EE6310, Image and Video Processing

Image Analysis Techniques

February 14, 2023

Image Analysis Techniques

- Image Quality Assessment
- Edge Detection
- Hough Transform
- Template matching, SIFT

Image Quality Assessment

- Big question: what determines image **quality**?
- Ultimate receiver is the **human eye – subjective judgment** is all that matters
- How to assess quality **objectively** – i.e., by an **algorithm**?

Image Quality Assessment – Subjective Quality

Figure: One method!

Image Quality Assessment – Classification

Techniques can be broadly classified into:

- Full-reference: **both** test image and reference image available (discussed in this class)
- Reduced reference: test image and **certain characteristics** of reference/distortion available
- No reference: **only** test image available

An **active** research area with **immediate practical** applications!

Image Quality Assessment – MSE/PSNR

- The **mean squared error** is the long-standing most widely used IQA method
- Definition: given **reference** image I and **test/observed** image J of size $M \times N$,
  $$MSE(I,J) = \frac{1}{NM} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} [I(i,j) - J(i,j)]^2$$
- **Peak Signal to Noise Ratio (PSNR)** is defined as: $PSNR(I,J) = 10\log_{10} \frac{L^2}{MSE(I,J)}$ where L is the range of allowable gray values

Image Quality Assessment – MSE/PSNR Pros and Cons

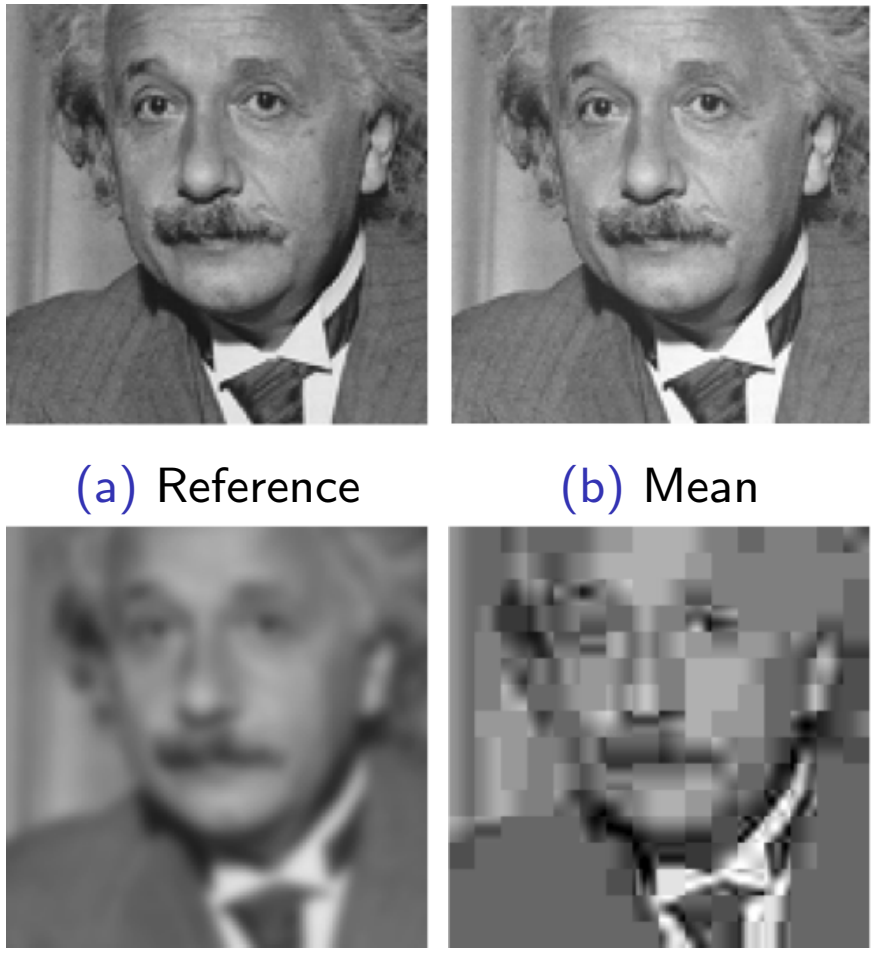
Advantages

- **Computationally simple**
- Analytically easy to work with
- Easy to **optimize** w.r.t. MSE
- **Effective** in high SNR situations

Disadvantages

- **Poor** correlation with human visual perception!

Image Quality Assessment – MSE/PSNR Failure



(a) Reference (b) Mean

(c) Blurred (d) JPEG

Figure: All images have same MSE (309)!

Image Quality Assessment – Human Subjectivity

- **Human opinion** is the ultimate gauge of image quality
- **Calibrated test conditions** required to **measure true image quality**
- The resulting **mean opinion score (MOS)** correlated with QA algorithm performance
- Until recently, MSE had **little competition** despite 40 years of research!

Image Quality Assessment – Structural Similarity Index

- Modern, successful approach: measure **loss** of **structure** in a distorted image
- Basic idea: combine local measures of similarity of **luminance, contrast, structure** into local measure of quality
- Perform **weighted average** of local measure across image

Image Quality Assessment – Structural Similarity Index

The structural similarity (SSIM) index between I and J at a point (i, j) is expressed as

$$\text{SSIM}_{I,J}(i,j) = L_{I,J}(i,j)C_{I,J}(i,j)S_{I,J}(i,j) \text{ where}$$

- $L_{I,J}(i,j)$ is a measure of local **luminance similarity**
- $C_{I,J}(i,j)$ is a measure of local **contrast similarity**
- $S_{I,J}(i,j)$ is a measure of local **structure similarity**

Image Quality Assessment – Luminance Similarity

Luminance similarity is defined as

$$L_{I,J}(i,j) = \frac{2\mu_I(i,j)\mu_J(i,j)+C_1}{\mu_I(i,j)^2+\mu_J(i,j)^2+C_1} = \frac{2\mu_I\mu_J+C_1}{\mu_I^2+\mu_J^2+C_1}$$

where

$$\mu_I(i,j) = \sum_{p=-P}^{P}\sum_{q=-O}^{Q} w(p,q)I(i+p,j+q),$$

$$\sum_{p=-P}^{P}\sum_{q=-Q}^{Q} w(p,q) = 1.$$

$w(p,q)$ is an **isotropic, unit area weighting function** and C_1 is a stabilizing constant

Image Quality Assessment – Contrast Similarity

Contrast similarity is defined as

$$C_{I,J}(i,j) = \frac{2\sigma_I(i,j)\sigma_J(i,j)+C_2}{\sigma_I(i,j)^2+\sigma_J(i,j)^2+C_2} = \frac{2\sigma_I\sigma_J+C_2}{\sigma_I^2+\sigma_J^2+C_2}$$

where

$$\sigma_I(i,j) = \sqrt{\sum_{p=-P}^{P}\sum_{q=-O}^{Q} w(p,q)[I(i+p,j+q)-\mu_I(i,j)]^2},$$

C_2 is a stabilizing constant

Image Quality Assessment – Structural Similarity

Structural similarity is defined as

$$S_{I,J}(i,j) = \frac{\sigma_{I,J}(i,j)+C_3}{\sigma_I(i,j)\sigma_J(i,j)+C_3} = \frac{\sigma_{I,J}+C_3}{\sigma_I\sigma_J+C_3}$$

where

$\sigma_{I,J}(i,j) =$

$$\sum_{p=-P}^{P}\sum_{q=-O}^{Q} w(p,q)[I(i+p,j+q)-\mu_I(i,j)][J(i+p,j+q)-\mu_J(i,j)],$$

C_3 is a stabilizing constant

Image Quality Assessment – SSIM Properties

- Symmetry: $\mathrm{SSIM}_{I,J}(i,j) = SSIM_{J,I}(i,j)$
- Boundedness: $-1 \leq \mathrm{SSIM}(I,J) \leq 1$
- Unique maximum: $\mathrm{SSIM}(i,j) = 1$ iff I, J are locally identical: $I \diamond w = J \diamond w$
- C_1, C_2, C_3 are **stabilizers** in case the local means or contrasts are **very small**
- For 8-bit gray scale images, $C_1 = (0.01 * 255)^2$, $C_2 = (0.03 * 255)^2$, $C_3 = C_2/2$ work well

Image Quality Assessment – SSIM Map

- Displaying SSIM(i, j) as an image is called an **SSIM Map**. It is an effective way of **visualizing** where the images I, J differ
- The **SSIM map** depicts where the quality of one image differs from the other

