MSCCS 1.1

M.Sc.

(COMPUTER SCIENCE)

SEMESTER - I

(REVISED SYLLABUS AS PER NEP 2020)

APPLIED SIGNAL AND IMAGE PROCESSING

Prof. Ravindra Kulkarni
Vice-Chancellor,
University of Mumbai,

Prin. Dr. Ajay Bhamare
Pro Vice-Chancellor,
University of Mumbai,

Prof. Shivaji Sargar
Director,
CDOE, University of Mumbai,

Programme Co-ordinator : **Shri. Mandar Bhanushe**
Head, Faculty of Science and Technology,
CDOE, University of Mumbai, Mumbai

Course Co-ordinator : **Mr. Sumedh Shejole**
Asst. Professor,
CDOE, University of Mumbai, Mumbai

Editor : **Dr. Rajeshri Pravin Shinkar**
Asst. Professor,
SIES Nerul College of Arts, Science and Commerce (Autonomous)

Course Writers : **Prachi Abhijeet Surve**
Assistant Professor
Hindi Vidya Prachar Samiti's Ramniranjan Jhunjhunwala College Of Arts, Science & Commerce (Empowered Autonomous)

: **Punam Sindhu**
Assistant Professor,
Hindi Vidya Prachar Samiti's Ramniranjan Jhunjhunwala College Of Arts, Science & Commerce (Empowered Autonomous)

: **Sonali Sambare**
Assistant Professor,
SIES College of Arts, Science and Commerce (Autonomous)

: **Dr. Mitali Shewale**
Assistant Professor,
Veermata Jijabai Technological Institute, Mumbai

November 2024, Print - I

Published by : Director,
Centre for Distance and Online Education,
University of Mumbai,
Vidyanagari, Mumbai - 400 098.

DTP Composed : Mumbai University Press
Printed by Vidyanagari, Santacruz (E), Mumbai - 400 098

CONTENTS

Semester- I

Programme Name: M.Sc. Computer Science (Semester I)	**Course Name:** Applied Signal and Image Processing
Total Credits: 04	**Total Marks:** 100
College assessment: 50	**University assessment:** 50

Prerequisite: Fundamental knowledge of mathematics, digital signal processing, programming, and image processing.

Course Outcome:

- Understand and apply the fundamentals of digital signal processing and frequency domain operations for image analysis.
- Gain proficiency in image processing techniques such as intensity transformations, histogram processing, and smoothing.
- Develop skills in edge detection and image segmentation using various algorithms and approaches.
- Utilize morphological operations for image enhancement, feature extraction, and noise reduction.
- Apply advanced image processing techniques including feature detection, descriptors, and segmentation algorithms for complex image analysis and understanding.

Course Code	Course Title	Total Credits
PSCS501	**Applied Signal and Image Processing**	**04**
MODULE - I **Unit 1: Fundamentals of Digital Signals Processing** Periodic signals, Spectral decomposition, Signals, Reading and writing Waves, Spectrums, Wave objects, Signal objects ,Noise: Uncorrelated noise, Integrated spectrum, Brownian noise, Pink Noise, Gaussian noise; Autocorrelation: Correlation, Serial correlation, Autocorrelation, Autocorrelation of periodic signals, Correlation as a dot product Frequency domain Operations: Representing Image as Signals, Sampling and Fourier Transforms, Discrete Fourier Transform, Convolution and Frequency Domain Filtering, Smoothing using lowpass filters, Sharpening using high-pass filters. Fast Fourier Transforms. **Unit 2:Image Processing fundamentals and Pixel Transformation** Definition, Application of Image Processing, Image Processing Pipeline, Tools and Libraries for Image Processing, Image types and files formats. Intensity Transformations- Log Transform, Power-law Transform, Contrast Stretching, Thresholding Histogram Processing- Histogram Equalization and Histogram Matching; Linear and Non-linear smoothing of Images, Sharpening of images Image Derivative: Derivatives and gradients, Laplacian, the effect of noise on gradient computation	**02**

MODULE - II **Unit 3:Structural and Morphological Operations** Edge Detection: Sobel, Canny Prewitt, Robert edge detection techniques, LoG and DoG filters, Image Pyramids: Gaussian Pyramid, Laplacian Pyramid Morphological Image Processing: Erosion, Dilation, Opening and closing, Hit-or-Miss Transformation, Skeletonizing, Computing the convex hull, removing small objects, White and black top-hats, Extracting the boundary, Grayscale operations **Unit 4: Advanced Image Processing Operations** Extracting Image Features and Descriptors: Feature detector versus descriptors, Boundary Processing and feature descriptor, Principal Components, Harris Corner Detector, Blob detector, Histogram of Oriented Gradients, Scale-invariant feature transforms, Haar-like features Image Segmentation: Hough Transform for detecting lines and circles, Thresholding and Otsu's segmentation, Edge-based/regionbased segmentation Region growing, Region splitting and Merging, Watershed algorithm, Active Contours, morphological snakes, and GrabCut algorithms	**02**

Text Books:

1. Digital Image Processing by Rafael Gonzalez & Richard Woods, Pearson; 4th edition, 2018.
2. Think DSP: Digital Signal Processing in Python by Allen Downey, O'Reilly Media; 1st edition (August 16, 2016).

Reference Books:

1. Understanding Digital Image Processing, VipinTyagi, CRC Press, 2018.
2. Digital Signal and Image Processing by Tamal Bose, John Wiley 2010.
3. Hands-On Image Processing with Python by SandipanDey,Packt Publishing, 2018.
4. Fundamentals of Digital Images Processing by A K Jain, Pearson, 2010

Unit 1

1

FUNDAMENTALS OF DIGITAL SIGNALS PROCESSING

Unit Structure :

1.0 OBJECTIVE

In today's digital world we all are connected with all around us with the help of signals.

Digital Signal Processing (DSP) is a magical tool that uses math and computers to analyze and improve signals.

In this chapter, we'll discuss the fundamentals of DSP With a programming-based approach based on the reference book Think DSP Digital Signal Processing in Python.

The examples and supporting code for this book are in Python.

The reader knows basic mathematics, including complex numbers.

The objective of this chapter is:

- To understand the basic concepts and techniques for processing signals and digital signal processing fundamentals.
- To Understand the processes of analog-to-digital and digital-to-analog conversion and relation between continuous-time and discrete time signals and systems.
- The impetus is to introduce a few real-world signal processing applications by analyze sound recordings and other signals, and generating new sounds.

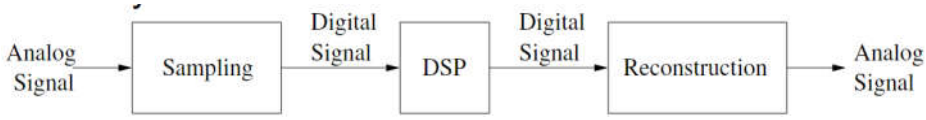
1.1 FUNDAMENTALS OF DIGITAL SIGNALS PROCESSING

What is Digital Signal Processing (DSP)?

- Digital Signal Processing (DSP) is an amazing field that uses advanced math and computers to understand and improve signals.
- Signals are everywhere in our digital world, like music, pictures, and wireless communication.
- DSP techniques are now used to analyze and process signals and data arising in many areas of engineering , science, medicine, economics and the social sciences.
- DSP is concerned with the numerical manipulation (treatment) of signals and data in sampled form. Using elementary operations as digital storage, delay, addition, subtraction and multiplication by constants, we can produce a wide variety of useful functions.
- For example to extract a wanted signal from unwanted noise, to assess the frequencies presented in a signal.

- The general purpose computer can be used for illustrating DSP theory and application.
- However, if high speed real time signal processing is required, it may use special purpose digital hardware.
- Programmable microprocessors attached to a general purpose host computer.
- Various terms are used to describe signals in the DSP environment. Discrete-time signal, we mean a signal which is defined only for a particular set of instants in time or sampling instants.

Basic DSP system:



Advantages :

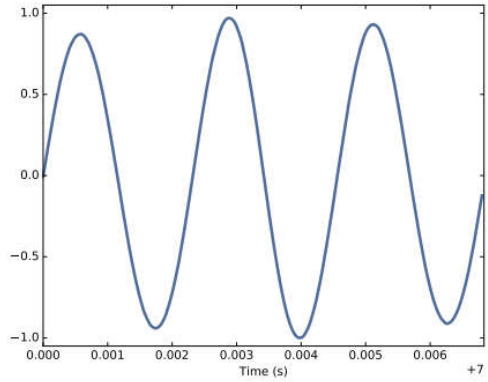
- DSP hardware is flexible and programmable
- DSP chips are relatively cheap (easily mass-produced)
- Digital storage is cheap
- Digital information can be encrypted, coded, and compressed

Disadvantages:

- Sampling leads to loss of information
- High-resolution ultra-fast A/D and D/A may be expensive
- Digital processing cannot always be done in real-time

1.1.1 Periodic signals

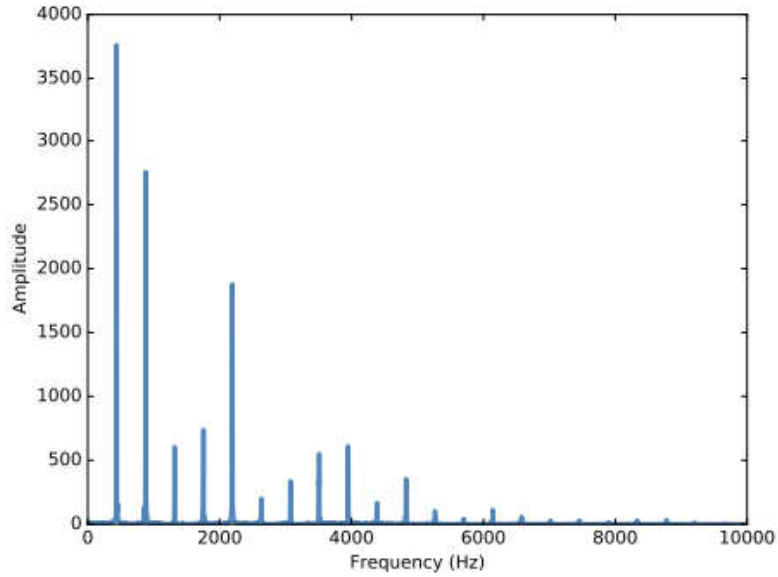
- Signals that repeat themselves after some period of time are called periodic signals.
- For example, if you strike a bell, it vibrates and generates sound.



