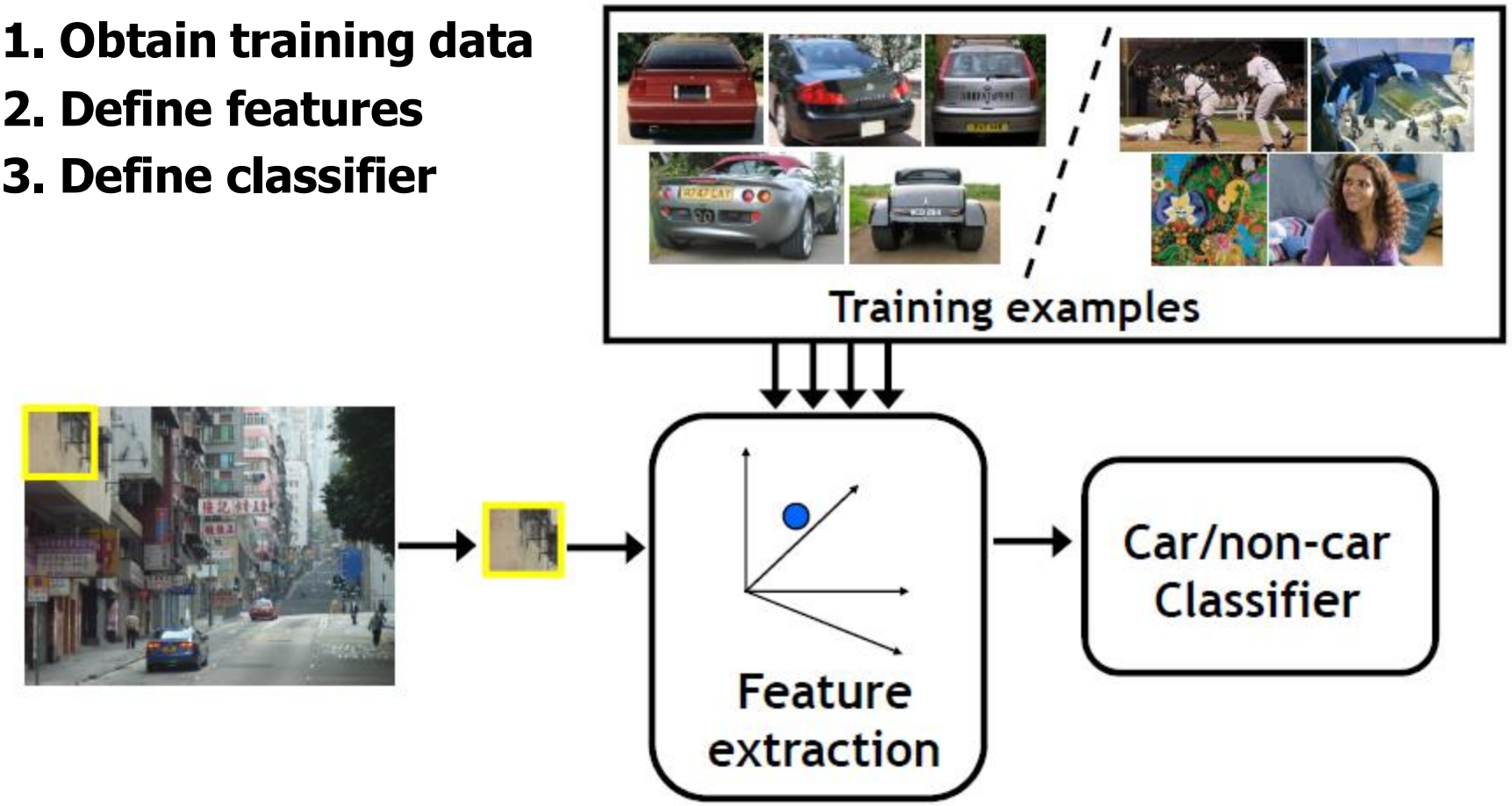


Object Detection
Via Learning and Classification

- **Basic component: a binary classifier**
  - **If object may be in a cluttered scene, slide a window around looking for it.**
  - **Consider all sub-windows in an image**
    - **Sample at multiple scales and positions (and orientations)**
  - **Make a decision per window:**
    - **"Does this contain object category X or not?"**

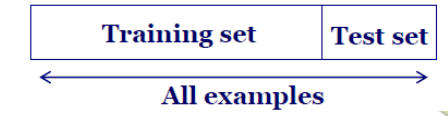
Object Detection
Via Learning and Classification

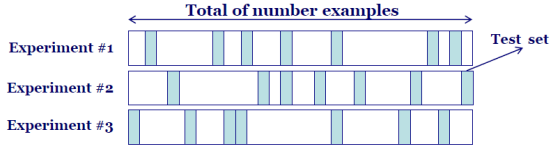
- **we need to:**
  - **1. Obtain training data**
  - **2. Define features**
  - **3. Define classifier**

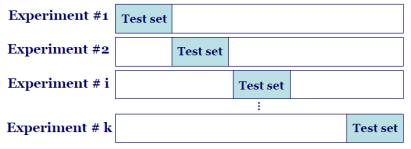


Object Detection
Data Partitioning(Three-way)

- **The data needs to be divided into three disjoint sets**
  - **Training set: a set of examples used for learning: to fit the parameters of the classifier**
  - **Validation set: a set of examples used to tune the parameters of a classifier**
  - **Test set: a set of examples used only to assess the performance of a fully-trained classifier**
- **The methods of data partitioning:**
  - **Holdout methods**
    - **Random Sampling**
    - **Resample with replacement Bootstrap**
    - **Multiple train-and-test experiment Bootstrap**
  - **Cross-validation**
    - **k-fold**
    - **Leave-one-out**







Object Detection
Feature Extraction

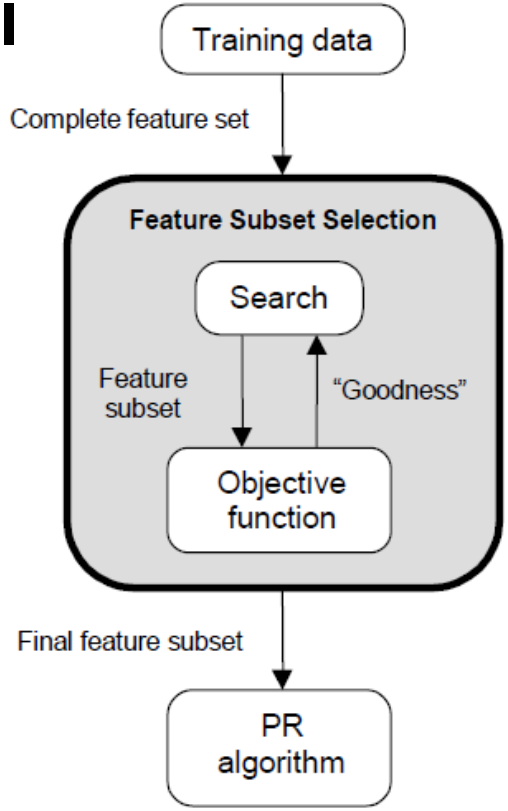
- **Simple holistic descriptions of image content**
  - **grayscale / color histogram**
    - **Pixel-based representations sensitive to small shifts, illumination and intra-class appearance variation**
- **Eigenvector-based (PCA) descriptors**
  - **Generate low dimensional representation of appearance with a linear subspace.**
- **Gradient-based representations**
  - **HoG Descriptor**
  - **SIFT Descriptor**
- **Texture pattern-based representations**
  - **LBP Descriptor**
  - **LPQ Descriptor**
  - **LTP Descriptor**
- **Other techniques**

Object Detection
Feature Selection

- **the classifier's performance usually will degrade for a large number of features.**
  - **Many explored domains have hundreds to tens of thousands of variables/features with many irrelevant and redundant one!**
  - **In domains with many features the underlying probability distribution can be very complex and very hard to estimate (e.g. dependencies between variables)!**
  - **Irrelevant and redundant features can "confuse" learners.**
  - **The required number of training samples (to achieve the same accuracy) grows exponentionally with the number of features!**
  - **In practice, the number of training examples is limited**
  - **Also computational resources is limited**
- **For higher recognition rate only a subset of the original features should be selected.**

Object Detection
Feature Selection

- **Feature selection is an optimization problem.**
  - **Search the space of possible feature subsets.**
  - **Pick the subset that is optimal or near-optimal with respect to a certain criterion.**

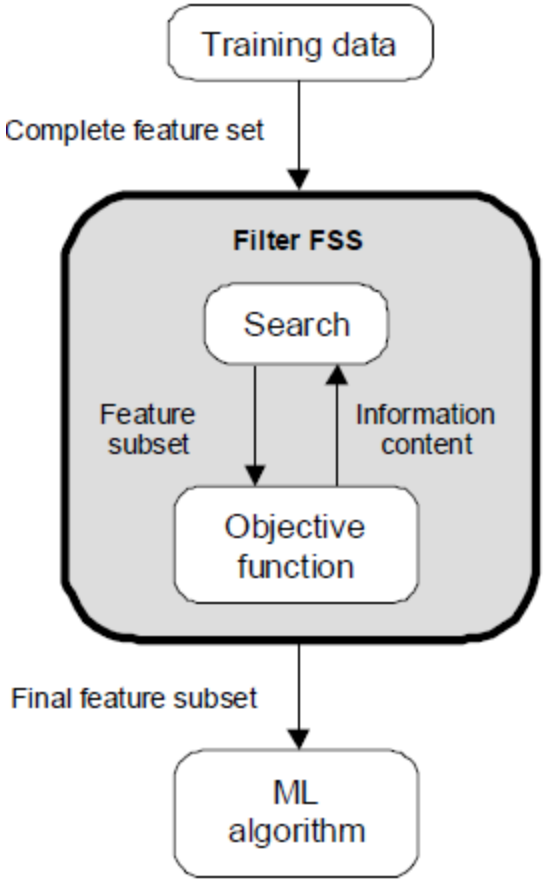


Search strategies
- **Optimum**
- **Heuristic**
- **Randomized**

Evaluation strategies
- **Filter methods**
- **Wrapper methods**

Object Detection
Feature Selection(Evaluation Strategies)

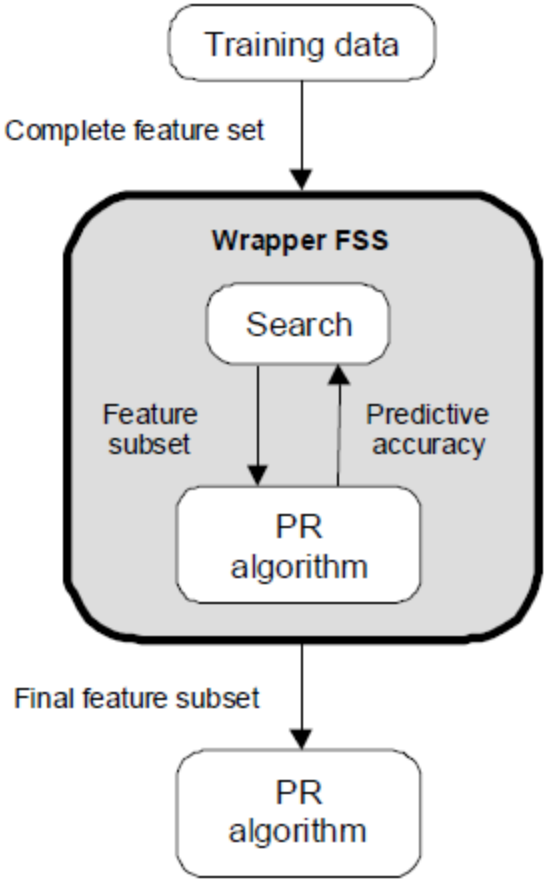
- **Filter Methods**
  - **Evaluation is independent of the classification algorithm.**
  - **The objective function evaluates feature subsets by their information content, typically interclass distance, statistical dependence or information-theoretic measures.**



- **Advantages**
  - **Fast execution: Filters generally involve a non-iterative computation on the dataset, which can execute much faster than a classifier training session**
  - **Generality: Since filters evaluate the intrinsic properties of the data, rather than their interactions with a particular classifier, their results exhibit more generality; the solution will be "good" for a large family of classifiers**
- **Disadvantages**
  - **Tendency to select large subsets: Filter objective functions are generally monotonic**

Object Detection
Feature Selection(Evaluation Strategies)

- **Wrapper Methods**
  - **Evaluation uses criteria related to the classification algorithm.**
  - **The objective function is a pattern classifier, which evaluates feature subsets by their predictive accuracy (recognition rate on test data) by statistical resampling or cross-validation.**
- **Advantages**
  - **Accuracy: wrappers generally have better recognition rates than filters since they tuned to the specific interactions between the classifier and the features.**
  - **Ability to generalize: wrappers have a mechanism to avoid over fitting, since they typically use cross-validation measures of predictive accuracy.**
- **Disadvantages**
  - **Slow execution**



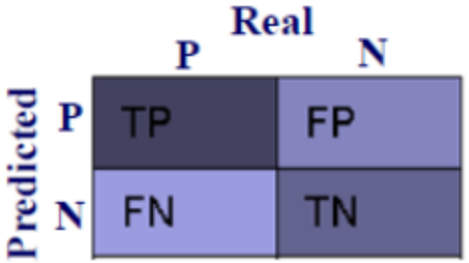
Object Detection
Feature Selection (Search Strategies)

- **Optimum**
  - **Select the best subset from $\binom{N}{M}$ possible combinations**
  - **Naïve Search**
    - **Sort the given N features in order of their probability of correct recognition and select the top M features**
  - **Statistical test**
    - **Employ two pair t-test and compare the significance of each feature against others. The most significant feature is then determined and repeat this for the next significant features.**
  - **PCA, ICA or LDA based feature reduction approaches**
- **Heuristic**
  - **Sequential forward/ backward selection (SFS, SBS or BDS)**
- **Randomized**
  - **Genetic Algorithm based feature selection**

Object Detection
Feature Classification

- **Classification**
  - predicts categorical class labels (discrete or nominal)
  - classifies data (constructs a model), based on the training set and the class labels, and uses it in classifying new data
  - The model is represented as classification rules, decision trees, probabilistic model, mathematical formulae and etc.

- **Classifier Performance Measures**
  - Performance measured using test set different from train set.
  - Accuracy: (TP+TN) / (TP+TN+FN+FP)
  - Specificity: TN / (FP+TN)
  - Sensitivity: TP / (FN+TP)
  - Index of Merit: (Specificity + Sensitivity) / 2 = (TP%+TN%) / 2
    - Also known as "percentage correct classifications"



Object Detection
Different Classifier

- **Parametric Classifiers**
  - **Bayesian classifier**
  - **Linear Discriminant Functions (such as LDA)**
- **Nonparametric Classifiers**
  - **The Nearest Neighbor (KNN)**
  - **Decision Trees (DT)**
  - **Artificial Neural Network (ANN)**
  - **Support Vector Machine (SVM)**
- **Combinational Classifiers**
  - **Ensemble or Bagging (bootstrap aggregating)**
    - **Uses distinct number of same classifier trained with different data**
  - **AdaBoost (adaptive boosting)**
    - **Uses base classifier and continuously adds different version of it in hierarchical fashion until desired low training error is achieved**

Object Detection
Sliding Window

- **Sliding window based object detection using global appearance descriptors is:**
  - **Simple detection protocol to implement**
  - **Good feature choices critical**
  - **Past successes for certain classes**
- **Limitations**
  - **High computational complexity**
  - **Not all objects are "box" shaped**
  - **Non-rigid, deformable objects not captured well with representations assuming a fixed 2d structure; or must assume fixed viewpoint**
  - **Objects with less-regular textures not captured well**
- **Models based on local features will alleviate some of these limitations**

Object Tracking
Motion and Flow

- **Why estimate visual motion?**
  - **Visual Motion can be annoying**
    - **Camera instabilities, jitter**
    - **Measure it; remove it (stabilize)**
  - **Visual Motion indicates dynamics in the scene**
    - **Moving objects, behavior**
    - **Track objects and analyze trajectories**
  - **Visual Motion reveals spatial layout**
    - **Motion parallax**

- **Motion estimation Techniques:**
  - **Simple gradient based motion estimation**
  - **Patch-based motion (optic flow)**

Object Tracking
Motion and Flow

Different Techniques for Motion Extraction are:

- **Feature-based methods**
  - **Extract visual features (corners, textured areas) and track them**
  - **Sparse motion fields, but possibly robust tracking**
  - **Suitable especially when image motion is large (>10 pixels)**
- **Direct methods**
  - **Directly recover image motion from spatio-temporal image brightness variations**
  - **Global motion parameters directly recovered without an intermediate feature motion calculation**
  - **Dense motion fields, but more sensitive to appearance variations**
  - **Suitable for video and when image motion is small (< 10 pixels)**

Object Tracking
Motion Estimation (Gradient-based)

- **Simple gradient based motion estimation:**

I J

$$d(x,y) = \begin{cases} 1 & if\ |I(x,y) - J(x,y)| > \varepsilon \\ 0 & otherwise \end{cases}$$

- **This method for each pixel (x,y) determines, it has changes in its intensity or not.**

Object Tracking
Motion Estimation (Gradient-based)

- **To reduce the effect of noise and other outliers, we can use patch based comparison:**

$$d(x,y)=\begin{cases}1 & if\ |\sum_{h,l\in\Omega_{x,y}} I(h,l)-\sum_{h,l\in\Omega_{x,y}} J(h,l)|>\varepsilon\\ 0 & otherwise\end{cases}$$

- **and use cumulative difference in successive frames:**

$$d(x,y)=\begin{cases}1 & if\ \sum_{k=1}^{N} a_k|I_t(x,y)-I_{t+k}(x,y)|>\varepsilon\\ 0 & otherwise\end{cases}$$

- **and also use both of them:**

$$d(x,y)=\begin{cases}1 & if\ \sum_{k=1}^{N} a_k|\sum_{h,l\in\Omega_{x,y}} I_t(h,l)-\sum_{h,l\in\Omega_{x,y}} I_{t+k}(h,l)|>\varepsilon\\ 0 & otherwise\end{cases}$$

Object Tracking
Motion Estimation (Optical Flow)

- **Based on Brightness Constancy Equation:**

$$J(x,y) \approx I\big(x + u(x,y), y + v(x,y)\big)$$

  - **Assume all change between frames is due to motion**

- **By linearizing (assuming small (u,v)) using Taylor series expansion we have:**

$$J(x,y) \approx I(x,y) + I_x(x,y).u(x,y) + I_y(x,y).v(x,y)$$

- **Based on brightness constancy, we should minimize:**

$$E(x,y) = (J(x,y) - I(x+u, y+v))^2$$
$$= (I_x(x,y).u(x,y) + I_y(x,y).v(x,y) + I(x,y) - I(x+u, y+v)^2$$
$$= (I_x.u + I_y.v + I_t)^2$$

Object Tracking
Motion Estimation (Optical Flow)

- **Gradient Constraint (or the Optical Flow Constraint):**

$$E(x,y) = (I_x.u + I_y.v + I_t)^2$$

- **Minimizing:** $\frac{\partial E(x,y)}{\partial u} = \frac{\partial E(x,y)}{\partial v} = 0$

$$I_x(I_x.u + I_y.v + I_t) = 0$$

$$I_y(I_x.u + I_y.v + I_t) = 0$$

- **In general** $I_x, I_y \neq 0$
- **Hence,** $I_x.u + I_y.v + I_t \approx 0$
- **We have:**

$$I_x = I * \begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix} \quad and \quad I_y = I * \begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix}$$

$$and \qquad I_t = I * \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} + J * \begin{bmatrix} -1 & -1 \\ -1 & -1 \end{bmatrix}$$

Object Tracking
Patch-based Motion Estimation

- **Assume a single velocity for all pixels within an image patch (Ω)**

$$E(x,y) = \sum_{x,y\in\Omega} (I_x(x,y).u + I_y(x,y).v + I_t)^2$$

- **Minimizing:** $(\sum_\Omega \nabla \mathrm{I} \nabla \mathrm{I}^{\mathrm{T}})\vec{U} = -2\sum_\Omega \nabla I I_t$

$$\begin{bmatrix} \sum_\Omega I_x^2 & \sum_\Omega I_x\ I_y \\ \sum_\Omega I_x\ I_y & \sum_\Omega I_y^2 \end{bmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = -2 \begin{pmatrix} \sum_\Omega I_x\ I_t \\ \sum_\Omega I_y\ I_t \end{pmatrix}$$

$$M\vec{U} = b$$

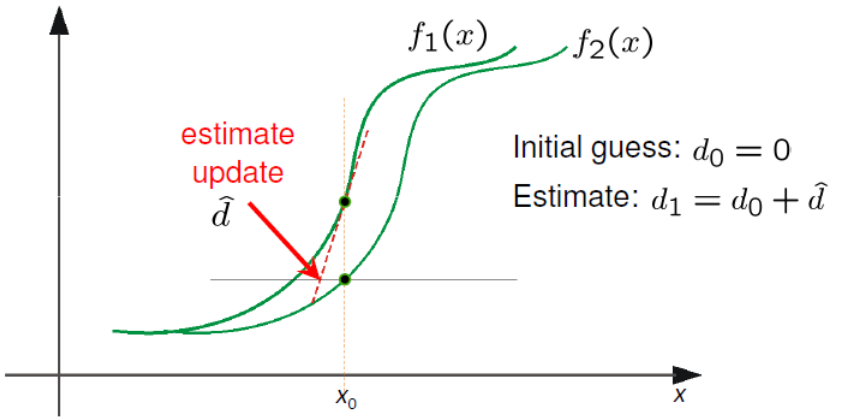
LHS: sum of the 2x2 outer product of the gradient vector

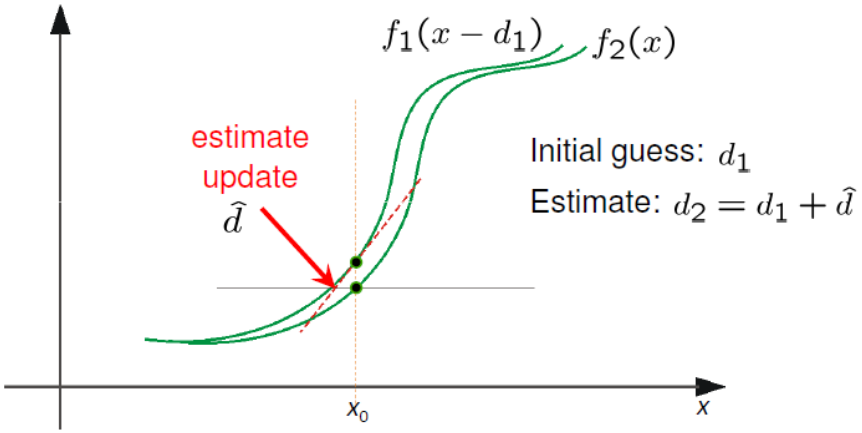
Object Tracking
Patch-based Motion Estimation

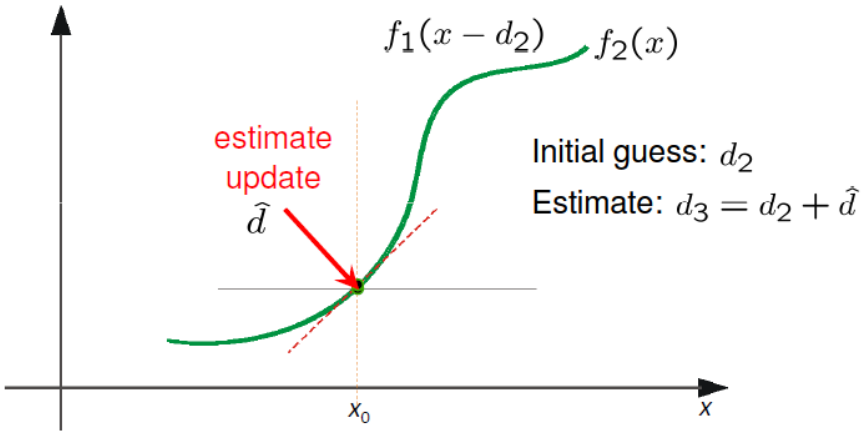
Algorithm:

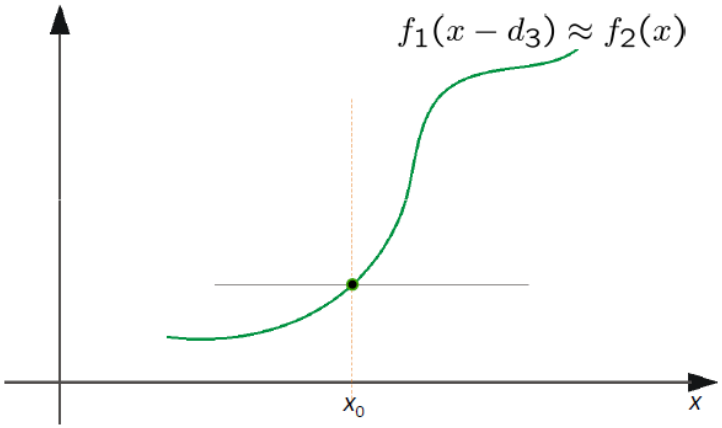
- **Select proper image patch (Ω)**
- **At each pixel compute U by solving $MU=b$ ($U = M^{-1}b$)**
- **M is singular if all gradient vectors point in the same direction**
  - **e.g., along an edge**
  - **of course, trivially singular if the summation is over a single pixel or there is no texture**
  - **i.e., only *normal flow* is available (aperture problem)**
  - **Corners and textured areas are OK**
- **Warp one image toward the other using the estimated flow field (easier said than done)**
- **Refine estimate by repeating the process**