Image Quality Assessment – SSIM Map Example

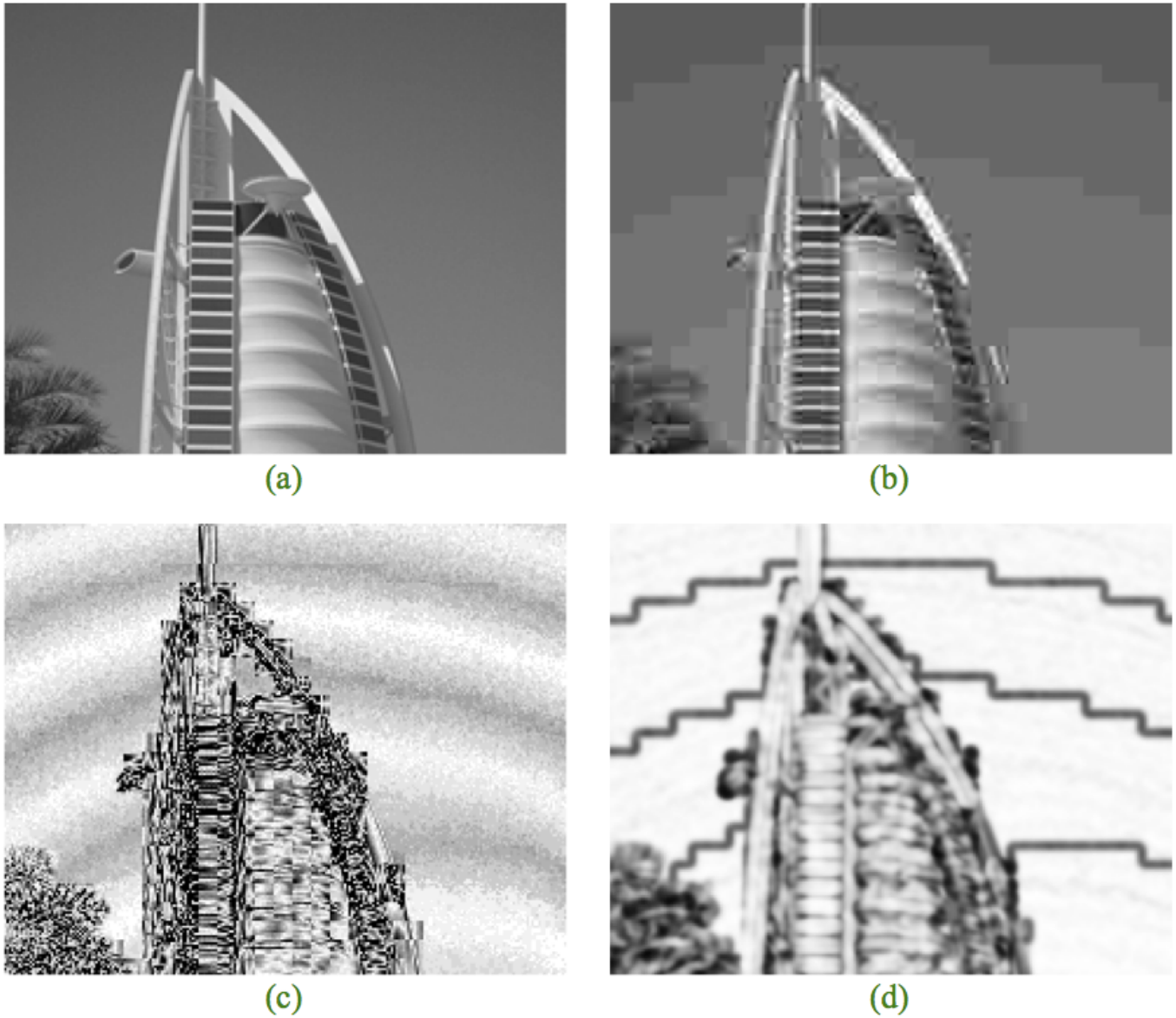


Figure: a: Reference; b: JPEG; c: Absolute diff; d: SSIM map

Image Quality Assessment – SSIM Map Example

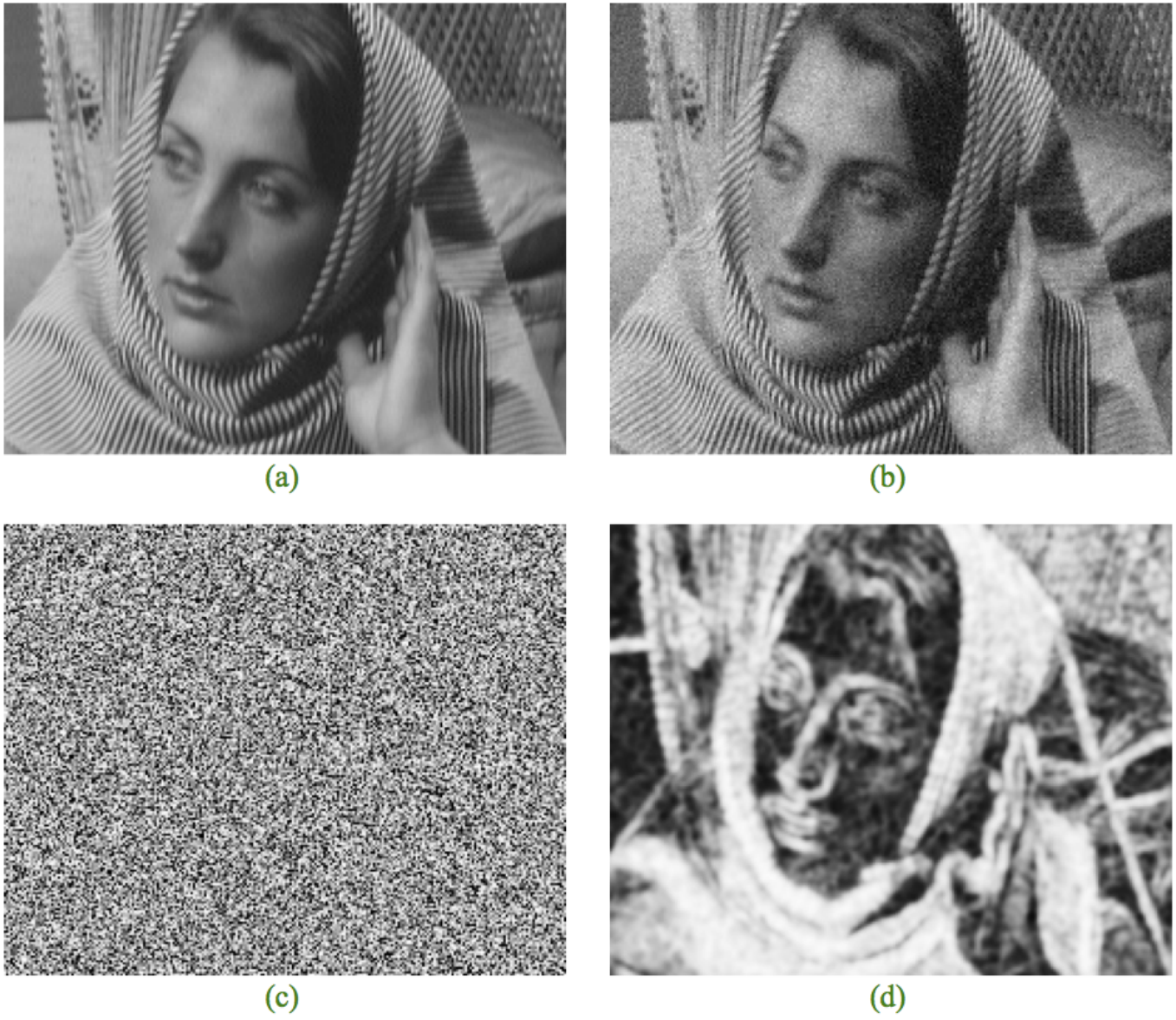


Figure: a: Reference; b: AWGN; c: Absolute diff; d: SSIM map

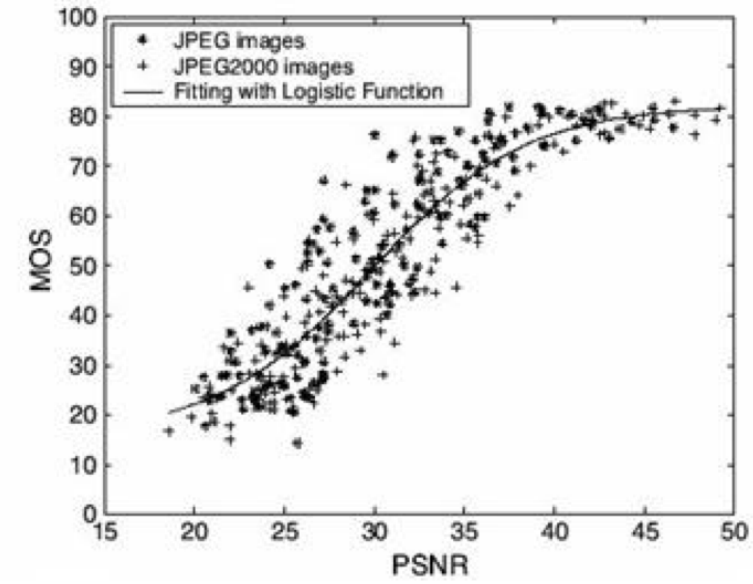
Image Quality Assessment – SSIM Map

- The mean SSIM is the average value of SSIM(i, j) over the entire image:

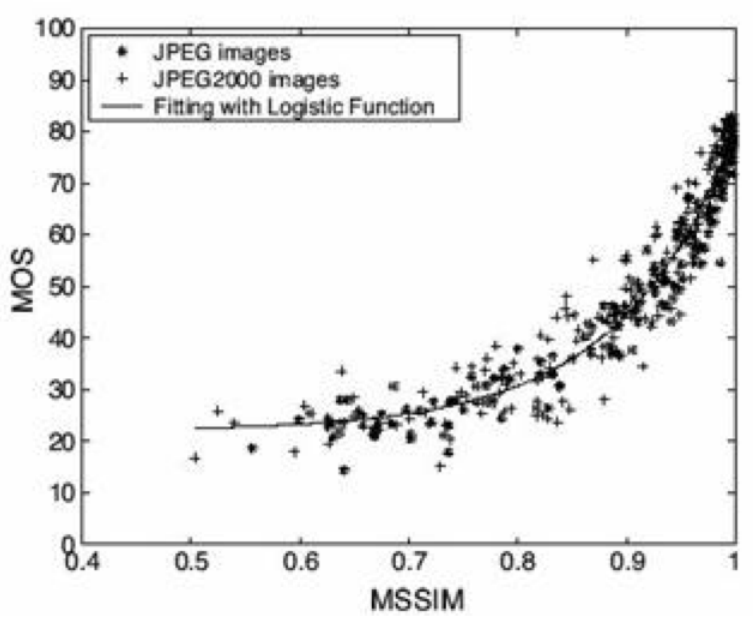
  $$\text{SSIM}(I, J) = \frac{1}{NM} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} SSIM_{I,J}(i, j)$$

- The mean SSIM correlates extraordinarily well with human response as measured by mean opinion score (MOS)

Image Quality Assessment – SSIM vs PSNR

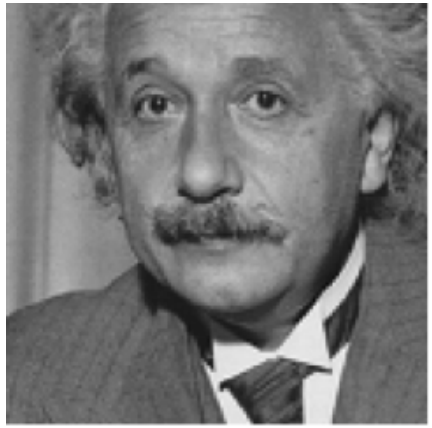


(a) PSNR vs MOS

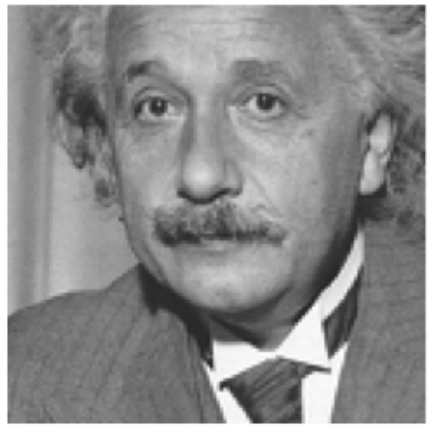


(b) MSSIM vs MOS

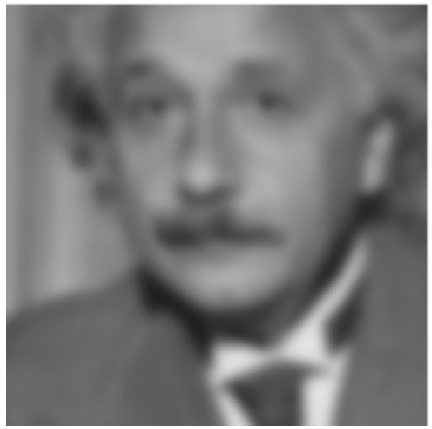
Image Quality Assessment – MSSIM Example