- Segment from a recording of a bell
- Signal is periodic, but the shape of the signal is more complex.
- The shape of a periodic signal is called the waveform.
- Most musical instruments produce waveforms more complex than a sinusoid.
- Sinusoid means signal that has the same shape as the trigonometric sine function.
- The shape of the waveform determines the musical timbre, which is our perception of the quality of the sound.
- People usually perceive complex waveforms as rich, warm and more interesting than sinusoids.
- The duration to show three full repetitions of the signal known as cycles.
- The duration of each cycle, called the period, is about 2.3 ms.
- The frequency of a signal is the number of cycles per second, which is the inverse of the period.
- The units of frequency are cycles per second, or Hertz, abbreviated "Hz".

1.1.2 Spectral decomposition

- Spectral decomposition is any signal that can be expressed as the sum of sinusoids with different frequencies.
- The discrete Fourier transform (DFT) takes a signal and produces its spectrum.
- The spectrum is the set of sinusoids that add up to produce the signal.
- The Fast Fourier transform (FFT) is an efficient way to compute the DFT.
- In the following figure :
  - The x-axis is the range of frequencies that make up the signal.
  - The y-axis shows the strength or amplitude of each frequency component.



- The lowest frequency component is called the fundamental frequency.
- The fundamental frequency of this signal is near 440 Hz (actually a little lower, or "flat").
- Dominant Frequency:In this signal the fundamental frequency has the largest amplitude, so it is also the dominant frequency. Normally the perceived pitch of a sound is determined by the fundamental frequency, even if it is not dominant.
- Harmonics:The other spikes in the spectrum are at frequencies 880, 1320, 1760, and 2200, which are integer multiples of the fundamental. These components are called harmonics because they are musically harmonious with the fundamental:
  - 880 is the frequency of A5, one octave higher than the fundamental. An octave is a doubling in frequency.
  - 1320 is approximately E6, which is a perfect fifth above A5.
  - 1760 is A6, two octaves above the fundamental.
  - 2200 is approximately C]7, which is a major third above A6.
  - These harmonics make up the notes of an A major chord, although not all in the same octave.

1.1.3 Signals

- Signal is anything that carries information.
- It can also be defined as a physical quantity that varies with time, temperature, pressure or with any independent variables such as speech signal or video signal.

- Signal processing: It is the process of operation in which the characteristics of a signal are Amplitude, shape,phase,frequency,etc. undergoes a change.
- In the Python module called thinkdsp.py that contains classes and functions for working with signals and spectrums.
- To represent signals, thinkdsp provides a class called Signal, which is the parent class for several signal types, including Sinusoid, which represents both sine and cosine signals.

`thinkdsp` provides functions to create sine and cosine signals:

```
cos_sig = thinkdsp.CosSignal(freq=440, amp=1.0, offset=0)
sin_sig = thinkdsp.SinSignal(freq=880, amp=0.5, offset=0)
```

`freq` is frequency in Hz. `amp` is amplitude in unspecified units where 1.0 is defined as the largest amplitude we can record or play back.

- Signals have an __add__ method, so you can use the + operator to add them:

mix = sin_sig + cos_sig

- The result is a SumSignal, which represents the sum of two or more signals.

A Signal is basically a Python representation of a mathematical function.
Most signals are defined for all values of t, from negative infinity to infinity.

- We represent ts and ys using NumPy arrays and encapsulate them in an object called a Wave.

Where,

ts - taking a sequence of points in time, ts

and ys - computing the corresponding values of the signal, ys

- Frame:A Wave represents a signal evaluated at a sequence of points in time. Each point in time is called a frame.
- Sample:The measurement itself is called a sample, although “frame” and “sample” are sometimes used interchangeably.
- Signal provides make_wave, which returns a new Wave object:

wave = mix.make_wave(duration=0.5, start=0, framerate=11025)

Where,

duration- is the length of the Wave in seconds.

start- is the start time, also in seconds.

framerate- is the (integer) number of frames per second, which is also the number of samples per second.

1.1.4 Reading and writing Waves

- thinkdsp provides read_wave, which reads a WAV file and returns a Wave:

  violin_wave = thinkdsp.read_wave('input.wav')

- Wave provides write, which writes a WAV file:

  wave.write(filename='output.wav')

- You can listen to the Wave with any media player that plays WAV files.

- thinkdsp also provides play_wave, which runs the media player as a subprocess:

  thinkdsp.play_wave(filename='output.wav', player='aplay')

- It uses aplay by default, but you can provide the name of another player.

1.1.5 Spectrums

The signal spectrum describes a signal's magnitude and phase characteristics as a function of frequency.

The system spectrum describes how the system changes signal magnitude and phase as a function of frequency.

- Wave provides make_spectrum, which returns a Spectrum:

spectrum = wave.make_spectrum()

- And Spectrum provides plot:

spectrum.plot()

- Spectrum provides three methods that modify the spectrum:

  - low_pass applies a low-pass filter, which means that components above a given cutoff frequency are attenuated (that is, reduced in magnitude) by a factor.

  - high_pass applies a high-pass filter, which means that it attenuates components below the cutoff.

  - band_stop attenuates components in the band of frequencies between two cutoffs.

- This example attenuates all frequencies above 600 by 99%:

spectrum.low_pass(cutoff=600, factor=0.01)

- A low pass filter removes bright, high-frequency sounds, so the result sounds muffled and darker.
- To hear what it sounds like, you can convert the Spectrum back to a Wave, and then play it.

wave = spectrum.make_wave() wave.play('temp.wav')

- The play method writes the wave to a file and then plays it.
- If you use Jupyter notebooks, you can use make_audio, which makes an Audio widget that plays the sound.

1.1.6 Wave objects

- In thinkdsp.py most of the functions it provides are thin wrappers around functions from NumPy and SciPy.
- The primary classes in thinkdsp are Signal, Wave, and Spectrum.
- Given a Signal, you can make a Wave.
- Given a Wave, you can make a Spectrum, and vice versa.
- These relationships are shown in following Figure:

Figure : Relationships among the classes in thinkdsp.

- A Wave object contains three attributes:

1. ys is a NumPy array that contains the values in the signal;
2. ts is an array of the times where the signal was evaluated or sampled; and
3. framerate is the number of samples per unit of time.

- The unit of time is usually seconds or sometimes it's days.
- Wave also provides three read-only properties:

1. start,
2. end, and
3. duration.

- If you modify ts, these properties change accordingly.
- To modify a wave, you can access the ts and ys directly.

- For example:

```
wave.ys *= 2

    wave.ts += 1
```

  - The first line scales the wave by a factor of 2, making it louder.
  - The second line shifts the wave in time, making it start 1 second later.
- Wave provides methods that perform many common operations.
- For example, the same two transformations could be written:

```
wave.scale(2)

wave.shift(1)
```

- You can read the documentation of these methods and others at http://greenteapress.com/thinkdsp.html

1.1.7 Signal objects

- Signal is a parent class that provides functions common to all kinds of signals, like make_wave.
- Child classes inherit these methods and provide evaluate, which evaluates the signal at a given sequence of times.
- For example, Sinusoid is a child class of Signal, with this definition:

```
class Sinusoid(Signal):

def __init__(self, freq=440, amp=1.0, offset=0, func=np.sin):

Signal.__init__(self)

self.freq = freq

self.amp = amp

self.offset = offset

self.func = func
```

- The parameters of __init__ are:
  - freq: frequency in cycles per second, or Hz.
  - amp: amplitude. The units of amplitude are arbitrary, usually chosen so 1.0 corresponds to the maximum input from a microphone or maximum output to a speaker.
  - offset: indicates where in its period the signal starts; offset is in units of radians, for reasons I explain below.

- func: a Python function used to evaluate the signal at a particular point in time.

- It is usually either np.sin or np.cos, yielding a sine or cosine signal.
- Like many init methods, this one just tucks the parameters away for future use.
- Signal provides make_wave, which looks like this:

```
def make_wave(self, duration=1, start=0, framerate=11025):

n = round(duration * framerate)

ts = start + np.arange(n) / framerate

ys = self.evaluate(ts)

return Wave(ys, ts, framerate=framerate)
```

Where,

- start and duration are the start time and duration in seconds.
- framerate is the number of frames (samples) per second.
- n is the number of samples, and ts is a NumPy array of sample times.

- To compute the ys, make_wave invokes evaluate, which is provided by Sinusoid:

```
def evaluate(self, ts):

phases = PI2 * self.freq * ts + self.offset

ys = self.amp * self.func(phases)

return ys
```

- Let's unwind this function one step at time:

1. self.freq is frequency in cycles per second, and each element of ts is a time in seconds, so their product is the number of cycles since the start time.
2. PI2 is a constant that stores 2π. Multiplying by PI2 converts from cycles to phase. You can think of phase as "cycles since the start time" expressed in radians. Each cycle is 2π radians.
3. self.offset is the phase when t is ts[0]. It has the effect of shifting the signal left or right in time.
4. If self.func is np.sin or np.cos, the result is a value between -1 and $+1$.
5. Multiplying by self.amp yields a signal that ranges from -self.amp to +self.amp.

In math notation, evaluate is written like this:

$y = A \cos(2\pi f t + \varphi 0)$

where

- A is amplitude,
- f is frequency,
- t is time, and
- φ0 is the phase offset.

- It may seem like I wrote a lot of code to evaluate one simple expression, but as we'll see, this code provides a framework for dealing with all kinds of signals, not just sinusoids.

1.2 NOISE

In signal processing, noise is a general term for unwanted (and, in general, unknown) modifications that a signal may suffer during capture, storage, transmission, processing, or conversion.

1.2.1 Uncorrelated noise

- The simplest way to understand noise is to generate it, and the simplest kind to generate is uncorrelated uniform noise (UU noise).
- "Uniform" means the signal contains random values from a uniform distribution.
- That is, every value in the range is equally likely.
- "Uncorrelated" means that the values are independent.
- That is, knowing one value provides no information about the others.
- Here's a class that represents UU noise:

```
class UncorrelatedUniformNoise(_Noise):

def evaluate(self, ts):

ys = np.random.uniform(-self.amp, self.amp, len(ts))

return ys
```

Where,

UncorrelatedUniformNoise inherits from _Noise, which inherits from Signal.

The evaluate function takes ts, the times when the signal should be evaluated.

It uses np.random.uniform, which generates values from a uniform distribution.

In this example, the values are in the range between -amp to amp.