Object Tracking
Patch-based Motion Estimation





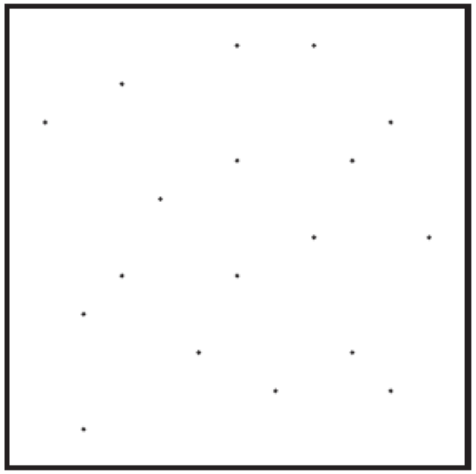




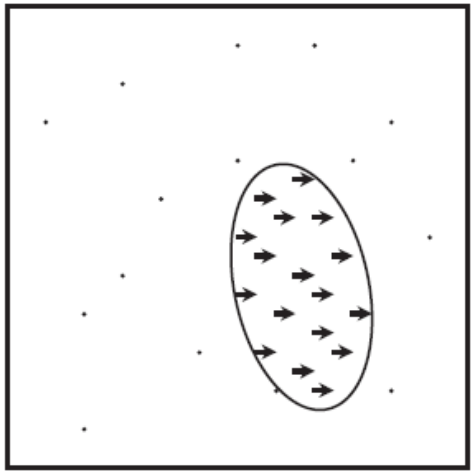
Object Tracking
Coherent Optical Flow

$$\theta = tg^{-1}\left(\frac{v}{u}\right)$$

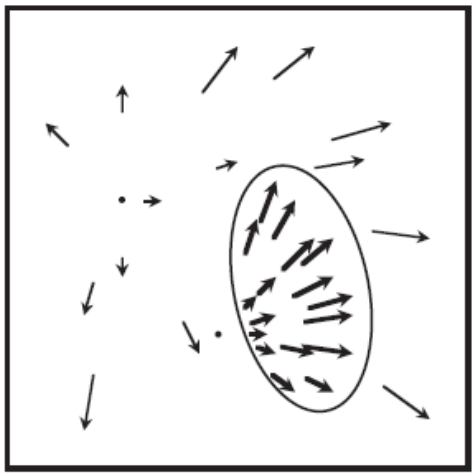
$$|\vec{V}| = \sqrt{u^2 + v^2}$$



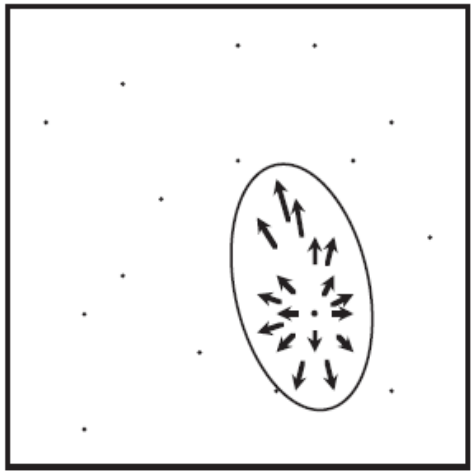
No Motion



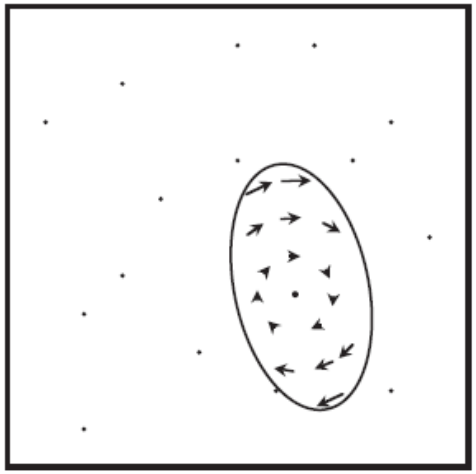
Object is moving to the right



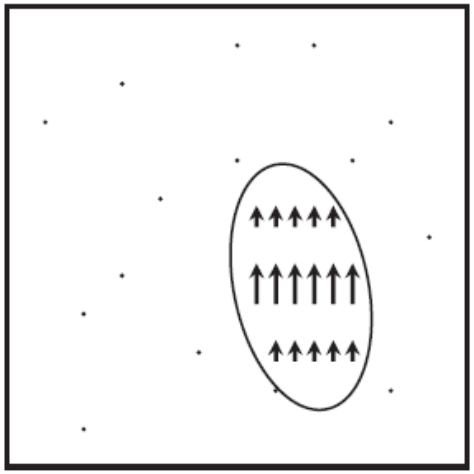
Camera is moving into the scene



Object is moving toward the camera



Object is rotating



Object is rotating about its x-axis

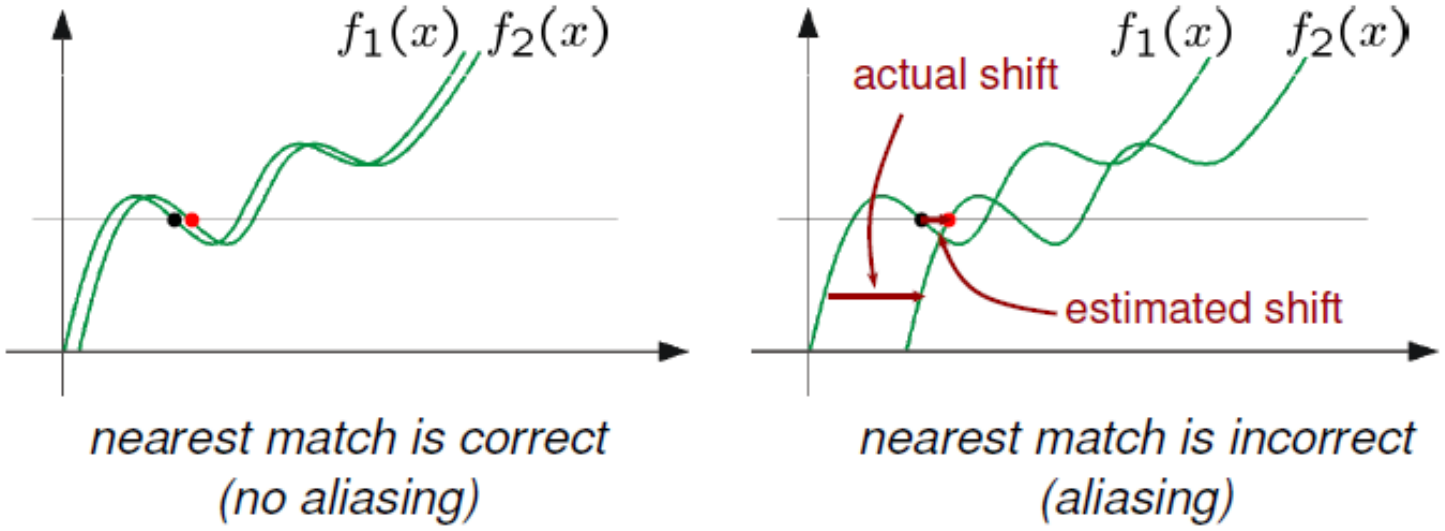
Object Tracking
Patch-based Motion Estimation

Some Implementation Issues:

- **Warping is not easy (ensure that errors in warping are smaller than the estimate refinement)**
- **Warp one image, take derivatives of the other so you don't need to re-compute the gradient after each iteration.**
- **Often it is useful to apply low-pass filter on the images before motion estimation (for better derivative estimation, and linear approximations to image intensity); also we can employ pyramid technique to overcome noise and aliasing problem.**

Aliasing Problem

- **Temporal aliasing causes ambiguities in optical flow because images can have many pixels with the same intensity.**
  - **i.e., how do we know which 'correspondence' is correct?**



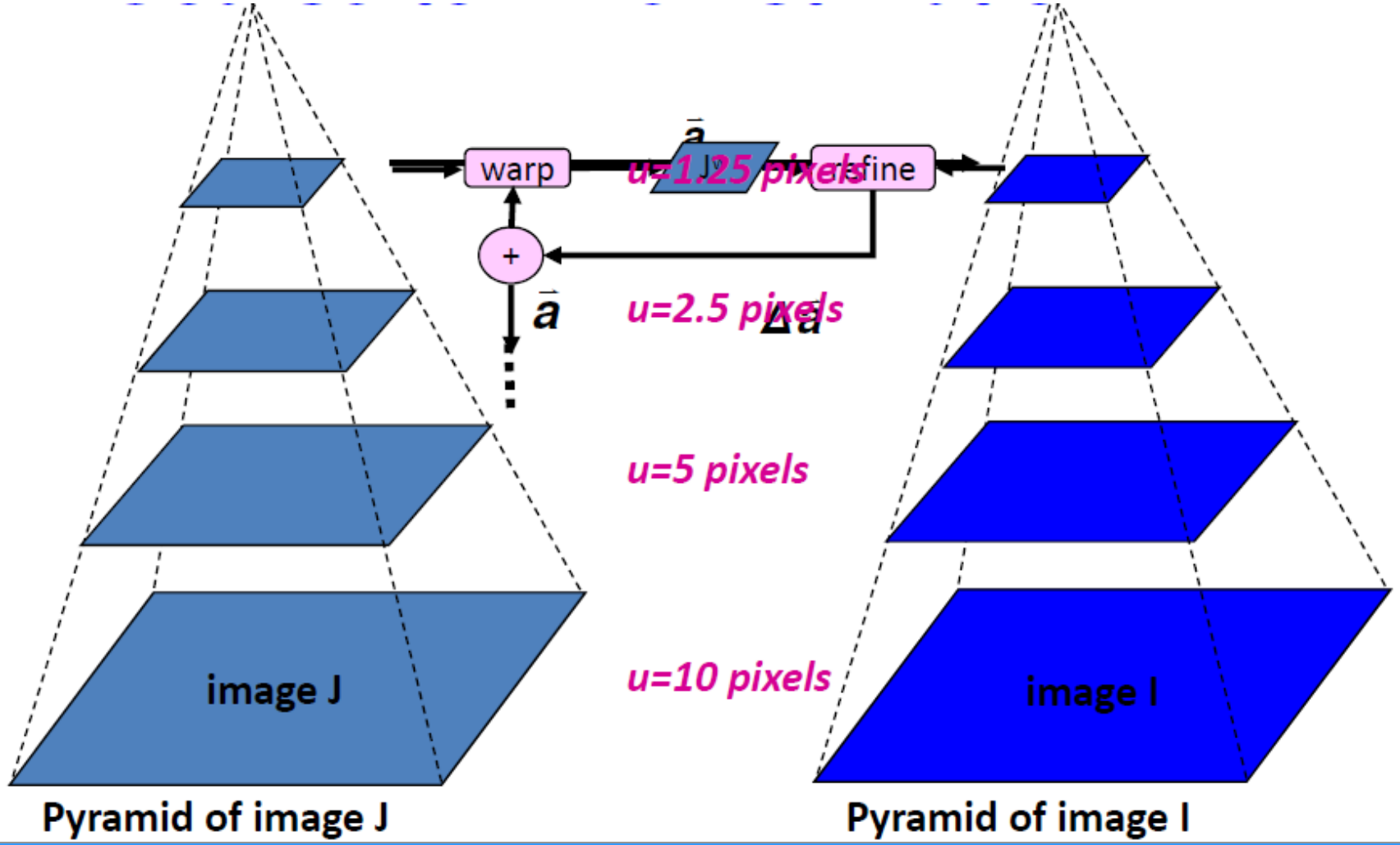
nearest match is correct (no aliasing)

nearest match is incorrect (aliasing)

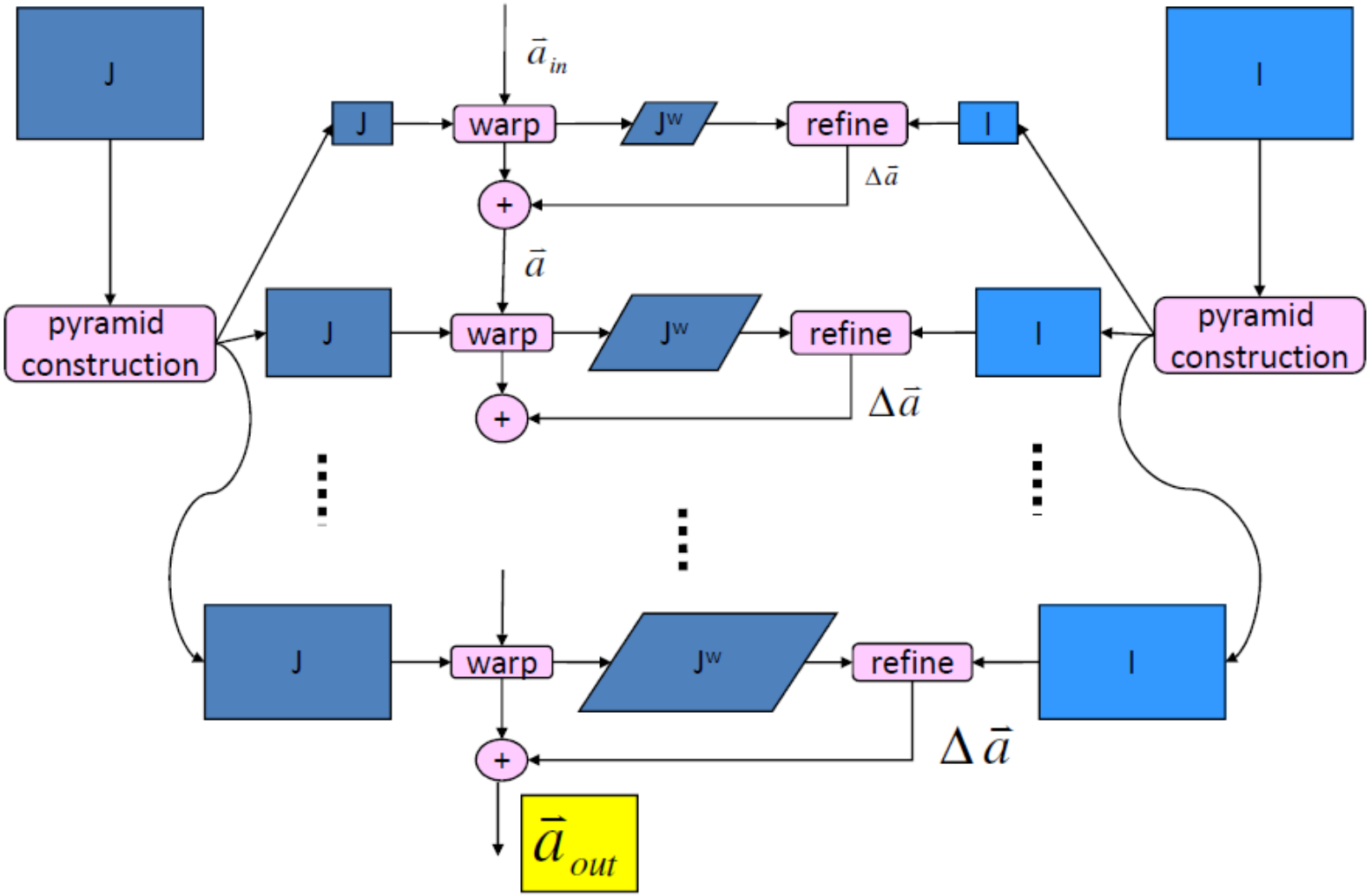
- **To overcome aliasing: coarse-to-fine estimation.**

Object Tracking
Patch-based Motion Estimation

Coarse-to-Fine Estimation



Object Tracking
Patch-based Motion Estimation



Object Tracking
Patch-based Motion Estimation

Limits of the gradient method:

- **Fails when intensity structure in window is poor**
- **Fails when the displacement is large (typical operating range is motion of 1 pixel)**
- **Linearization of brightness is suitable only for small displacements**
- **Also, brightness is not strictly constant in images**
  - **actually less problematic than it appears, since we can pre-filter images to make them look similar**

Object Tracking
Coherent Optical Flow

- **Gradient based motion estimation does not specify the velocity vector completely;**
  - **rather, it only provides the component in the direction of the brightest gradient**

- **To solve the problem completely, a smoothness constraint is introduced, that is, the velocity vector field changed slowly in a given neighborhood.**
  - **To this end we should minimize the error quantity as below:**

$$E(x,y) = (I_x.u + I_y.v + I_t)^2 + \lambda(u_x^2 + u_y^2 + v_x^2 + v_y^2)$$

  - where $u_x^2, u_y^2, v_x^2, v_y^2$ denote partial derivatives squared as error terms.
  - The first term represents a solution to gradient based motion equation , the second term is the smoothness criterion, and, λ is a Lagrange multiplier.

Object Tracking
Coherent Optical Flow

- **Using standard techniques [Horn and Schunk, 1981], this reduces to solving the differential equations and then simplified the estimation of motion vector as:**

$$u = \bar{u} - I_x \frac{P}{D}$$

$$v = \bar{v} - I_y \frac{P}{D}$$

  - Where $\bar{u}$ and $\bar{u}$ are mean value of u and v in the image and P and D are:

$$P = I_x \bar{u} + I_y \bar{v} + I_t$$

$$D = \lambda^2 + I_x^2 + I_y^2$$

- **The above estimation is repeated until $E(x, y) < \varepsilon$**
  - where ε is the maximum permitted error

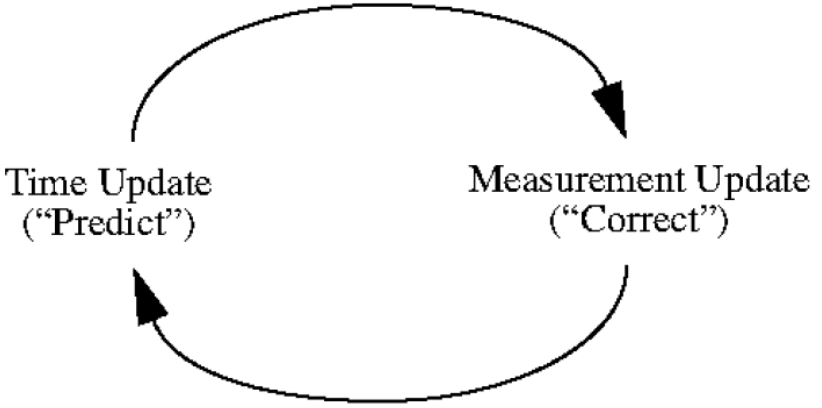
Object Tracking
Kalman filter

- **The Kalman Filter Key ideas:**
  - **Linear models interact uniquely well with Gaussian noise –**
    - make the prior Gaussian, everything else Gaussian and the calculations are easy
  - **Gaussians are really easy to represent**
    - once you know the mean and covariance, you're done

- **Tracking scenarios**
  - Make a measurement starting in the 0_{th} frame
  - Then: assume you have an estimate at the i *th* frame, after the measurement step.
  - Show that you can do prediction for the i+1*th* frame, and measurement for the i+1*th* frame.

Object Tracking
Kalman filter

- **Kalman filter graphical model and corresponding factorized joint probability**



- **Dynamic Model**

$$x_i \sim N(d_i x_{i-1}, \sigma_{d_i}^2)$$
$$y_i \sim N(m_i x_i, \sigma_{m_i}^2)$$

- **Notation**
  - $\bar{x}_i^-$ *as predicted mean of* $P(X_i|y_0, \ldots, y_{i-1})$
  - $\bar{x}_i^+$ *as corrected mean of* $P(X_i|y_0, \ldots, y_i)$
  - $\bar{\sigma}_i^-$ *as predicted standard deviation of* $P(X_i|y_0, \ldots, y_{i-1})$
  - $\bar{\sigma}_i^+$ *ascorrectedstandard deviation of* $P(X_i|y_0, \ldots, y_i)$
  - $\bar{x}_0^-$ *and* $\bar{\sigma}_0^-$ *are assumed to be known*

Object Tracking
Kalman filter

- **Time Update (Prediction) Equations:**

$$\overline{x}_i^- = d_i \overline{x}_{i-1}^+$$

$$\overline{\sigma}_i^- = \sqrt{\sigma_{d_i}^2 + (d_i \overline{\sigma}_{i-1}^+)^2}$$

- **Measurement Update (Correction) Equations:**

$$\overline{x}_i^+ = \frac{\overline{x}_i^- \sigma_{m_i}^2 + m_i y_i (\overline{\sigma}_i^-)^2}{\sigma_{m_i}^2 + m_i^2 (\overline{\sigma}_i^-)^2}$$

$$\overline{\sigma}_i^+ = \sqrt{\frac{\sigma_{m_i}^2 (\overline{\sigma}_i^-)^2}{\sigma_{m_i}^2 + m_i^2 (\overline{\sigma}_i^-)^2}}$$

Object Tracking
Kalman filter (Example)

Kalman filter model

$$d_i = 1,\ m_i = 1,\ \sigma_{d_i} = 0,\ \sigma_{m_i} = 1$$

$$x_i \sim N(d_i x_{i-1}, \sigma_{d_i})$$

$$y_i \sim N(m_i x_i, \sigma_{m_i})$$

Initial conditions

$$\overline{x}_0^- = 0 \quad \sigma_0^- = \infty$$

$\overline{x}_0^-$ and σ_0^- are known

Prediction

$$\overline{x}_i^- = d_i \overline{x}_{i-1}^+$$

$$\sigma_i^- = \sqrt{\sigma_{d_i}^2 + (d_i \sigma_{i-1}^+)^2}$$

Correction

$$x_i^+ = \left(\frac{\overline{x}_i^- \sigma_{m_i}^2 + m_i y_i (\sigma_i^-)^2}{\sigma_{m_i}^2 + m_i^2 (\sigma_i^-)^2} \right)$$

$$\sigma_i^+ = \sqrt{\left(\frac{\sigma_{m_i}^2 (\sigma_i^-)^2}{(\sigma_{m_i}^2 + m_i^2 (\sigma_i^-)^2)} \right)}$$

Iteration	0	1	2
$\overline{x}_i^-$	0	y_0	$\frac{y_0 + y_1}{2}$
$\overline{x}_i^+$	y_0	$\frac{y_0 + y_1}{2}$	$\frac{y_0 + y_1 + y_2}{3}$
σ_i^-	∞	1	$\frac{1}{\sqrt{2}}$
σ_i^+	1	$\frac{1}{\sqrt{2}}$	$\frac{1}{\sqrt{3}}$

Object Tracking
Kalman filter (Example)

$$x_i \sim N(d_i x_{i-1}, \sigma_{d_i})$$

$$y_i \sim N(m_i x_i, \sigma_{m_i})$$

$\overline{x_0}$ and σ_0^- are known

Prediction

$$\overline{x_i^-} = d_i \overline{x}_{i-1}^+$$

$$\sigma_i^- = \sqrt{\sigma_{d_i}^2 + (d_i \sigma_{i-1}^+)^2}$$

Correction

$$x_i^+ = \left(\frac{\overline{x_i^-}\,\sigma_{m_i}^2 + m_i y_i (\sigma_i^-)^2}{\sigma_{m_i}^2 + m_i^2 (\sigma_i^-)^2} \right)$$

$$\sigma_i^+ = \sqrt{\left(\frac{\sigma_{m_i}^2 (\sigma_i^-)^2}{(\sigma_{m_i}^2 + m_i^2 (\sigma_i^-)^2)} \right)}$$

Kalman filter model

$$d_i = 1,\ m_i = 1,\ \sigma_{d_i} = 1,\ \sigma_{m_i} = 1$$

Initial conditions

$$\overline{x}_0^- = 0 \quad \sigma_0^- = \infty$$

Iteration	0	1	2
$\overline{x}_i^-$	0	y_0	$\frac{y_0 + 2y_1}{3}$
$\overline{x}_i^+$	y_0	$\frac{y_0 + 2y_1}{3}$	$\frac{y_0 + 2y_1 + 5y_2}{8}$
σ_i^-	∞	$\sqrt{2}$	$\sqrt{\frac{5}{3}}$
σ_i^+	1	$\sqrt{\frac{2}{3}}$	$\sqrt{\frac{5}{8}}$

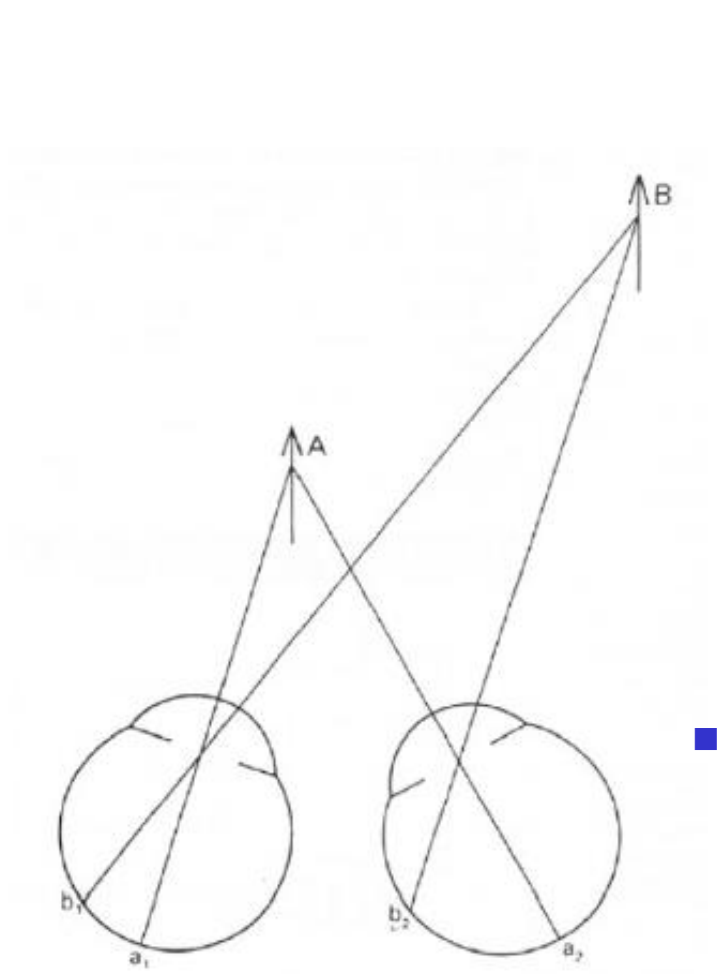
Stereo Vision

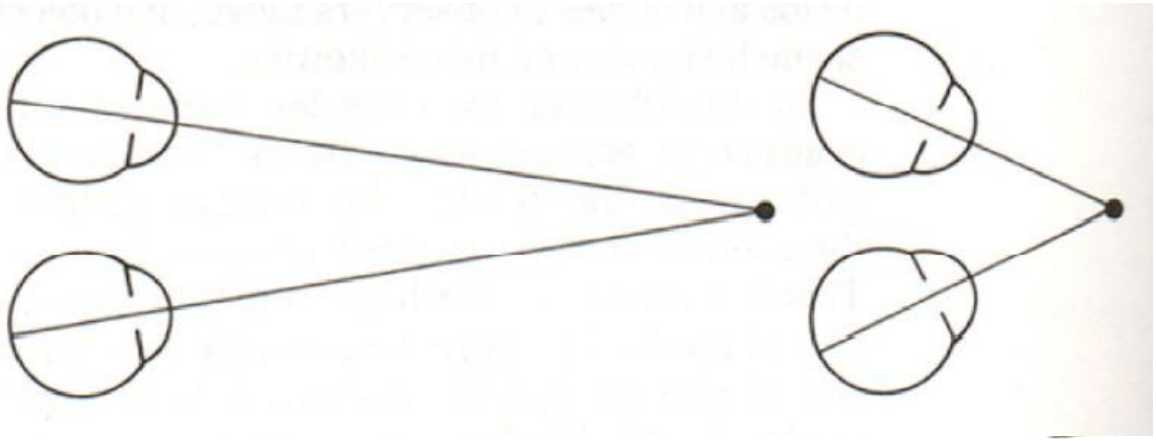
Stereo Vision

- **Contents:**
  - **Human stereopsis & stereograms**
  - **Epipolar geometry and the epipolar constraint**
    - **Case example with parallel optical axes**
    - **General case with calibrated cameras**
  - **Correspondence search**
  - **The Essential and the Fundamental Matrix**
  - **Multi-view stereo**

Stereo Vision

- **Human fixation, convergence**

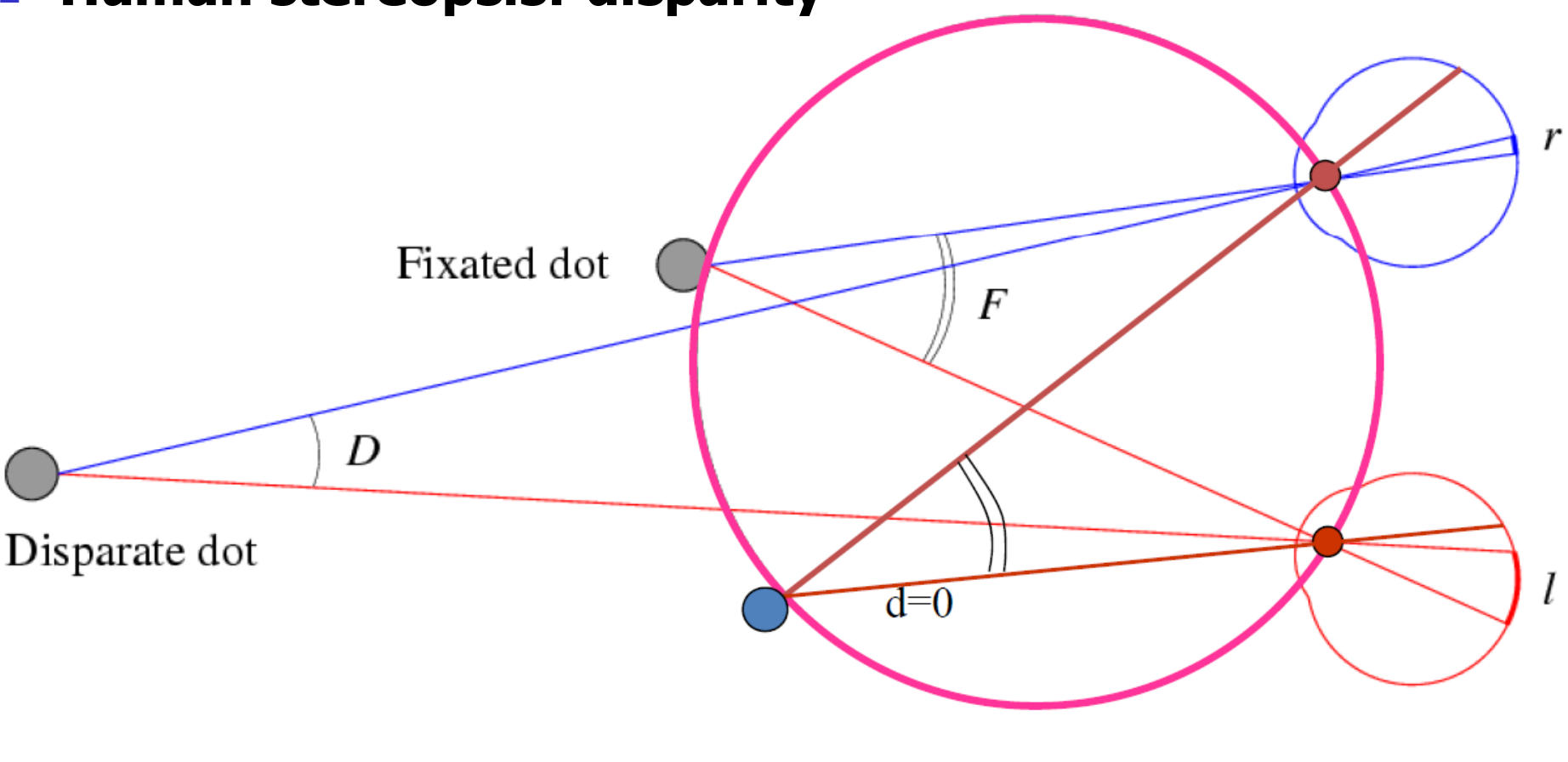




- **Human stereopsis disparity** occurs when eyes fixate on one object; others appear at different visual angles

Stereo Vision

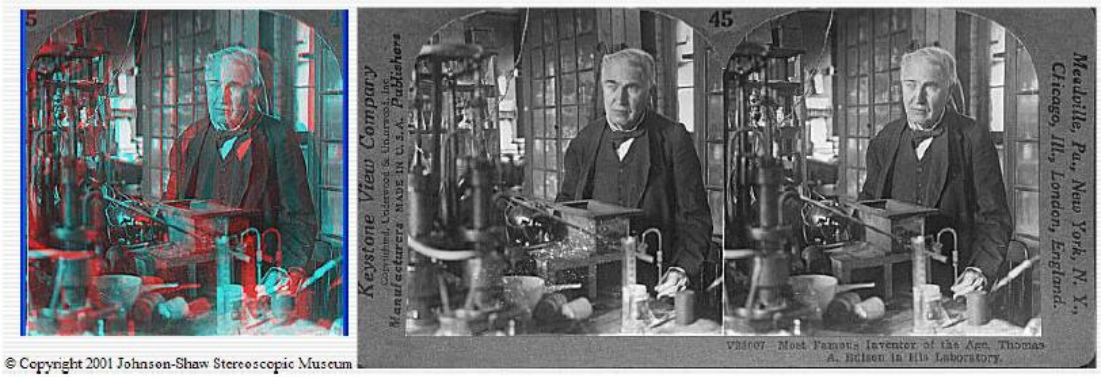
- **Human stereopsis: disparity**



Disparity: $d = r\text{-}l = D\text{-}F$.