(a) Reference, MSSIM = 1



(b) Mean, MSSIM = 0.91



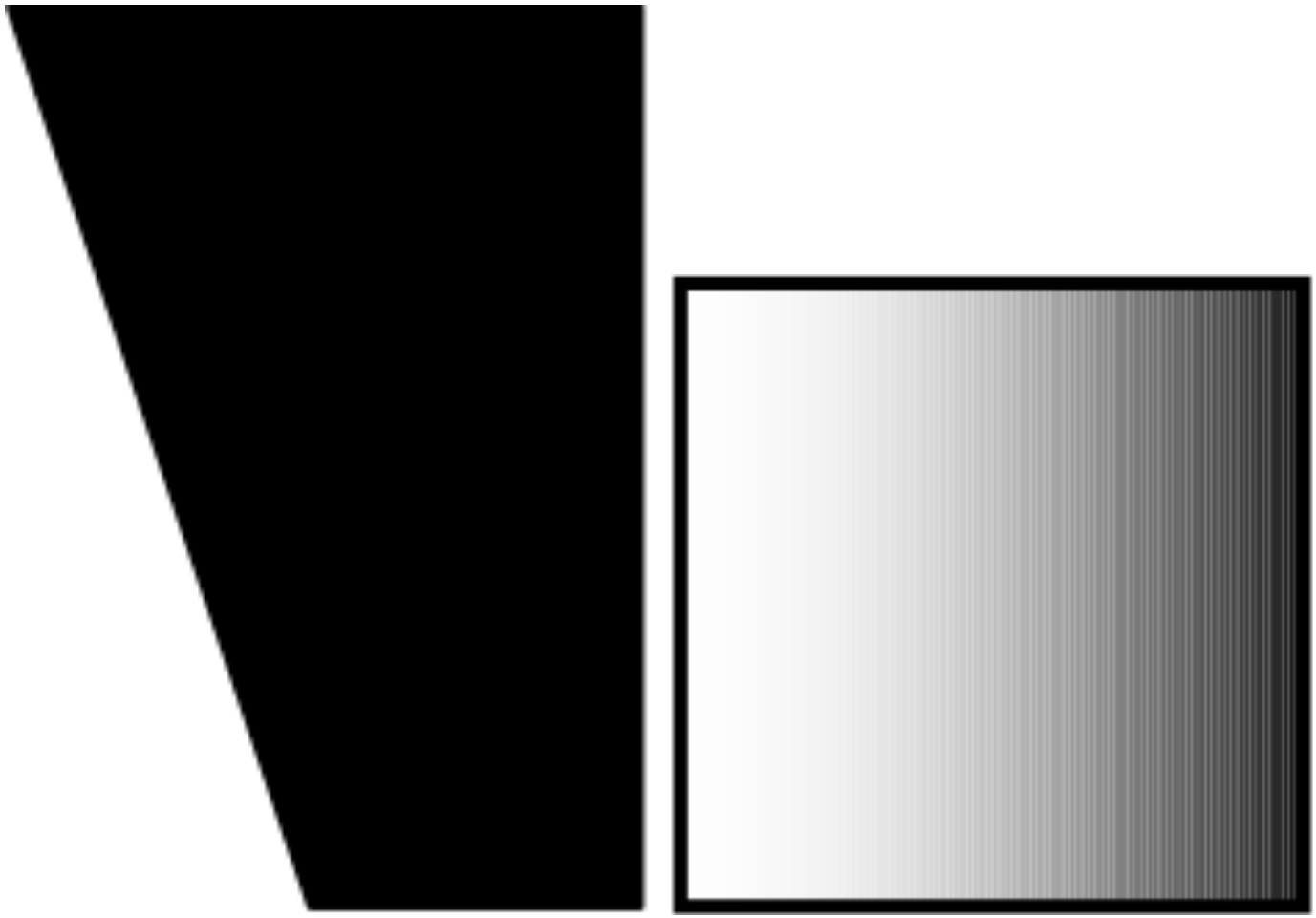
(c) Blurred, MSSIM = 0.64

(d) JPEG, MSSIM = 0.58

Figure: All images have same MSE (309)!

Edge Detection – Edge Examples

Where are the edges?



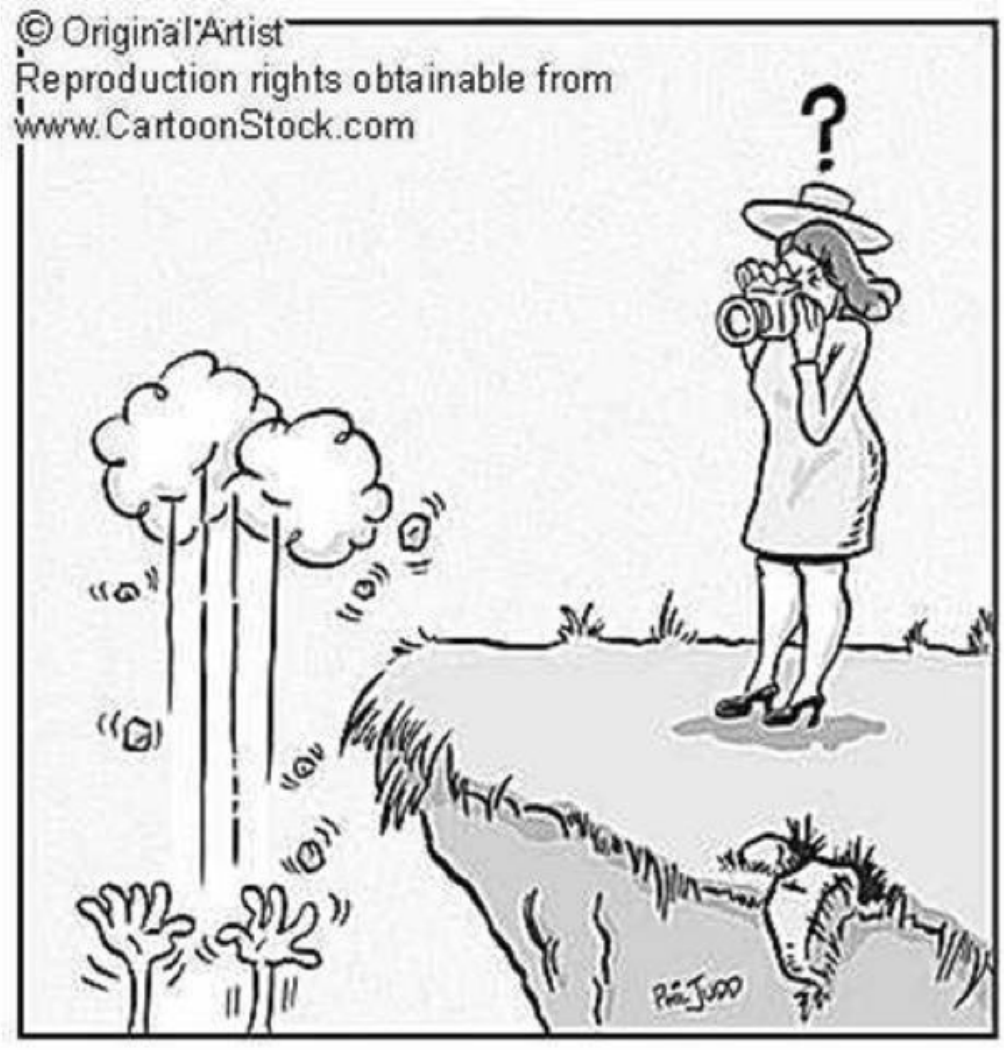
Edge Detection – Introduction

- Probably the **most exhaustively** studied topic in image analysis
- **Edges** are sudden, sustained changes in **AVERAGE** image intensity that extent along a countour
- **Edge detection** used as a precursor to most practical image analysis tasks
- Reasons:
  - **Enormous information** contained in image edges. Image can largely be recognized from edges alone!
  - An **edge map** requires **much less** storage than the image itself – it is a **binary** contour plot

Edge Detection – Overview of Methods

- Methods for edge detection studied since mid-1960s
- Easily the most studied problem in image analysis!
- **Hundreds** of approaches exist – based on a myriad of **math techniques**
- Problem became well understood in the 1980s
- In this class, we'll study **three fundamental methods:**
  - **Gradient** edge detectors
  - **Laplacian** edge detectors
  - **Diffusion-based** edge detectors

Edge Detection – Before Computers!



Edge Detection – Gradient Edge Detectors

- Oldest (Roberts 1965) but still very valuable class of edge detectors
- For a 2-D continuous function $f(x, y)$, the **gradient** is a two-element **vector**: $\nabla f(x, y) = [f_x(x, y),\ f_y(x, y)]^T$ where
  - $f_x(x, y) = \frac{\partial}{\partial x} f(x, y)$,
  - $f_y(x, y) = \frac{\partial}{\partial y} f(x, y)$

  are the **directional derivatives** of $f(x, y)$ along the x and y directions respectively

Edge Detection – Gradient Measurements

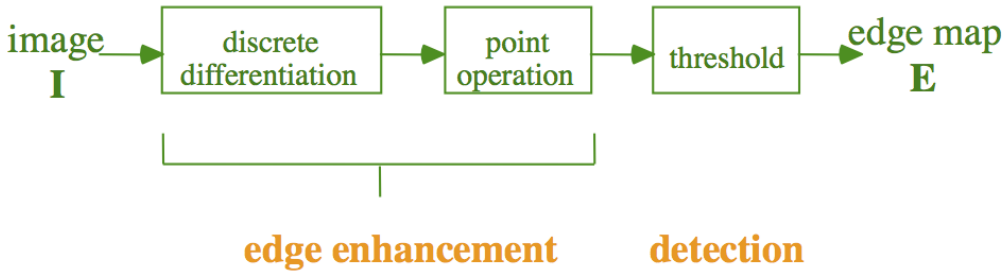
- The **direction** of the **fastest rate of change** of $f(x, y)$ at a point (x, y) is the **gradient orientation** $\theta_f(x, y) = \tan^{-1}(\frac{f_y(x,y)}{f_x(x,y)})$
- The rate of change is the **gradient magnitude** $M_f(x, y) = \sqrt{f_x(x, y)^2 + f_y(x, y)^2}$
- Gradient is appealing since we can expect edges to locally exhibit the **greatest rate of change of** what?

Edge Detection – Isotropic Gradient Magnitude

- Taking derivate in **any two perpendicular directions,** say $(x^{'}, y^{'})$ **does not change** $M_f(x, y)$
- $M_f(x, y)$ is **rotationally symmetric** or **isotropic**
- **Isotropicity** desirable since we want to detect edges regardless of orientation

Edge Detection – Gradient Edge Detector Diagram

Described by the following block diagram:



- **Digital differentiation:** digitally approximate ∇I
- **Point operation:** estimate $M_I = |\nabla I|$
- **Threshold:** decide which large values of M_I are likely locations

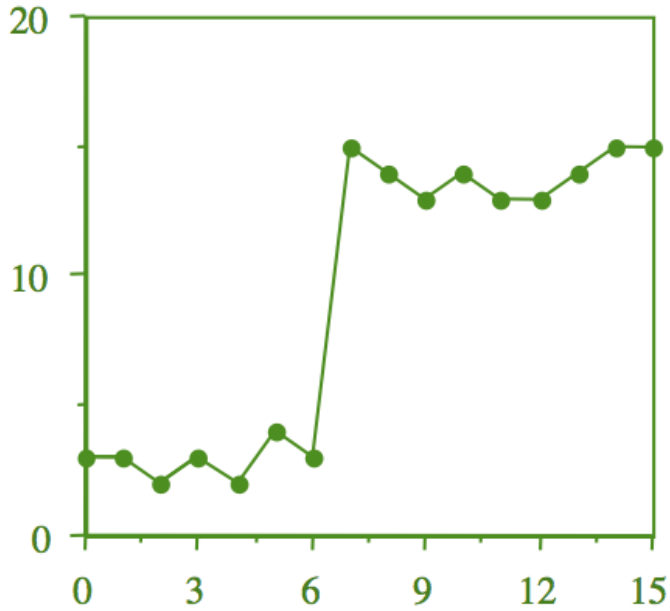
Edge Detection – Digital Differentiation

- Digital differentiation is **differencing**. For a 1-D function $f(x)$ that has been sampled to produce $f(i)$, either:
  - $\frac{d}{dx} f(x)|_{x=i} \approx f(i) - f(i-1)$ or
  - $\frac{d}{dx} f(x)|_{x=i} \approx \frac{f(i+1)-f(i-1)}{2}$
- Advantage of first method is that the current value $f(i)$ is used in computation
- Advantage of second method is that it is **centered**