- The following example generates UU noise with duration 0.5 seconds at 11,025 samples per second.

```
signal = thinkdsp.UncorrelatedUniformNoise()

wave = signal.make_wave(duration=0.5, framerate=11025)
```

- If you play this wave, it sounds like the static you hear if you tune a radio between channels.
- Following Figure shows what the waveform looks like.
- As expected, it looks pretty random.

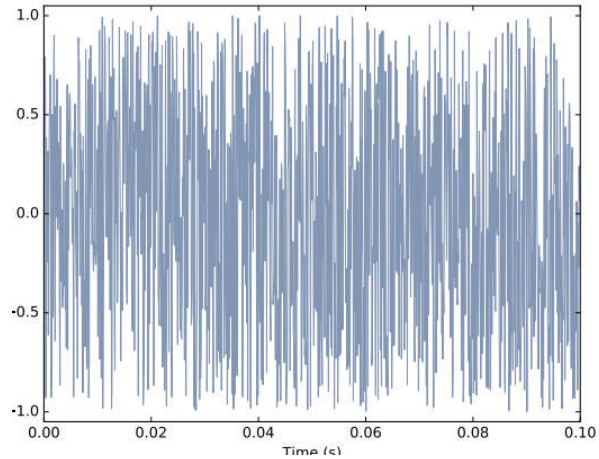


Figure: Waveform of uncorrelated uniform noise.

- The spectrum:

```
spectrum = wave.make_spectrum()

spectrum.plot_power()
```

  - Spectrum.plot_power is similar to Spectrum.plot, except that it plots power instead of amplitude.
  - Power is the square of amplitude.
  - We switch from amplitude to power here because it is more conventional in the context of noise.

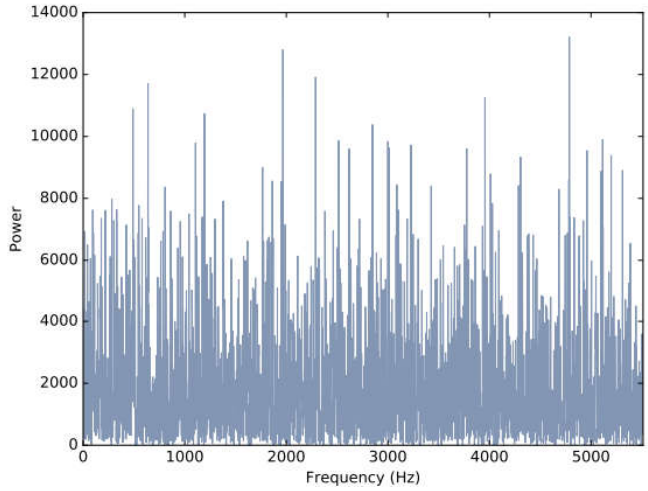


Figure : Power spectrum of uncorrelated uniform noise

- There are at least three things about a noise signal or its spectrum:

1. Distribution: The distribution of a random signal is the set of possible values and their probabilities.

For example, in the uniform noise signal, the set of values is the range from -1 to 1, and all values have the same probability.

An alternative is Gaussian noise, where the set of values is the range from negative to positive infinity, but values near 0 are the most likely, with probability that drops off according to the Gaussian or "bell" curve.

2. Correlation: Each value in UU noise is independent.

Brownian noise is an alternative.

It is where each value is the sum of the previous value and a random "step".

So if the value of the signal is high at a particular point in time, we expect it to stay high, and if it is low, we expect it to stay low.

3. Relationship between power and frequency: In the spectrum of UU noise, the power at all frequencies is drawn from the same distribution.

That is, the average power is the same for all frequencies.

An alternative is pink noise, where power is inversely related to frequency.

That is, the power at frequency f is drawn from a distribution whose mean is proportional to 1/f.

1.2.2 Integrated spectrum

- Integrated spectrum is used for UU noise so we can see the relationship between power and frequency more clearly.

- It is a function of frequency, f, that shows the cumulative power in the spectrum up to f.
- Spectrum provides a method that computes the IntegratedSpectrum:

```
def make_integrated_spectrum(self):
cs = np.cumsum(self.power)
cs /= cs[-1]
return IntegratedSpectrum(cs, self.fs)
```

where,

self.power is a NumPy array containing power for each frequency.

np.cumsum computes the cumulative sum of the powers.

- Dividing through by the last element normalizes the integrated spectrum so it runs from 0 to 1.
- The result is an IntegratedSpectrum.
- Here is the class definition:

```
class IntegratedSpectrum(object):
def __init__(self, cs, fs):
self.cs = cs
self.fs = fs
```

- Like Spectrum, IntegratedSpectrum provides plot_power, so we can compute and plot the integrated spectrum like this:

```
integ = spectrum.make_integrated_spectrum()
integ.plot_power()
```

- The result, shown in the following Figure, is a straight line, which indicates that power at all frequencies is constant, on average.

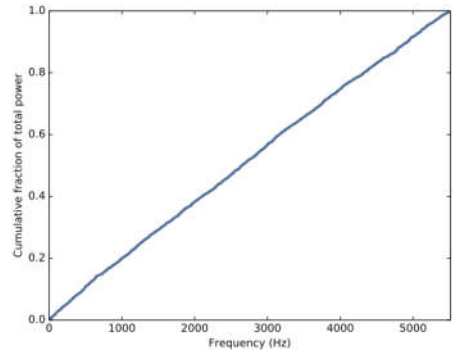


Figure: Integrated spectrum of uncorrelated uniform noise.

- Noise with equal power at all frequencies is called white noise by analogy with light, because an equal mixture of light at all visible frequencies is white.

1.2.3 Brownian noise

- UU noise is uncorrelated, which means that each value does not depend on the others.
- An alternative is Brownian noise, in which each value is the sum of the previous value and a random "step".
- It is called "Brownian" by analogy with Brownian motion, in which a particle suspended in a fluid moves apparently at random, due to unseen interactions with the fluid.
- Brownian motion is often described using a random walk, which is a mathematical model of a path where the distance between steps is characterized by a random distribution.
- In a one-dimensional random walk, the particle moves up or down by a random amount at each time step.
- The location of the particle at any point in time is the sum of all previous steps.
- This observation suggests a way to generate Brownian noise: generate uncorrelated random steps and then add them up.
- Here is a class definition that implements this algorithm:

```
class BrownianNoise(_Noise):

def evaluate(self, ts):

dys = np.random.uniform(-1, 1, len(ts))

ys = np.cumsum(dys)

ys = normalize(unbias(ys), self.amp)

return ys
```

Where,

evaluate uses np.random.uniform to generate an uncorrelated signal and

np.cumsum to compute their cumulative sum.

- Since the sum is likely to escape the range from -1 to 1, we have to use unbias to shift the mean to 0, and normalize to get the desired maximum amplitude.
- Here's the code that generates a BrownianNoise object and plots the waveform.

```
signal = thinkdsp.BrownianNoise()

wave = signal.make_wave(duration=0.5, framerate=11025)

wave.plot()
```

- Following Figure shows the result.

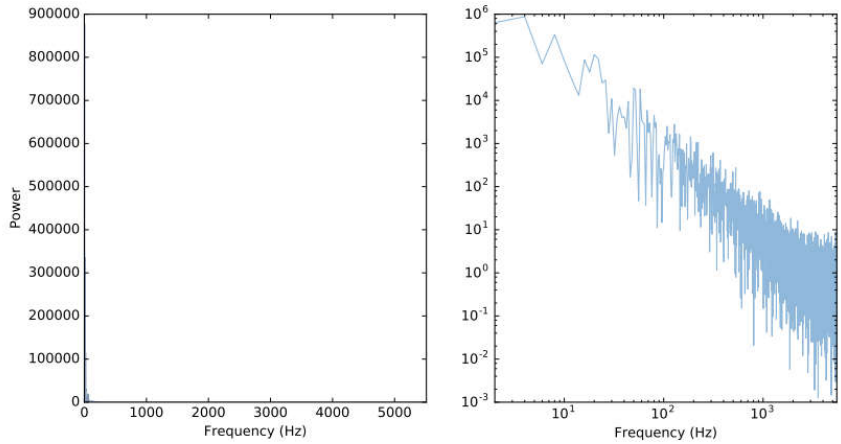


Figure: Spectrum of Brownian noise on a linear scale (left) and log-log scale (right).

- The waveform wanders up and down, but there is a clear correlation between successive values.
- When the amplitude is high, it tends to stay high, and vice versa.
- If you plot the spectrum of Brownian noise on a linear scale, as in above Figure (left), it doesn't look like much.
- Nearly all of the power is at the lowest frequencies; the higher frequency components are not visible.
- To see the shape of the spectrum more clearly, we can plot power and frequency on a log-log scale.
- Here's the code:

```
import matplotlib.pyplot as plt

spectrum = wave.make_spectrum()

spectrum.plot_power(linewidth=1, alpha=0.5)

plt.xscale('log') plt.yscale('log')
```

- The result is in the above Figure (right).
- The relationship between power and frequency is noisy, but roughly linear.
- Spectrum provides estimate_slope, which uses SciPy to compute a least squares fit to the power spectrum:

```
#class Spectrum

def estimate_slope(self):

x = np.log(self.fs[1:])

y = np.log(self.power[1:])

t = scipy.stats.linregress(x,y)

return t
```

- It discards the first component of the spectrum because this component corresponds to f = 0, and log 0 is undefined.
- estimate_slope returns the result from scipy.stats.linregress which is an object that contains the estimated slope and intercept, coefficient of determination (R2), p-value, and standard error.
- For our purposes, we only need the slope.
- For Brownian noise, the slope of the power spectrum is -2, so we can write this relationship:

$\log P = k - 2 \log f$

where,

P is power,

f is frequency, and

k is the intercept of the line, which is not important for our purposes.

- Exponentiating both sides yields:

$P = K/f^2$

where,

K is e^k, but still not important.

More relevant is that power is proportional to $1/f^2$, which is characteristic of Brownian noise.

- Brownian noise is also called red noise, for the same reason that white noise is called “white”.
- If you combine visible light with power proportional to $1/f^2$, most of the power would be at the low-frequency end of the spectrum, which is red.
- Brownian noise is also sometimes called “brown noise”.

1.2.4 Pink Noise

- For red noise, the relationship between frequency and power is

$P = K/f^2$

- There is nothing special about exponent 2.
- We can synthesize noise with any exponent, β.

$P = K/f^{\beta}$

  - When β = 0, power is constant at all frequencies, so the result is white noise.
  - When β = 2 the result is red noise.
  - When β is between 0 and 2, the result is between white and red noise, so it is called pink noise.