Stereo Vision

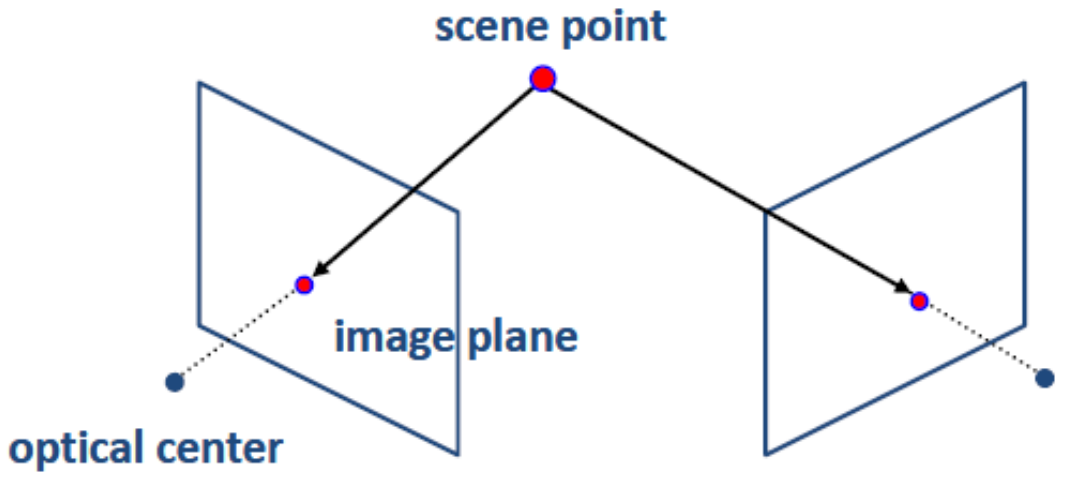
- **Stereo photography and stereo viewers**
  - **Take two pictures of the same subject from two slightly different viewpoints and display so that each eye sees only one of the images.**



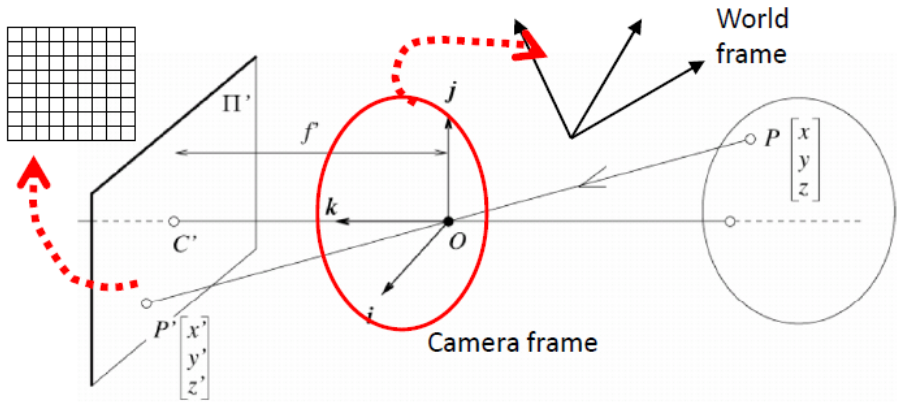
Stereo Vision
Depth with Stereo: Basic Idea

- **Basic Principle: Triangulation**
  - Gives reconstruction as intersection of two rays
  - Requires
    - camera pose (calibration)
    - point correspondence

- **Main Steps**
  - Calibrate cameras
  - Rectify images
  - Compute disparity
  - Estimate depth



Stereo Vision
Depth with Stereo: Camera calibration



Extrinsic parameters:
Camera frame ←→ Reference frame

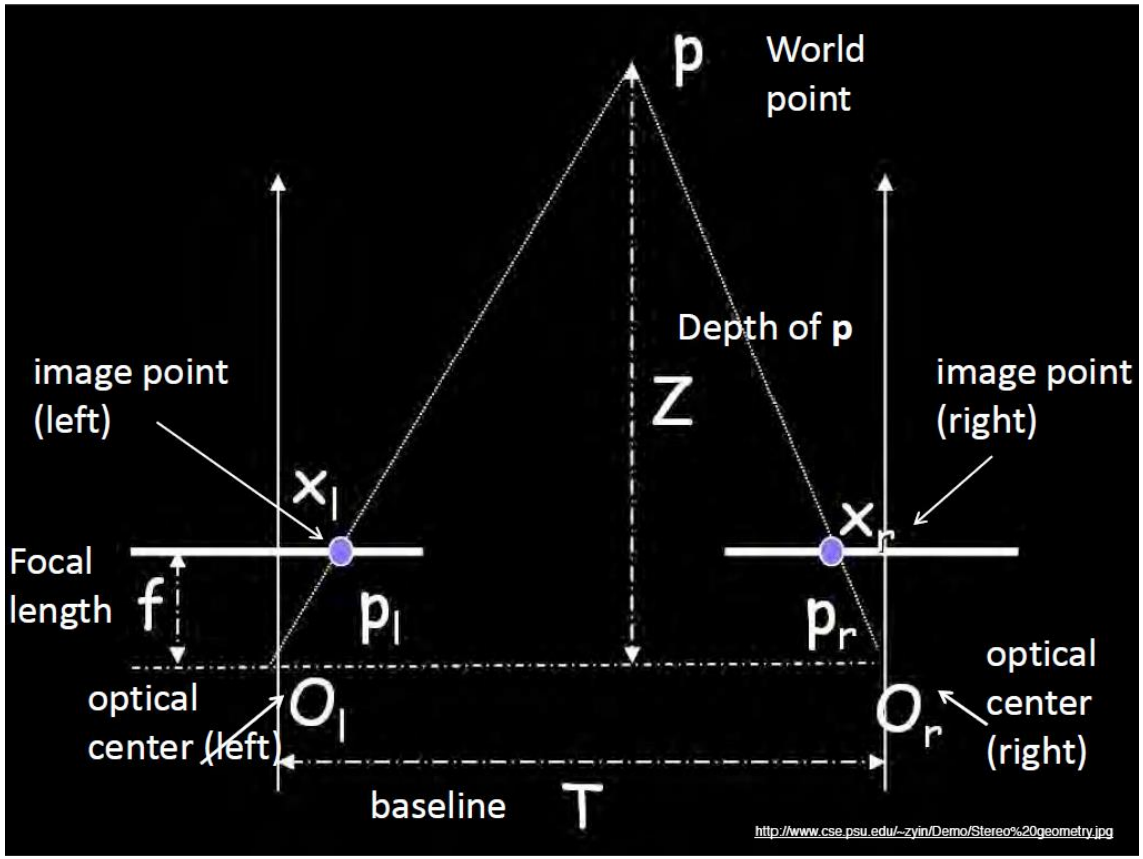
Intrinsic parameters:
Image coordinates relative to camera
←→ Pixel coordinates

- ***Extrinsic* parameters:**
  - **rotation matrix and translation vector**
- ***Intrinsic* parameters:**
  - **focal length, pixel sizes (mm), image center point, radial distortion parameters**

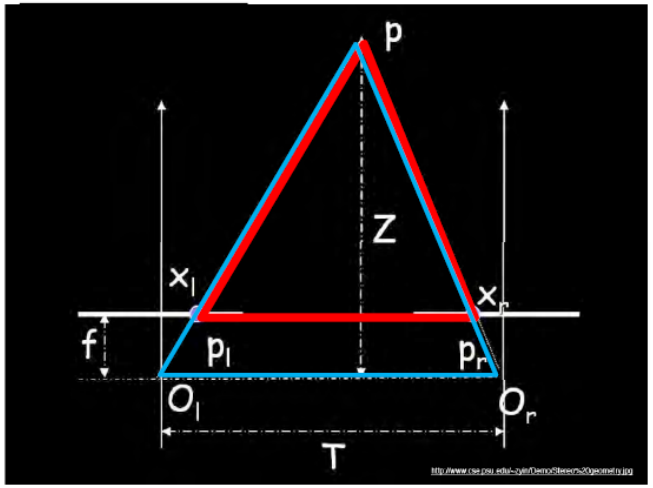
- **We'll assume for f now that these parameters are given and fixed.**

Stereo Vision
Depth with Stereo: Basic Idea

- **Geometry for a simple stereo system**
  - **Assume parallel optical axes, known camera parameters (i.e., calibrated cameras). We can use triangulate as below**



Stereo Vision
Depth with Stereo: Basic Idea



Similar triangles (p_l, P, p_r) and (O_l, P, O_r):

$$\frac{T + x_l - x_r}{Z - f} = \frac{T}{Z}$$

$$Z = f\frac{T}{x_r - x_l}$$

disparity

image I(x,y)

Disparity map D(x,y)

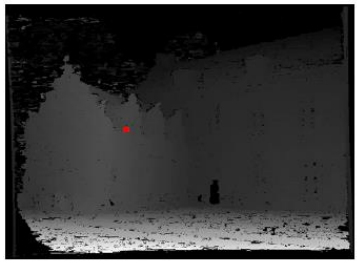
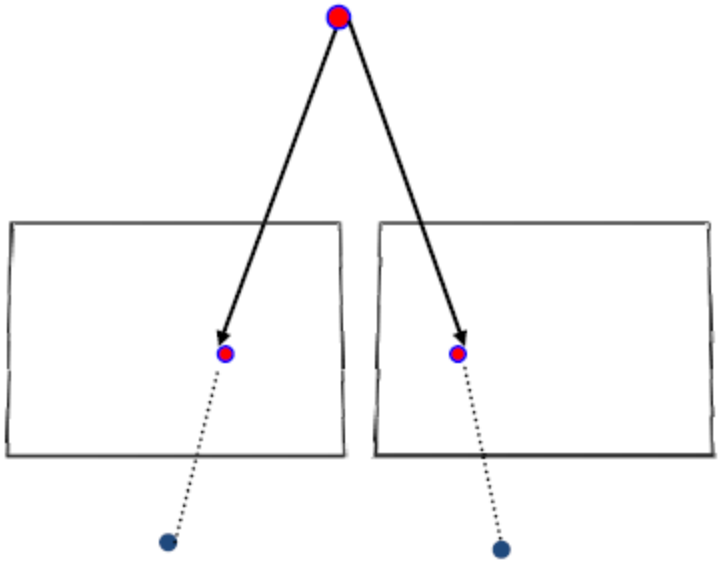


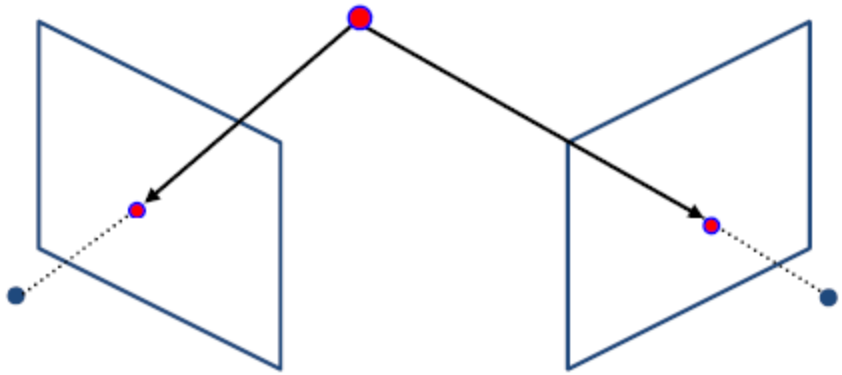
image I´(x´,y´)

(x´,y´)=(x+D(x,y), y)

Stereo Vision
Depth with Stereo: Basic Idea

- **General case, with calibrated cameras**
  - **The two cameras need not have parallel optical axes.**

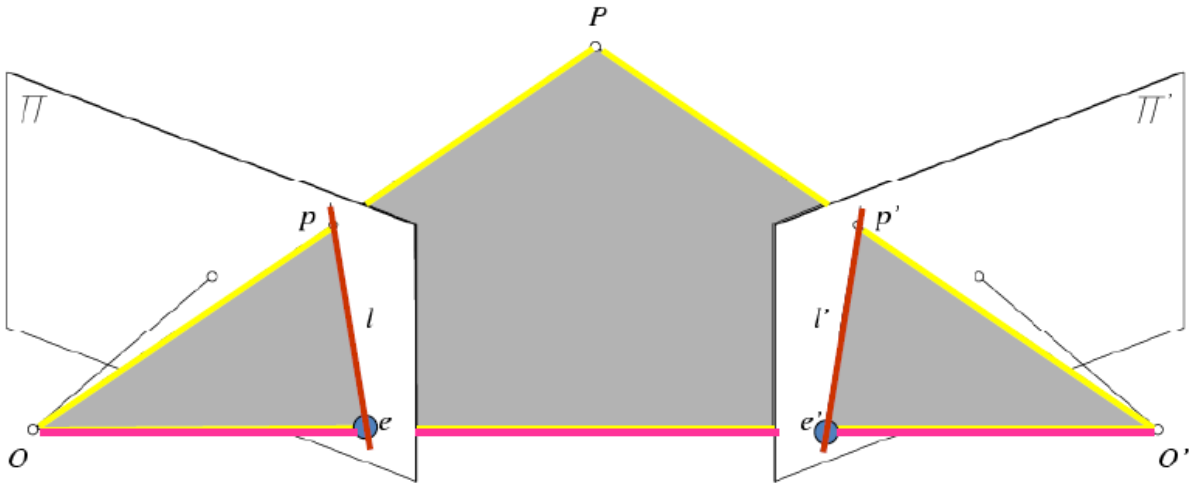




- **Stereo correspondence constraints**
  - **Geometry of two views allows us to constrain where the corresponding pixel for some image point in the first view must occur in the second view.**

Stereo Vision
Depth with Stereo: Basic Idea

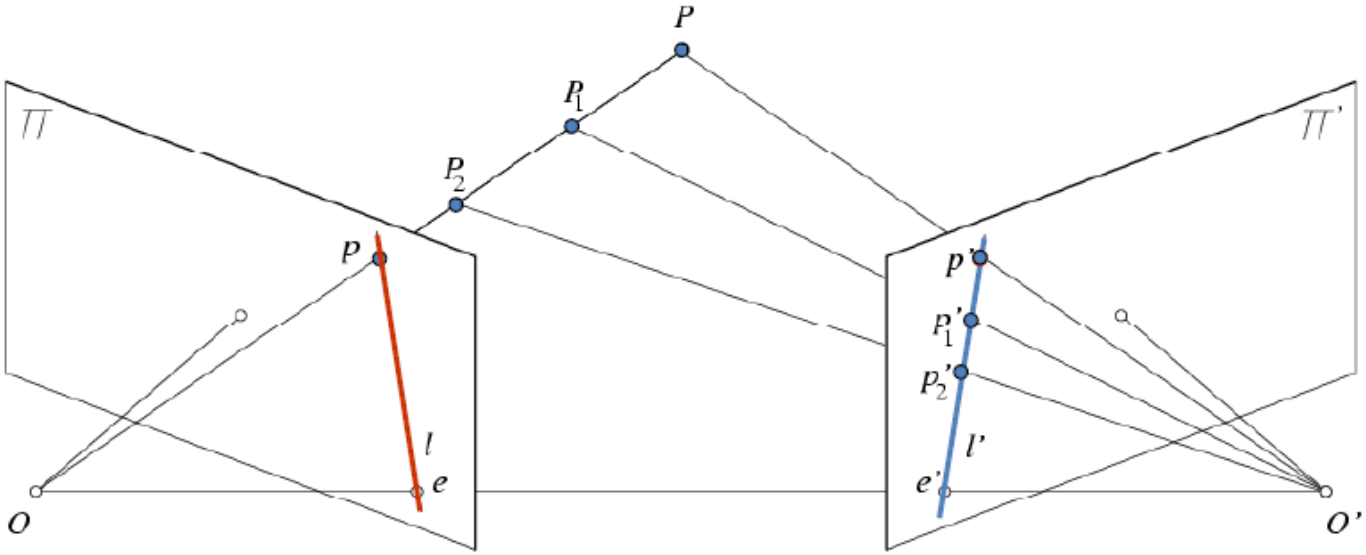
- **Epipolar constraint**: Why is this useful?
  - **Reduces correspondence problem to 1D search along *conjugate epipolar lines***



  - **Baseline**: line joining the camera centers
  - **Epipole**: point of intersection of baseline with the image plane
  - **Epipolar plane**: plane containing baseline and world point
  - **Epipolar line**: intersection of epipolar plane with the image plane

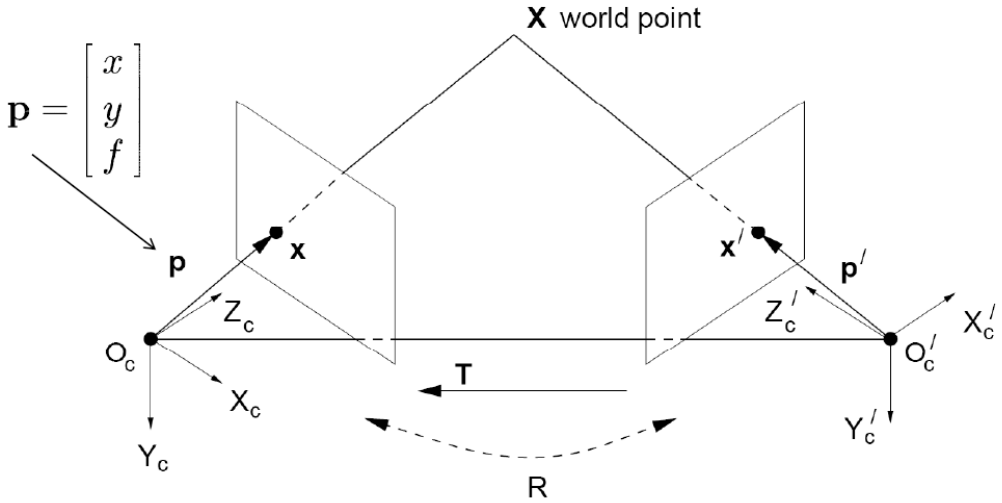
Stereo Vision
Depth with Stereo: Basic Idea

- **Epipolar constraint:**
  - All epipolar lines intersect at the epipole
  - An epipolar plane intersects the left and right image planes in epipolar lines
  - Potential matches for *p* have to lie on the corresponding epipolar line *l′*.
  - Potential matches for *p′* have to lie on the corresponding epipolar line *l*.



Stereo Vision
Depth with Stereo: Epipolar Constraints

- **Stereo geometry, with calibrated cameras**
- **If the rig is calibrated, we know :**
  - how to **rotate** and **translate** camera reference frame 1 to get to camera reference frame 2.
    - Rotation: 3 x 3 matrix; translation: 3 vector.



- Camera-centered coordinate systems are related by known rotation **R** and translation **T** as: $\mathbf{X}' = \mathbf{R}\mathbf{X} + \mathbf{T}$

Stereo Vision
Depth with Stereo: Epipolar Constraints

- **Stereo geometry, with calibrated cameras**
- Based on vector cross product, we have:
  - $\vec{a} \times \vec{b} = \vec{c}$ and $\vec{a} \times \vec{a} = 0$
  - takes two vectors and returns a third vector perpendicular to them.
- So we can write:

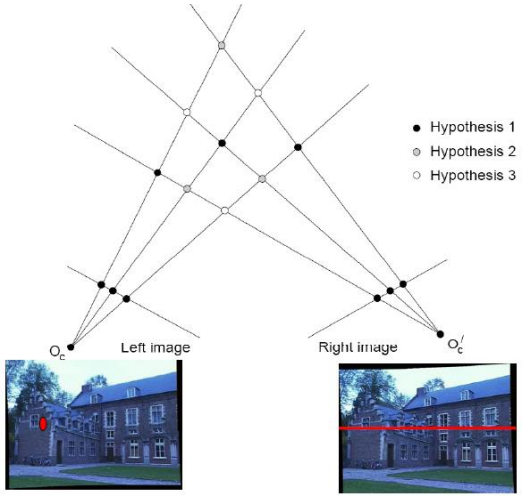
$$X'.(T \times X') = X'.\big(T \times (RX + T)\big) = X'.(T \times RX + T \times T)$$
$$= X'.(T \times RX) = X'.(T_x RX) = X'^{T}(T_x R)X = X'^{T}\boldsymbol{E}X = 0$$

  - which **E** is called the **essential matrix** which relates corresponding image points.
- **Epipolar constraint**: if we observe point **p** in one image, then its position **p'** in second image must satisfy this equation.

$$\boldsymbol{p'^{T} E\, p = 0}$$

Stereo Vision
Depth with Stereo: Correspondence problem

- **Multiple match hypotheses satisfy epipolar constraint, but which is correct?**
- **Beyond the hard constraint of epipolar geometry, there are "soft" constraints to help identify corresponding points:**
  - **Similarity**
  - **Uniqueness**
  - **Ordering**
  - **Disparity gradient**



- **To find matches in the image pair, we will assume**
  - **Most scene points visible from both views**
  - **Image regions for the matches are similar in appearance**

Stereo Vision
Depth with Stereo: Weak calibration

- **Want to estimate world geometry without requiring calibrated cameras**
  - **Archival videos**
  - **Photos from multiple unrelated users**
  - **Dynamic camera system**
- **Main idea:**
  - **Estimate epipolar geometry from a (redundant) set of point correspondences between two uncalibrated cameras**
- **For a given camera with known calibration parameters M_{int} we have:**

$$\overline{p} = M_{int}\ p$$

  - **p is camera coordinate and $\overline{p}$ is image coordinate**

Stereo Vision
Depth with Stereo: Weak calibration

- **So, for two cameras (left and right) we have:**

$$p_{(left)} = M_{int,left}^{-1}\,\overline{p}_{(left)}$$
$$p_{(right)} = M_{int,right}^{-1}\,\overline{p}_{(right)}$$

Camera coordinates — **Internal calibration matrices, one per camera** — **Image pixel coordinates**

- **From before, the essential matrix E :** $p_{(right)}^{T}\,E\,p_{(left)} = 0$

$$(M_{int,right}^{-1}\,\overline{p}_{(right)})^{T}\,E\,(M_{int,left}^{-1}\,\overline{p}_{(left)}) = 0$$
$$\overline{p}_{(right)}^{\,T}\,(M_{int,right}^{-T}\,E\,M_{int,left}^{-1})\,\overline{p}_{(left)} = 0$$
$$\overline{p}_{(right)}^{\,T}\,F\,\overline{p}_{(left)} = 0$$

  - **F is called as Fundamental matrix:** $F = M_{int,right}^{-T}\,E\,M_{int,left}^{-1}$

Stereo Vision
Depth with Stereo: Weak calibration

- **Fundamental matrix relates pixel coordinates in the two views**
- **More general form than essential matrix**
  - **we remove need to know intrinsic parameters**
- **If we estimate fundamental matrix from correspondences in pixel coordinates, can reconstruct epipolar geometry without intrinsic or extrinsic parameters**

- **Cameras are uncalibrated**
  - **we don't know E or left or right M_{int} matrices**
- **Estimate F from 8+ point correspondences.**
  - Each point correspondence generates one constraint on F

Stereo Vision
Depth with Stereo: Weak calibration

- **Estimate F from 8+ point correspondences.**
  - Each point correspondence generates one constraint on F

$$\bar{p}_{(right)}{}^{T} F\, \bar{p}_{(left)} = 0$$

$$\begin{bmatrix} u' & v' & 1 \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = 0$$

- **Collect n of these constraints:**

$$\begin{bmatrix} u'_1u_1 & u'_1v_1 & u'_1 & v'_1u_1 & v'_1v_1 & v'_1 & u_1 & v_1 & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{bmatrix} = 0$$

- **Solve for f , vector of parameters.**

Stereo Vision
Depth with Stereo: Weak calibration

- **So, where to start with uncalibrated cameras?**
  - Need to find fundamental matrix F **and** the correspondences (pairs of points: **(u',v')** ↔ **(u,v)**).