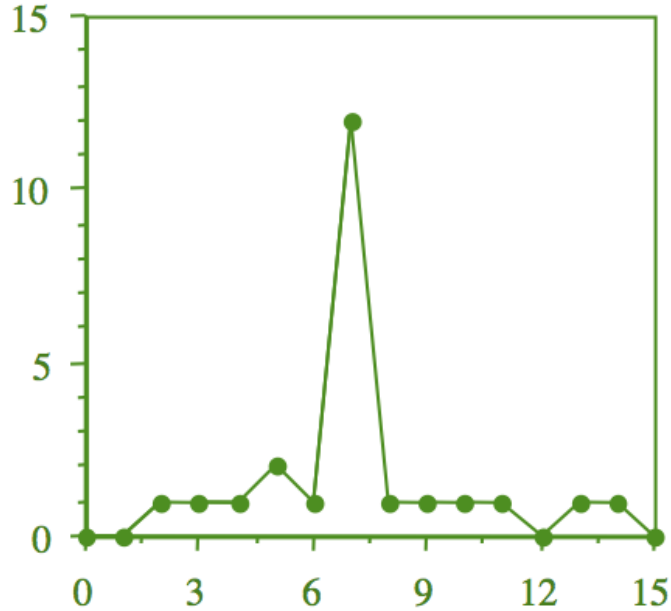
Edge Detection – 2-D Differencing

- Similarly extended to **2-D**
- Current
  - $\frac{\partial}{\partial x} f(x,y)|_{(x,y)=(i,j)} \approx f(i,j) - f(i-1,j)$ or
  - $\frac{\partial}{\partial y} f(x,y)|_{(x,y)=(i,j)} \approx f(i,j) - f(i,j-1)$
- Centered
  - $\frac{\partial}{\partial x} f(x,y)|_{(x,y)=(i,j)} \approx \frac{f(i+1,j)-f(i-1,j)}{2}$ or
  - $\frac{\partial}{\partial y} f(x,y)|_{(x,y)=(i,j)} \approx \frac{f(i,j+1)-f(i,j-1)}{2}$

Edge Detection – Example 1-D Differencing

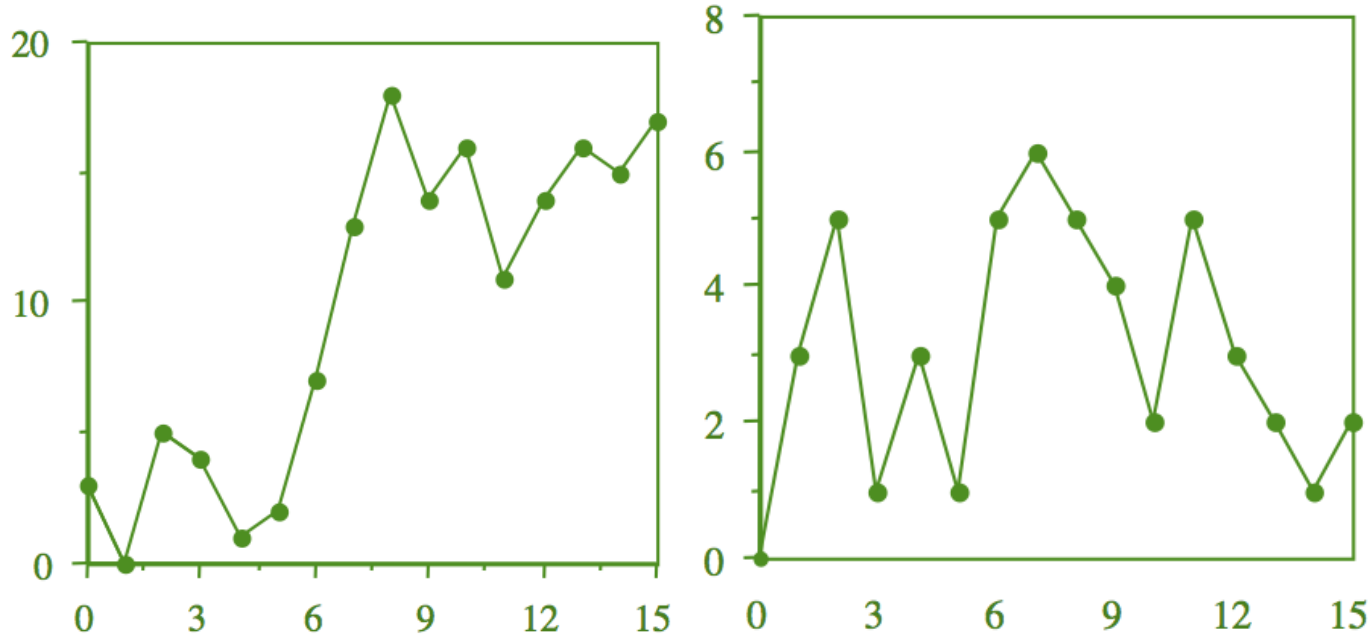


(a) $I(i)$



(b) $D_I(i) = |I(i) - I(i-1)|$

Edge Detection – Example 1-D Differencing with Noise



(a) $J(i) = I(i) + N(i)$ (b) $D_J(i) = |J(i) - J(i-1)|$

Noise is a **huge** problem because differentation **always** emphasises high frequencies (like noise)

Edge Detection – Gradient Edge Detectors

Define **convolutional edge templates** Δ_x and Δ_y that produce directional derivative estimates:

- Adjacent: $\Delta_x = \boxed{\begin{matrix} -1 & +1 \end{matrix}}$, $\Delta_y = \boxed{\begin{matrix} -1 \\ +1 \end{matrix}}$
- Centered: $\Delta_x = \boxed{\begin{matrix} -1 & 0 & +1 \end{matrix}}/2$, $\Delta_y = \boxed{\begin{matrix} -1 \\ 0 \\ +1 \end{matrix}}/2$
- Roberts: $\Delta_x = \boxed{\begin{matrix} -1 & 0 \\ 0 & +1 \end{matrix}}$, $\Delta_y = \boxed{\begin{matrix} 0 & -1 \\ +1 & 0 \end{matrix}}$

All three give very **similar** performance.

Edge Detection – Noise Reducing Detectors

Designed to reduce noise effects by averaging along rows and columns:

- Prewitt: $\Delta_x = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix}/3$, $\Delta_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{bmatrix}/3$
- Sobel: $\Delta_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}/4$, $\Delta_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}/4$

Edge Detection – Gradient Magnitude

- The **point operation** combines directional derivative estimates Δ_x and Δ_y into a single estimate of **gradient estimate**
- The usual estimates:
  - A: $M(i,j) = \sqrt{\Delta_x^2(i,j) + \Delta_y^2(i,j)}$
  - B: $M(i,j) = |\Delta_x(i,j)| + |\Delta_y(i,j)|$
  - C: $M(i,j) = \max\{|\Delta_x(i,j)|, |\Delta_y(i,j)|\}$
- The following always holds: $\mathrm{C} \leq \mathrm{A} \leq \mathrm{B}$
- A is the **correct** interpretation but B, C are **cheaper** – no squaring/root operations
- B often **overestimates** and C often **underestimates** edge magnitude

Edge Detection – Gradient Magnitude

- D: $M(i,j) = \max\{|\Delta_x(i,j)|, |\Delta_y(i,j)|\} + \frac{1}{4}\min\{|\Delta_x(i,j)|, |\Delta_y(i,j)|\}$
- D better than B or C. **Slight** difference between A and D

Edge Detection – Magnitude Thresholding

- Once edge magnitude is computed, it must be **thresholded** to find plausible edge locations
- This produces binary **edge map E:**

$$E(i,j) = \begin{cases} 1; & M(i,j) > \tau \\ 0; & M(i,j) \leq \tau \end{cases}$$

- Thus:
  - '1': edge **present** at (i,j)
  - '0': edge **absent** at (i,j)
- Threshold τ constrains the **sharpness** and **magnitude** of edges that are detected

Edge Detection – Gradient Detection Advantages

- Simple, computationally efficient
- Natural definition
- Works well on “clean” images

Edge Detection – Gradient Detection Disadvantages

- Extremely **noise-sensitive**
- Requires a **threshold** – difficult to select – usually requires **interactive selection**
- Gradient magnitude estimate **falls above threshold** over a few pixels distance from the true edge. So edges are **a few pixels wide**
- This usually requires some sort of **"edge thinning"** operation - usually heuristic
- The edge contours are often **broken** – gaps appear. This requires an **"edge linking"** operation – usually heuristic

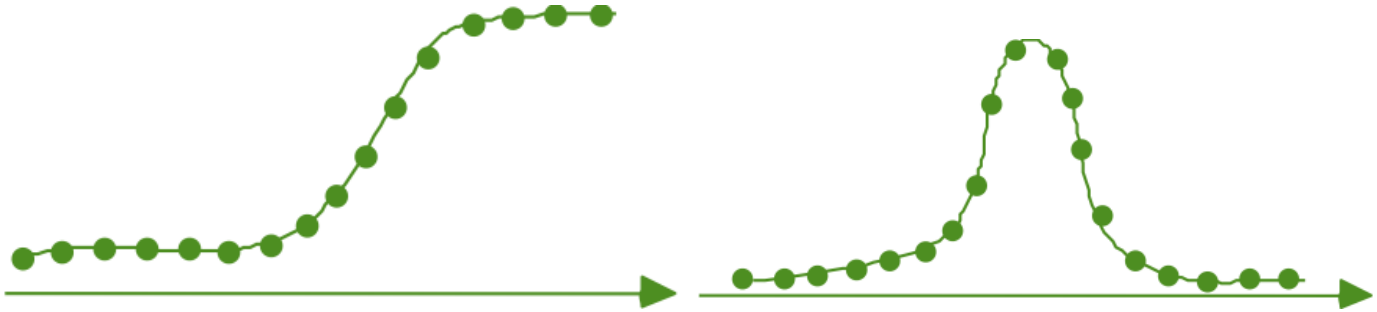
Edge Detection – Laplacian Edge Detectors

- Edge detectors are based on **second derivates**
- For a continuous 2-D function $F(x, y)$, the **Laplacian** is defined as: $\nabla^2 f(x, y) = \frac{\partial^2}{\partial x^2} f(x, y) + \frac{\partial^2}{\partial y^2} f(x, y)$
- It is a **scalar**
- **Fact:** Laplacian remains **unchanged** if directional derivates taken w.r.t. **any other set of orthogonal directions**

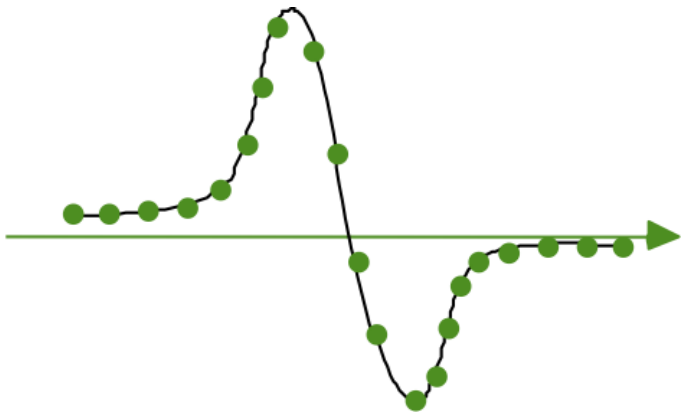
Edge Detection – Laplacian Edge Detectors

- The **Laplacian edge detection** process can be broken down into:
  - Discrete differentation that causes **edge enhancement**
  - Zero-crossing detection to **detect** edges
- **Digital differentiation:** Digitally approximate $\nabla^2 I$
- **Zero-crossing detection:** Discover where the Laplacian **crosses** the zero level

Edge Detection – Laplacian Edge Detectors



(a) 1-D edge profile (b) Differentiated once



(c) Differentiated twice

A **zero crossing** or **ZC** occurs near the center of the edge where the **slope of the slope changes sign**