- There are several ways to generate pink noise.
- The simplest is to generate white noise and then apply a low-pass filter with the desired exponent.
- thinkdsp provides a class that represents a pink noise signal:

```
class PinkNoise(_Noise):

def __init__(self, amp=1.0, beta=1.0):

self.amp = amp

self.beta = beta
```

Where,

amp is the desired amplitude of the signal.

beta is the desired exponent.

- PinkNoise provides make_wave, which generates a Wave.

```
def make_wave(self, duration=1, start=0, framerate=11025):

signal = UncorrelatedUniformNoise()

wave = signal.make_wave(duration, start, framerate)

spectrum = wave.make_spectrum()

spectrum.pink_filter(beta=self.beta)

wave2 = spectrum.make_wave()

wave2.unbias()

wave2.normalize(self.amp)

return wave2
```

Where,

- duration is the length of the wave in seconds.
- start is the start time of the wave; it is included so that make_wave has the same interface for all types of signal, but for random noise, start time is irrelevant.
- And framerate is the number of samples per second.

- make_wave creates a white noise wave, computes its spectrum, applies a filter with the desired exponent, and then converts the filtered spectrum back to a wave.
- Then it unbiases and normalizes the wave.
- Spectrum provides pink_filter:
- def pink_filter(self, beta=1.0):
- denom = self.fs ** (beta/2.0)
- denom[0] = 1
- self.hs /= denom

Where,

pink_filter divides each element of the spectrum by $f^{\beta/2}$.

Since power is the square of amplitude, this operation divides the power at each component by f^{β} .

It treats the component at f = 0 as a special case, partly to avoid dividing by 0, and partly because this element represents the bias of the signal, which we are going to set to 0 anyway.

Following Figure shows the resulting waveform.

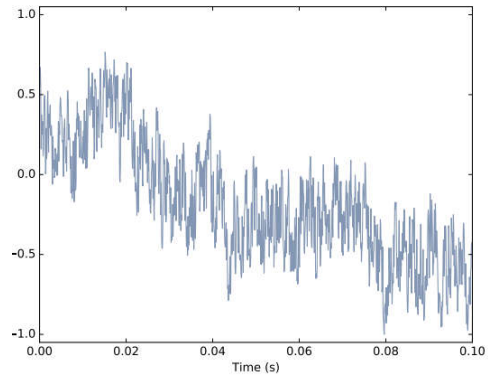


Figure : Waveform of pink noise with β = 1.

Like Brownian noise, it wanders up and down in a way that suggests correlation between successive values, but at least visually, it looks more random.

Finally, the following Figure shows a spectrum for white, pink, and red noise on the same log-log scale.The relationship between the exponent, β, and the slope of the spectrum is apparent in this figure.

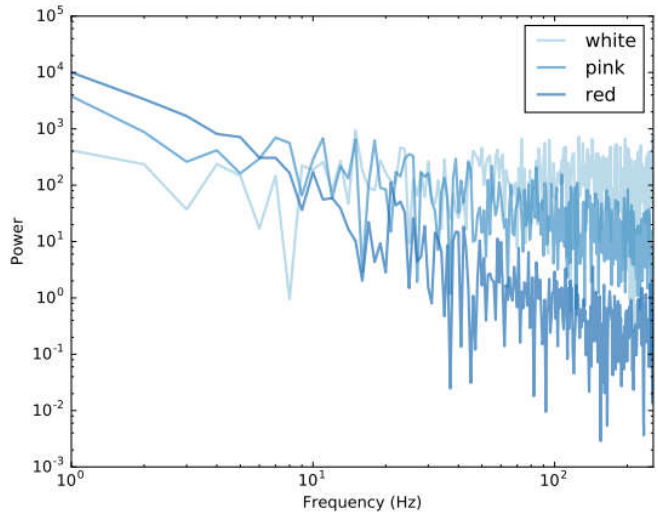


Figure : Spectrum of white, pink, and red noise on a log-log scale.

1.2.5 Gaussian noise

- In uncorrelated uniform (UU) noise it showed that, because its spectrum has equal power at all frequencies, on average, UU noise is white.
- But, more often they mean uncorrelated Gaussian (UG) noise.
- thinkdsp provides an implementation of UG noise:

```
class UncorrelatedGaussianNoise(_Noise):

def evaluate(self, ts):

ys = np.random.normal(0, self.amp, len(ts))

return ys
```

Where,

np.random.normal returns a NumPy array of values from a Gaussian distribution.

- In this case with mean 0 and standard deviation self.amp.
- In theory the range of values is from negative to positive infinity, but we expect about 99% of the values to be between -3 and 3.
- UG noise is similar in many ways to UU noise.
- The spectrum has equal power at all frequencies, on average, so UG is also white.

- It has one other interesting property: the spectrum of UG noise is also UG noise.
- The real and imaginary parts of the spectrum are uncorrelated Gaussian values.
- We can generate the spectrum of UG noise and then generate a "normal probability plot", which is a graphical way to test whether a distribution is Gaussian.

```
signal = thinkdsp.UncorrelatedGaussianNoise()

wave = signal.make_wave(duration=0.5, framerate=11025)

spectrum = wave.make_spectrum()

thinkstats2.NormalProbabilityPlot(spectrum.real)

thinkstats2.NormalProbabilityPlot(spectrum.imag)
```

- NormalProbabilityPlot is provided by thinkstats2, which is included in the repository.
- Following Figure shows the results.

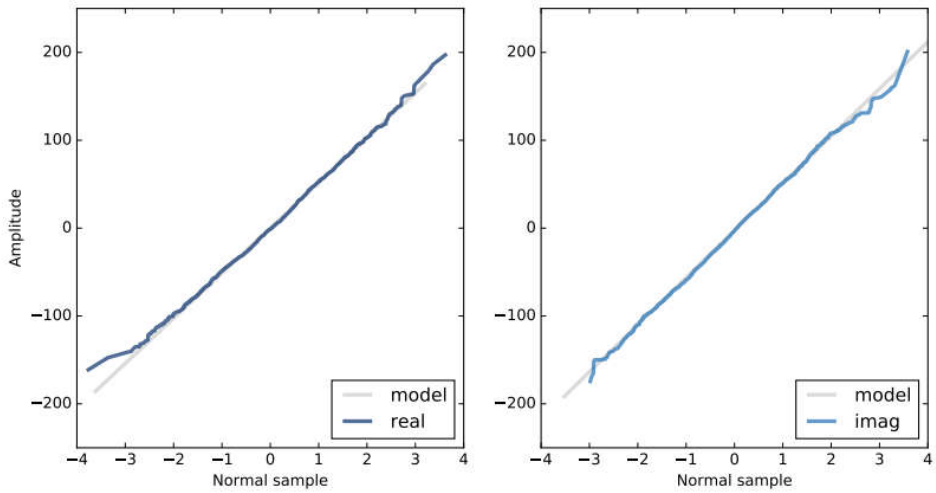


Figure : Normal probability plot for the real and imaginary parts of the spectrum of Gaussian noise.

- The gray lines show a linear model fit to the data; the dark lines show the data.
- A straight line on a normal probability plot indicates that the data come from a Gaussian distribution.
- Except for some random variation at the extremes, these lines are straight, which indicates that the spectrum of UG noise is UG noise.
- The spectrum of UU noise is also UG noise, at least approximately.

- By the Central Limit Theorem, the spectrum of almost any uncorrelated noise is approximately Gaussian, as long as the distribution has finite mean and standard deviation, and the number of samples is large.

1.3 AUTOCORRELATION

Autocorrelation, sometimes known as serial correlation in the discrete time case, is the correlation of a signal with a delayed copy of itself as a function of delay. Informally, it is the similarity between observations of a random variable as a function of the time lag between them.

1.3.1 Correlation

- Correlation between variables means that if you know the value of one, you have some information about the other.
- There are several ways to quantify correlation, but the most common is the Pearson product-moment correlation coefficient, usually denoted ρ.
- For two variables, x and y, that each contain N values:

$$\rho = \frac{\sum_i (x_i - \mu_x)(y_i - \mu_y)}{N \sigma_x \sigma_y}$$

Where,

μ_x and μ_y are the means of x and y, and

σ_x and σ_y are their standard deviations.

- Pearson's correlation is always between -1 and +1 (including both).
- If ρ is positive, we say that the correlation is positive, which means that when one variable is high, the other tends to be high.
- If ρ is negative, the correlation is negative, so when one variable is high, the other tends to be low.
- The magnitude of ρ indicates the strength of the correlation.
- If ρ is 1 or -1, the variables are perfectly correlated, which means that if you know one, you can make a perfect prediction about the other.
- If ρ is near zero, the correlation is probably weak, so if you know one, it doesn't tell you much about the others.
- I say "probably weak" because it is also possible that there is a nonlinear relationship that is not captured by the coefficient of correlation.
- Nonlinear relationships are often important in statistics, but less often relevant for signal processing, so I won't say more about them here.

- Python provides several ways to compute correlations.
- np.corrcoef takes any number of variables and computes a correlation matrix that includes correlations between each pair of variables.
- I'll present an example with only two variables.
- First, I define a function that constructs sine waves with different phase offsets:

```
def make_sine(offset):

signal = thinkdsp.SinSignal(freq=440, offset=offset)

wave = signal.make_wave(duration=0.5, framerate=10000)

return wave
```

- Next I instantiate two waves with different offsets:

```
wave1 = make_sine(offset=0)

wave2 = make_sine(offset=1)
```

- Following Figure shows what the first few periods of these waves look like.

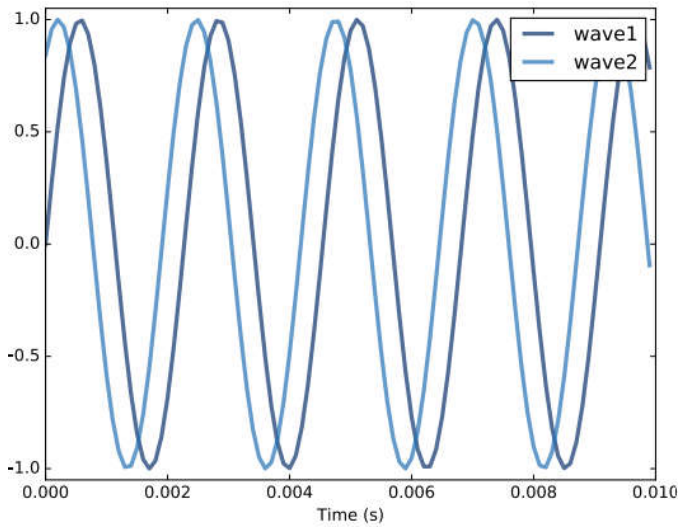


Figure: Two sine waves that differ by a phase offset of 1 radian; their coefficient of correlation is 0.54.