1) Find interest points in image (more on this later)

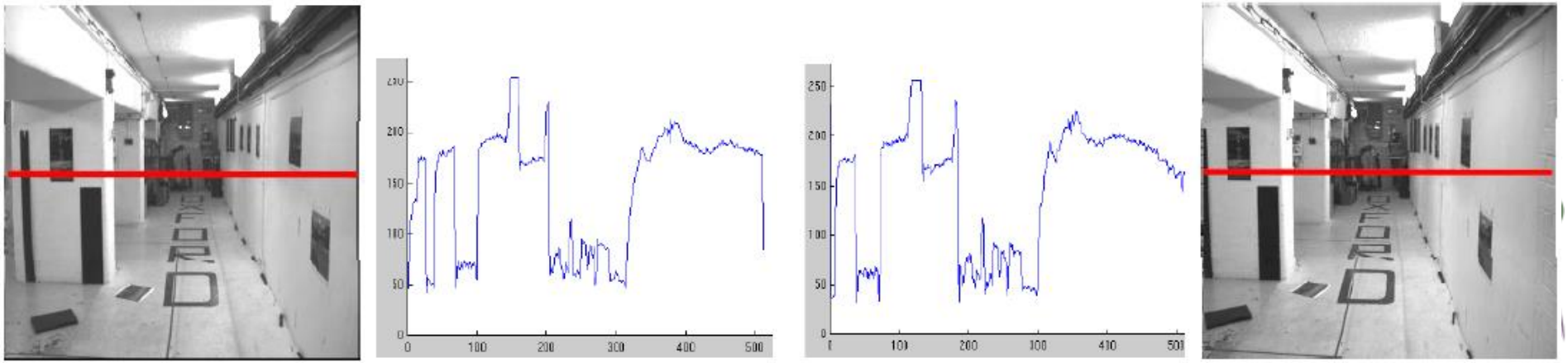
2) Compute correspondences

3) Compute epipolar geometry

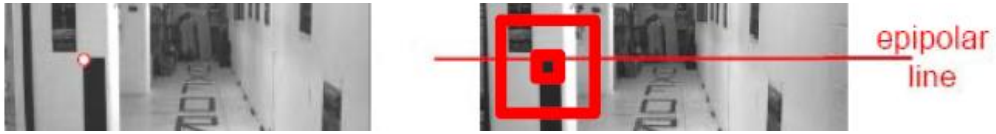
4) Refine

Stereo Vision
Depth with Stereo: Weak calibration

- **Correspondence problem**
  - Epipolar lines are corresponding, but also noise and ambiguity are existing.



  - Neighborhood of corresponding points are similar in intensity patterns.
    - To solve it, we can use correlation-based window matching
    - Want window large enough to have sufficient intensity variation, yet small enough to contain only pixels with about the same disparity.



Stereo Vision
Depth with Stereo: Weak calibration

- **Correspondence problem**
  - Restrict search to sparse set of detected features
  - Rather than pixel values (or lists of pixel values) use *feature descriptor* and an associated *feature distance*
  - Still narrow search further by epipolar geometry
- **Uniqueness**
  - For opaque objects, up to one match in right image for every point in left image
- **Ordering constraint**
  - Points on **same surface** (opaque object) will be in same order in both views
  - Won't always hold, e.g. consider transparent object, or an occluding surface

Learning and Classification (review)

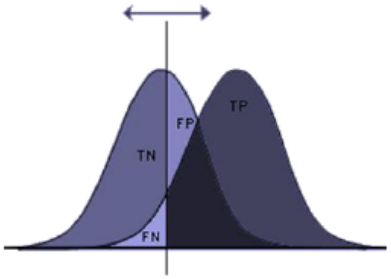
Introduction

- **Classification**
  - predicts categorical class labels (discrete or nominal)
  - classifies data (constructs a model), based on the training set and the class labels, and uses it in classifying new data

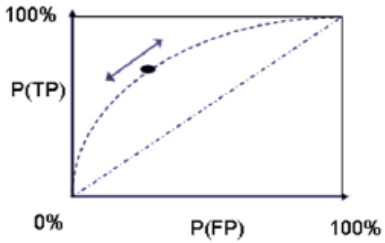
- **Model construction**
  - Each sample is assumed to belong to a predefined class, as determined by the class label
  - The set of samples used for model construction is called "training set"
  - The model is represented as classification rules, decision trees, probabilistic model, mathematical formulae and etc.

Evaluating Classification Methods

- **Performance**
  - classifier performance: predicting class label of new data
    - {true positive, true negative}, {false positive, false negative}



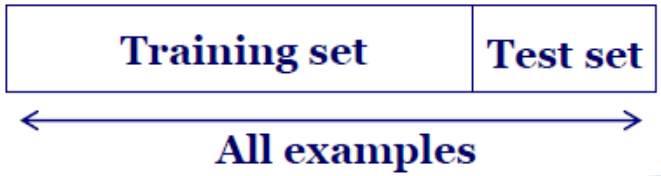
Predicted \ Real	P	N
P	TP	FP
N	FN	TN



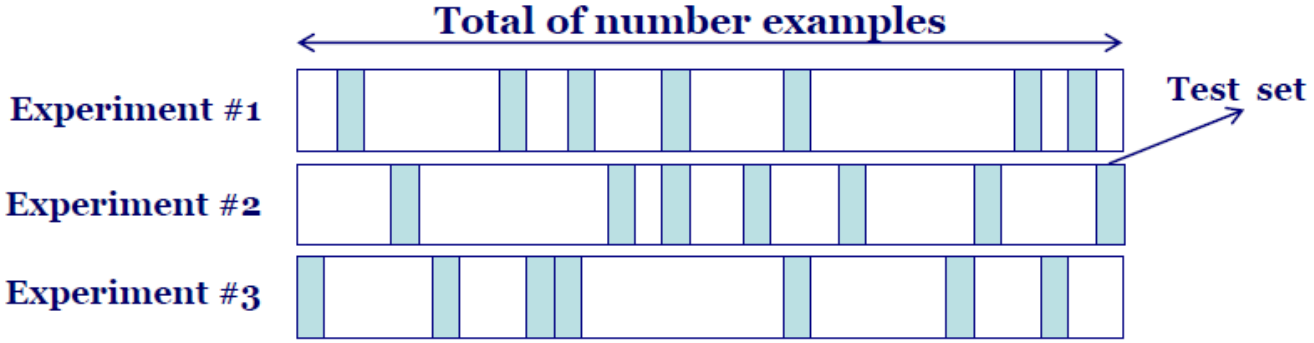
  - Accuracy: (TP+TN) / (TP+TN+FN+FP)
  - Specificity: TN / (FP+TN)
  - Sensitivity: TP / (FN+TP)
  - Index of Merit: (Specificity + Sensitivity) / 2 = (TP%+TN%) / 2
    - Also known as "percentage correct classifications"
- **Time Complexity**
  - time to construct the model (training time)
  - time to use the model (classification time)

Data Partitioning (Holdout)

- **Holdout methods: Random Sampling**
  - data is randomly partitioned into two independent sets
  - Always size of train set is twice of test set
  - Assumption: data is uniformly distributed

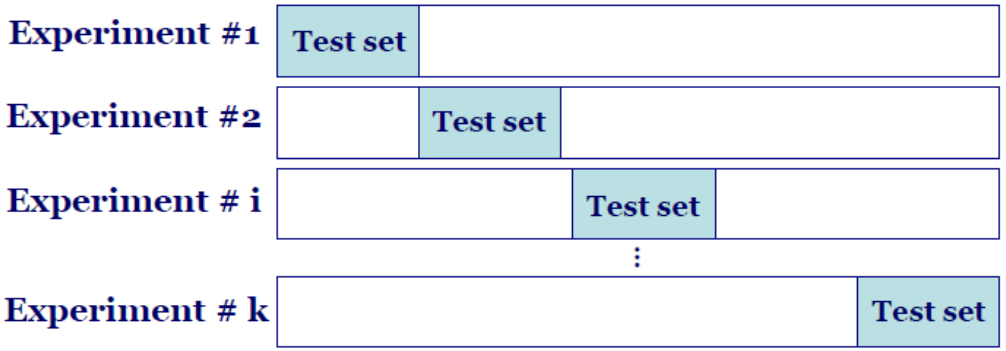


- **Holdout methods: Bootstrap**
  - Resample with replacement N samples of original data as training set.
  - Some numbers in the original sample may be included several times in the bootstrap sample (63.2% of samples are distinct)
- **Holdout methods: Multiple train-and-test experiment Bootstrap**



Data Partitioning(Cross-Validation)

- **Cross-validation (k-fold, where k = 10 is most popular)**
  - Randomly partition the data into k mutually exclusive subsets, each approximately equal size
  - At i^{th} iteration, use D_i as test set and others as training set
  - The mean of measures obtained in all iterations used as output performance measure



- **Leave-one-out**
  - k folds where *k* = #the number of samples, for small sized data
- **Stratified cross-validation**
  - folds are stratified so that class distributions in each fold is approximate the same as that in the initial data

Data Partitioning(Cross-Validation)

- How many folds are needed?
  - With a large number of folds
    - + The bias of the true error rate estimator will be small (the estimator will be very accurate)
    - -The variance of the true error rate estimator will be large
    - -The computational time will be very large as well (many experiments)
  - With small number of folds
    - + The number of experiments and, computation time are reduced
    - + The variance of the estimator will be small
    - -The bias of the estimator will be large(conservative or higher than the true error rate)
  - In practice, the choice of the number of folds depends on the size of the dataset
    - For large datasets, even 3-Fold Cross Validation will be quite accurate
    - For very sparse datasets, we may have to use leave-one-out in order to train on as many examples as possible

Data Partitioning(Three-way)

- If **model selection** and **true error estimates** are to be computed simultaneously, the data needs to be divided into three disjoint sets
  - **Training set:** a set of examples used for learning: to fit the parameters of the classifier
  - **Validation set:** a set of examples used to tune the parameters of a classifier
  - **Test set:** a set of examples used only to assess the performance of a fully-trained classifier
- Why separate test and validation sets?
  - The error rate estimate of the final model on validation data will be biased(smaller than the true error rate) since the validation set is used to select the final model
  - After assessing the final model with the test set, **YOU MUST NOT** tune the model any further

Parametric Classifiers

- Bayesian Decision Theory
  - Prior Probabilities
  - Class-Conditional Probabilities
  - Posterior Probabilities
  - Probability of Error
  - Conditional Risk
  - MinMax Criteria
  - Naive Bayes
- Discriminant Functions
- Probability Density Function Estimation

Classifiers based on Bayes Decision Theory

- **Bayesian Decision Theory is a fundamental statistical approach that quantifies the tradeoffs between various decisions using probabilities and costs that accompany such decisions.**
  - First, we will assume that all probabilities are known.
  - Then, we will study the cases where the probabilistic structure is not completely known.
- **Feature vectors are treated as random vectors.**
- **For input feature vector** $\underline{x} = [x_1, x_2, \ldots, x_l]^T$
- **Assign the pattern represented by feature vector** $\underline{x}$ **to the most probable of the available classes** $\omega_1, \omega_2, \ldots, \omega_M$

That is $\underline{x} \rightarrow \omega_i : P(\omega_i | \underline{x})$ **maximum**

Bayesian Classifiers: Prior Probability

- Prior probabilities reflect our knowledge of how likely each class type will appear before we actually see it.
- How can we choose $P(w_1)$ and $P(w_2)$?
  - Set $P(w_1) = P(w_2)$ if they are equiprobable (uniform priors).
  - May use different values depending on the problem.
- Assume there are no other types $P(\omega_1) + P(\omega_2) = 1$
  - (exclusivity and exhaustivity).
- How can we make a decision with only the prior information?
  - Decide $\begin{cases} \omega_1 & P(\omega_1) > P(\omega_2) \\ \omega_2 & Otherwise \end{cases}$
- What is the probability of error for this decision?
  - P(error) = min{$P(w_1)$, $P(w_2)$}

Bayesian Classifiers: Class-Conditional Probability

- **Let's try to improve the decision using the lightness measurement x.**
  - Let x be a continuous random variable.
  - Define $P(x|w_j)$ as the class-conditional probability density (probability of x given that the state of nature is w_j for j = 1, 2).
  - $P(x|w_1)$ and $P(x|w_2)$ describe the hypothetical class-conditional probability density functions for two Classes.
- **How can we make a decision with only the class-conditional probabilities?**
  - Decide $\begin{cases} \omega_1 & P(x|\omega_1) > P(x|\omega_2) \\ \omega_2 & Otherwise \end{cases}$
  - **Class-conditional is known as "Maximum Likelihood".**
  - **Looks good, but prior information are not used. It may degrade decision performance**

Bayesian Classifiers: Posterior Probability

- **Computation of a-posteriori probabilities**
  - **Assume known**
    - **a-priori probabilities** $P(\omega_1), P(\omega_2)\ldots, P(\omega_M)$

      and
    - **likelihood probabilities** $p(\underline{x}|\omega_i), i = 1,2,\ldots, M$

      (This is also known as the likelihood of $\underline{x}$ $w.r.\ to\ \omega_i$. **)**
  - **The Bayes rule (For M=2)**

$$p(\underline{x})P(\omega_i|\underline{x}) = p(\underline{x}|\omega_i)P(\omega_i) \Rightarrow$$

$$P(\omega_i|\underline{x}) = \frac{p(\underline{x}|\omega_i)P(\omega_i)}{p(\underline{x})} \approx \frac{\text{Likelihood}\times\text{Priori}}{\text{Evidence}}$$

where

$$p(\underline{x}) = \sum_{i=1}^{2} p(\underline{x}|\omega_i)P(\omega_i)$$

Bayesian Classifiers: Posterior Probability

- **The Bayes classification rule (for two classes $M=2$)**
  - **Given $\underline{x}$ classify it according to the rule**

$$If\ P(\omega_1|\underline{x}) > P(\omega_2|\underline{x})\ \ \underline{x} \rightarrow \omega_1$$

$$If\ P(\omega_2|\underline{x}) > P(\omega_1|\underline{x})\ \ \underline{x} \rightarrow \omega_2$$

  - **Equivalently: classify $\underline{x}$ according to the rule**

$$p(\underline{x}|\omega_1)P(\omega_1)(><)p(\underline{x}|\omega_2)P(\omega_2)$$

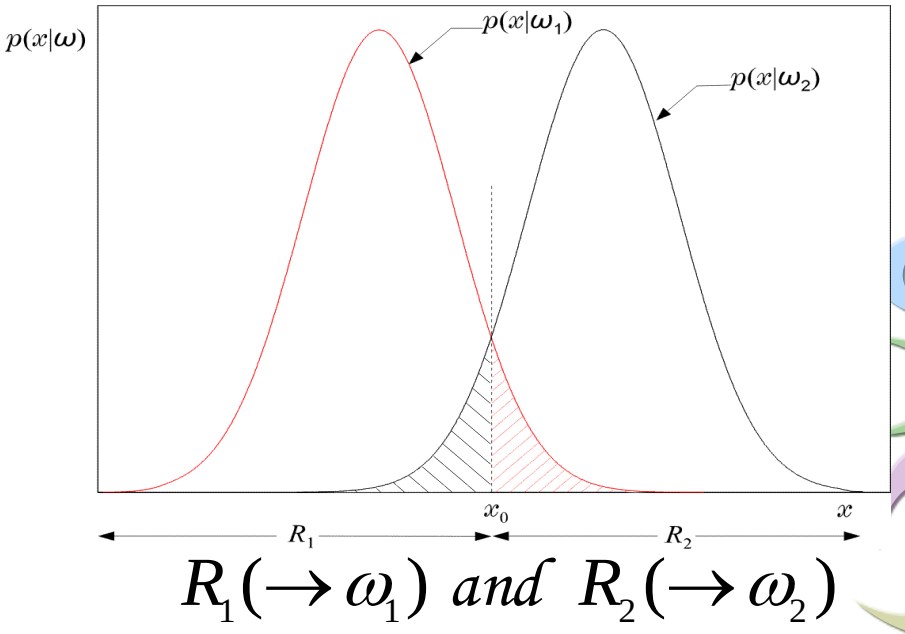
  - **For equiprobable classes the test becomes**

$$p(\underline{x}|\omega_1)(><)p(\underline{x}|\omega_2)$$

Bayesian Classifiers: Posterior Probability

- **Equivalently in words: Divide space in two regions**

$$\text{If } \underline{x} \in R_1 \Rightarrow \underline{x} \text{ in } \omega_1$$
$$\text{If } \underline{x} \in R_2 \Rightarrow \underline{x} \text{ in } \omega_2$$

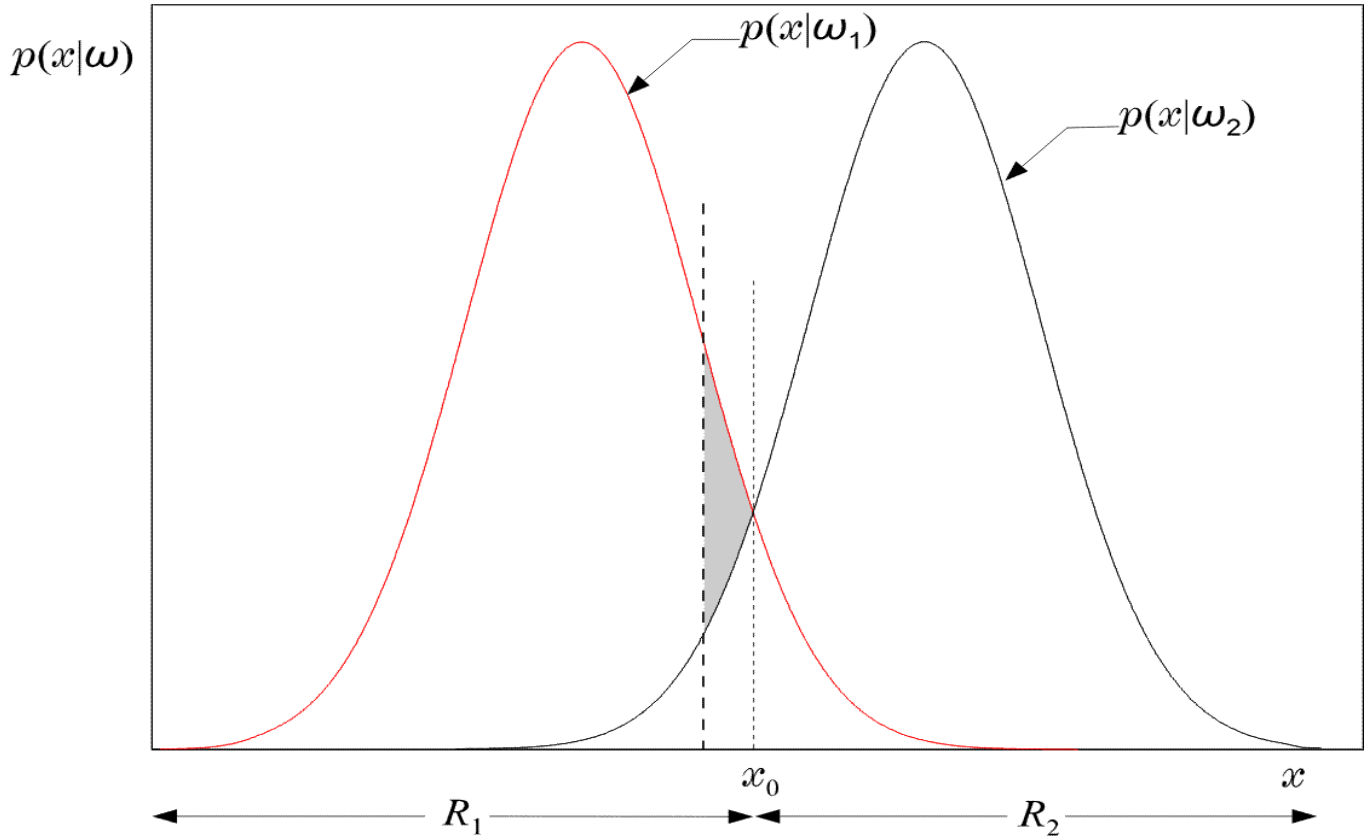


- **Probability of error**
  - **Total shaded area**

$$P_e = \frac{1}{2}\int_{-\infty}^{x_0} p(x|\omega_2)dx + \frac{1}{2}\int_{x_0}^{+\infty} p(x|\omega_1)dx$$

- **Bayesian classifier is OPTIMAL with respect to minimising the classification error probability!!!!**

Bayesian Classifiers: Posterior Probability



- **Indeed: Moving the threshold the total shaded area INCREASES by the extra "gray" area.**

Bayesian Classifiers: Probability Error

- **What is the probability of error for each decision?**

$$p(error \mid x) = \begin{cases} p(\omega_1 \mid x) & if\ we\ decide\ \omega_2 \\ p(\omega_2 \mid x) & if\ we\ decide\ \omega_1 \end{cases}$$

- **What is the average probability of error?**

$$p(error) = \int_{-\infty}^{+\infty} p(error, x)dx = \int_{-\infty}^{+\infty} p(error \mid x)p(x)dx$$

- **Bayes decision rule minimizes this error because**

$$p(error \mid x) = \min\{p(\omega_1 \mid x), p(\omega_2 \mid x)\}$$

Bayesian Classifiers: Naive Bayes

- **NAIVE – BAYES CLASSIFIER**
  - **Let $\underline{x} \in \Re^{\ell}$ and the goal is to estimate $p(\underline{x} \mid \omega_i)$ for $i = 1, 2, \ldots, M$. For a "good" estimate of the pdf one would need, say, N^{ℓ} points.**
  - **Assume $x_1, x_2, \ldots, x_{\ell}$ mutually independent. Then:**

$$p(\underline{x} \mid \omega_i) = \prod_{j=1}^{\ell} p(x_j \mid \omega_i)$$

  - **In this case, one would require, roughly, N points for each pdf. Thus, a number of points of the order $N \cdot \ell$ would suffice.**
  - **It turns out that the Naïve – Bayes classifier works reasonably well even in cases that violate the independence assumption.**

DISCRIMINANT FUNCTIONS DECISION SURFACES

- **discriminant function divides the feature space by a decision surface**
- **If R_i, R_j are contiguous:** $g(\underline{x}) \equiv P(\omega_i|\underline{x}) - P(\omega_j|\underline{x}) = 0$

$$R_i : P(\omega_i|\underline{x}) > P(\omega_j|\underline{x})$$

$$+ \quad \text{---------- } g(\underline{x}) = 0$$

$$-$$

$$R_j : P(\omega_j|\underline{x}) > P(\omega_i|\underline{x})$$

g(x) is the surface separating the regions. On the one side is positive (+), on the other is negative (-).

It is known as Decision Surface.

DISCRIMINANT FUNCTIONS DECISION SURFACES

- **If $f(.)$ monotonically increasing, the rule remains the same if we use:**

$$\underline{x} \to \omega_i \text{ if : } f(P(\omega_i|\underline{x})) > f(P(\omega_j|\underline{x})) \ \forall i \neq j$$

- $g_i(\underline{x}) \equiv f(P(\omega_i|\underline{x}))$ **is a discriminant function.**

- **In general, discriminant functions can be defined independent of the Bayesian rule.**
  - **They lead to suboptimal solutions, yet, if chosen appropriately, they can be computationally more tractable.**
  - **Moreover, in practice, they may also lead to better solutions. This, for example, may be case if the nature of the underlying pdf's are unknown.**

Linear Discriminant Function (LDF)

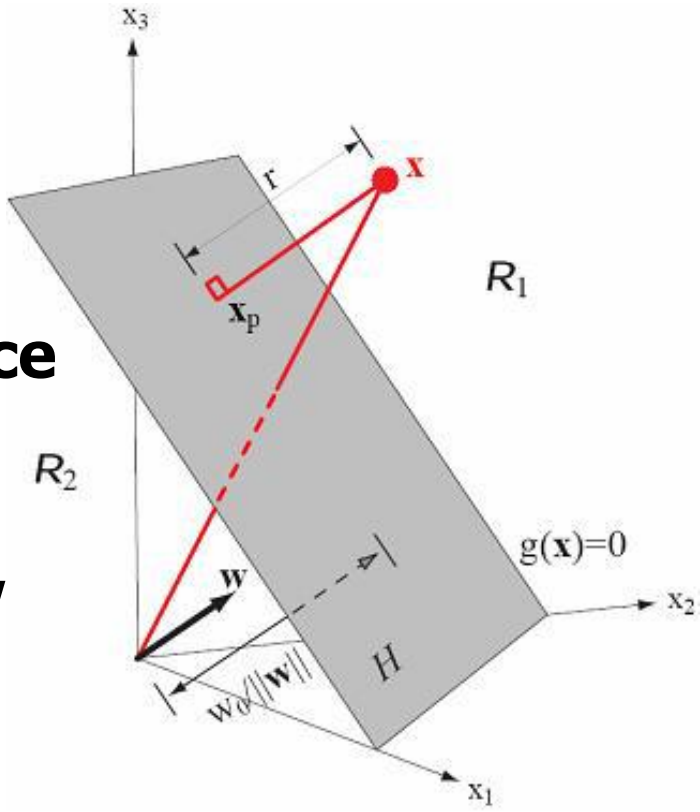
- **Definition:**
  - **LDF is a function that is a linear combination of the components of x**

$$g(x) = w^t x + w_0$$

    - where w is the weight vector and w_0 the bias, or threshold weight.
- **A two-category classifier with a discriminant function of the above form uses the following rule:**
  - **Decide w_1 if g(x) > 0 and w_2 if g(x) < 0**
    - Decide w_1 if $w^t x > -w_0$ and w_2 otherwise
    - The value g(x) of is called functional margin
  - **If g(x) = 0 then x can be assigned to either class**
    - The equation g(x) = 0 defines the decision surface that separates points assigned to the category w_1 from points assigned to the category w_2
    - When g(x) is linear, the decision surface is a hyperplane.

Linear Discriminant Function (LDF)

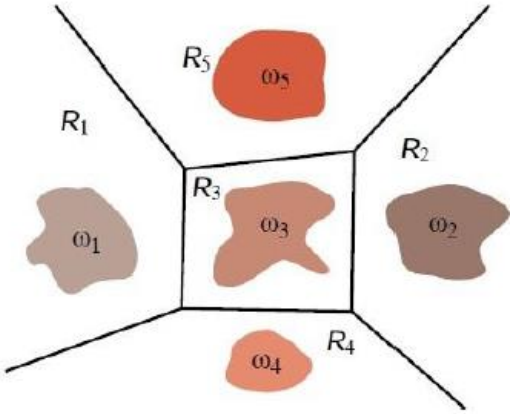
- A linear discriminant function divides the feature space by a hyperplane decision surface
- Decision boundary *g(x)=0* corresponds to(d-1)-dimensional hyperplane in d-dimensional x-space
- The orientation of the surface is determined by the normal vector w and the location of the surface is determined by the bias w_0
  - We can view Fisher method (LDA) as a linear discriminant function, too.