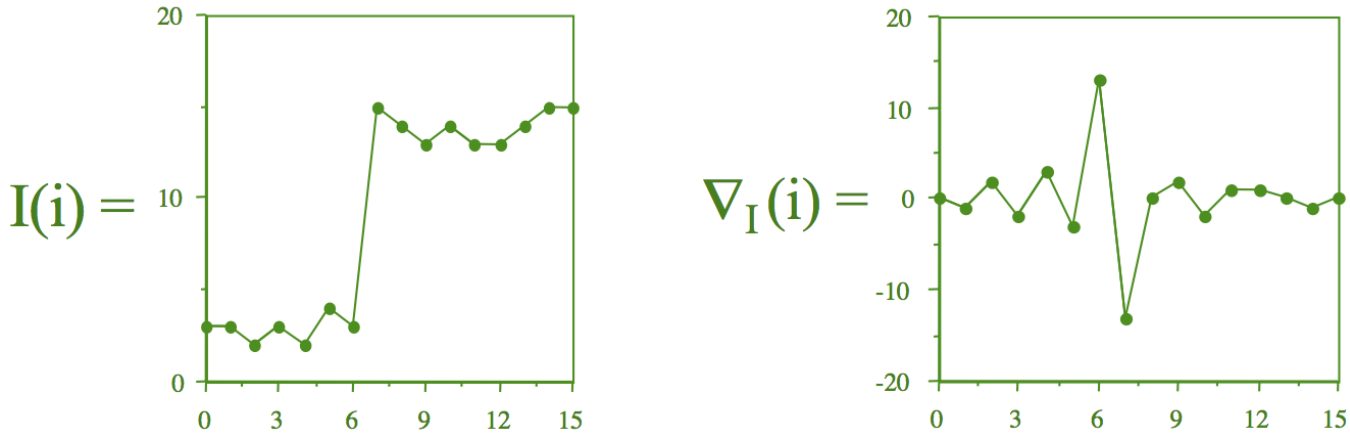
Edge Detection – Digital Twice Difference

- For a 1-D function $f(x)$ we use:
  $\frac{d}{dx} f(x)|_{x=i} \approx f(i) - f(i-1) = y(i)$
- $\frac{d^2}{dx^2} f(x)|_{x=1} \approx y(i) - y(i-1) = f(i+1) - 2f(i) + f(i-1)$
  with the **convolution template**: $\boxed{\begin{array}{ccc} 1 & -2 & 1 \end{array}}$
- In 2 dimensions: $\nabla^2 I(x,y) \approx$
  $[I(i+i,j) - 2I(i,j) + I(i-1,j)] + [I(i,j+1) - 2I(i,j) + I(i,j-1)]$
- **Convolution template:** $\boxed{\begin{array}{ccc} 1 & -2 & 1 \end{array}} + \boxed{\begin{array}{c} 1 \\ -2 \\ 1 \end{array}} =$

$$\boxed{\begin{array}{ccc} 0 & +1 & 0 \\ +1 & -4 & +1 \\ 0 & +1 & 0 \end{array}}$$

Edge Detection – Digital Twice Difference Example

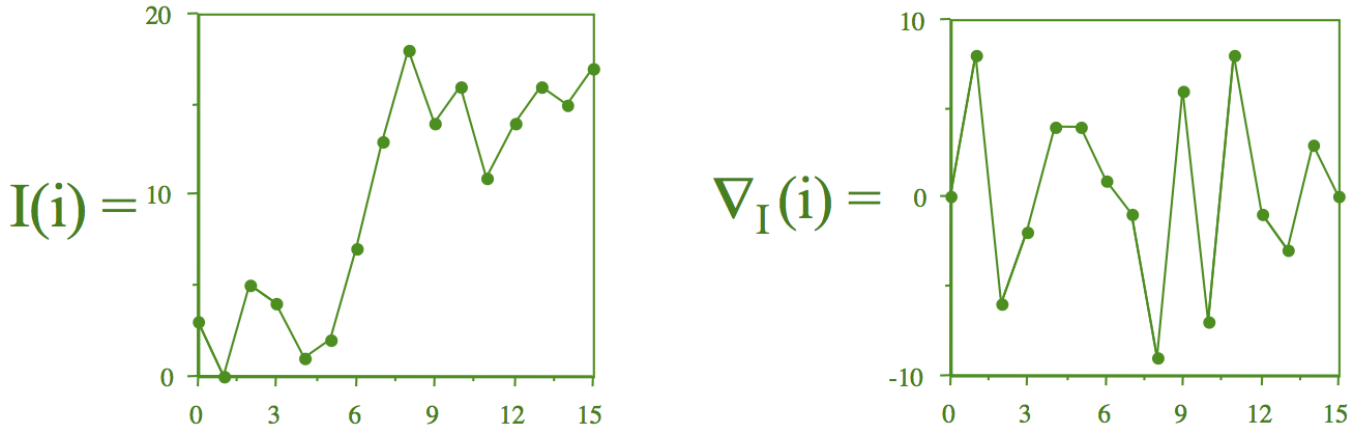
- $\nabla_I(i) = I(i+1) - 2I(i) + I(i-1)$



- Clearly reveals a sharp edge location: a single **large slope ZC** and several smaller ZCs

Edge Detection – Digital Twice Difference Example

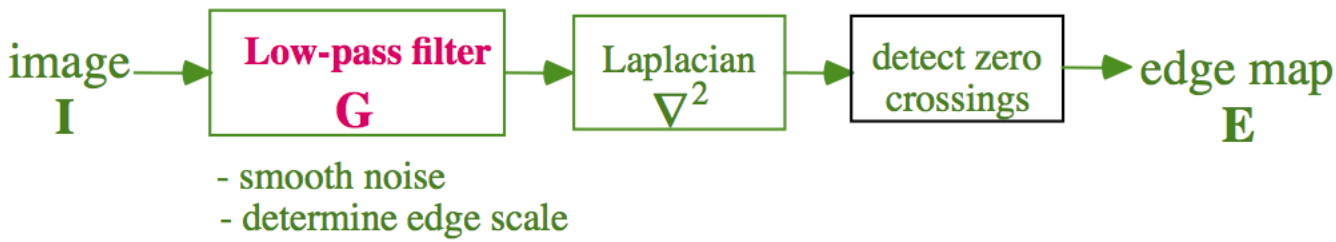
- $I(i,j) = J(i,j) + N(i,j)$



- Numerous **spurious** ZCs
- **Noise** an **even bigger problem**. Differentiating twice creates **highly amplified noise**

Edge Detection – Smoothed/Multi-scale Laplacian Edge Detection

- Laplacian operator too noise sensitive to be practical
- A simple modification makes it very powerful
- Basic idea:



- Difference: a **linear blur** applied prior to Laplacian operator

Edge Detection – Low pass pre-filter

- The **main purpose** of a low pass pre-filter to the Laplacian is to **attenuate high-frequency noise** while retaining significant image structure
- The **secondary purpose** of smoothing is to **constrain the scale** over which edges are detected
- A **high pass** operation followed by a **low pass** operation results in a **band pass** filter if their passbands overlap

Edge Detection – Gaussian pre-filter

- A lot of research has been done on how to select the low pass filter G
- It has been found that the optimal smoothing filter in the following two senses:
  1. best **edge location accuracy**
  2. **maximum** signal-to-noise ratio (SNR)

  is a **Gaussian** filter: (K is an irrelevant constant)
  $G(i,j) = K.\exp\{-[i^2 + j^2]/2\sigma^2\}$

Edge Detection – Laplacian of Gaussian Edge Detector

- Define the **Laplacian of a Gaussian** or **LoG** on I:
  $J(i,j) = \nabla^2[G(i,j) * I(i,j)] = G(i,j) * \nabla^2 I(i,j) =$
  $\nabla^2 G(i,j) * I(i,j)$
- Above 3 forms equivalent since **linear operations commute**
- Best approach: pre-compute **LoG**:
  $\nabla^2 G(i,j) \propto [1 - \frac{i^2+j^2}{\sigma^2}].\exp[-\frac{i^2+j^2}{2\sigma^2}]$ and convolve with image I
- Constant multiplier omitted since only ZC of interest to us

Edge Detection – Polar Form of LoG

- The LoG is **isotropic** and can be written in polar form as: $\nabla^2 G(r) \propto (1 - \frac{r^2}{\sigma^2}).\exp(-\frac{r^2}{2\sigma^2})$

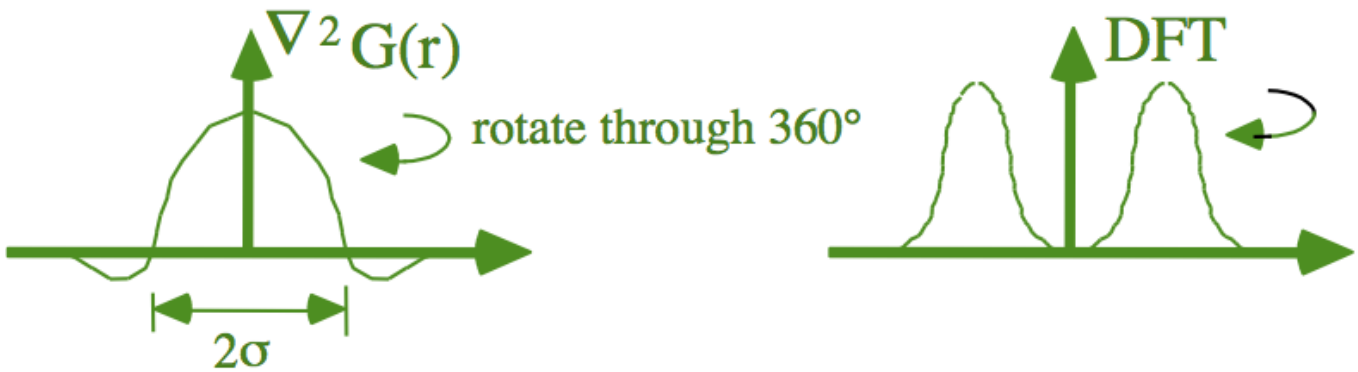
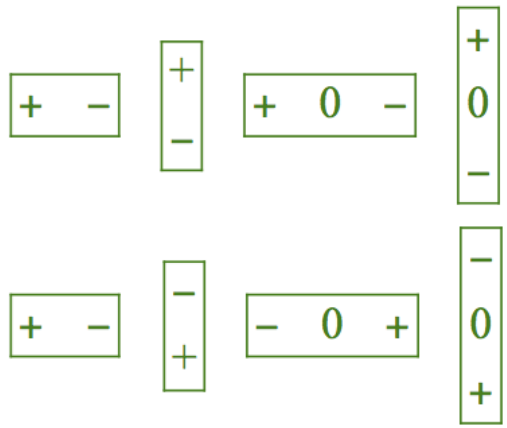


Figure: LoG in space and frequency

Edge Detection – ZC Detection

- The last stage of edge detection is **zero-crossing detection**
- Let $J = J(i, j)$ be the result of LoG filtering
- A ZC is a **crossing** of the zero level: the algorithm must search for pixel occurences of the form:



- By convention, one sign or the other is **marked** as the edge unless finer resolution (sub-pixel) is required

Edge Detection – Scale of LoG

- The larger the value of σ used, the greater the degree of **smoothing** by the low-pass pre-filter G
- If σ is **large**, then noise will be greatly smoothed - but so will **less significant edges**
- **Noise sensitivity increases** with decrease in σ but LoG detector then detects more detail

Edge Detection – Digital Implementation of LoG

- Use the **sampled LoG:**
  $\nabla^2 G(i,j) \propto [1 - \frac{i^2+j^2}{\sigma^2}].\exp[-\frac{i^2+j^2}{2\sigma^2}]$
- Specific rules of thumb should be followed
- **Enough** of $\nabla^2 G(i,j)$ must be sampled. LoG will not work unless the template contains both **main** and **minor lobes**
- In practice, the **radius** R of the LoG in space should satisfy $R \geq 4\sigma$ (in pixels)
- Adjust template to **sum to zero** by subtracting mean – why?
- LoG will not work unless $\sigma \geq 1$