- When one wave is high, the other is usually high, so we expect them to be correlated.

```
>>> corr_matrix = np.corrcoef(wave1.ys, wave2.ys, ddof=0) [[ 1. 0.54]  [ 0.54 1. ]]
```

- The option ddof=0 indicates that corrcoef should divide by N, as in the equation above, rather than use the default, N − 1.

- The result is a correlation matrix:
  - The first element is the correlation of wave1 with itself, which is always 1.
  - The last element is the correlation of wave2 with itself.
- The off-diagonal elements contain the value we're interested in, the correlation of wave1 and wave2.
- The value 0.54 indicates that the strength of the correlation is moderate.
- As the phase offset increases, this correlation decreases until the waves are 180 degrees out of phase, which yields correlation -1.
- Then it increases until the offset differs by 360 degrees.
- At that point we have come full circle and the correlation is 1.
- Following Figure shows the relationship between correlation and phase offset for a sine wave.

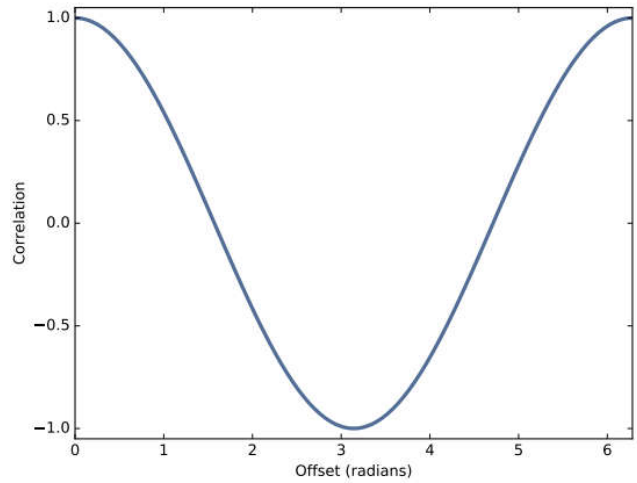


Figure: The correlation of two sine waves as a function of the phase offset between them. The result is a cosine.

- The shape of that curve should look familiar; it is a cosine.
- thinkdsp provides a simple interface for computing the correlation between waves:

```
>>> wave1.corr(wave2) 0.54
```

1.3.2 Serial correlation

- Signals often represent measurements of quantities that vary in time.
- For example, the sound signals we've worked with represent measurements of voltage (or current), which correspond to the changes in air pressure we perceive as sound.

- Measurements like this almost always have serial correlation, which is the correlation between each element and the next (or the previous).
- To compute serial correlation, we can shift a signal and then compute the correlation of the shifted version with the original.

```
def serial_corr(wave, lag=1):

n = len(wave)

y1 = wave.ys[lag:]

y2 = wave.ys[:n-lag]

corr = np.corrcoef(y1, y2, ddof=0)[0, 1]

return corr
```

- serial_corr takes a Wave object and lag, which is the integer number of places to shift the wave.
- It computes the correlation of the wave with a shifted version of itself.
- We expect UU noise to be uncorrelated, based on the way it's generated (not to mention the name):

```
signal = thinkdsp.UncorrelatedGaussianNoise()

wave = signal.make_wave(duration=0.5, framerate=11025)

serial_corr(wave)
```

- When I ran this example, I got 0.006, which indicates a very small serial correlation.
- You might get a different value when you run it, but it should be comparably small.
- In a Brownian noise signal, each value is the sum of the previous value and a random "step", so we expect a strong serial correlation:

```
signal = thinkdsp.BrownianNoise()

wave = signal.make_wave(duration=0.5, framerate=11025)

serial_corr(wave)
```

- Sure enough, the result I got is greater than 0.999.
- Since pink noise is in some sense between Brownian noise and UU noise, we might expect an intermediate correlation:

```
signal = thinkdsp.PinkNoise(beta=1)

wave = signal.make_wave(duration=0.5, framerate=11025)

serial_corr(wave)
```

- With parameter β = 1, I got a serial correlation of 0.851.
- As we vary the parameter from β = 0, which is uncorrelated noise, to β = 2, which is Brownian, serial correlation ranges from 0 to almost 1, as shown in following Figure.

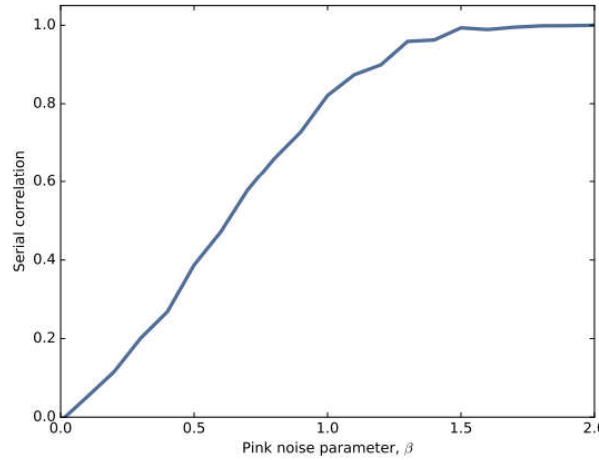


Figure: Serial correlation for pink noise with a range of parameters.

1.3.3 Autocorrelation

- In the previous section we computed the correlation between each value and the next, so we shifted the elements of the array by 1.
- But we can easily compute serial correlations with different lags.
- You can think of serial_corr as a function that maps from each value of lag to the corresponding correlation, and we can evaluate that function by looping through values of lag: def autocorr(wave):

  lags = range(len(wave.ys)//2)

  corrs = [serial_corr(wave, lag) for lag in lags]

  return lags, corrs

  - autocorr takes a Wave object and returns the autocorrelation function as a pair of sequences: lags is a sequence of integers from 0 to half the length of the wave;
  - corrs is the sequence of serial correlations for each lag.
- Following Figure shows autocorrelation functions for pink noise with three values of β.

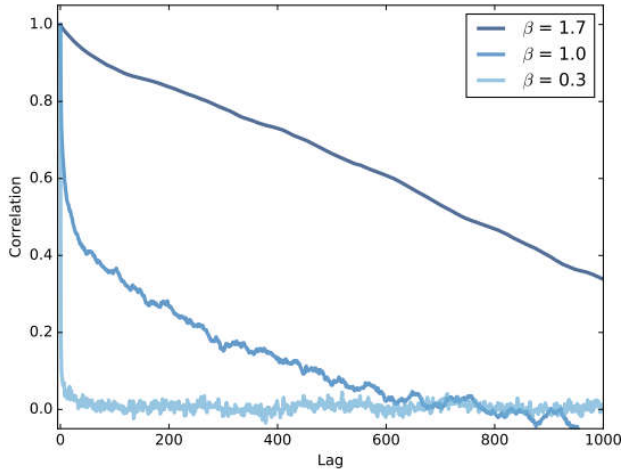


Figure : Autocorrelation functions for pink noise with a range of parameters.

- For low values of β, the signal is less correlated, and the autocorrelation function drops off to zero quickly.
- For larger values, serial correlation is stronger and drops off more slowly.
- With β = 1.7 serial correlation is strong even for long lags; this phenomenon is called long-range dependence, because it indicates that each value in the signal depends on many preceding values.

1.3.4 Autocorrelation of periodic signals

- The autocorrelation of pink noise has interesting mathematical properties.
- The autocorrelation of pink noise has limited applications.
- The autocorrelation of periodic signals is more useful.
- As an example, I downloaded from freesound.org a recording of someone singing a chirp; the repository for this book includes the file: 28042_ _bcjordan__voicedownbew.wav.
- Following Figure shows the spectrogram of this wave.

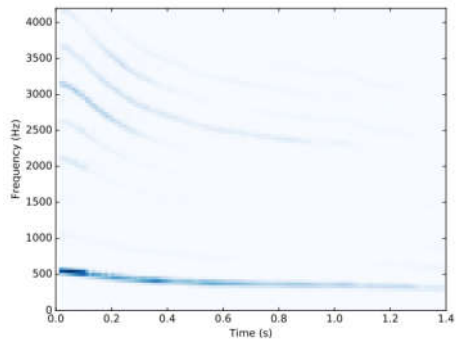


Figure: Spectrogram of a vocal chirp.

- The fundamental frequency and some of the harmonics show up clearly.
- The chirp starts near 500 Hz and drops down to about 300 Hz, roughly from C5 to E4.
- To estimate pitch at a particular point in time, we could use the spectrum, but it doesn't work very well.
- To see why not, I'll take a short segment from the wave and plot its spectrum:

```
duration = 0.01

segment = wave.segment(start=0.2, duration=duration)

spectrum = segment.make_spectrum()

spectrum.plot(high=1000)
```

- Following Figure shows its spectrum,where the segment starts at 0.2 seconds and lasts 0.01 seconds..

Figure 5.6: Spectrum of a segment from a vocal chirp.

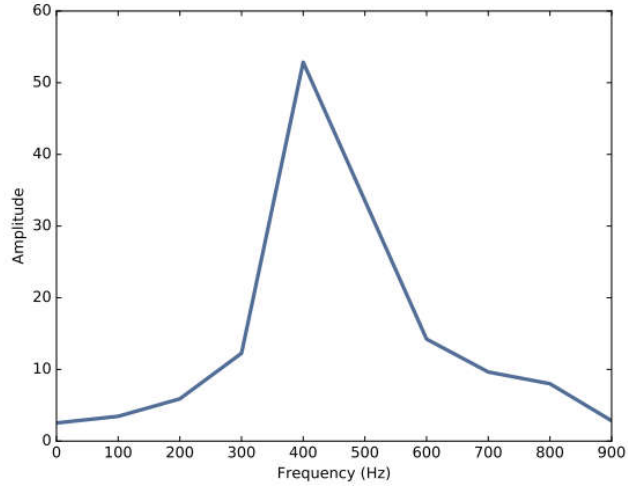


Figure : Spectrum of a segment from a vocal chirp.