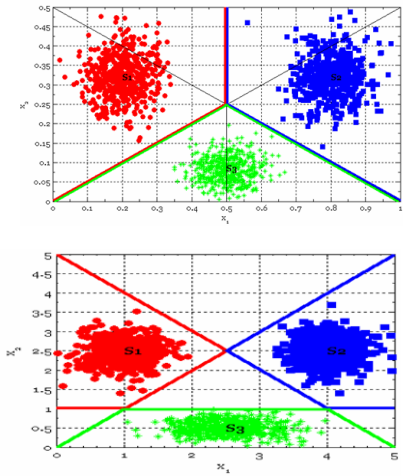


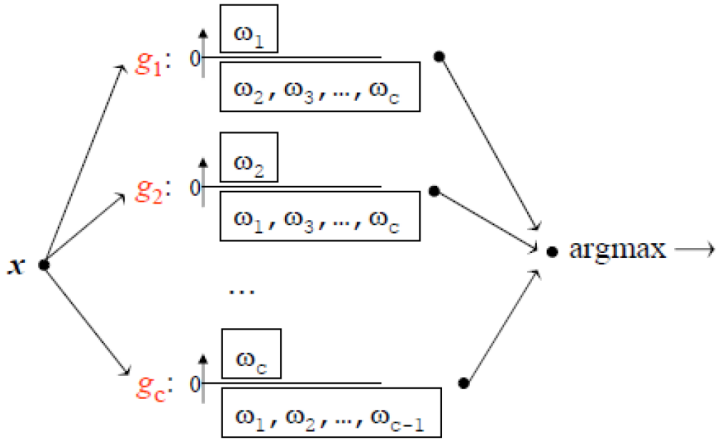
Linear Discriminant Function (LDF) Linear Machine

- **Suppose we have an n-classes classification problem, and we want to separate them with linear discriminant functions.**
  - **If we use the following rule for classification, this is linear machine rule:**

$$x \in \omega_i \Leftrightarrow g_i(x) > g_j(x);\ \forall j \neq i$$







  - **If we use the following rule for classification, this is completely linearly separation rule:**

$$if\ g_i(x) > 0 \ \Rightarrow\ x \in \omega_i\ and\ if\ g_i(x) < 0 \ \Rightarrow\ x \notin \omega_i$$

- **The decision regions for linear machine are convex and this restriction limits the flexibility of the classifier.**

Linear Discriminant Function Design

- **Definition:**
  - **LDF is a function that is a linear combination of the components of x**

$$g(x) = w^t x + w_0$$

    - where w is the weight vector and w_0 the bias, or threshold weight.
- **Main problem**
  - **How to create the discriminant functions for each class(how obtain w)?**
- **Many methods exist for this purpose, such as:**
  - **Probabilistic Methods**
  - **Error Minimization Methods**
    - **Least Mean Squared Error Method**
    - **Sum of Squared Error Method**
    - **Stochastic Minimization Methods**
    - **Ho-Kashyap Method**
  - **Perceptorn Method**
  - **etc.**

Linear Discriminant Function Design (Probabilistic Methods)

- **Maximum likelihood**
  - $g_i(x) = P(x|w_i)$
- **Bayesian Classifier**
  - $g_i(x) = P(w_i|x)$
    - $g_i(x) = P(x|w_i)P(w_i)$
    - $g_i(x) = Ln(P(x|w_i)) + Ln(P(w_i))$
- **Expected Loss (Conditional Risk)**
  - $g_i(x) = -R(a_i|x)$

Linear Discriminant Function Design (Augmented Space)

- **Consider a linear discriminant function $g(x)$ in feature space x**

$$g(x) = w^t x + w_0$$

- **We can use augmented space to get ride of w_0 in new augmented feature space x^* which map each data point x in to new augmented space x^* according:**

$$x^* = \begin{bmatrix} x \\ 1 \end{bmatrix} \quad and \quad w^* = \begin{bmatrix} w \\ w_0 \end{bmatrix}$$

- **So the new discriminant function will be:** $g(x^*) = w^{*t} x^*$
- **Then:**

$$g(x^*) > 0 \; for \; positive \; calss$$
$$g(x^*) < 0 \; for \; negative \; calss$$

- **If we invert the training data of negative class then for all data point we must have**

$$x^* = \begin{bmatrix} x_{11} & \cdots & x_{1d} & 1 \\ x_{21} & \cdots & x_{2d} & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_{k1} & \cdots & x_{kd} & 1 \\ x_{(k+1)1} & \cdots & x_{(k+1)d} & -1 \\ \vdots & \vdots & \vdots & \vdots \\ x_{n1} & \vdots & x_{nd} & -1 \end{bmatrix} \Rightarrow for \; all \; data \quad g(x^*) > 0$$

Linear Discriminant Function Design (Least Mean Squared Error)

- **The steps to Compute a solution, i.e., a hyperplane w^***
  - **Define a cost function ($J(w^*)$) to be minimized**
  - **Choose an algorithm to minimize the cost function**
  - **The minimum corresponds to a solution**

- **We want to choose the w^* that minimizes the mean-squared-error criterion function:**

$$J(w^*) = E\left[|y - g(x^*)|^2\right] = E\left[(y - w^{*t}x^*)^2\right]$$

$$\widehat{w^*} = arg \min_{w^*} J(w^*)$$

  - **y is the corresponding desired responses**

Linear Discriminant Function Design (Least Mean Squared Error)

- **Minimizing**

$J(\underline{w}^*)$ w.r. to $\underline{w}^*$ results in :

$$\frac{\partial J(\underline{w}^*)}{\partial \underline{w}^*} = \frac{\partial}{\partial \underline{w}^*} E[(y - \underline{w}^{*T} x^*)^2] = 2E[x^*(y - x^{*T}\underline{w}^*)] = 0$$

$$\Rightarrow E[x^* x^{*T}]\underline{w}^* = E[x^* y] \Rightarrow \quad \boxed{\underline{\hat{w}} = R_{x^*}^{-1} E[x^* y]}$$

- **where R_x is the autocorrelation matrix**

$$R_{x^*} \equiv E[\underline{x}^* \underline{x}^{*T}] = \begin{bmatrix} E[x_1^* x_1^*] & E[x_1^* x_2^*] \ldots & E[x_1^* x_n^*] \\ \ldots\ldots\ldots & \ldots\ldots\ldots\ldots & \ldots\ldots\ldots \\ E[x_n^* x_1^*] & E[x_n x_2] \ldots & E[x_n x_n] \end{bmatrix}$$

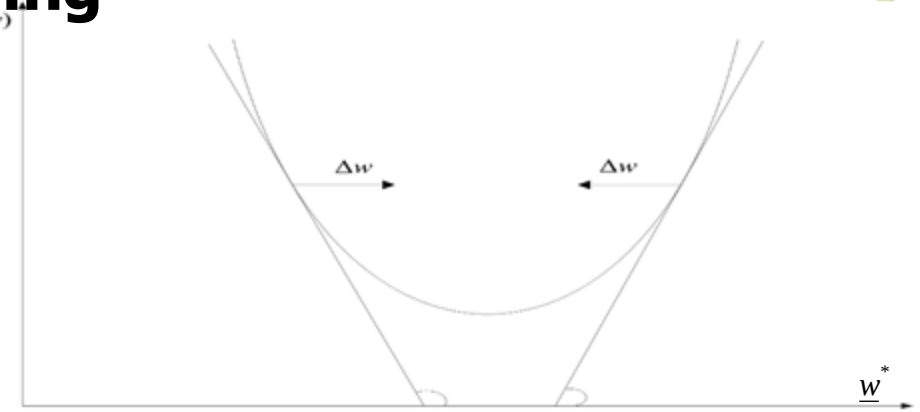
and $E[x^* y] = \begin{bmatrix} E[x_1^* y] \\ \ldots \\ E[x_n^* y] \end{bmatrix}$ **the crosscorrelation vector**

Linear Discriminant Function Design (Least Mean Squared Error)

- **Remark:** **The MSE criterion belongs to a more general class of cost function with the following important property:**
  - **The value of $g_i(x^*)$ is an estimate, in the MSE sense, of the a-posteriori probability $P(\omega_i|x^*)$, provided that the desired responses used during training are $y_i = 1\ if\ x^* \in \omega_i$ and 0 otherwise.**
- **We can also use the gradient descent rule for updating w^* instead of analytical solving**

$$\underline{w}^*(\text{new}) = \underline{w}^*(\text{old}) + \Delta\underline{w}^*$$

$$\Delta\underline{w}^* = -\mu \frac{\partial J(\underline{w}^*)}{\partial \underline{w}^*}\Big|_{\underline{w}^* = \underline{w}^*(\text{old})}$$



Linear Discriminant Function Design (Sum of Squared Error)

- **In SSE we want to choose the w^* that minimizes the sum of squared error as objective function**
  - **Also known as Pseudo inverse matrix method**

$$J(\underline{w}^*) = \sum_{i=1}^{N} (y_i - \underline{w}^{*T} \underline{x}_i^*)^2$$

$(y_i, \underline{x}_i^*)$: training pairs that, the input is $\underline{x}_i$ and its corresponding class label is y_i (±1).

$$\frac{\partial J(\underline{w}^*)}{\partial \underline{w}^*} = \frac{\partial}{\partial \underline{w}^*} \sum_{i=1}^{N} (y_i - \underline{w}^{*T} \underline{x}_i^*)^2 = 0 \Rightarrow$$

$$(\sum_{i=1}^{N} \underline{x}_i^* \underline{x}_i^{*T}) \underline{w}^* = \sum_{i=1}^{N} \underline{x}_i^* y_i$$

Linear Discriminant Function Design (Sum of Squared Error)

- **Pseudo inverse Matrix**
  - **Define**

$$X = \begin{bmatrix} \underline{x}_1^T \\ \underline{x}_2^T \\ \dots \\ \underline{x}_N^T \end{bmatrix} \quad (N \times (d+1) \text{ matrix}) \qquad \underline{y} = \begin{bmatrix} y_1 \\ \dots \\ y_N \end{bmatrix} \text{ corresponding desired responses}$$

  - $X^T = [\underline{x}_1, \underline{x}_2, \dots, \underline{x}_N] \quad (\text{an } (d+1) \times N \text{ matrix})$
  - $X^T X = \sum_{i=1}^{N} \underline{x}_i \underline{x}_i^T$ **and** $X^T \underline{y} = \sum_{i=1}^{N} \underline{x}_i y_i$

- **Thus** $(\sum_{i=1}^{N} \underline{x}_i^T \underline{x}_i)\underline{\hat{w}} = (\sum_{i=1}^{N} \underline{x}_i y_i) \Rightarrow (X^T X)\underline{\hat{w}} = X^T \underline{y} \Rightarrow \underline{\hat{w}} = (X^T X)^{-1} X^T \underline{y} = X^{\neq} \underline{y}$

$$X^{\neq} \equiv (X^T X)^{-1} X^T$$ **Pseudo inverse of X**

Linear Discriminant Function Design (Sum of Squared Error)

- **Assume $N = d+1$ $\Rightarrow$ X square and invertible. Then**

$$(X^T X)^{-1} X^T = X^{-1} X^{-T} X^T = X^{-1} \Rightarrow \quad \boxed{X^{\neq} = X^{-1}}$$

- **Assume $N > d+1$. Then, in general, there is no solution to satisfy all equations simultaneously:**

$$X\underline{w} = \underline{y}: \begin{cases} \underline{x}_1^T \underline{w} = y_1 \\ \underline{x}_2^T \underline{w} = y_2 \\ \cdots \\ \underline{x}_N^T \underline{w} = y_N \end{cases} \quad N \text{ equations} > d+1 \text{ unknowns}$$

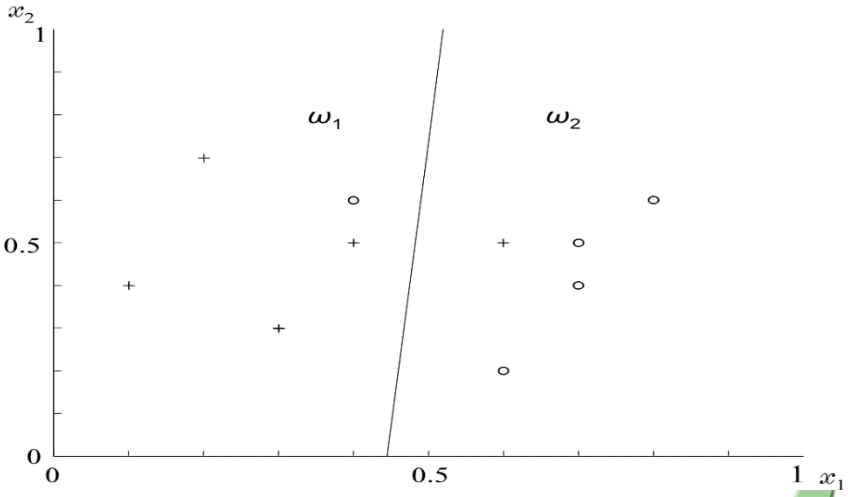
- **The "solution" $\underline{w} = X^{\neq} \underline{y}$ corresponds to the minimum sum of squares solution**

Linear Discriminant Function Design (Sum of Squared Error)

- **Example:**

$$\omega_1: \begin{bmatrix}0.4\\0.5\end{bmatrix}, \begin{bmatrix}0.6\\0.5\end{bmatrix}, \begin{bmatrix}0.1\\0.4\end{bmatrix}, \begin{bmatrix}0.2\\0.7\end{bmatrix}, \begin{bmatrix}0.3\\0.3\end{bmatrix}$$

$$\omega_2: \begin{bmatrix}0.4\\0.6\end{bmatrix}, \begin{bmatrix}0.6\\0.2\end{bmatrix}, \begin{bmatrix}0.7\\0.4\end{bmatrix}, \begin{bmatrix}0.8\\0.6\end{bmatrix}, \begin{bmatrix}0.7\\0.5\end{bmatrix}$$



$$X = \begin{bmatrix}0.4 & 0.5 & 1\\0.6 & 0.5 & 1\\0.1 & 0.4 & 1\\0.2 & 0.7 & 1\\0.3 & 0.3 & 1\\0.4 & 0.6 & 1\\0.6 & 0.2 & 1\\0.7 & 0.4 & 1\\0.8 & 0.6 & 1\\0.7 & 0.5 & 1\end{bmatrix} \text{ and } \underline{y} = \begin{bmatrix}1\\1\\1\\1\\1\\-1\\-1\\-1\\-1\\-1\end{bmatrix} \Rightarrow$$

$$X^T X = \begin{bmatrix}2.8 & 2.24 & 4.8\\2.24 & 2.41 & 4.7\\4.8 & 4.7 & 10\end{bmatrix}, \; X^T \underline{y} = \begin{bmatrix}-1.6\\0.1\\0.0\end{bmatrix}$$

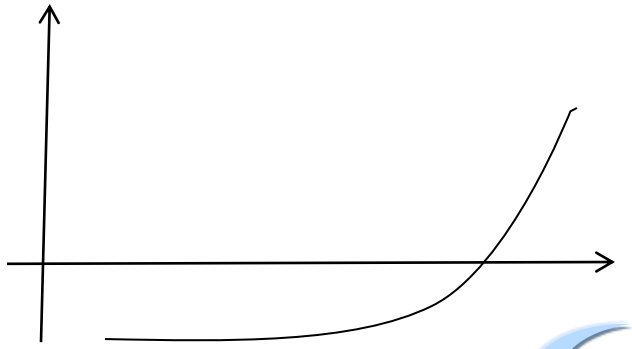
$$\underline{w} = (X^T X)^{-1} X^T \underline{y} = \begin{bmatrix}-3.13\\0.24\\1.34\end{bmatrix}$$

Discriminant Function Design (Stochastic Minimization)

- **Our goal:**

$$E\{F(x_k, w)\} = 0, \quad x_k : \text{Sequence of samples}$$

$$w_k = w_{k-1} + \varepsilon_k F(x_k, w_{k-1}), \quad \begin{cases} \sum \varepsilon_k \to \infty \\ \sum \varepsilon_k^2 \to 0 \end{cases}$$



- **Simple Model:**

$$E\{x_k - w\} = 0, \quad x_k : \text{Sequence of samples}$$

$$w_k = w_{k-1} + \frac{1}{k}(x_k - w_{k-1})$$

$$w_k = \frac{k-1}{k} w_{k-1} + \frac{1}{k} x_k \Rightarrow w_k = \frac{1}{k} \sum_{r=1}^{k} x_r$$

- **Widrow-Hoff Learning:**

$$w_k = w_{k-1} + \varepsilon_k x_k \left(y_k - x_k^T w_{k-1}\right)$$

LMS Method with fixed step size: $0 < \varepsilon < \frac{2}{trace(R_x)}$

Linear Discriminant Function Design (Ho-Kashyap Method)

- **The main limitation of the SSE is lack of guarantees that a separating hyperplane will be found in the linearly separable case**
  - The SSE rule tries to minimize $\|w^t x + y\|^2$
  - Finding a separating hyperplane depends on how suitably the outputs **y** are selected

- **If the two classes are linearly separable, there must exists vectors w and b such that $w^t x = y > 0$**
  - if **y** were known, to compute the separating hyperplane, the SSE solution will be $w = x - y$
  - Nevertheless, since **y** is unknown, one must solve the equation for both w and y

- **A possible algorithm is the Ho-Kashyap procedure:**
  - **1. Find the target values y with gradient descent**
  - **2. compute the weight vector w from the SSE solution**
  - **3. Repeat 1 and 2 until convergence**

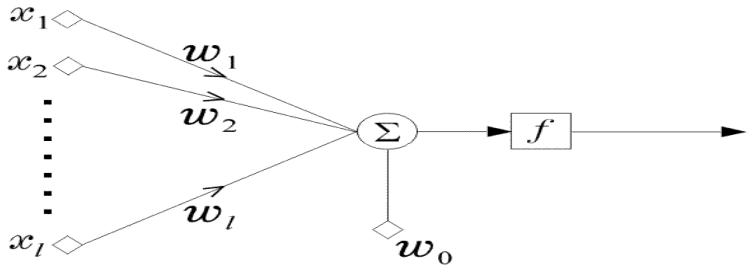
Linear Discriminant Function Design (Ho-Kashyap Method)

- $g(x) > 0$ **can be rewrite as** $g(x) = y\,; y > 0$
  - **How we can determine** y**?**
- **Objective function in this case is** $J(w, y) = \|w^t x - y\|^2$
- **Ho-Kashyap method offers an iterative method for obtaining w and** y**, using following steps:**
  - **Keep** y **constant and optimize** J **relative to** w **(using obtained** y **from last step)**
    - Using previous method we have: $w(t+1) = x^t y(t)$
  - **Keep** w **constant and optimize** J **relative to** y **(using obtained** w **from last step)**
    - The objective is to minimize $\frac{dJ}{dw} = -2w^t x - y$
    - Using Gradient descent method we have: $y(t+1) = y(t) + 2\eta\, w^t(t)x - y$
    - To hold the constraint $y > 0$, we set $(w^t x - y)$ in this rule to zero if it becomes negative, then the rule will be:

$$y(t+1) = y(t) + \eta[w^t(t)x - y + |w^t(t)x - y|]$$

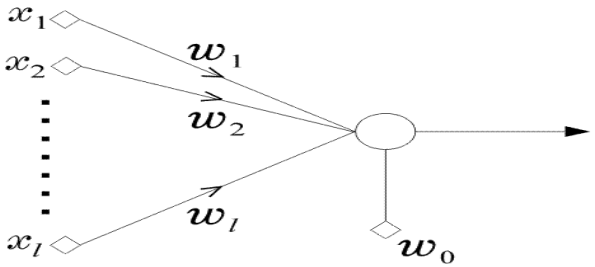
Linear Discriminant Function Design (Perceptorn Method)

- **The network is called perceptron or neuron**



$w_i's$ synapses or synaptic weights

w_0 threshold



- **It is a learning machine that learns from the training vectors via the perceptron algorithm**

$$\mathbf{w}(k+1)=\begin{cases}\mathbf{w}(k)+\mu\mathbf{x} & x\in\omega_1 \text{ and } \mathbf{w}^T(k)\mathbf{x}<0\\ \mathbf{w}(k)-\mu\mathbf{x} & x\in\omega_2 \text{ and } \mathbf{w}^T(k)\mathbf{x}>0\end{cases}$$

$$\mu>\frac{\left|\mathbf{w}^T(k)\mathbf{x}\right|}{\mathbf{x}^T\mathbf{x}}$$

Nonparametric Classification

K Nearest Neighbor (KNN)

- **The Nearest Neighbor Rule**
  - **Choose k out of the N training vectors, identify the k nearest ones to $\underline{x}$**
  - **Out of these k identify k_i that belong to class ω_i**
  - $$\text{Assign } \underline{x} \rightarrow \omega_i : \; k_i > k_j \quad \forall i \neq j$$
  - **The simplest version is : $k=1$!!!**
  - **For large N this is not bad. It can be shown that: if P_B is the optimal Bayesian error probability, then:**

$$P_B \leq P_{NN} \leq P_B\left(2 - \frac{M}{M-1} P_B\right) \leq 2P_B$$

    - **M is the number of classes, and** $k \rightarrow \infty, \; P_{kNN} \rightarrow P_B$

Nonparametric Classification

Decision Trees (DT)

- **This is a family of non-linear classifiers. They are multistage decision systems, in which classes are sequentially rejected, until a finally accepted class is reached. To this end:**
  - **The feature space is split into unique regions in a sequential manner.**
  - **Upon the arrival of a feature vector, sequential decisions, assigning features to specific regions, are performed along a path of nodes of an appropriately constructed tree.**
  - **The sequence of decisions is applied to individual features, and the queries performed in each node are of the type:**

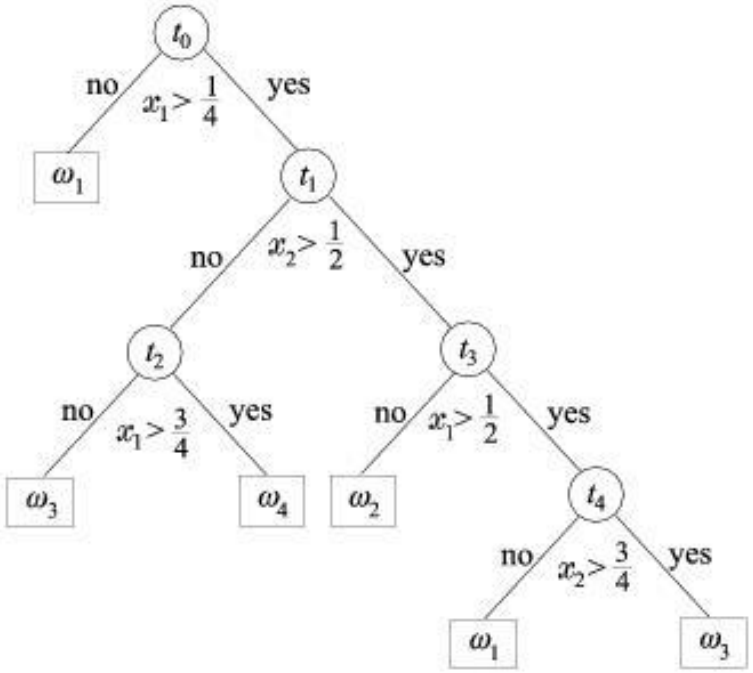
$$is\ Feature\ x_i \leq \alpha$$

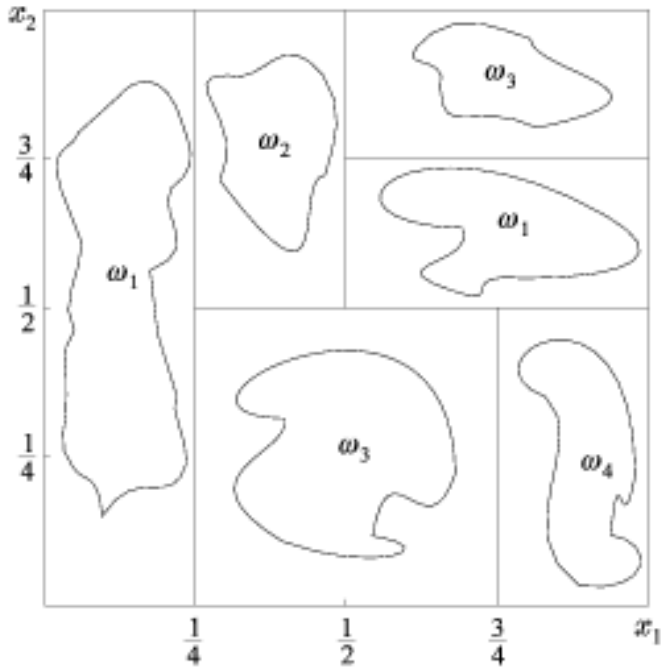
  - **where α is a pre-chosen (during training) threshold.**

Nonparametric Classification

Decision Trees (DT)

- **The figures below are such examples. This type of trees is known as Ordinary Binary Classification Trees (OBCT).**
  - **The decision hyperplanes, splitting the space into regions, are parallel to the axis of the spaces. Other types of partition are also possible, yet less popular.**





Nonparametric Classification

Decision Trees (DT)

- **Design Elements that define a decision tree.**
  - **Each node, t, is associated with a subset $X_t \subseteq X$, where X is the training set. At each node, X_t is split into two (binary splits) disjoint descendant subsets $X_{t,Y}$ and $X_{t,N}$, where**

$$X_{t,Y} \cap X_{t,N} = \emptyset$$
$$X_{t,Y} \cup X_{t,N} = X_t$$

$X_{t,Y}$ is the subset of X_t for which the answer to the query at node t is YES. $X_{t,N}$ is the subset corresponding to NO. The split is decided according to an adopted question (query).

Nonparametric Classification

Decision Trees (DT)

- A **splitting** criterion must be adopted for the **best** split of X_t into $X_{t,Y}$ and $X_{t,N}$.
- A **stop-splitting** criterion must be adopted that controls the growth of the tree and a node is declared as **terminal (leaf)**.
- A rule is required that assigns each (terminal) leaf to a class.
- **Set of Questions:** In OBCT trees the set of questions is of the type

$$is \quad x_i \leq \alpha \ ?$$

The choice of the specific x_i and the value of the threshold α, for each node t, are the results of searching, during training, among the features and a set of possible threshold values. The final combination is the one that results to the **best value** of a criterion.

Nonparametric Classification

Decision Trees (DT)

- **Splitting Criterion:** The main idea behind splitting at each node is the resulting descendant subsets $X_{t,Y}$ and $X_{t,N}$ to be more **class homogeneous** compared to X_t. Thus the criterion must be in harmony with such a goal. A commonly used criterion is the **node impurity**:

$$I(t) = -\sum_{i=1}^{M} P(\omega_i \mid t) \log_2 P(\omega_t \mid t)$$

and

$$P(\omega_i \mid t) \approx \frac{N_t^i}{N_t}$$

where N_t^i is the number of data points in X_t that belong to class ω_i. The **decrease in node impurity** is defined as:

$$\Delta I(t) = I(t) - \frac{N_{t,\mathrm{Y}}}{N_t} I(t_{\mathrm{Y}}) - \frac{N_{t,N}}{N_t} I(t_N)$$

Nonparametric Classification

Decision Trees (DT)

Razi University

- The goal is to choose the parameters in each node (feature and threshold) that result in a split with the highest decrease in impurity.

- Why highest decrease? Observe that the highest value of $I(t)$ is achieved if all classes are equiprobable, i.e., X_t is the least homogenous.

- Stop - splitting rule. Adopt a threshold T and stop splitting a node (i.e., assign it as a leaf), if the impurity decrease is less than T. That is, node t is "pure enough".