Edge Detection – Advantages of LoG

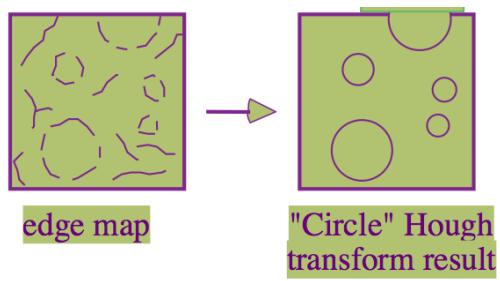
- Usually **doesn't require thresholding**
- Yields **single pixel-width edges always**
- Yields **connected edges always**
- Appears close to what happens in **biological vision**

Edge Detection – Disdvantages of LoG

- **More computation** than gradient edge detectors
- ZCs continuity property often leads to ZCs that **meander across the image**
- ZC contours tend to be **over-smooth near corners**

Hough Transform

- The **Hough Transform** is a simple, generalizable tool for finding **instances of curves of a specific shape** in a **binary edge map**
- Pictorially:



Hough Transform – Advantages

- Highly **noise-insensitive** - can "pick" edges from among many spurious edges
- Able to **reconstruct "partial" curves** containing gaps and breaks to the "ideal" form
- Can be **generalized** to **almost any desired shape**

Hough Transform – Disadvantages

- It is **computation** and **memory-intensive**

Basic Hough Transform

- Assume that it is desired to find the locations of curves that
  - Can be expressed as a function of (i, j)
  - Have a set (vector) of parameters $\mathbf{a} = [a_1, \ldots, a_n]^T$ that specify the exact **size, shape** and **location** of the curves
- Thus curves of the form: $f(i, j; \mathbf{a}) = 0$

Basic Hough Transform – Curves With Parameters

- 2-D **lines** have a **slope-intercept** form
  $f(i, j; \mathbf{a}) = j - mi - b = 0$, where $\mathbf{a} = (m, b)$ = (slope, j-intercept)
- 2-D **circles** have the form
  $f(i, j; \mathbf{a}) = (i - i_0)^2 + (j - j_0)^2 - r^2 = 0$, where $\mathbf{a} = (i_0, j_0, r)$ = (center coordinates, radius)

Basic Hough Transform

- **Input** to Hough Transform: An **edge map** or other representation of **image contour**
- From this map, a **record** is made of **possible instances** of the shape in the image
- The likelihood of a shape is found by **counting the number of edge points that fall on the shape countour**

Hough Example

- Straight line/line segment detection

- The line indicated is the "most likely" since there are **seven** pixels that contribute **evidence** for its existence
- There are other "less likely" segments

Hough Accumulator

- The **Hough accumulator A** is a **matrix** that **accumulates evidence** of instances of the curve of interest via **counting**
- The Hough accumulator **A** is **n-dimension**, where n is the number of **parameters** in **a**: $\mathbf{a} = [a_1, a_2, \ldots, a_n]^T$
- Each parameter $a_i, i = 1, \ldots, n$ can take on only a **finite** number of values N_i
- The accumulator **A** is an $N_1 \times N_2 \times \ldots N_{n-1} \times N_n$ matrix containing $N_1.N_2.\ldots.N_{n-1}.N_n$ slots

Size of Hough Accumulator

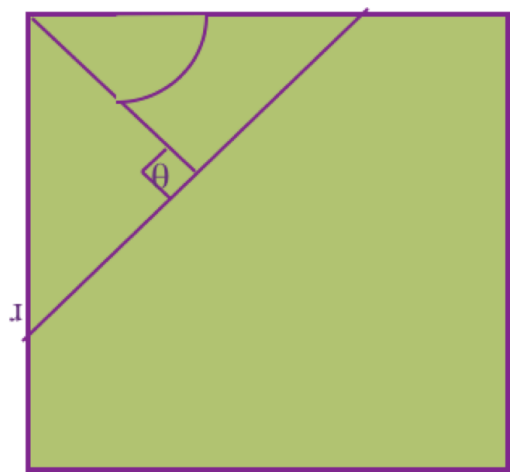
- The accumulator **A** becomes very large if:
  - Many parameters are used
  - Parameters are allowed to take many values
- The accumulator can be **much larger** than the image!
- It is practical to implement the accumulator **A** as a **single vector**, otherwise many matrix entries may always be **empty**, taking up valuable space

Accumulator Design

- The **design** of the Hough accumulator **A** is critical to keep its **dimensions** and **size** manageable
- Creating a manageable Hough accumulator is an art
- The following are **general** steps to follow

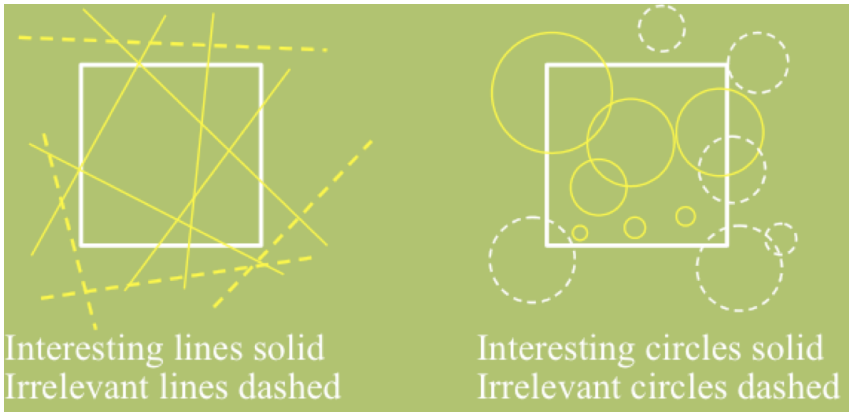
Accumulator Design – Step One

- Use **appropriate** curve equations. For example, in **line detection** the slope-intercept version is poor since **nearly vertical** lines have **large slopes**
- A better line representation: **polar form**
  $f(i, j; \mathbf{a}) = i\cos\theta + j\sin\theta - r = 0$, where $\mathbf{a} = (r, \theta) =$ (distance, angle)



Accumulator Design – Step Two

- **Bound** the parameter space to only allow the parameters for curves that **sufficiently intersect** the image
- What is "allowable" depends on the application – perhaps circles
  - Must lie **completely inside** the image
  - Must be of some **min** and **max** radius



Accumulator Design – Step Two Example

- In an $N \times N$ image indexed $0 \leq i, j \leq N-1$, **detect lines** of the form $f(i, j; \mathbf{a}) = i\cos\theta + j\sin\theta - r = 0$ that **intersect** the image
- Thus each line must intersect the four sides of the image in **two places**
- **Either** the i-intercept or the j-intercept must fall in the range $0, \ldots, N-1$:
  - i-intercept $= r/\cos\theta$
  - j-intercept $= r/\sin\theta$
- So, can bound as follows: **either**
  $0 \leq r/\cos\theta \leq N-1$ **or**
  $0 \leq r/\sin\theta \leq N-1$

Accumulator Design – Step Three

- **Quantize** the accumulator space. Curves are **digital** so the accumulator is a **finite array**
- Decide how "finely-tuned" the detector is to be
- For example, for circles the choice may be easy – let circle centers (i_0, j_0) and circle radii be **integer values** only
- For lines, the selection can be more complex since digital lines aren't always "straight"

Accumulator Design – Step Three Example

- In an $N \times N$ image, **detect circles** of the form $f(i, j; \mathbf{a}) = (i - i_0)^2 + (j - j_0)^2 - r^2 = 0$
- Radius and center constraints: $3 \leq r \leq 10$ and $r \leq i_0 \leq (N-1) - r$ and $r \leq j_0 \leq (N-1) - r$
- Assume radii and circle centers are **integers**: $r \in \{3, \ldots, 10\}$ and for **each** r the circles are completely constrained within the image: $r \leq i_0 \leq (N-1) - r$ and $r \leq j_0 \leq (N-1) - r$
- Then the accumulator contains the following number of slots: $\#$ circles types $= [N - 2r]^2 = 8N^2 - 208N + 1520 \approx 8N^2$
- This is about 8 times the image size!
- Circle counting algorithms often assume only a few radii, example in coin counting only a few coin sizes are considered

Accumulator Design – Step Four

- Initialize accumulator **A** to 0
- At each **edge coordinate** (i, j), $[E(i, j) = 1]$, increment **A**: for all accumulator elements **a** such that (ϵ small) $|f(i, j; \mathbf{a})| \leq \epsilon$ set $\mathbf{A}(\mathbf{a}) = \mathbf{A}(\mathbf{a}) + 1$ until all edge points have been examined
- **Theshold** the accumulator. Those parameters **a** such that $\mathbf{A}(\mathbf{a}) \geq \tau$ represent instances of the curve being detected
- Since local **regions** of the accumulator may fall above threshold, detection of local **maxima** is usually done

Comments on the Hough Transform

- **Substantial memory** required even for simple problems such as line detection
- **Computationally intensive;** all accumulator points are examined and potentially incremented as each image pixel is examined
- **Refinements** of the Hough transform are able to deal with these problems to some degree

Refining the Hough Transform

- The best way to reduce **both** computation **and** memory requirement is to reduce the size of the Hough array. Here is a **very general** modified approach:
  1. **Coarsely quantize** the accumulator. Define **error threshold** ϵ_0 and **variable thresholds** τ_m, ϵ_m
  2. **Increment** the Hough array wherever $(\epsilon_m > \epsilon_0)$ $|f(i, j; \mathbf{a})| \leq \epsilon_m$
  3. **Threshold** the Hough array using a threshold τ_m
  4. **Redefine** the Hough array by
     - Only allowing values similar to those $\geq \tau_m$
     - Requantize the Hough array more finely over those values
  5. Unless $\epsilon_m < \epsilon_0$, set $\epsilon_{m+1} < \epsilon_m, \tau_{m+1} < \tau_m$ and goto (2)
  6. The exact approach taken in such a strategy will vary with the application and available resources, but generally the Hough array can be made much smaller (hence faster)