- There is a clear peak near 400 Hz, but it is hard to identify the pitch precisely.
- The length of the segment is 441 samples at a framerate of 44100 Hz, so the frequency resolution is 100 Hz (see Section 3.5).
- That means the estimated pitch might be off by 50 Hz; in musical terms, the range from 350 Hz to 450 Hz is about 5 semitones.
- We could get better frequency resolution by taking a longer segment, but since the pitch is changing over time, we would also get "motion blur"; that is, the peak would spread between the start and end pitch of the segment.

- We can estimate pitch more precisely using autocorrelation.
- If a signal is periodic, we expect the autocorrelation to spike when the lag equals the period.
- To show why that works, I'll plot two segments from the same recording.

```
import matplotlib.pyplot as plt

        def plot_shifted(wave, offset=0.001, start=0.2):

segment1 = wave.segment(start=start, duration=0.01)

segment1.plot(linewidth=2, alpha=0.8)

segment2 = wave.segment(start=start-offset, duration=0.01)

segment2.shift(offset)

segment2.plot(linewidth=2, alpha=0.4)

corr = segment1.corr(segment2)

text = r'$\rho =$ %.2g' % corr

plt.text(segment1.start+0.0005, -0.8, text)

plt.xlabel('Time (s)')
```

- Following Figure 5.7 shows the result of One segment starts at 0.2 seconds; the other starts 0.0023 seconds later.

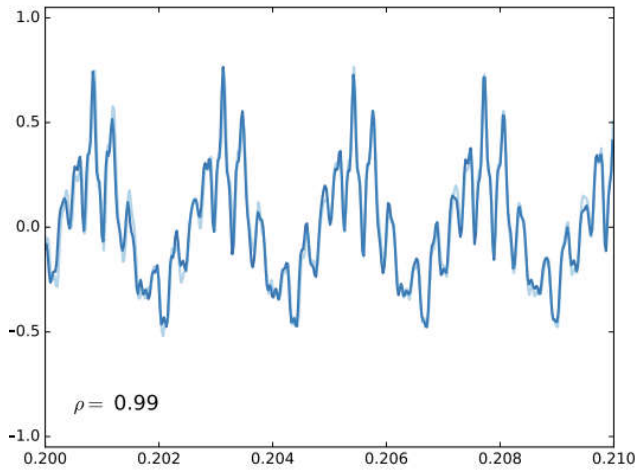


Figure : Two segments from a chirp, one starting 0.0023 seconds after the other

- The segments are similar, and their correlation is 0.99.
- This result suggests that the period is near 0.0023 seconds, which corresponds to a frequency of 435 Hz.

- For this example, I estimated the period by trial and error.
- To automate the process, we can use the autocorrelation function.

  lags, corrs = autocorr(segment)

  plt.plot(lags, corrs)

- Following Figure shows the autocorrelation function for the segment starting at t = 0.2 seconds.

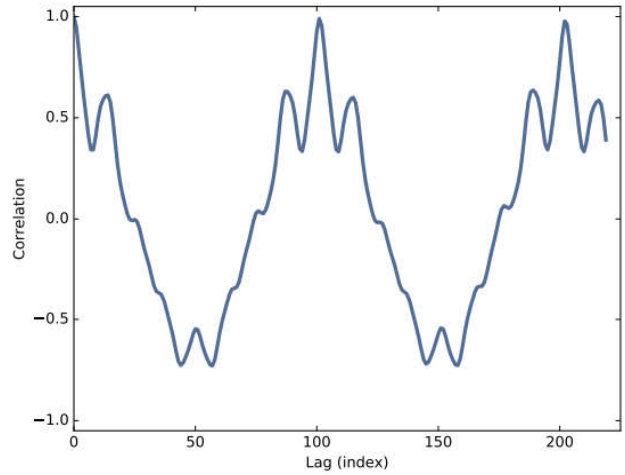


Figure : Autocorrelation function for a segment from a chirp.

- Here,the first peak occurs at lag=101.
- We can compute the frequency that corresponds to that period like this:

period = lag / segment.framerate

frequency = 1 / period

- The estimated fundamental frequency is 437 Hz.
- To evaluate the precision of the estimate, we can run the same computation with lags 100 and 102, which correspond to frequencies 432 and 441 Hz.
- The frequency precision using autocorrelation is less than 10 Hz, compared with 100 Hz using the spectrum.
- In musical terms, the expected error is about 30 cents (a third of a semitone).

1.3.5 Correlation as a dot product

- We started with this definition of Pearson's correlation coefficient:

$$\rho = \frac{\sum_i (x_i - \mu_x)(y_i - \mu_y)}{N \sigma_x \sigma_y}$$

- Then I used ρ to define serial correlation and autocorrelation.

- That's consistent with how these terms are used in statistics, but in the context of signal processing, the definitions are a little different.
- In signal processing, we are often working with unbiased signals, where the mean is 0, and normalized signals, where the standard deviation is 1.
- In that case, the definition of ρ simplifies to:

$$\rho = \frac{1}{N} \sum_i x_i y_i$$

- And it is common to simplify even further:

$$r = \sum_i x_i y_i$$

- This definition of correlation is not "standardized", so it doesn't generally fall between -1 and 1.
- But it has other useful properties. If you think of x and y as vectors, you might recognize this formula as the dot product, x · y.
- The dot product indicates the degree to which the signals are similar. If they are normalized so their standard deviations are 1,

  $x \cdot y = \cos \theta$

  where θ is the angle between the vectors.
- And that explains why the following Figure is a cosine curve.

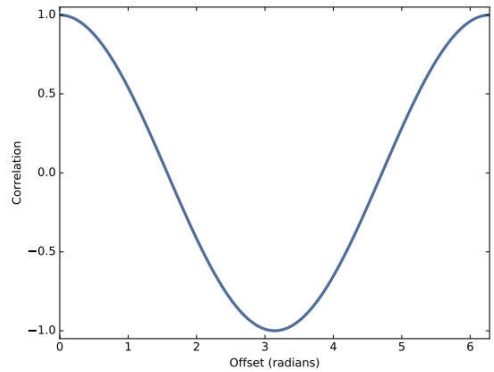


Figure : The correlation of two sine waves as a function of the phase offset between them. The result is a cosine.

1.4 SUMMARY

In this chapter we have discussed:

Fundamental concepts and techniques for processing signals and digital signal processing, Process of conversion of analog to digital as well as digital to analog conversion, Digital Signal Processing (DSP): Digital Signal Processing (DSP) is an amazing field that uses advanced math and

computers to understand and improve signals. Signals are everywhere in our digital world, like music, pictures, and wireless communication. Applications of real -world signal processing problems.

Periodic signals: Signals that repeat themselves after some period of time are called periodic signals. For example, if you strike a bell, it vibrates and generates sound.

Noise: In signal processing, noise is a general term for unwanted (and, in general, unknown) modifications that a signal may suffer during capture, storage, transmission, processing, or conversion. The simplest way to understand noise is to generate it, and the simplest kind to generate is uncorrelated uniform noise (UU noise).

Spectrum: The spectrum is the set of sinusoids that add up to produce the signal.

Autocorrelation: sometimes known as serial correlation in the discrete time case, is the correlation of a signal with a delayed copy of itself as a function of delay. Informally, it is the similarity between observations of a random variable as a function of the time lag between them.

Correlation: Correlation between variables means that if you know the value of one, you have some information about the other.

1.5 EXERCISE

Answer the following:

1. What is Digital Signal Processing (DSP)? Explain.
2. Define Digital Signal Processing (DSP). Draw basic DSP system. Note down the advantages and disadvantages of DSP.
3. Write a short note on Periodic signals.
4. Explain following terms about Periodic signals:
   a. Periodic signals
   b. Waveform
   c. Sinusoid
   d. Cycles
   e. Period
   f. Frequency
5. What is Spectral decomposition? Explain.
6. What is Spectral decomposition? Explain Harmonics in detail.
7. Write a short note on the signal.
8. What is signal and signal processing? Explain the term SumSignal.

9. Explain following term for signal:
    a. Signal
    b. Signal processing
    c. SumSignal
    d. Frame
    e. Sample
    f. wave
10. What is spectrum? Explain three methods that modify the spectrum.
11. Write a short note on a wave object.
12. Write a short note on : Signal objects. Explain The parameters of __init__.
13. What is noise? Explain Uncorrelated noise.
14. What is the spectrum in noise ? Explain following three things about a noise signal or its spectrum :
    a. Distribution
    b. Correlation
    c. Relationship between power and frequency
15. Write a short note on: Integrated spectrum.
16. What is Brownian noise? Explain in detail.
17. What is Pink Noise? Explain in detail.
18. What is Gaussian noise? Explain.
19. Write a short note on :Correlation.
20. Write a short note on :Serial correlation.
21. Write a short note on: Autocorrelation.
22. Explain: Autocorrelation of periodic signals.
23. Explain: Correlation as a dot product

1.6 REFERENCES

Text Books/Reference Books:

1. Think DSP: Digital Signal Processing in Python by Allen Downey, O'Reilly Media; 1st edition (August 16, 2016).

https://greenteapress.com/thinkdsp/thinkdsp.pdf

2. The code and sound samples used in this chapter are available from

https: //github.com/AllenDowney/ThinkDSP

❖❖❖❖

Unit 1

2

FUNDAMENTALS OF DIGITAL SIGNALS PROCESSING - II

Unit Structure :

2.0 OBJECTIVE

The objective of this chapter is :

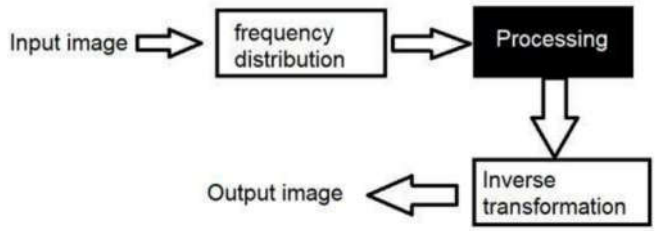
- To Understand the meaning of frequency domain filtering, and how it differs from filtering in the spatial domain.
- Be familiar with the concepts of sampling, function reconstruction, and aliasing.
- Understand convolution in the frequency domain, and how it is related to filtering.
- Know how to obtain frequency domain filter functions from spatial kernels, and vice versa.
- Know the steps required to perform filtering in the frequency domain.

- Understand the mechanics of the fast Fourier transform, and how to use it effectively in image processing.
- Some solved examples based on DFT and FFT.