- Class Assignment Rule: Assign a leaf to a class ω_j, where:

$$j = \arg\max_i P(\omega_i \mid t)$$

Nonparametric Classification

Decision Trees (DT)

- **Remarks:**
  - **A critical factor in the design is the size of the tree. Usually one grows a tree to a large size and then applies various pruning techniques.**
  - **Decision trees belong to the class of unstable classifiers.**
    - **This can be overcome by a number of "averaging" techniques. Bagging is a popular technique. Using bootstrap techniques in X, various trees are constructed, T_i, $i=1, 2, \ldots, B$. The decision is taken according to a majority voting rule.**

Nonparametric Classification

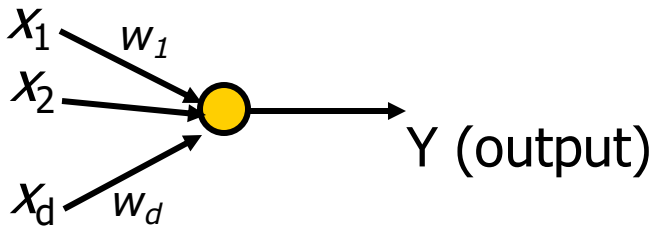
Artificial Neural Network (ANN)

- **Biological networks achieve excellent recognition performance via dense interconnection of simple computational elements (neurons)**
  - **Number of neurons ≈ $10^{10} – 10^{12}$**
  - **Number of interconnections/neuron ≈ $10^{3} – 10^{4}$**
  - **Total number of interconnections ≈ 10^{14}**
- **Massive parallelism is essential for complex recognition tasks (e.g., speech & image recognition)**
  - **Humans take only a few hundred milliseconds for most cognitive tasks; this suggests parallel computation in human brain**

Nonparametric Classification

Artificial Neural Network (ANN)

- **Nodes in neural networks are simple computational nonlinear elements, typically analog**



$$Y = f(\sum_{i=1}^{d} w_i x_i - \theta)$$

where θ is internal threshold or offset

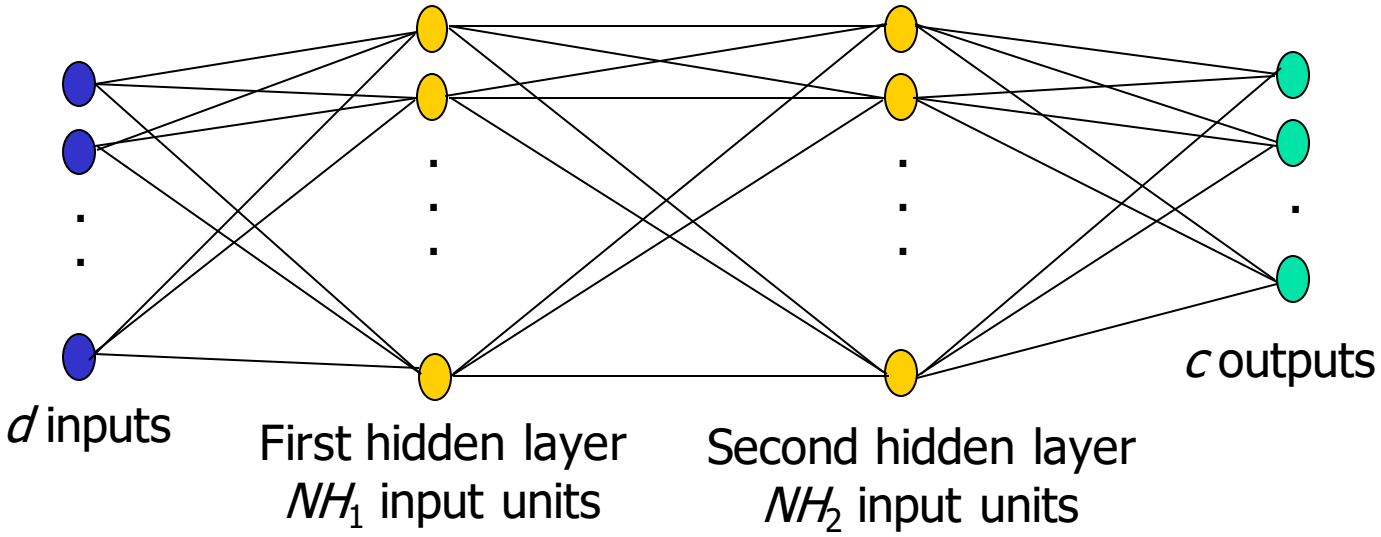
- **Feed-forward networks with one or more layers (hidden) between input & output nodes**

- **Networks are trained by back-propagation training algorithm**

Nonparametric Classification

Artificial Neural Network (ANN)

- **A three-layer network can generate arbitrary complex decision regions**

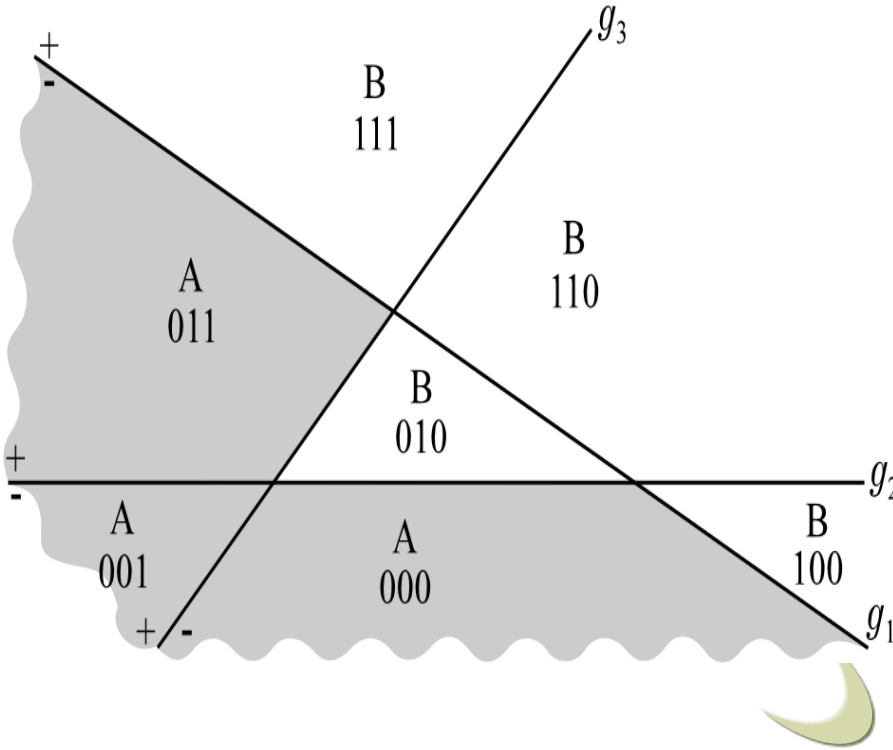


- **This is capable to classify vectors into classes consisting of ANY union of polyhedral regions.**
- **Intersections of these hyperplanes form regions in the l-dimensional space. Each region corresponds to a vertex of the H_p unit hypercube.**

Nonparametric Classification

Artificial Neural Network (ANN)

- **The reasoning**
  - **For each vertex, corresponding to class, say A, construct a hyperplane which leaves THIS vertex on one side (+) and ALL the others to the other side (-).**
  - **The output neuron realizes an OR gate**



- **Overall:**

The first layer of the network forms the hyperplanes, the second layer forms the regions and the output neuron forms the classes.

- **The Backpropagation Algorithm**
  - **This is an algorithmic procedure that computes the synaptic weights iteratively, so that an adopted cost function is minimized (optimized)**
  - **In a large number of optimizing procedures, computation of derivatives are involved. Hence, discontinuous activation functions pose a problem, i.e.,**

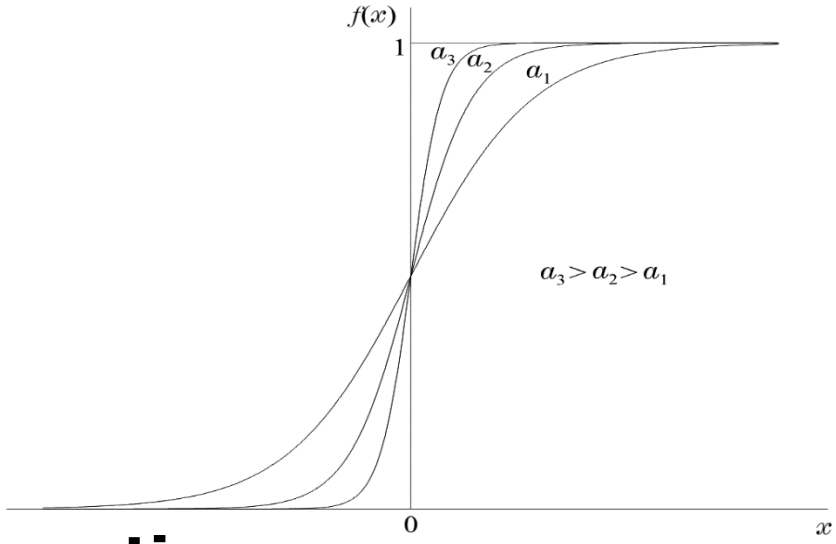
$$f(x)=\begin{cases}1 & x>0\\ 0 & x<0\end{cases}$$

  - **There is always an escape path!!!**
  - **The logistic/Sigmoid function** $f(x)=\dfrac{1}{1+\exp(-ax)}$

is an example. Other functions are also possible and in some cases more desirable.

Nonparametric Classification

Artificial Neural Network (ANN)



- **Common Alternatives:**
  - **Sigmoid** $f(x)=\frac{1}{1+e^{-x}} \Rightarrow f'(x)=f(x)\left(1-f(x)\right)!$
  - **Hyperbolic** $f(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \Rightarrow f'(x)=1-f(x)^{2}$
  - **Linear** $f(x)=x \Rightarrow f'(x)=1$

Nonparametric Classification

Artificial Neural Network (ANN)

- **The steps:**
  - **Adopt an optimizing cost function, e.g.,**
    - **Least Squares Error**
    - **Relative Entropy**

    between desired responses and actual responses of the network for the available training patterns.

    That is, from now on we have to live with errors. We only try to minimize them, using certain criteria.
  - **Adopt an algorithmic procedure for the optimization of the cost function with respect to the synaptic weights e.g.,**
    - **Gradient descent** $\underline{w}_1^r(\text{new}) = \underline{w}_1^r(\text{old}) + \Delta\underline{w}_1^r \qquad where \quad \Delta\underline{w}_1^r = -\mu\frac{\partial J}{\partial w_1^r}$
    - **Newton's algorithm**
    - **Conjugate gradient**

Nonparametric Classification

Artificial Neural Network (ANN)

- **The Procedure:**
  - Initialize unknown weights randomly with small values.
  - Compute the gradient terms backwards, starting with the weights of the last (3^{rd}) layer and then moving towards the first
  - Update the weights
  - Repeat the procedure until a termination procedure is met

- **Two major philosophies:**
  - **Batch mode**: The gradients of the last layer are computed once ALL training data have appeared to the algorithm, i.e., by summing up all error terms.
  - **Pattern mode**: The gradients are computed every time a new training data pair appears. Thus gradients are based on successive individual errors.

Artificial Neural Network (ANN) Some Important Points

- **Algorithm Termination:**
  - **Condition on Cost function (J<ε)**
  - **Condition on Gradient of Cost Function (Grad(J) <ε) –Between two successive Epochs**
  - **Small Changes in Weights!**

- **Problem with Local Minimum:**
  - **Problem comes from high complexity of mapping function**
  - **Solving methods:**
    - **Re-Initialization**
    - **Add white noise to data**
    - **Change the cost function**
    - **Data Re-Shuffling (some case!)**

Artificial Neural Network (ANN) Some Important Points

- **What do, when see no improvement in cost function:**
  - **Add complexity is NOT best and first choice!**
  - **A common case is local minima!**
  - **Add complexity if required!**

- **How to use from our data:**
  - **For large dataset (Holdout Method)**
    - **Training Set:**
      - **Estimation Set**
      - **Validation Set**
    - **Test Set**
  - **For small dataset (Leave-One-Out Method):**
    - **Use N-1 samples for training and 1 excluding for testing (N possible experiments) and choose the best one!**

Artificial Neural Network (ANN) Some Heuristic Interpretation

- **For large Dataset uses Pattern-Mode**
  - **Problem of Correlated data in Batch-Mode**
- **Use Valuable Training set (Rich-Contents):**
  - **Use Samples with high training Errors**
  - **Use Samples which differ from others**
  - **Random Re-shuffling in each Epoch**
  - **Train hard samples (High Training Error) more**
- **Use Anti-Symmetric (odd) function:**

$$f(x)=\alpha\tanh(\beta x)\xrightarrow[\beta=0.6667]{\alpha=1.7150}\begin{cases} f(1)=-f(-1)=1 \\ f'(0)=\alpha\beta=1.1424 \\ f''(1)=Max f''(x) \end{cases}$$

Artificial Neural Network (ANN) Some Heuristic Interpretation

- **Normalized output [0 1], [-1 +1]**
  - **Consider a margin [ε,1-ε], [-1+ε , 1-ε]**
  - **Former example: ε=0.7159→[-1 +1] as target!**

- **Input Normalization, zero mean**
  - **Input Whitening (Uncorrelated data), PCA/KL Transform**
  - **Normalized to unique Covariance value (Same Speed of convergence for each dimension)**
  - **Gram-Schmidt orthogonalization**

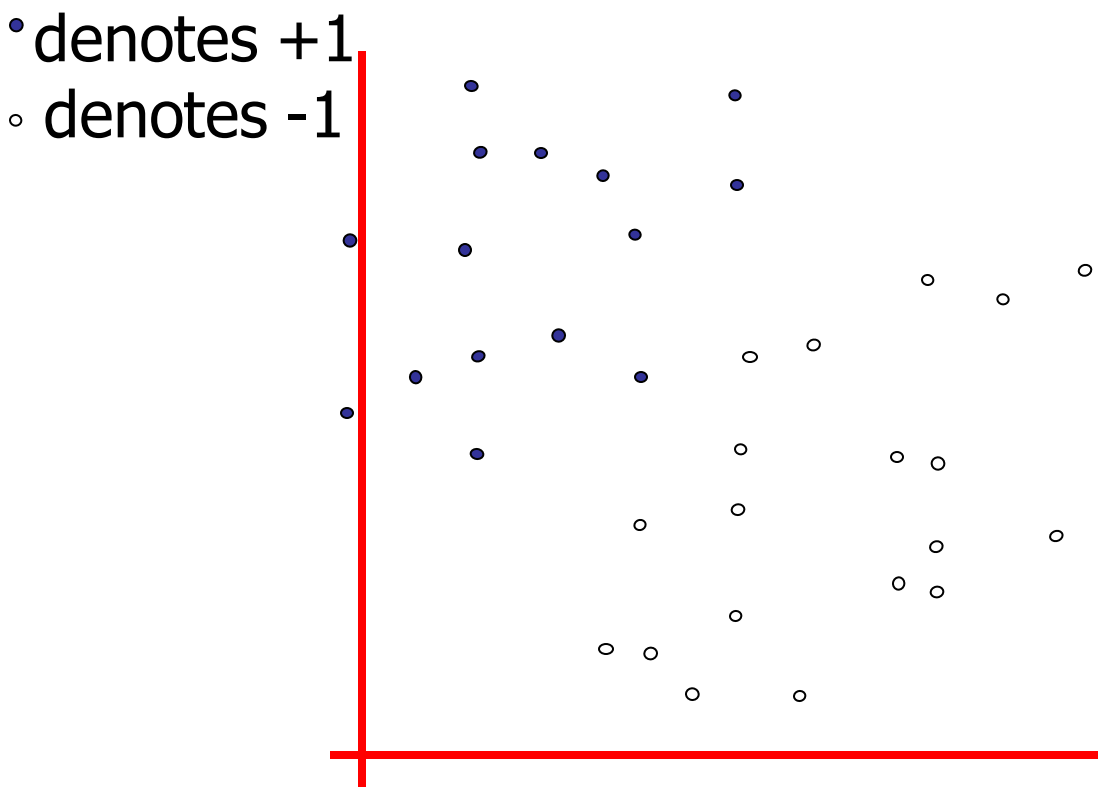
- **Initialization**
  - **Medium value for all weights (Problem with large and small values)**
  - **Uniform $(0,\sigma^2)$**

Nonparametric Classification
Support Vector Machine (SVM)

- **Separable Classes**
  - If we have two-class (ω_1, ω_2) linearly separable and $\boldsymbol{x} = \{x_i, i = 1,2, \dots, N\}$ as feature vectors of the training set, then we design linear classifiers as:
$$g(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b$$
  - To correctly classifies all the training vectors. Such a hyperplane is not unique. The perceptron algorithm may converge to any one of the possible solutions.
  - Let us now quantify the term *margin* that a hyperplane leaves from both classes.
  - Every hyperplane is characterized by its direction (determined by $\boldsymbol{w}$) and its exact position in space (determined by b). Since we want to give no preference to either of the classes, then it is reasonable for each direction to select that hyperplane which has the same distance from the respective nearest points in ω_1 and ω_2.
  - ***Our goal is to search for the direction that gives the maximum possible margin.***

Nonparametric Classification

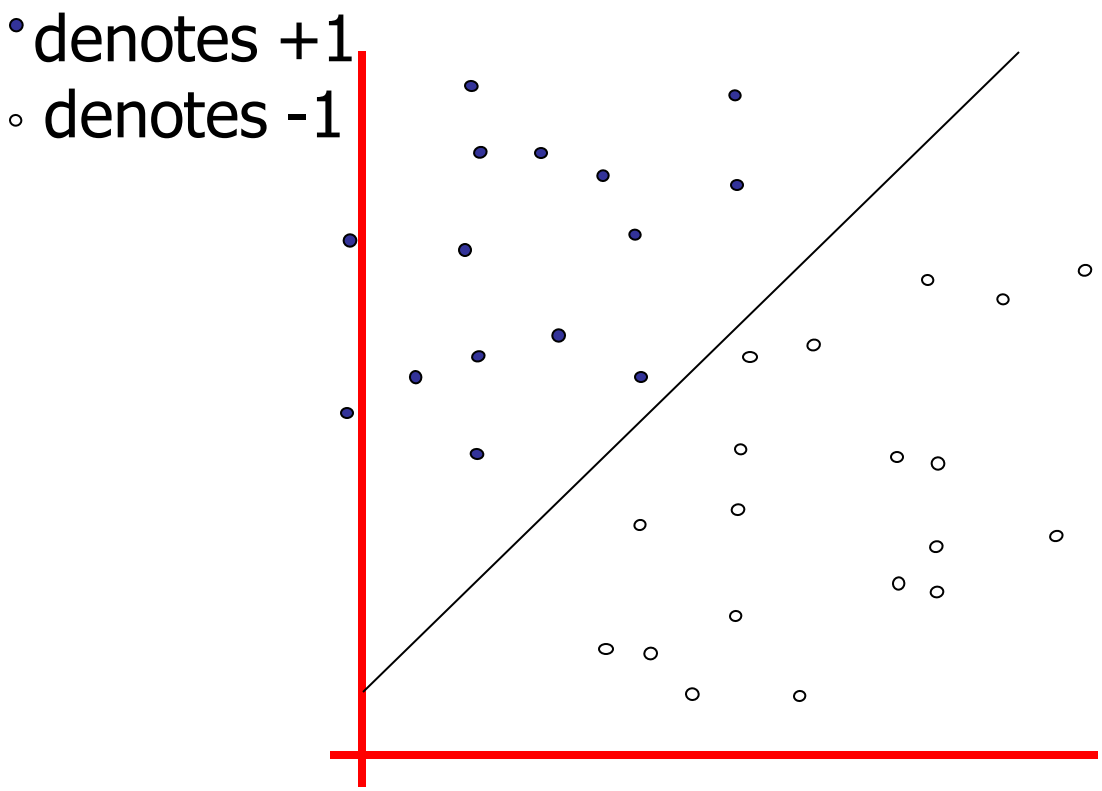
Support Vector Machine (SVM)



How would you classify this data?

Nonparametric Classification

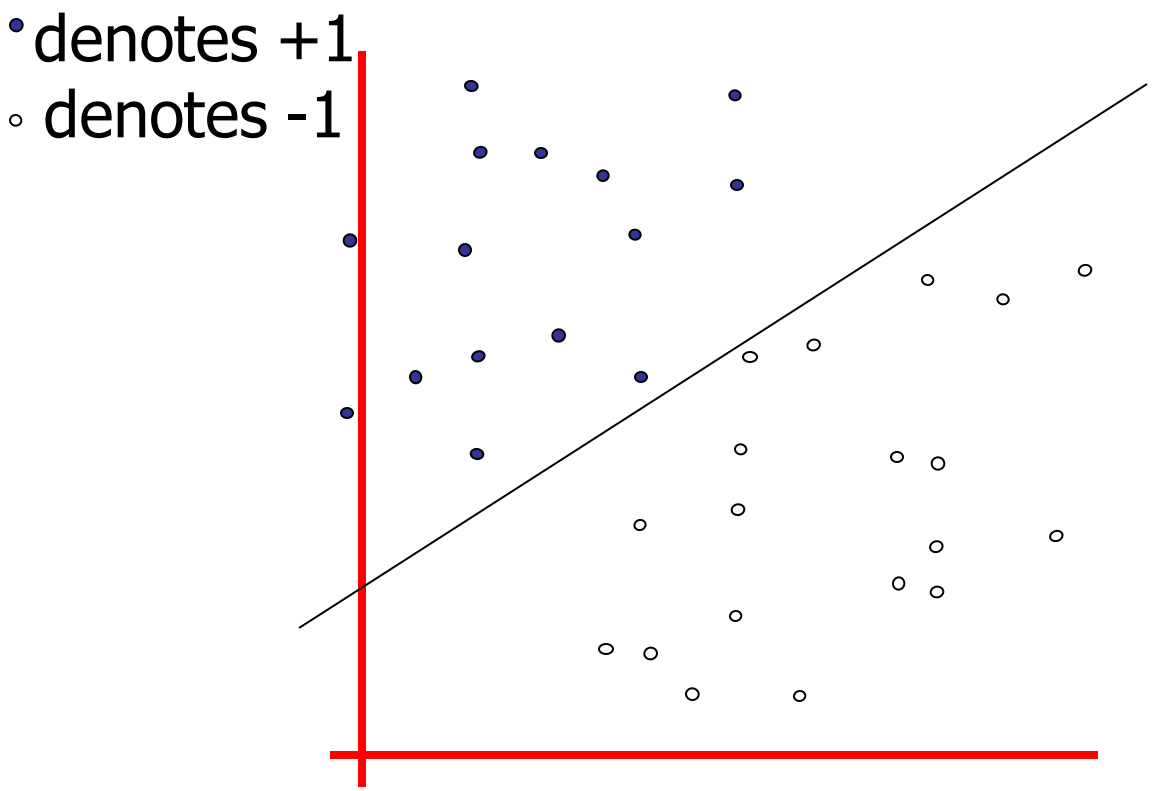
Support Vector Machine (SVM)



How would you classify this data?

Nonparametric Classification

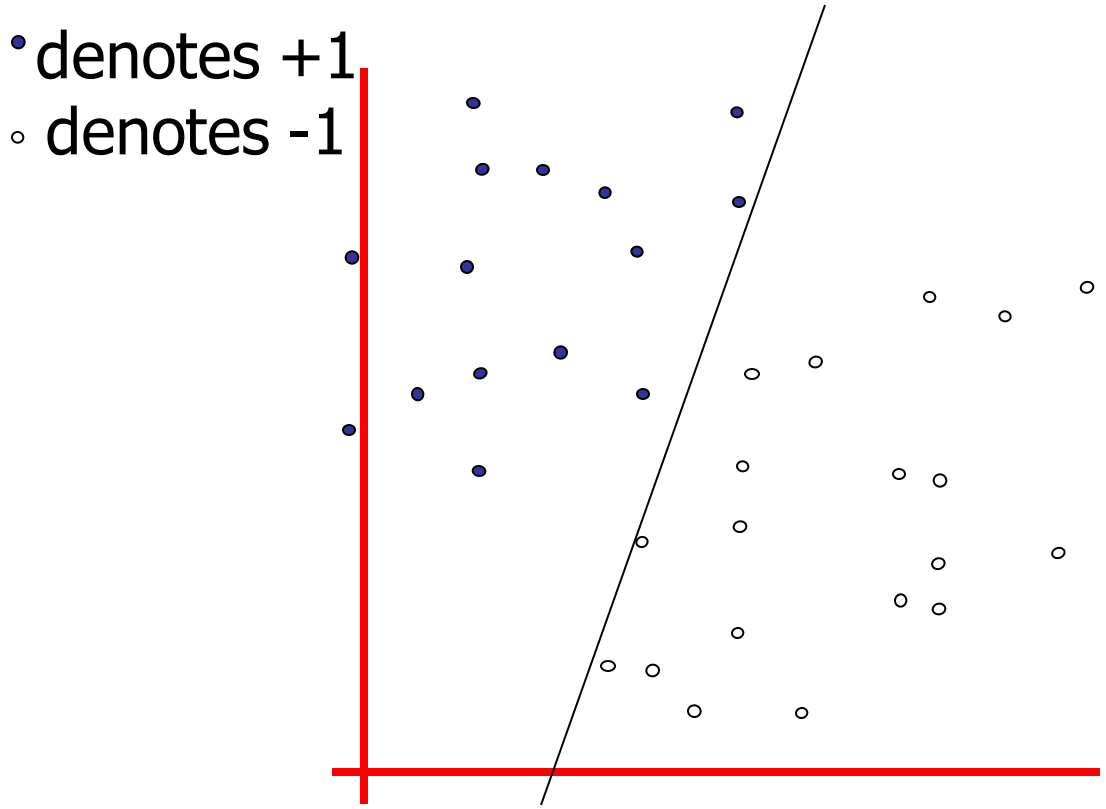
Support Vector Machine (SVM)



How would you classify this data?

Nonparametric Classification

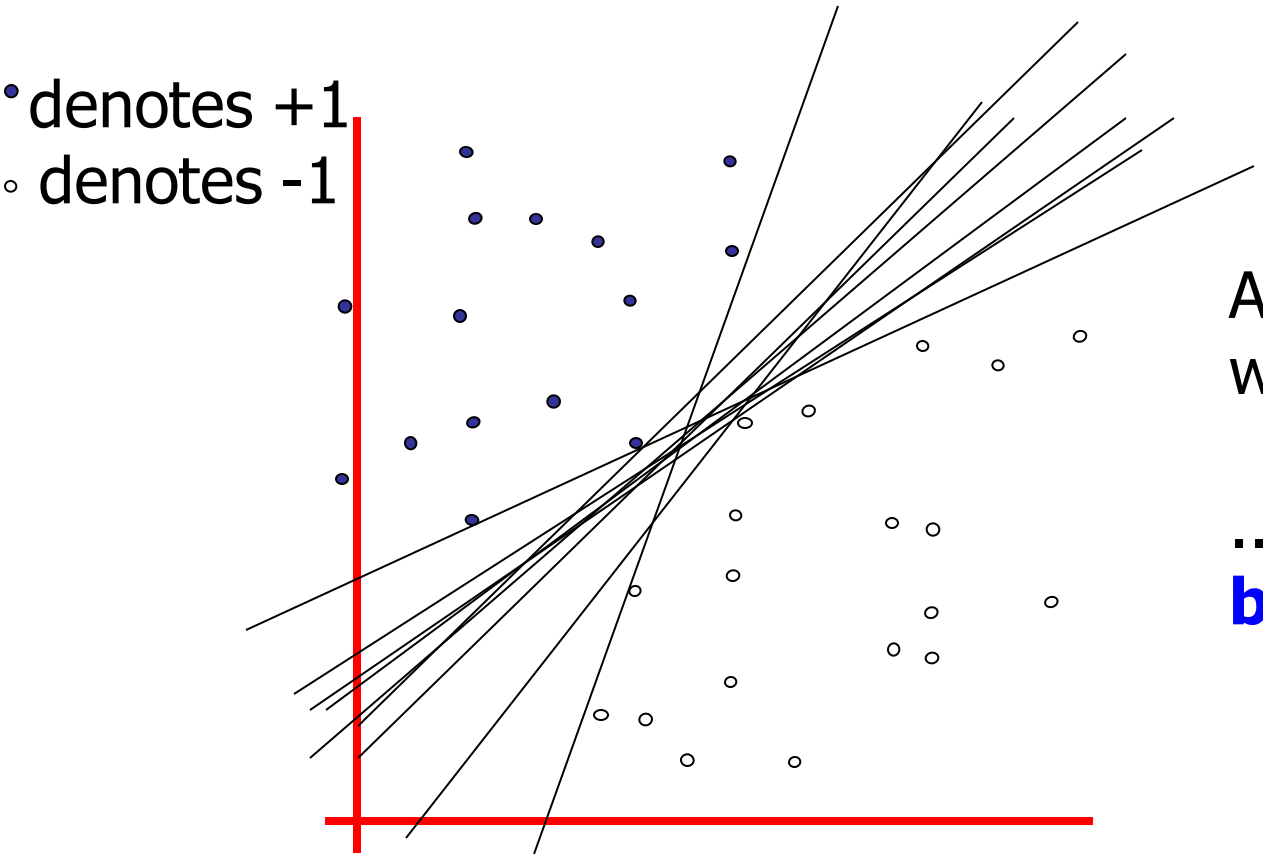
Support Vector Machine (SVM)



How would you classify this data?