Hough Transform – Unusual Circles

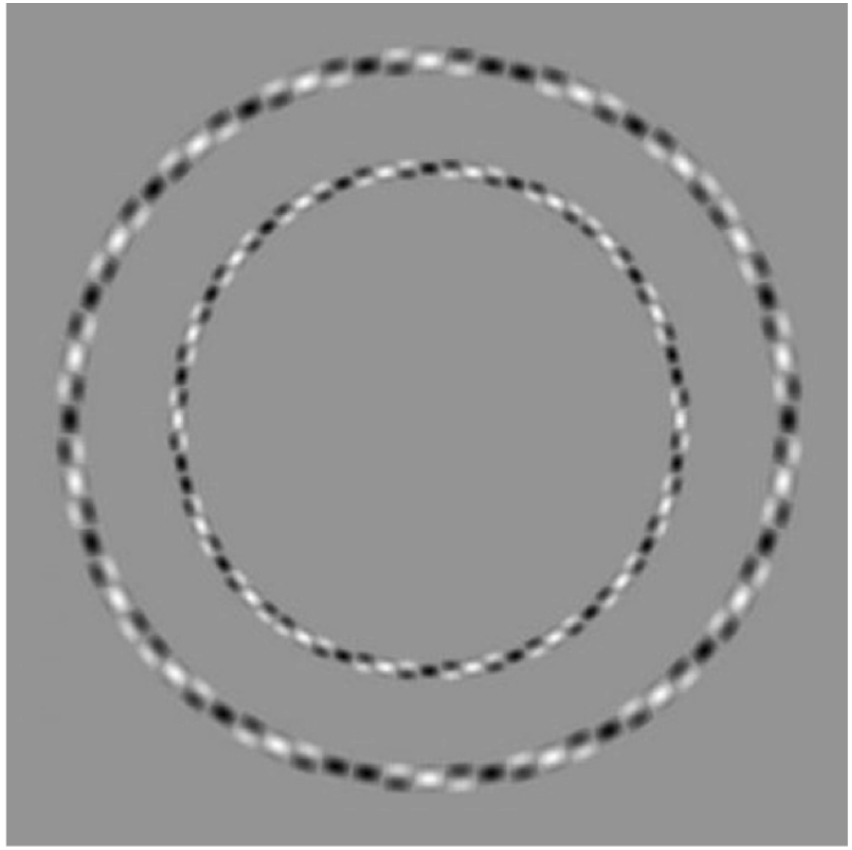


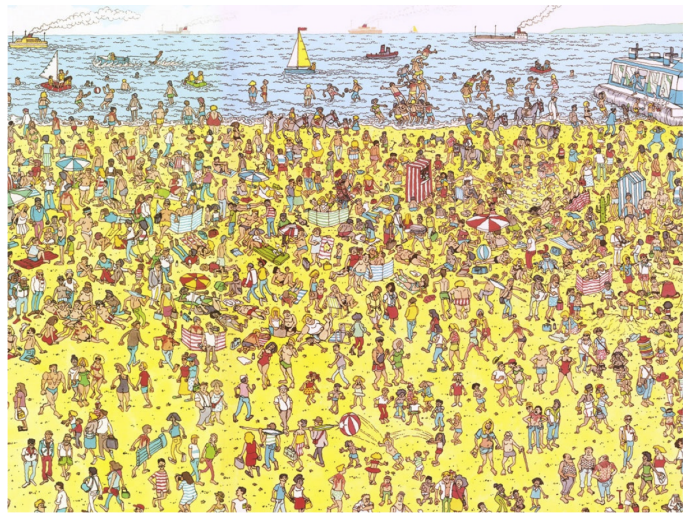
Figure: Could you find these using the Hough transform?

Template Matching

- Consider **finding instances** of a certain **sub-image**, example a certain object, written character etc
- This is called **visual search**
- Visual search is a **hard problem!**

Where's Waldo?

Waldo



Template Matching

- **Template matching** is a window-based technique to accomplish this task
- A **template** is a **sub-image:**

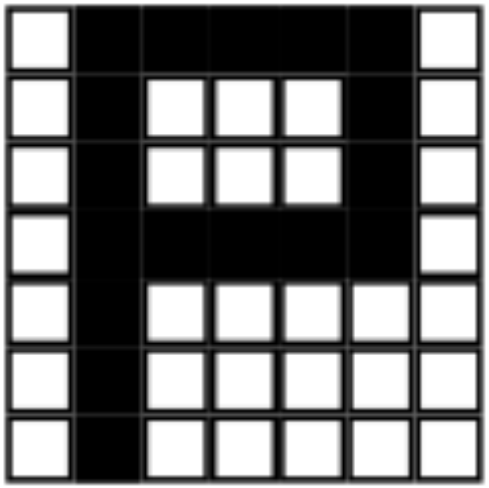


Figure: Template image of 'P'

Find The Faces

Figure: 11 faces!

Template

- Associated with a **template T** is a window $\mathbf{B_T}$ which defines its domain:
  $\mathbf{T} = \{T(m,n); (m,n) \in \mathbf{B_T}\}$
- We will also use window $\mathbf{B_T}$ to search $\mathbf{I}$
- As before, the **windowed set** at (i,j) is:
  $\mathbf{B_T} \diamond \mathbf{I}(i,j) = \{I(i+m, j+n); (m,n) \in \mathbf{B_T}\}$

Matching the Template to the Image

- Goal: Measure **goodness-of-match** of template $\mathbf{T}$ with image patches $\mathbf{B_T} \diamond \mathbf{I}(i, j)$ for all (i, j)
- Result: A **match image J** taking large values where $\mathbf{B_T} \diamond \mathbf{I}(i, j)$ and $\mathbf{T}$ are highly similar
- We start with various **mismatch measures** and will derive a **match measure** from one of them

Mismatch Measures

- **Mismatch** $\{\mathbf{B_T} \diamond \mathbf{I}(i,j), \mathbf{T}\}$
  1. Max absolute error: $\max_{(m,n)\in\mathbf{B_T}} |I(i+m, j+n) - T(m,n)|$
  2. Mean absolute error: $\frac{1}{MN} \sum_{(m,n)\in\mathbf{B_T}} \sum |I(i+m, j+n) - T(m,n)|$
  3. Mean squared error: $\frac{1}{MN} \sum_{(m,n)\in\mathbf{B_T}} \sum |I(i+m, j+n) - T(m,n)|^2$
- All three are reasonable mismatch criteria
- Only (3) leads to an easy **match measure**

Mismatch Measures

- Decompose MSE into **three terms:**
  $$\sum_{(m,n)\in \mathbf{B_T}} \sum [I(i+m, j+n)^2] \text{ -}$$
  $$2 \sum_{(m,n)\in \mathbf{B_T}} \sum I(i+m, j+n) T(m,n) + \sum_{(m,n)\in \mathbf{B_T}} \sum [T(m,n)]^2$$
  $$= E_{\mathbf{B_T} \diamond \mathbf{I}(i,j)} - 2C_{\mathbf{B_T} \diamond \mathbf{I}(i,j), \mathbf{T}} + E_{\mathbf{T}}$$
- MSE is **small** when the match is good. Only the correlation term contributes to the match measure

Cross-Correlation Matching

- **Schwarz Inequality**
$$\sum \sum_{(m,n)} A(m,n)B(m,n) \leq$$
$$\sqrt{\sum \sum_{(m,n)} [A(m,n)]^2 \sum \sum_{(m,n)} [B(m,n)]^2}$$
- Equality holds **iff** $A(m,n) = KB(m,n)$ for all (m,n) where K is **any constant**

Upper Bound on Cross-Correlation Matching

- Upper bound the cross-correlation:

$$C_{\mathbf{B_T} \diamond \mathbf{I}(i,j),\mathbf{T}} = \sum_{(m,n)\in\mathbf{B_T}} \sum I(i+m, j+n)\, T(m,n)$$

$$\leq \sqrt{\sum_{(m,n)\in\mathbf{B_T}} \sum [I(i+m, j+n)]^2 \sum_{(m,n)\in\mathbf{B_T}} \sum [T(m,n)]^2}$$

$$= \sqrt{E_{\mathbf{B_T} \diamond \mathbf{I}(i,j)} \cdot E_{\mathbf{T}}}$$

- **Equality** holds **iff** $I(i+m, j+n) = KT(m,n)$ for all $(m,n) \in \mathbf{B_T}$

Normalized Cross-Correlation

- Cross-correlation is a **good match measure** but we need to compare it with the theoretical upper bound: $\sqrt{E_{\mathbf{B_T} \diamond \mathbf{I}(i,j)} \cdot E_{\mathbf{T}}}$ or normalize with respect to it
- Define $\hat{C}_{\mathbf{B_T} \diamond \mathbf{I}(i,j),\mathbf{T}} = \frac{C_{\mathbf{B_T} \diamond \mathbf{I}(i,j),\mathbf{T}}}{\sqrt{E_{\mathbf{B_T} \diamond \mathbf{I}(i,j)} \cdot E_{\mathbf{T}}}}$
- $0 \leq \hat{C}_{\mathbf{B_T} \diamond \mathbf{I}(i,j),\mathbf{T}} \leq 1$ for all (i,j)
- Define the **normalized cross-correlation image:** $\mathbf{J} = \mathrm{CORR}[\mathbf{I}, \mathbf{T}, \mathbf{B_T}]$
- $J(i,j) = \hat{C}_{\mathbf{B_T} \diamond \mathbf{I}(i,j),\mathbf{T}}$ if $0 \leq i \leq N-1, 0 \leq j \leq M-1$

Thresholding

- A good way to compute the **best** match is to find the **largest value** in the match image
  $\mathbf{J} = \text{CORR}[\mathbf{I}, \mathbf{T}, \mathbf{B_T}]$
- This assumes there is **exactly one match**. Alternately, **threshold** the match image:
  $K(i,j) = 1$ if $J(i,j) \geq \tau$
- This will hopefully identify all points having a sufficiently good match
- If $\tau = 1$, the only **perfectly matches** will be found. Rarely happens due to noise
- τ usually chosen to be **close to** but **less than** one
- Found by trial and error

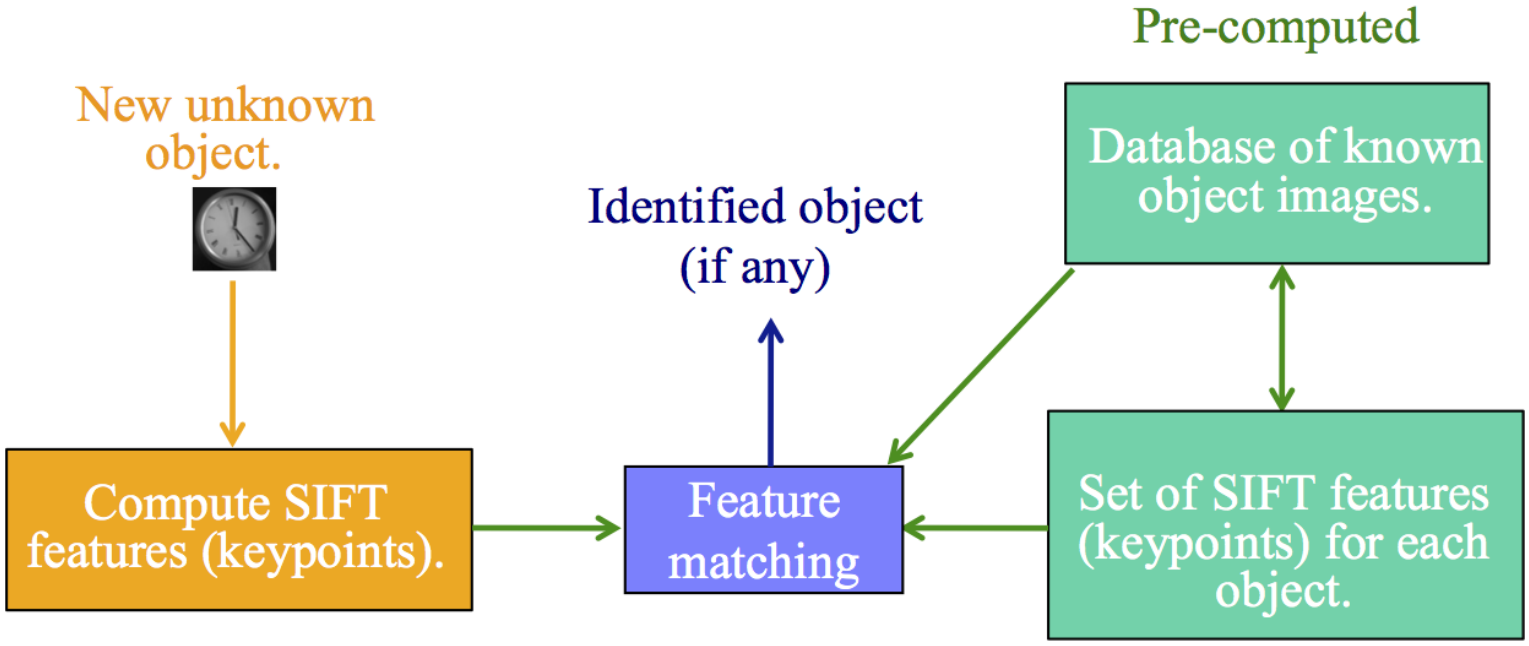
Limitations of Template Matching

- Template matching is quite **sensitive to object variations**
- Noise, rotation, scaling, stretching, occlusion and many other changes can confound the matching process