2.1 FREQUENCY DOMAIN OPERATIONS

2.1.1 Introduction to Frequency domain:

- Frequency domain analysis and Fourier transforms are a cornerstone of signal and system analysis.
- Processing signals /images in the frequency domain with the context Fourier series and frequency domain is purely mathematics.
- We will try to minimize that math's part and focus more on its use in Digital Image Processing.
- In Frequency Domain we first transform the image to its frequency distribution.
- Then our black box system performs whatever processing it has to perform, and the output of the black box in this case is not an image, but a transformation.
- After performing inverse transformation, it is converted into an image which is then viewed in spatial domain.
- It can be pictorially viewed as



- In the frequency domain, a digital image is converted from spatial domain to frequency domain.
- In the frequency domain, image filtering is used for image enhancement for a specific application.
- A Fast Fourier transformation is a tool of the frequency domain used to convert the spatial domain to the frequency domain.
- For smoothing an image, low filter is implemented and for sharpening an image, high pass filter is implemented.
- When both the filters are implemented, it is analyzed for the ideal filter, Butterworth filter and Gaussian filter.

- The frequency domain is a space which is defined by Fourier transform.
- Fourier transform has a very wide application in image processing.
- Frequency domain analysis is used to indicate how signal energy can be distributed in a range of frequency.
- The basic principle of frequency domain analysis in image filtering is to computer 2D discrete Fourier transform of the image.

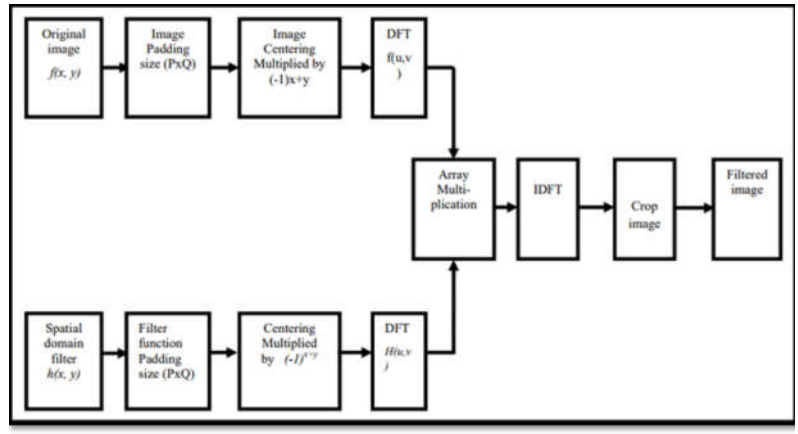


Fig: Frequency domain filtering process

2.1.2 Representing Image as Signals

- **Fourier series :** Any function that periodically repeats itself can be expressed as the sum of sines and/or cosines of different frequencies, each multiplied by a different coefficient.
- **Fourier transform :** Even functions that are not periodic (but whose area under the curve is finite) can be expressed as the integral of sines and/or cosines multiplied by a weighting function.
- **The frequency domain :** refers to the plane of the two dimensional discrete Fourier transform of an image.
- The purpose of the Fourier transform is to represent a signal as a linear combination of sinusoidal signals of various frequencies.
- Following Figure shows sum of the four functions :

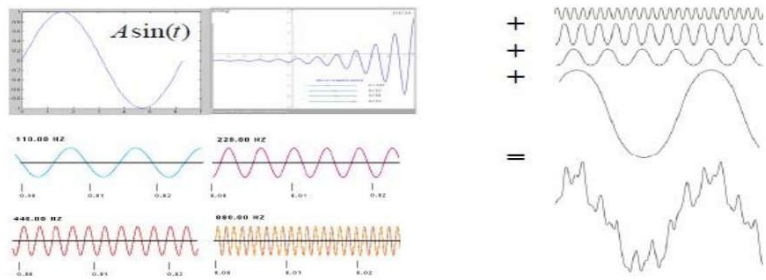


Figure :The function at the bottom is the sum of the four functions above it.

FUNDAMENTAL STEPS IN DIGITAL IMAGE PROCESSING

The two broad categories are:

1. methods whose input and output are images, and
2. methods whose inputs may be images, but whose outputs are attributes extracted from those images.

Fundamental steps in digital image processing are as follow:

Image acquisition

- It is the first process in digital image processing.
- Acquisition could be as simple as being given an image that is already in digital form.
- Generally, the image acquisition stage involves preprocessing, such as scaling.

Image enhancement

- It is the process of manipulating an image so the result is more suitable than the original for a specific application.
- It establishes at the outset that enhancement techniques are problem oriented.
- Thus, for example, a method that is quite useful for enhancing X-ray images may not be the best approach for enhancing satellite images taken in the infrared band of the electromagnetic spectrum.

Image restoration

- It is an area that also deals with improving the appearance of an image.
- However, unlike enhancement, which is subjective, image restoration is objective, in the sense that restoration techniques tend to be based on mathematical or probabilistic models of image degradation.
- Enhancement, on the other hand, is based on human subjective preferences regarding what constitutes a “good” enhancement result.

Color image processing

- It is an area that has been gaining in importance because of the significant increase in the use of digital images over the internet.
- Color is used also as the basis for extracting features of interest in an image.

Wavelets

- Wavelets are the foundation for representing images in various degrees of resolution.

Compression

- As the name implies, it deals with techniques for reducing the storage required to save an image, or the bandwidth required to transmit it.
- Although storage technology has improved significantly over the past decade, the same cannot be said for transmission capacity.
- This is true particularly in uses of the internet, which are characterized by significant pictorial content. Image compression
- It is familiar (perhaps inadvertently) to most users of computers in the form of image file extensions, such as the jpg file extension used in the JPEG (Joint Photographic Experts Group) image compression standard.

Morphological processing

- It deals with tools for extracting image components that are useful in the representation and description of shape.

Segmentation

- Segmentation partitions an image into its constituent parts or objects.
- In general, autonomous segmentation is one of the most difficult tasks in digital image processing.
- A rugged segmentation procedure brings the process a long way toward successful solution of imaging problems that require objects to be identified individually.
- On the other hand, weak or erratic segmentation algorithms almost always guarantee eventual failure. In general, the more accurate the segmentation, the more likely automated object classification is to succeed.

Feature extraction

- It almost always follows the output of a segmentation stage, which usually is raw pixel data, constituting either the boundary of a region (i.e., the set of pixels separating one image region from another) or all the points in the region itself.
- Feature extraction consists of feature detection and feature description.
- Feature detection refers to finding the features in an image, region, or boundary.
- Feature description assigns quantitative attributes to the detected features.
- For example, we might detect corners in a region, and describe those corners by their orientation and location; both of these descriptors are quantitative attributes.

- Feature processing methods discussed in this chapter are subdivided into three principal categories, depending on whether they are applicable to boundaries, regions, or whole images.
- Some features are applicable to more than one category. Feature descriptors should be as insensitive as possible to variations in parameters such as scale, translation, rotation, illumination, and viewpoint.

Image pattern classification

- It is the process that assigns a label (e.g., "vehicle") to an object based on its feature descriptors.
- We will discuss methods of image pattern classification ranging from "classical" approaches such as minimum-distance, correlation, and Bayes classifiers, to more modern approaches implemented using deep neural networks.
- In particular, we will discuss in detail deep convolutional neural networks, which are ideally suited for image processing work.

2.1.3 Sampling and Fourier Transforms:

SAMPLING

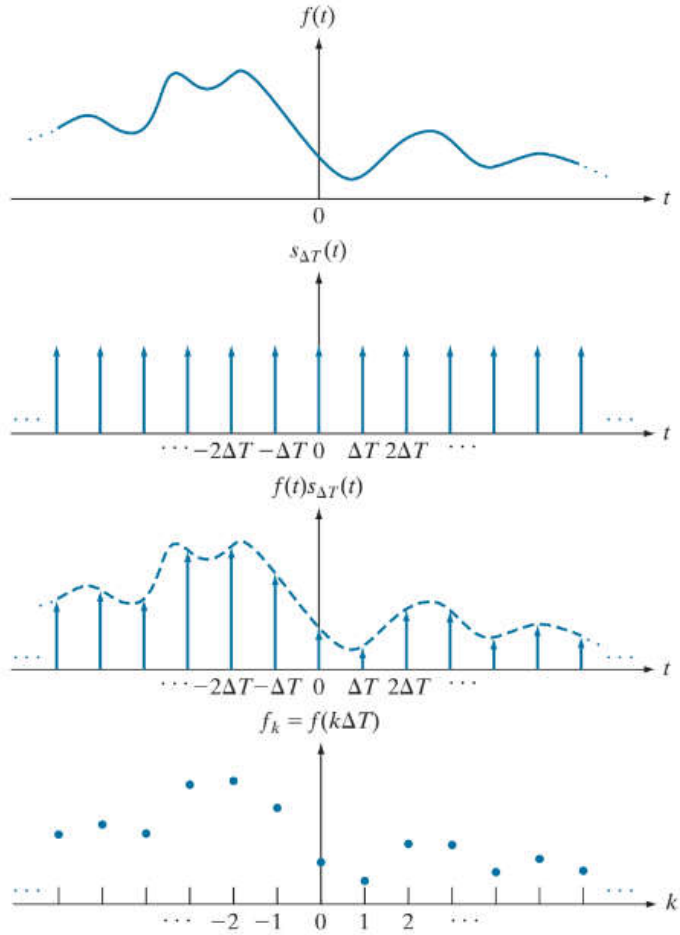
- Continuous functions have to be converted into a sequence of discrete values before they can be processed in a computer.
- This requires sampling and quantization.
- In the following discussion, we examine sampling in more detail.
- Consider a continuous function, f (t), that we wish to sample at uniform intervals, ΔT, of the independent variable t (see below figure).
- In the following FIGURE we will see,

(a) A continuous function.

(b) Train of impulses used to model sampling.

(c) Sampled function formed as the product of (a) and (b).

(d) Sample values obtained by integration and using the sifting property of impulses. (The dashed line in (c) is shown for reference. It is not part of the data.)



- We assume initially that the function extends from $-\infty$ to ∞ with respect to t.
- One way to model sampling is to multiply f (t) by a sampling function equal to a train of impulses ΔT units apart.
- That is,

$$\tilde{f}(t) = f(t)s_{\Delta T}(t) = \sum_{n=-\infty}^{\infty} f(t)\delta(t - n\Delta T)$$

where $\overline{f}$ (t) denotes the sampled function.