Nonparametric Classification

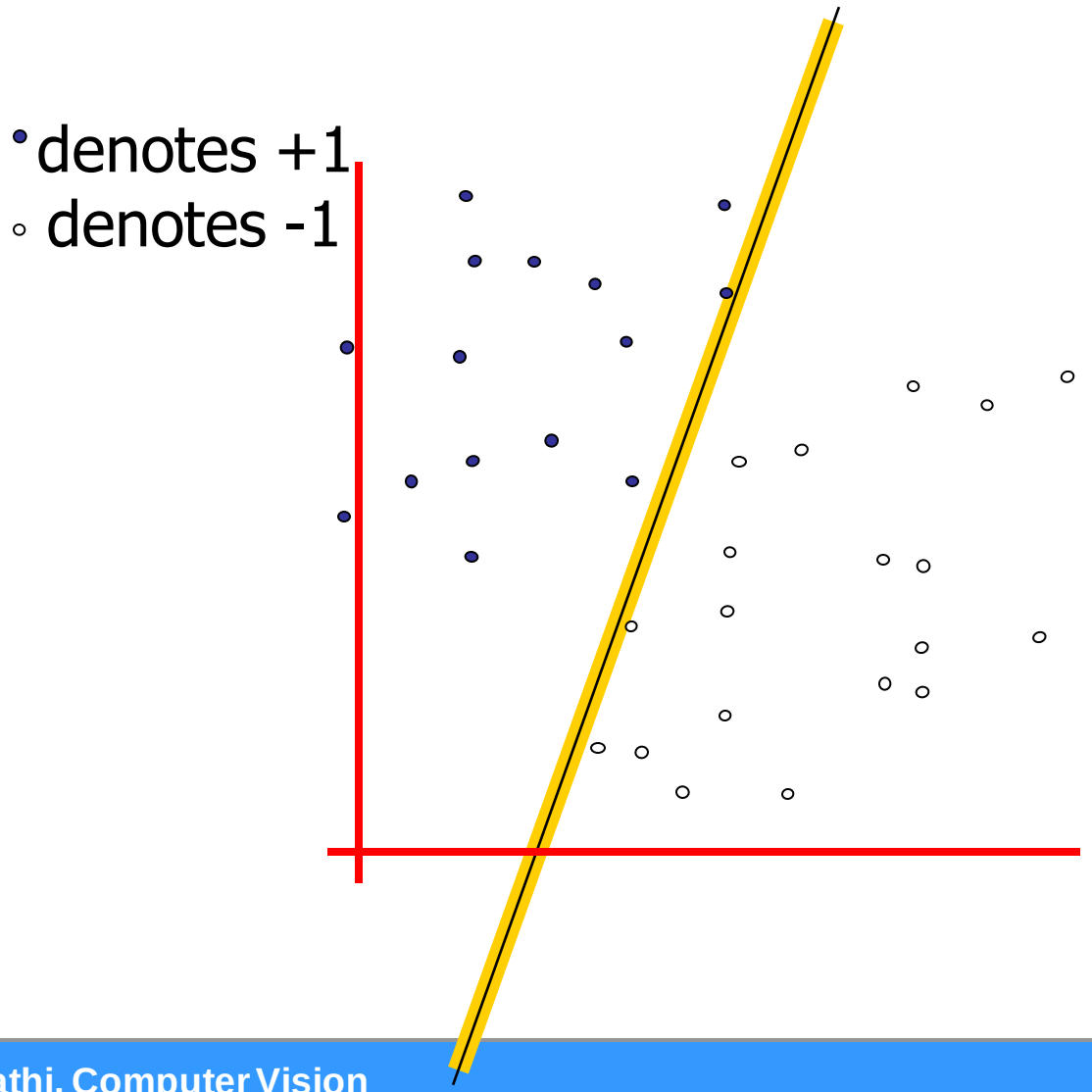
Support Vector Machine (SVM)



Any of these would be fine..

..but which is **best**?

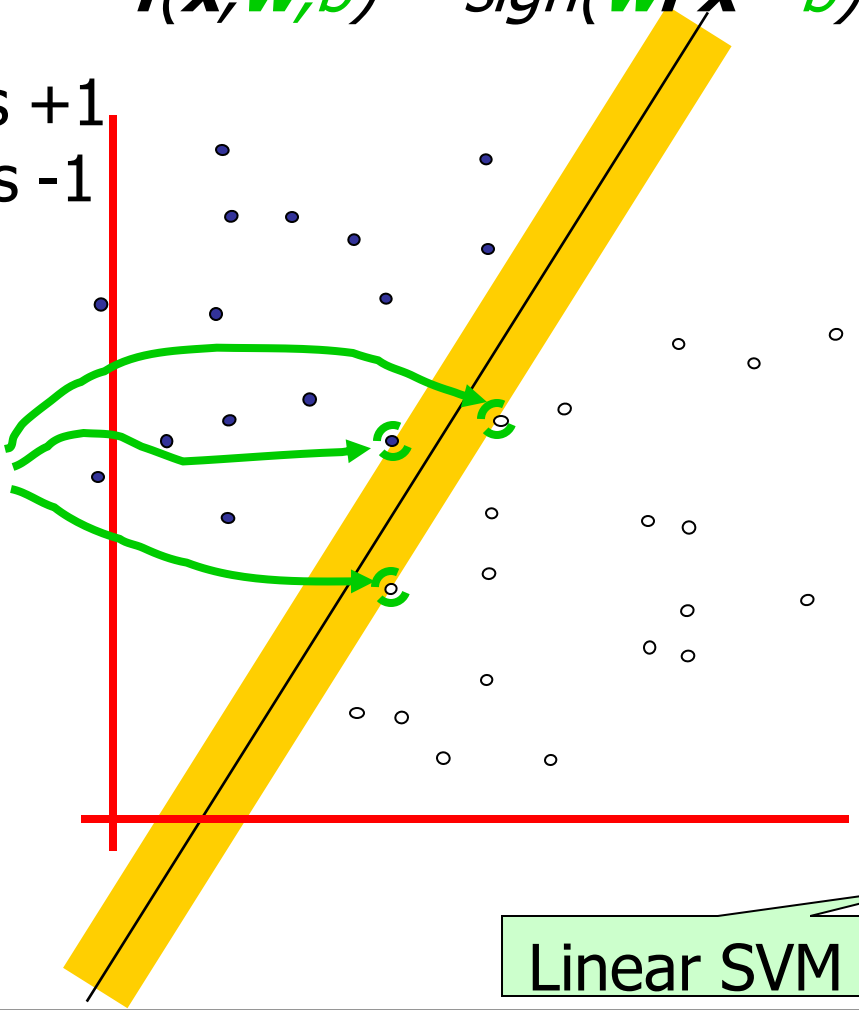
Support Vector Machine (SVM) Classifier Margin



Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.

Support Vector Machine (SVM) Maximum Margin

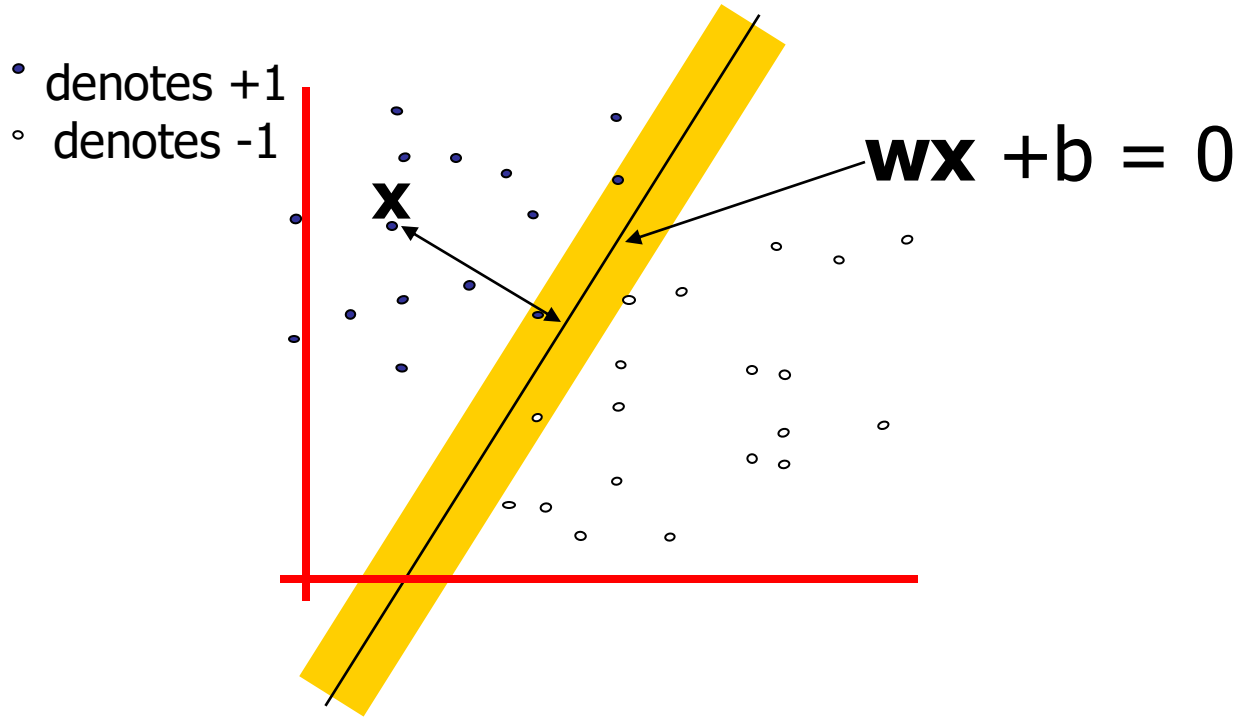
$$f(\mathbf{x}, \mathbf{w}, b) = sign(\mathbf{w}. \mathbf{x} - b)$$



Support Vectors are those datapoints that the margin pushes up against

The maximum margin linear classifier is the linear classifier with the, um, maximum margin. This is the simplest kind of SVM (Called an LSVM)

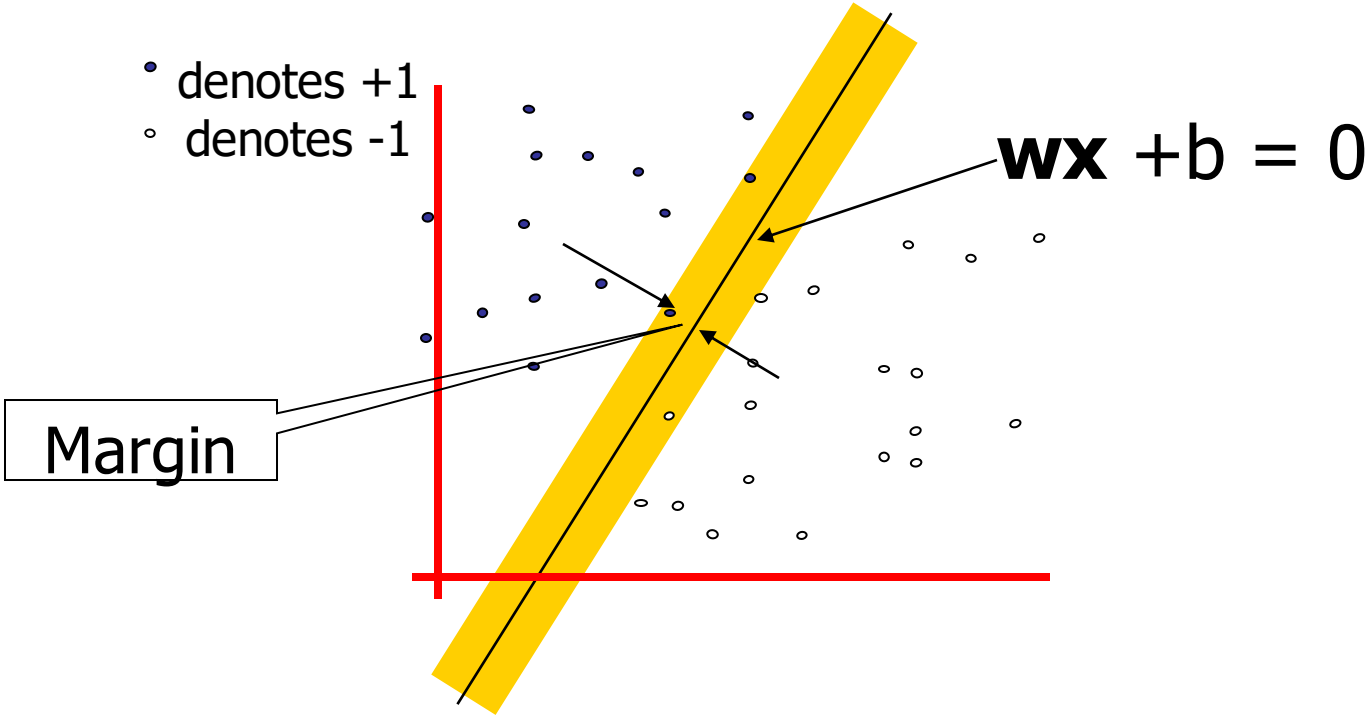
Support Vector Machine (SVM) Estimate the Margin



- **What is the distance expression for a point x to a line wx+b= 0?**

$$d(\mathbf{x}) = \frac{|\mathbf{x}\cdot\mathbf{w}+b|}{\sqrt{\|\mathbf{w}\|_2^2}} = \frac{|\mathbf{x}\cdot\mathbf{w}+b|}{\sqrt{\sum_{i=1}^{d} w_i^2}}$$

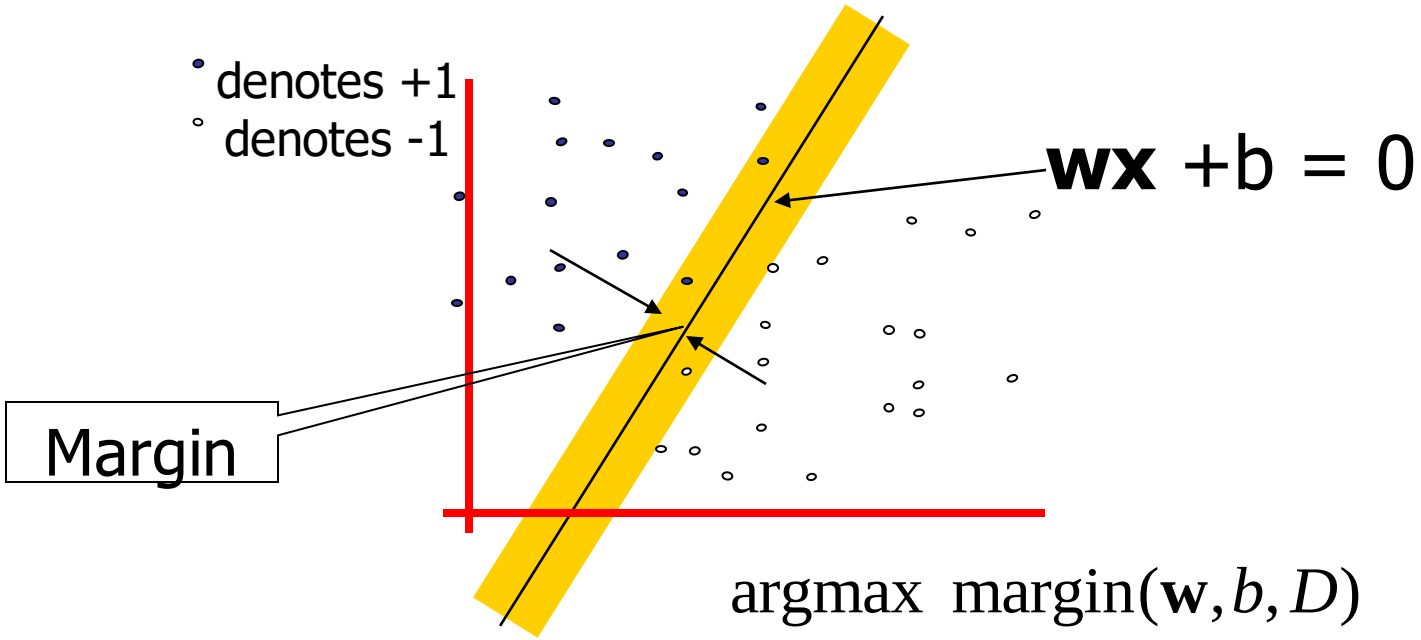
Support Vector Machine (SVM) Estimate the Margin



- **What is the expression for margin?**

$$\text{margin} \equiv \min_{\mathbf{x}\in D} d(\mathbf{x}) = \min_{\mathbf{x}\in D} \frac{\left|\mathbf{x}\cdot\mathbf{w}+b\right|}{\sqrt{\sum_{i=1}^{d} w_i^2}}$$

Support Vector Machine (SVM)
Maximize Margin

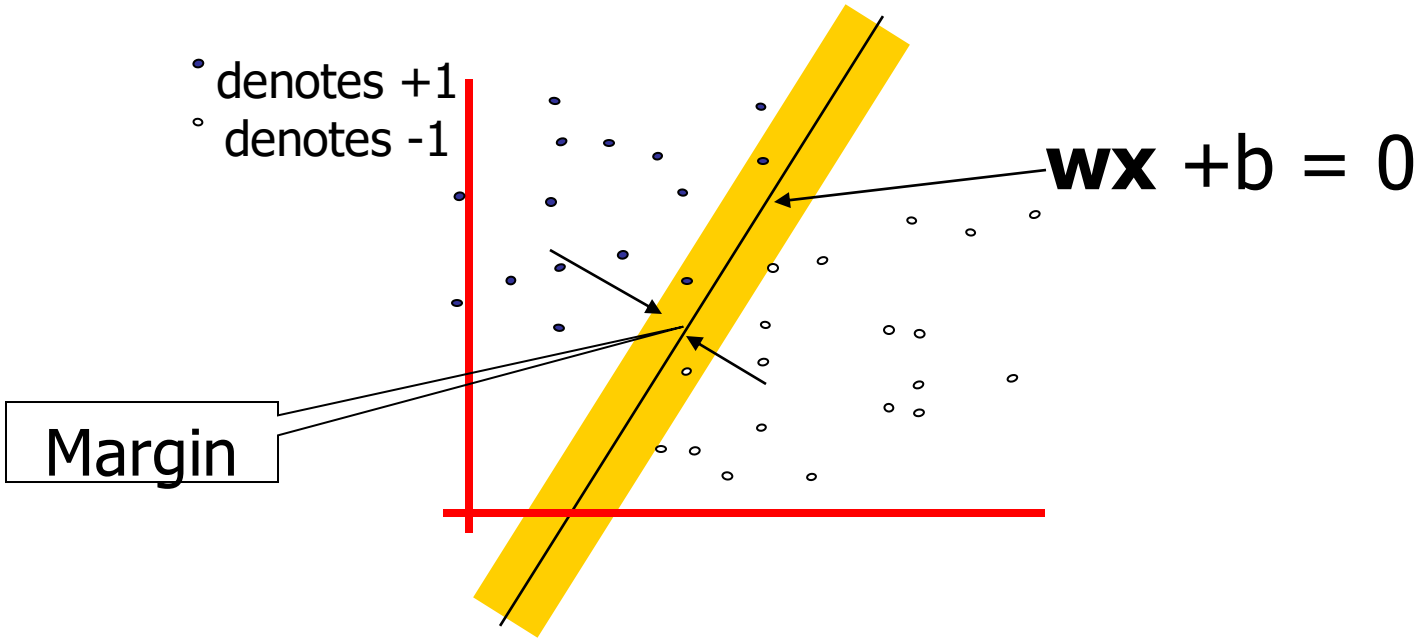


$$\underset{\mathbf{w},b}{\text{argmax}}\ \text{margin}(\mathbf{w},b,D)$$

$$= \underset{\mathbf{w},b}{\text{argmax}} \min_{\mathbf{x}_i \in D} d(\mathbf{x}_i)$$

$$= \underset{\mathbf{w},b}{\text{argmax}} \min_{\mathbf{x}_i \in D} \frac{\left| b + \mathbf{x}_i \cdot \mathbf{w} \right|}{\sqrt{\sum_{i=1}^{d} w_i^2}}$$

Support Vector Machine (SVM) Maximize Margin

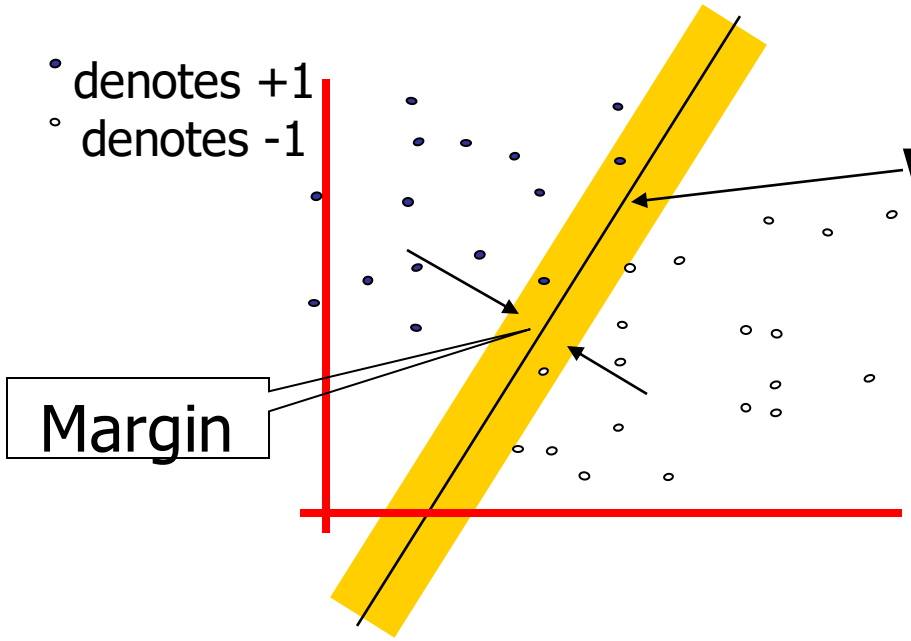


$$\underset{\mathbf{w},b}{\operatorname{argmax}} \min_{\mathbf{x}_i \in D} \frac{\left| b + \mathbf{x}_i \cdot \mathbf{w} \right|}{\sqrt{\sum_{i=1}^{d} w_i^2}}$$

$$\text{subject to } \forall \mathbf{x}_i \in D : y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b \right) > 0$$

- **Min-max problem → game problem**

Support Vector Machine (SVM)
Maximize Margin



$$\underset{\mathbf{w},b}{\operatorname{argmax}} \min_{\mathbf{x}_i \in D} \frac{\left| b + \mathbf{x}_i \cdot \mathbf{w} \right|}{\sqrt{\sum_{i=1}^{d} w_i^2}}$$

subject to $\forall \mathbf{x}_i \in D : y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b \right) \geq 0$

Strategy:

$$\forall \mathbf{x}_i \in D : \left| b + \mathbf{x}_i \cdot \mathbf{w} \right| \geq 1$$

$$\underset{\mathbf{w},b}{\operatorname{argmin}} \sum_{i=1}^{d} w_i^2$$

subject to $\forall \mathbf{x}_i \in D : y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b \right) \geq 1$

Support Vector Machine (SVM) Maximum Margin Linear Classifier

$$\{\vec{w}^*, b^*\} = \underset{\vec{w},b}{\operatorname{argmin}} \sum_{k=1}^{d} w_k^2$$

subject to

$$y_1\left(\vec{w}\cdot\vec{x}_1 + b\right) \geq 1$$

$$y_2\left(\vec{w}\cdot\vec{x}_2 + b\right) \geq 1$$

....

$$y_N\left(\vec{w}\cdot\vec{x}_N + b\right) \geq 1$$

- **quadratic programming is a well-studied class of optimization algorithms can be used to maximize a quadratic function of some real-valued variables subject to linear constraints.**

Support Vector Machine (SVM) Quadratic Programming

Find $\arg\min_{\mathbf{u}} \; c + \mathbf{d}^T\mathbf{u} + \frac{\mathbf{u}^T R\mathbf{u}}{2}$ ← Quadratic criterion

Subject to

$$\begin{aligned} a_{11}u_1 + a_{12}u_2 + \ldots + a_{1m}u_m &\le b_1 \\ a_{21}u_1 + a_{22}u_2 + \ldots + a_{2m}u_m &\le b_2 \\ &\vdots \\ a_{n1}u_1 + a_{n2}u_2 + \ldots + a_{nm}u_m &\le b_n \end{aligned}$$

n additional linear inequality constraints

And subject to

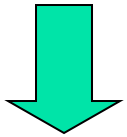
$$\begin{aligned} a_{(n+1)1}u_1 + a_{(n+1)2}u_2 + \ldots + a_{(n+1)m}u_m &= b_{(n+1)} \\ a_{(n+2)1}u_1 + a_{(n+2)2}u_2 + \ldots + a_{(n+2)m}u_m &= b_{(n+2)} \\ &\vdots \\ a_{(n+e)1}u_1 + a_{(n+e)2}u_2 + \ldots + a_{(n+e)m}u_m &= b_{(n+e)} \end{aligned}$$

e additional linear equality constraints

Support Vector Machine (SVM) Quadratic Programming

$$\{\vec{w}^*, b^*\} = \min_{\vec{w},b} \sum_i w_i^2$$

subject to $y_i\left(\vec{w}\cdot\vec{x}_i + b\right) \geq 1$ for all training data $(\vec{x}_i, y_i)$



$$\{\vec{w}^*, b^*\} = \underset{\vec{w},b}{\text{argmax}} \left\{0 + \vec{0}\cdot\vec{w} - \vec{w}^T \mathbf{I_n} \vec{w}\right\}$$

$$\left.\begin{array}{l} y_1\left(\vec{w}\cdot\vec{x}_1 + b\right) \geq 1 \\ y_2\left(\vec{w}\cdot\vec{x}_2 + b\right) \geq 1 \\ .... \\ y_N\left(\vec{w}\cdot\vec{x}_N + b\right) \geq 1 \end{array}\right\} \text{inequality constraints}$$

Support Vector Machine (SVM) The Dual Form of QP

Maximize $\sum_{k=1}^{R} \alpha_k - \frac{1}{2} \sum_{k=1}^{R} \sum_{l=1}^{R} \alpha_k \alpha_l Q_{kl}$ where $Q_{kl} = y_k y_l (\mathbf{x}_k \cdot \mathbf{x}_l)$

Subject to these constraints: $0 \le \alpha_k \le C \quad \forall k$ $\sum_{k=1}^{R} \alpha_k y_k = 0$

Then define:

$$\mathbf{w} = \sum_{k=1}^{R} \alpha_k y_k \mathbf{x}_k$$

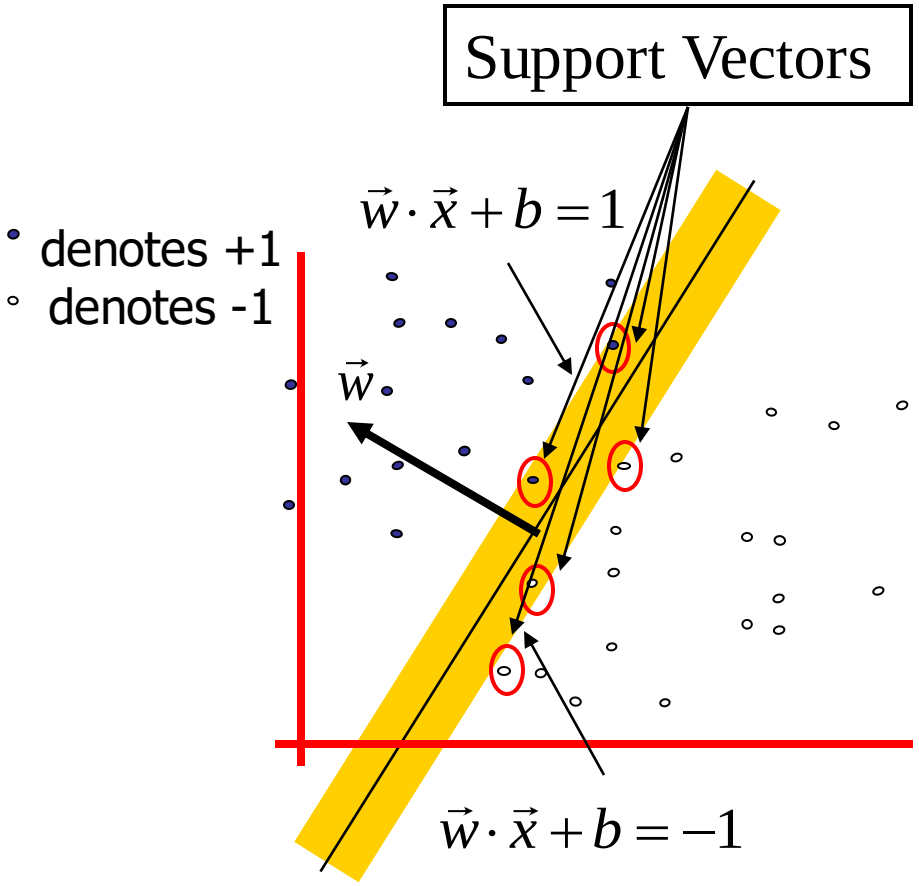
Datapoints with $\alpha_k > 0$ will be the support vectors

..so this sum only needs to be over the support vectors.