Scale Invariant Feature Transform

- **SIFT** (Scale Invariant Feature Transform) is a powerful and popular method of **visual search/object matching**
- Brainchild of Prof. David Lowe of University of British Columbia
- It can find objects that have been **rotated** and **scaled**
- **Limited ability** to match **objects** that have been **deformed**
- We will outline the **steps involved** and the **features** that are used in SIFT

SIFT – Basic Idea

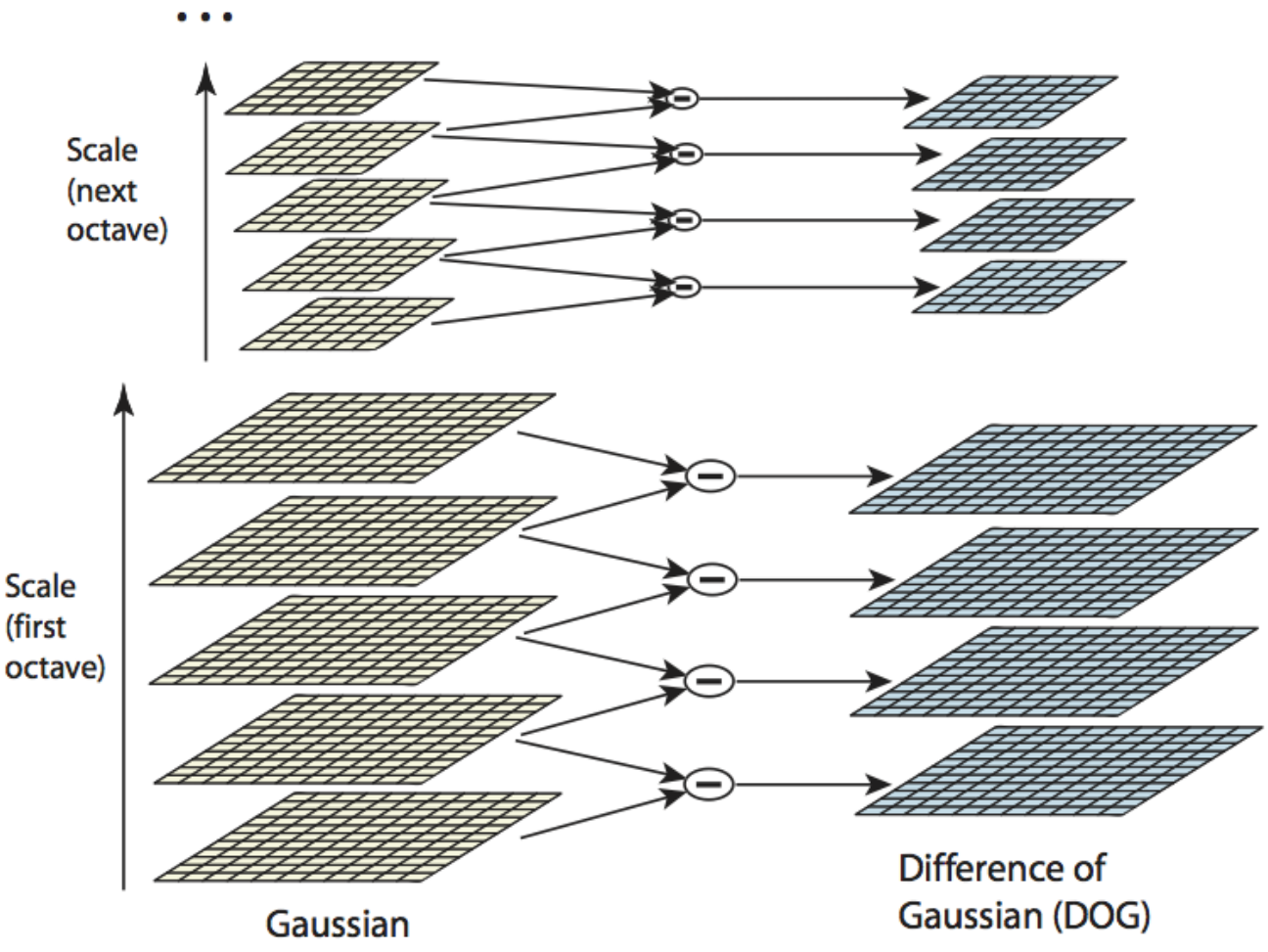


The number of "keypoint" SIFT features may be large. They are related to edges.

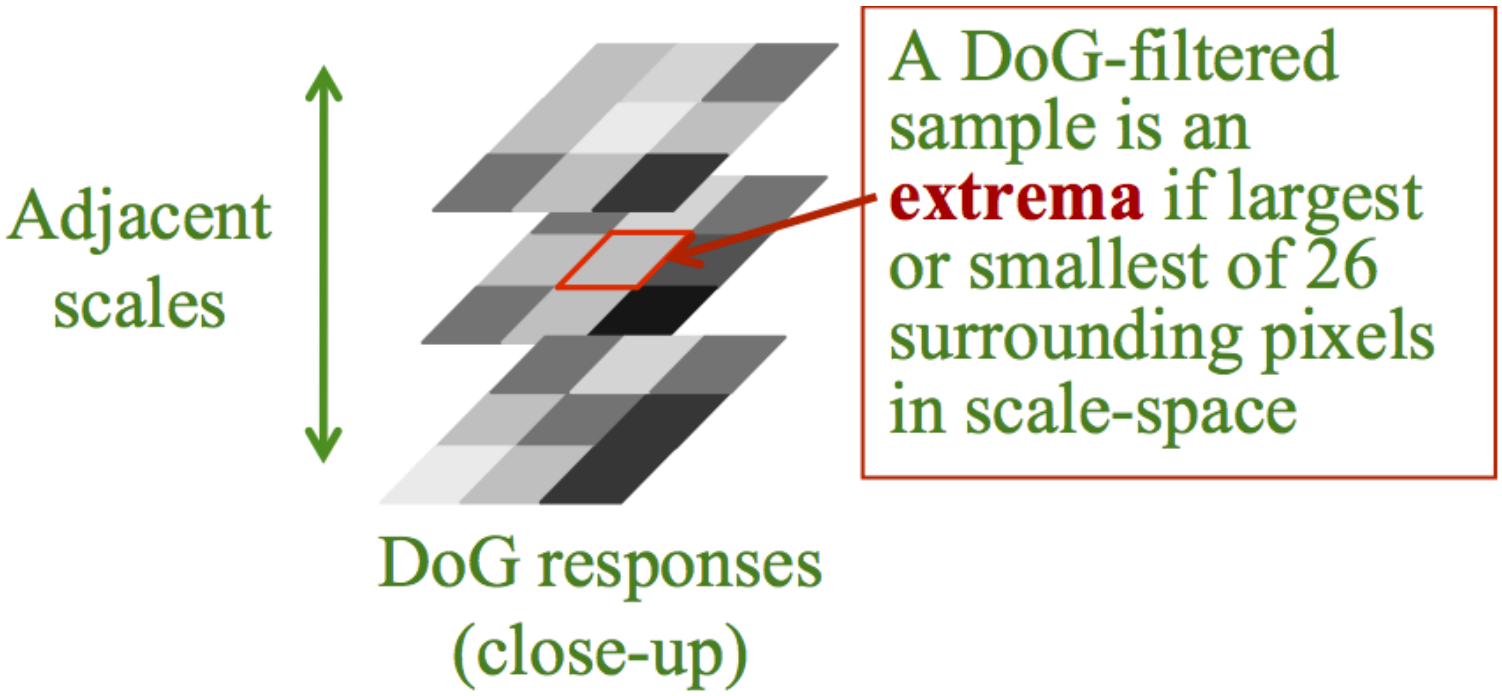
SIFT Features

- Keypoint **candidates** are the maxima of **DoG-filtered responses.** Given gaussian at **scale** σ: $G_\sigma(i,j) = \frac{1}{2\pi\sigma^2}.\exp\{-[i^2+j^2]/2\sigma^2\}$
- **DoG response** to image I is: $D_\sigma(i,j) = [G_{k\sigma}(i,j) - G_\sigma(i,j)] * I(i,j)$ where $k = 2^{(1/s)}$ and often $s = 2$
- Note: $G_{k\sigma}(i,j) - G_\sigma(i,j) \approx (k-1)\sigma\nabla^2 G_\sigma(i,j)$

DoGs Computer Over Scale



Extrema Detection



All **extrema** are found over **all DoG scales**

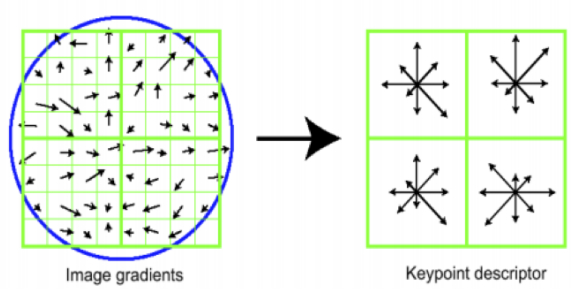
Extrema Processing

- Each "extrema" keypoint is then **carefully examined** to see whether:
  - It is actually **located on an edge**
  - If so, if the edge has a **strong enough contrast**

  Otherwise it is **rejected**
- Remaining extrema are **assigned an orientation** that is the gradient orientation of the **DoG response**

SIFT Features at Keypoints

- All **keypoints** for an image stored in the database and associated with **location, scale, and orientation**



Depiction of histograms of gradients from local patches around each keypoint.

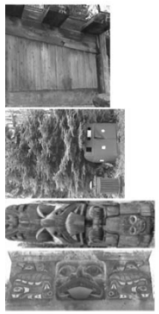
Length of each arrow is (distance-weighted) sum of gradient magnitudes near that direction.

- At the **location** and **scale** of each keypoint, the **gradient magnitudes and orientations** (**relative** to the keypoint orientation) are found at all points in a patch around the keypoint
- These are formed into **descriptors** that vectorize the **histograms** of the gradient magnitudes/orientations from sub-patches of the larger patch
- Lastly, these vectors are made **unit-length** to reduce the effects of illumination

Matching

- When a new image is to be matched, the **SIFT descriptors** are computed from it
- The database consists of a set of images, possibly large, each with **associated SIFT descriptors**
- **Matching** the **image descriptors** to the **database** requires a process of **search** which can be done in many possible ways. Lowe uses **nearest-neighbor** search (Euclidean distance)
- Lowe also uses a **Hough Transform**–like method to **cluster** and **count keypoint descriptors**. This matching process also uses keypoint location, orientation, and scale **relative** to those found in the database
- Lowe's paper has more details

Lowe's Example



Recognition results

Outer boundaries show matches and keypoints are squares

Comments on SIFT

- SIFT uses both **fundamental** and **ad-hoc** ideas. Yet, works very well within its domain (finding rigid bodies)
- It has **strong invariance** to:
  - Rotations
  - Translations
  - Scale changes
  - Illumination changes
- It has **reasonable invariance** to:
  - Occlusions
  - Affine transformations
- It has **weak invariance** to:
  - Objects that deform or change (like faces)