Then classify with:

$f(\mathbf{x}, \mathbf{w}, b) = sign(\mathbf{w}. \mathbf{x} - b)$

Support Vector Machine (SVM)
Support Vectors



$$\forall i : \alpha_i \left(y_i \left(\vec{w} \cdot \vec{x}_i + b \right) - \left(1 - \varepsilon_i \right) \right) = 0$$

$\alpha_i = 0$ for non-support vectors

$\alpha_i \neq 0$ for support vectors

$$\mathbf{w} = \sum_{k=1}^{R} \alpha_k y_k \mathbf{x}_k$$

Decision boundary is determined only by those support vectors !

Support Vector Machine (SVM) Kernel Functions

- ***K(a,b)=(a . b +1)^d*** **is an example of an SVM Kernel Function**
- **Beyond polynomials there are other very high dimensional basis functions that can be made practical by finding the right Kernel Function**
  - **Radial-Basis-style Kernel Function:** $K(\mathbf{a},\mathbf{b}) = \exp\left(-\frac{(\mathbf{a}-\mathbf{b})^2}{2\sigma^2}\right)$
  - **Neural-net-style Kernel Function:** $K(\mathbf{a},\mathbf{b}) = \tanh(\kappa\mathbf{a}.\mathbf{b}-\delta)$
- **Not all functions are kernel functions**
  - **Need to be decomposable**
    - **K(a,b) = ϕ(a) $\cdot$ ϕ(b)**

Support Vector Machine (SVM) Kernel Tricks

- **Mercer's condition**
  - **To expand Kernel function K(x,y) into a dot product, i.e. $K(x,y)=\Phi(x)\cdot\Phi(y)$, K(x, y) has to be positive semi-definite function, i.e., for any function f(x) whose $\int f^2(x)dx$ is finite, the following inequality holds**

$$\int dxdyf(x)K(x,y)f(y) \geq 0$$

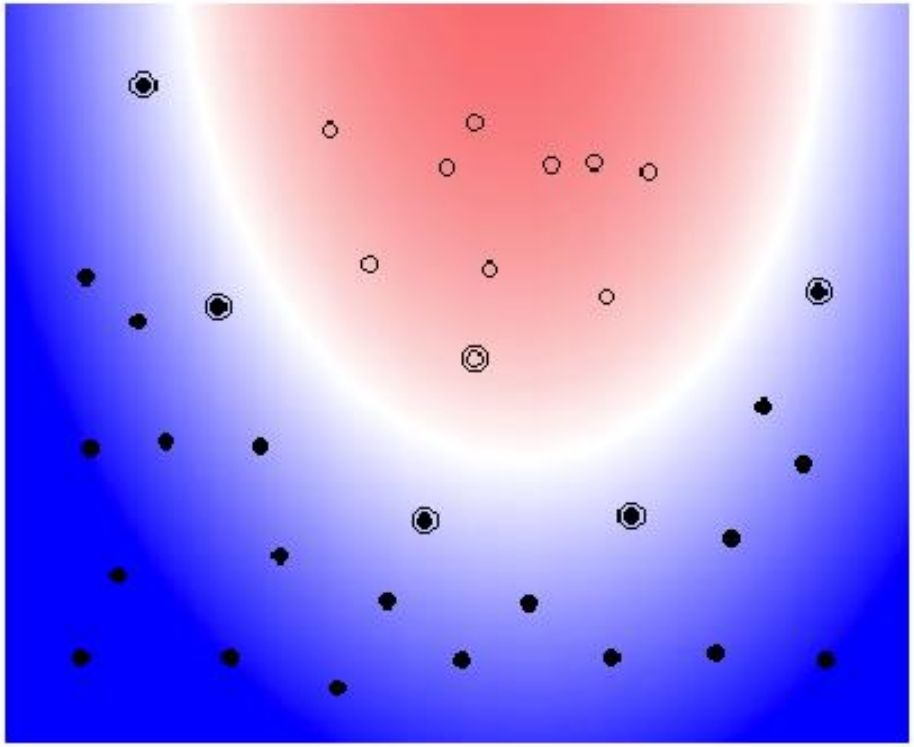
  - **Could K(a,b) = $(a-b)^3$ be a kernel function ?**
  - **Could K(a,b) = $(a-b)^4 - (a+b)^2$ be a kernel function?**
  - **Could $K(\vec{x},\vec{y}) = \left(\sum_i x_i y_i\right)^p$ be a kernel function?**
- **Pro**
  - **Introducing nonlinearity into the model**
  - **Computational cheap**
- **Con**
  - **Still have potential overfitting problems**

Support Vector Machine (SVM) Nonlinear Kernel (I)

Example: SVM with Polynomial of Degree 2

Kernel: $K(\vec{x}_i,\vec{x}_j) = [\vec{x}_i \cdot \vec{x}_j + 1]^2$

plot by Bell SVM applet

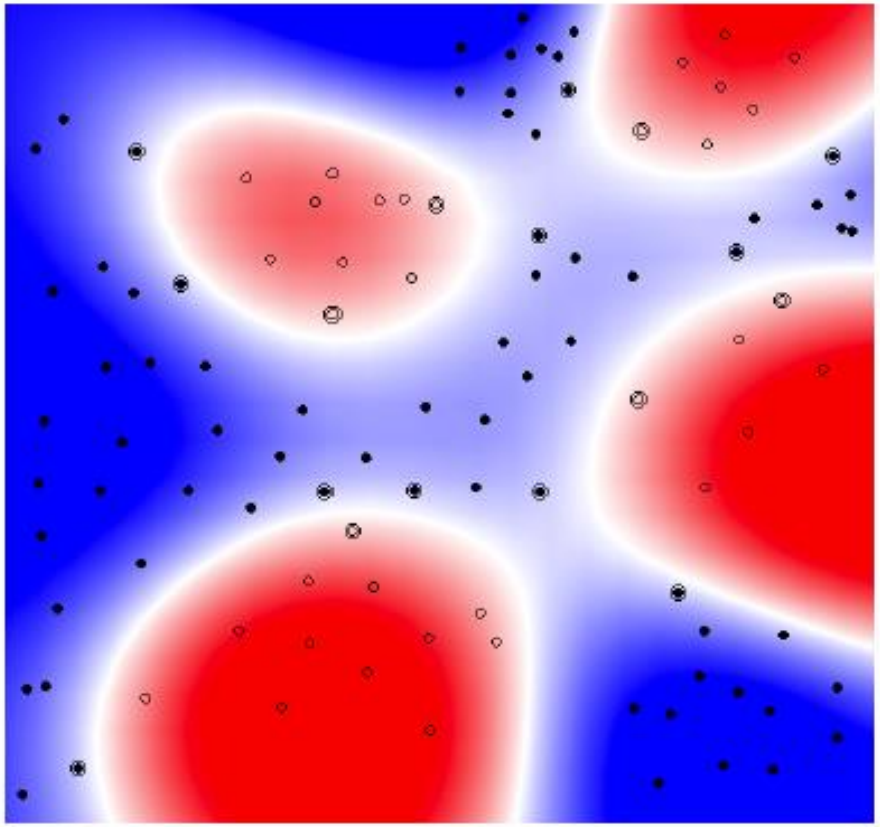


Support Vector Machine (SVM) Nonlinear Kernel (II)

Example: SVM with RBF-Kernel

Kernel: $K(\vec{x}_i,\vec{x}_j) = \exp(-|\vec{x}_i - \vec{x}_j|^2/\sigma^2)$

plot by Bell SVM applet



Support Vector Machine (SVM)
Kernelize Logistic Regression

$$p(y \mid \vec{x}) = \frac{1}{1+\exp(-y\vec{x}\cdot\vec{w})}$$

$$l_{reg}(\vec{\alpha}) = \sum_{i=1}^{N} \log \frac{1}{1+\exp(-y\vec{x}\cdot\vec{w})} - c\sum_{k=1}^{N} w_k^2$$

How can we introduce the nonlinearity into the logistic regression?

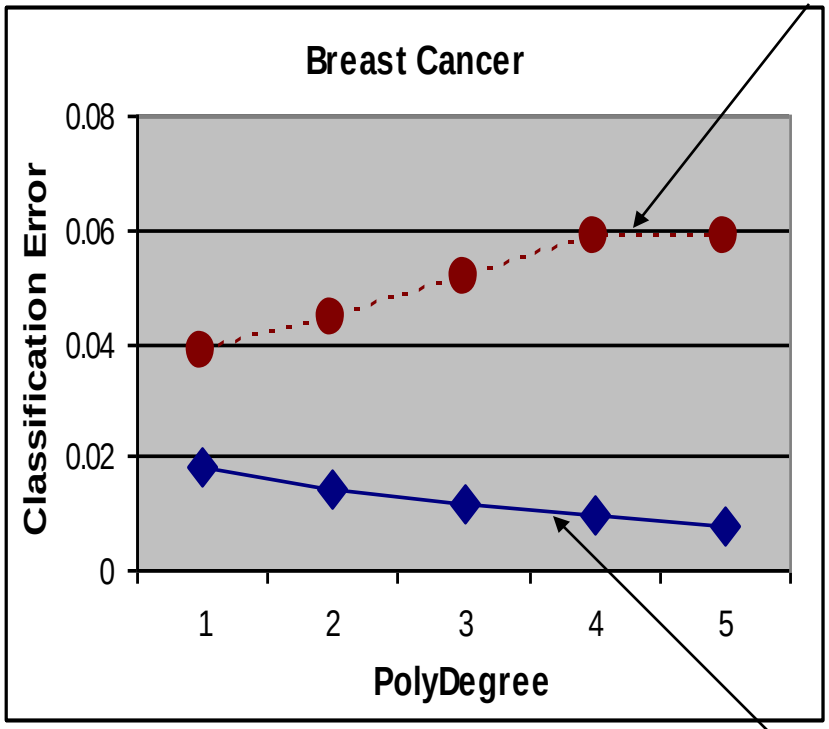
$$\vec{x} \to \vec{\phi}(\vec{x}),\ \vec{w} = \sum_{i=1}^{N} \alpha_i \vec{\phi}(\vec{x}_i) \quad , \quad K(\vec{w},\vec{x}) = \sum_{i=1}^{N} \alpha_i K(\vec{x}_i,\vec{x})$$

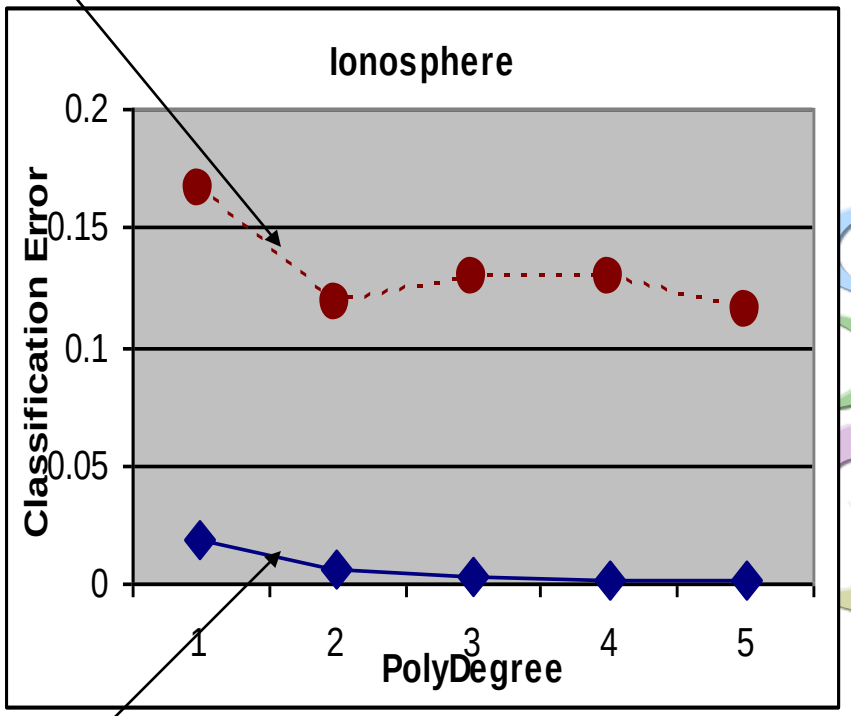
$$p(y \mid \vec{x}) = \frac{1}{1+\exp(-yK(\vec{x},\vec{w}))} = \frac{1}{1+\exp\left(-y\sum_{i=1}^{N} \alpha_i K(\vec{x}_i,\vec{x})\right)}$$

$$l_{reg}(\vec{\alpha}) = \sum_{i=1}^{N} \log \frac{1}{1+\exp\left(-y_i\sum_{j=1}^{N} \alpha_j K(\vec{x}_j,\vec{x}_i)\right)} - c\sum_{i,j=1}^{N} \alpha_i\alpha_j K(\vec{x}_i,\vec{x}_j)$$

Support Vector Machine (SVM)
Overfitting in SVM

Testing Error





Training Error

Support Vector Machine (SVM) Diffusion Kernel

- **Kernel function describes the correlation or similarity between two data points**

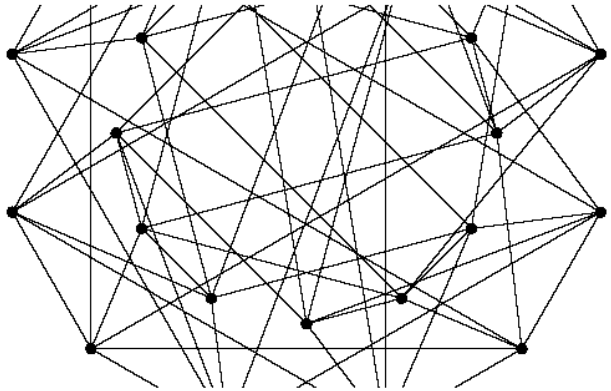
- **Given that I have a function s(x,y) that describes the similarity between two data points. Assume that it is a non-negative and symmetric function. How can we generate a kernel function based on this similarity function?**

- **A graph theory approach …**

Support Vector Machine (SVM) Diffusion Kernel

- **Create a graph for the data points**
  - **Each vertex corresponds to a data point**
  - **The weight of each edge is the similarity s(x,y)**
- **Graph Laplacian**

$$L_{i,j} = \begin{cases} s(x_i, x_j) & i \neq j \\ -\sum_{k \neq i} s(x_i, x_k) & i = j \end{cases}$$



- **Properties of Laplacian**
  - **Negative semi-definite**

- **Consider a simple Laplacian**

$$L_{i,j} = \begin{cases} 1 & x_i \text{ and } x_j \text{ are connected} \\ -\sum_{x_k \in N(x_i)} 1 & i = j \end{cases}$$

- **Consider** $L^2, L^4, \ldots$
  - **What do these matrixes represent?**

- **A diffusion kernel**

$$K_\beta = e^{\beta L} = \lim_{n\to\infty} \left(I + \frac{\beta}{n} L\right)^n$$

Support Vector Machine (SVM) Diffusion Kernel: Properties

- **Positive definite**
- **Local relationships L induce global relationships**

$$K_\beta = e^{\beta L}, \text{ or } \frac{d}{d\beta} K_\beta = L K_\beta$$

- **Works for undirected weighted graphs with similarities**

$$s(x_i, x_j) \neq s(x_j, x_i)$$

- **How to compute the diffusion kernel** $e^{\beta L}$

Support Vector Machine (SVM) Computing Diffusion Kernel

- **Singular value decomposition of Laplacian *L***

$$\mathbf{L} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{U}^T = (\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_m) \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_m \end{pmatrix} (\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_m)^T$$

$$= \sum_{i=1}^{m} \lambda_i \mathbf{u}_i \mathbf{u}_i^T$$

$$\forall i, j : \mathbf{u}_i^T \mathbf{u}_j = \delta_{i,j}$$

- **What is $\mathbf{L}^2$?**

$$\mathbf{L}^2 = \left(\sum_{i=1}^{m} \lambda_i \mathbf{u}_i \mathbf{u}_i^T \right)^2$$

$$= \sum_{i,j=1}^{m} \lambda_i \lambda_j \mathbf{u}_i \mathbf{u}_i^T \mathbf{u}_j \mathbf{u}_j^T = \sum_{i,j=1}^{m} \lambda_i \lambda_j \mathbf{u}_i \mathbf{u}_j^T \delta_{i,j}$$

$$= \sum_{i=1}^{m} \lambda_i^2 \mathbf{u}_i \mathbf{u}_i^T$$

Support Vector Machine (SVM) Computing Diffusion Kernel

- **What about L^n ?** $\mathbf{L}^2 = \sum_{i=1}^{m} \lambda_i^n \mathbf{u}_i \mathbf{u}_i^T$

- **Compute diffusion kernel** $e^{\beta L}$

$$e^{\beta L} = \sum_{n=1}^{\infty} \frac{\beta^n L^n}{n!} = \sum_{n=1}^{\infty} \frac{\beta^n}{n!} \left(\sum_{i=1}^{m} \lambda_i^n \mathbf{u}_i \mathbf{u}_i^T \right)$$

$$= \sum_{i=1}^{m} \mathbf{u}_i \mathbf{u}_i^T \left(\sum_{n=1}^{\infty} \frac{\lambda_i^n \beta^n}{n!} \right) = \sum_{i=1}^{m} \mathbf{u}_i \mathbf{u}_i^T e^{\beta \lambda_i}$$

Support Vector Machine (SVM) Doing multi-class classification

- **SVMs can only handle two-class outputs (i.e. a categorical output variable with arity 2).**
- **What can be done?**
- **Answer: with output arity N, learn N SVM's**
  - **SVM 1 learns "Output==1" vs "Output != 1"**
  - **SVM 2 learns "Output==2" vs "Output != 2"**
  - **:**
  - **SVM N learns "Output==N" vs "Output != N"**
- **Then to predict the output for a new input, just predict with each SVM and find out which one puts the prediction the furthest into the positive region.**

Combining Classifiers

- **Just like different features capturing different properties of a pattern, different classifiers also capture different structures and relationships of these patterns in the feature space.**
  - **An empirical comparison of different classifiers can help us choose one of them as the best classifier for the problem at hand.**
  - **However, although most of the classifiers may have similar error rates, sets of patterns misclassified by different classifiers do not necessarily overlap.**
  - **Not relying on a single decision but rather combining the advantages of different classifiers is intuitively promising to improve the overall accuracy of classification.**
- **Such combinations are variously called *combined classifiers*, *ensemble classifiers*, *mixture-of-expert models*, or *pooled classifiers*.**

Combining Classifiers

- **In summary, we may have different feature sets, training sets, classification methods, and training sessions, all resulting in a set of classifiers whose outputs may be combined.**

- **Combination architectures can be grouped as:**
  - ***Parallel*: all classifiers are invoked independently and then their results are combined by a combiner.**
  - ***Serial (cascading)*: individual classifiers are invoked in a linear sequence where the number of possible classes for a given pattern is gradually reduced.**
  - ***Hierarchical (tree)*: individual classifiers are combined into a structure, which is similar to that of a decision tree, where the nodes are associated with the classifiers.**

Combining Classifiers

- **Selecting and training of individual classifiers:**
  - **Combination of classifiers is especially useful if the individual classifiers are largely independent.**
  - **This can be explicitly forced by using different training sets, different features and different classifiers.**
- **classifier combination schemes :**
  - **Majority voting (each classifier makes a binary decision (vote) about each class and the final decision is made in favor of the class with the largest number of votes),**
  - **Sum, product, maximum, minimum and median of the posterior probabilities computed by individual classifiers,**
  - **Class ranking (each class receives m ranks from m classifiers, the highest (minimum) of these ranks is the final score for that class),**
  - **Weighted combination of classifiers.**

Combining Classifiers

- **Combining Classifiers**

The basic philosophy behind the combination of different classifiers lies in the fact that even the "best" classifier fails in some patterns that other classifiers may classify correctly. Combining classifiers aims at exploiting this complementary information residing in the various classifiers.

Thus, one designs different optimal classifiers and then combines the results with a specific rule.

- **Assume that each of the, say, L designed classifiers provides at its output the posterior probabilities:**

$$P(\omega_i \mid \underline{x}), i = 1, 2, \ldots, M$$

Combining Classifiers
Majority Voting Rule

- **Majority Voting Rule:** **Assign $\underline{x}$ to the class for which there is a consensus or when at least ℓ_c of the classifiers agree on the class label of $\underline{x}$ where:**

$$\ell_c = \begin{cases} \frac{L}{2}+1, \text{ L even} \\ \frac{L+1}{2}, \text{ L odd} \end{cases}$$

otherwise the decision is rejection, that is no decision is taken.

Thus, correct decision is made if the majority of the classifiers agree on the correct label, and wrong decision if the majority agrees in the wrong label.

Combining Classifiers
Product and Sum Rule

- **Product Rule: Assign $\underline{x}$ to the class ω_i :**

$$i = \arg\max_k \prod_{j=1}^{L} P_j(\omega_k \mid \underline{x})$$

where $P_j(\omega_k \mid \underline{x})$ is the respective posterior probability of the j^{th} classifier.

- **Sum Rule: Assign $\underline{x}$ to the class ω_i :**

$$i = \arg\max_k \sum_{j=1}^{L} P_j(\omega_k \mid \underline{x})$$

Combining Classifiers
Dependency

- **Dependent or not Dependent classifiers?**
  - **Although there are not general theoretical results, experimental evidence has shown that the more independent in their decision the classifiers are, the higher the expectation should be for obtaining improved results after combination. However, there is no guarantee that combining classifiers results in better performance compared to the "best" one among the classifiers.**
- **Towards Independence: A number of Scenarios.**
  - **Train the individual classifiers using different training data points. To this end, choose among a number of possibilities:**
    - **Bootstrapping: This is a popular technique to combine unstable classifiers such as decision trees (Bagging belongs to this category of combination).**
    - **Stacking: Train the combiner with data points that have been excluded from the set used to train the individual classifiers.**
    - **Use different subspaces to train individual classifiers: According to the method, each individual classifier operates in a different feature subspace. That is, use different features for each classifier.**

Combining Classifiers
Bagging

- ***Bagging (bootstrap aggregating)*** **uses multiple versions of the training set, each created by bootstrapping the original training data.**
  - **Each of these bootstrap data sets is used to train a different component classifier.**
  - **The final classification decision is based on the vote of each component classifier.**
    - **Traditionally, the component classifiers are of the same general form (e.g., all neural networks, all decision trees, etc.) where their differences are in the final parameter values due to their different sets of training patterns.**
      - **A classifier/learning algorithm is informally called unstable if small changes in the training data lead to significantly different classifiers and relatively large changes in accuracy (like decision trees and neural networks).**
  - **In general, bagging improves recognition for unstable classifiers.**

Combining Classifiers
boosting

- **In boosting, each training pattern receives a weight that determines its probability of being selected for the training set for an individual component classifier.**
  - **Adopt a weak classifier known as the base classifier.**
    - **Employing the base classifier, design a series of classifiers, in a hierarchical fashion, each time employing a different weighting of the training samples. Emphasis in the weighting is given on the hardest samples, i.e., the ones that keep "failing".**
  - **If a training pattern is accurately classified, its chance of being used again in a subsequent component classifier is reduced.**
  - **Conversely, if the pattern is not accurately classified, its chance of being used again is increased.**
  - **The final classification decision is based on the weighted sum of the votes of the component classifiers where the weight for each classifier is a function of its accuracy.**

Combining Classifiers
AdaBoost

- **The popular *AdaBoost (adaptive boosting)* algorithm allows continuous adding of classifiers until desired low training error has been achieved.**
  - **Let $a^t(x_i)$ denote the weight of pattern x_i at trial t, where $a^1(x_i) = 1/n$ for every x_i.**
  - **At each trial $t = 1, \ldots, T$, a classifier C^t is constructed from the given patterns under the distribution at where $a^t(x_i)$ reflects occurrence probability of x_i.**
    - The error ε^t of this classifier is also measured with respect to the weights, and consists of the sum of the weights of the patterns that it misclassifies.
      - **If ε^t is greater than 0.5, the trials are terminated and T is set to $t-1$.**
      - **Conversely, if C^t correctly classifies all patterns so that ε^t is zero, the trials are terminated and T becomes t.**
      - **Otherwise, the weights a^{t+1} for the next trial are generated by multiplying the weights of patterns that C^t classifies correctly by the factor $\beta^t = \varepsilon^t/(1-\varepsilon^t)$ and then are renormalized so that $\sum_{i=1}^{n} a^t(x_i) = 1$**
  - **The boosted classifier C^* is obtained by summing the votes of the classifiers C^1,..., C^T , where the vote for classifier C^t is also weighted by $log(1/\beta^t)$).**

Combining Classifiers
AdaBoost

- **Provided that ε^t is always less than 0.5, it was shown that the error rate of C^* on the given patterns under the original uniform distribution a^1 approaches zero exponentially quickly as T increases.**

- **A succession of weak classifiers $\{C^t\}$ can thus be boosted to a strong classifier C^* that is at least as accurate as, and usually much more accurate than, the best weak classifier on the training data.**

- **However, note that there is no guarantee of the generalization performance of a bagged or boosted classifier on unseen patterns.**

End