- Each component of this summation is an impulse weighted by the value of f (t) at the location of the impulse, as above Fig.(c) shows.
- The value of each sample is given by the "strength" of the weighted impulse, which we obtain by integration. That is, the value, f_k , of an arbitrary sample in the sampled sequence is given by

$$\begin{aligned} f_k &= \int_{-\infty}^{\infty} f(t)\delta(t - k\Delta T)\,dt \\ &= f(k\Delta T) \end{aligned}$$

where we used the sifting property of δ .

- Above Equation holds for any integer value k = ...,-2 ,-1 ,0,1,2,....
- Above Figure (d) shows the result, which consists of equally spaced samples of the original function.

THE FOURIER TRANSFORM OF SAMPLED FUNCTIONS

- Let F(μ) m denote the Fourier transform of a continuous function f (t).
- As discussed in the corresponding sampled function, $\overline{f}$ (t) , is the product of f (t) and an impulse train.
- We know from the convolution theorem that the Fourier transform of the product of two functions in the spatial domain is the convolution of the transforms of the two functions in the frequency domain.
- Thus, the Fourier transform of the sampled function is:

$$\begin{aligned}\tilde{F}(\mu) &= \Im\{\tilde{f}(t)\} = \Im\{f(t)s_{\Delta T}(t)\} \\ &= (F \star S)(\mu)\end{aligned}$$

where,

$S(\mu) = \frac{1}{\Delta T}\sum_{n=-\infty}^{\infty} \delta(\mu - \frac{n}{\Delta T})$ is the Fourier transform of the impulse train $s_{\Delta t}$(t).

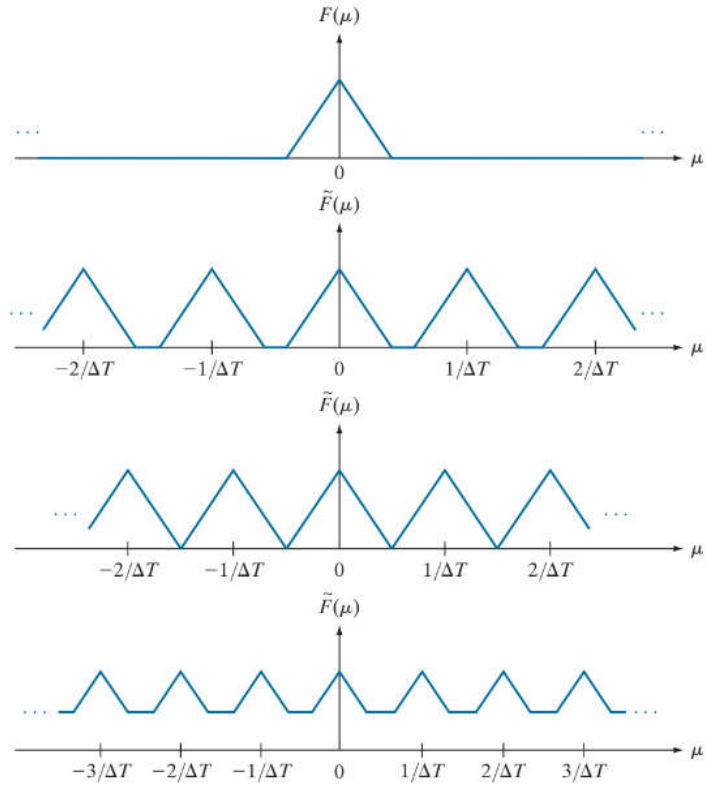
- We obtain the convolution of F(μ) and S(μ) directly from the 1-D definition of convolution:

$$\begin{aligned}\tilde{F}(\mu) = (F \star S)(\mu) &= \int_{-\infty}^{\infty} F(\tau)S(\mu-\tau)d\tau \\ &= \frac{1}{\Delta T}\int_{-\infty}^{\infty} F(\tau)\sum_{n=-\infty}^{\infty}\delta\left(\mu-\tau-\frac{n}{\Delta T}\right)d\tau \\ &= \frac{1}{\Delta T}\sum_{n=-\infty}^{\infty}\int_{-\infty}^{\infty} F(\tau)\delta\left(\mu-\tau-\frac{n}{\Delta T}\right)d\tau \\ &= \frac{1}{\Delta T}\sum_{n=-\infty}^{\infty} F\left(\mu-\frac{n}{\Delta T}\right)\end{aligned}$$

- where the final step follows from the sifting property of the impulse.
- The summation in the last line of Eq. shows that the Fourier transform $\overline{F}(\mu)$ of the sampled function $\overline{f}(t)$ is an infinite, periodic sequence of copies of the transform of the original, continuous function.
- The separation between copies is determined by the value of 1/ΔT.

- Observe that although $\overline{f}(t)$ is a sampled function, its transform, $\overline{F}(\mu)$, is continuous because it consists of copies of F(μ), which is a continuous function.

Following Figure is a graphical summary of the preceding results.



- Figure (a) is a sketch of the Fourier transform, F(μ), of a function f (t), and
- Figure (b) shows the transform, F(μ), of the sampled function, f (t).
- As mentioned in the previous section, the quantity 1/ΔT is the sampling rate used to generate the sampled function.
- So, in Figure (b) the sampling rate was high enough to provide sufficient separation between the periods, and thus preserve the integrity (i.e., perfect copies) of F(μ).
- In Figure(c), the sampling rate was just enough to preserve F(μ), but in Figure (d), the sampling rate was below the minimum required to maintain distinct copies of F(μ) ,and thus failed to preserve the original transform.
- Figure (b) is the result of an over-sampled signal, while Figs.(c) and (d) are the results of critically sampling and under-sampling the signal, respectively.

2.2 DISCRETE FOURIER TRANSFORM

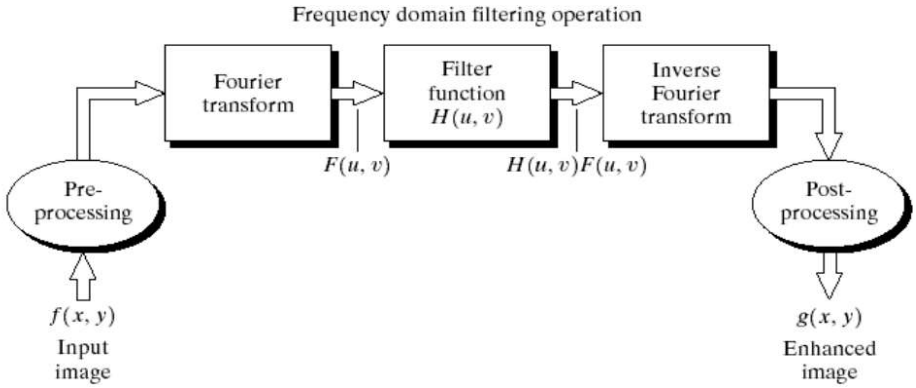
One of the principal goals of this chapter is the derivation of the discrete Fourier transform (DFT) starting from basic principles.

The material up to this point may be viewed as the foundation of those basic principles, so now we have in place the necessary tools to derive the DFT.

The DFT and Image Processing

To filter an image in the frequency domain:

1. Compute F(u,v) the DFT of the image
2. Multiply F(u,v) by a filter function H(u,v)
3. Compute the inverse DFT of the result



2.2.1 Convolution and Frequency Domain Filtering

CONVOLUTION:

- Convolution of two functions involves flipping (rotating by 180°) one function about its origin and sliding it past the other.
- At each displacement in the sliding process, we perform a computation, which, for discrete interested in the convolution of two continuous functions, f (t) and h (t), of one continuous variable, t, so we have to use integration instead of a summation.
- The convolution of these two functions, denoted as before by the operator , is defined as

$$(f \star h)(t) = \int_{-\infty}^{\infty} f(T)h(t-T)dT$$

- where the minus sign accounts for the flipping just mentioned, t is the displacement needed to slide one function past the other, and t is a dummy variable that is integrated out.
- We assume for now that the functions extend from $-\infty$ to ∞.

- We are interested in finding the Fourier transform:

$$\Im\{(f \star h)(t)\} = \int_{-\infty}^{\infty} \left[\int_{-\infty}^{\infty} f(\tau) h(t-\tau) d\tau \right] e^{-j2\pi\mu t} dt$$
$$= \int_{-\infty}^{\infty} f(\tau) \left[\int_{-\infty}^{\infty} h(t-\tau)\, e^{-j2\pi\mu t} dt \right] d\tau$$

- The term inside the brackets is the Fourier transform of h(t − T).
- We will show,

$$\Im\{h(t-\tau)\} = H(\mu) e^{-j2\pi\mu\tau}$$

where H(μ) m is the Fourier transform of h (t).

- Using this in the preceding equation gives us

$$\Im\{(f \star h)(t)\} = \int_{-\infty}^{\infty} f(\tau) \left[H(\mu) e^{-j2\pi\mu\tau} \right] d\tau$$
$$= H(\mu) \int_{-\infty}^{\infty} f(\tau) e^{-j2\pi\mu\tau}\, d\tau$$
$$= H(\mu) F(\mu)$$
$$= (H \bullet F)(\mu)$$

where " . " indicates multiplication.

- Remember, convolution is commutative, so the order of the functions in convolution expressions does not matter.
- If we refer to the domain of t as the spatial domain, and the domain of μ as the frequency domain, the preceding equation tells us that the Fourier transform of the convolution of two functions in the spatial domain is equal to the product in the frequency domain of the Fourier transforms of the two functions.
- Conversely, if we have the product of the two transforms, we can obtain the convolution in the spatial domain by computing the inverse Fourier transform.
- In other words, f✶h and H · F are a Fourier transform pair.
- This result is one-half of the convolution theorem and is written as

(f ✶ h) ⇔ (H·F) (μ)

The double arrow indicates that the expression on the right is obtained by taking the forward Fourier transform of the expression on the left,

while the expression on the left is obtained by taking the inverse Fourier transform of the expression on the right.

- Following a similar development would result in the other half of the convolution theorem:

$(f \cdot h)(t) \Leftrightarrow (H \star F)(\mu)$

which states that convolution in the frequency domain is analogous to multiplication in the spatial domain, the two being related by the forward and inverse Fourier transforms, respectively.

- The convolution theorem is the foundation for filtering in the frequency domain.
- The steps in filtering are given below.
  - At first step we have to do some pre – processing an image in spatial domain, means increase its contrast or brightness
  - Then we will take discrete Fourier transform of the image
  - Then we will center the discrete Fourier transform, as we will bring the discrete Fourier transform in center from corners
  - Then we will apply filtering, means we will multiply the Fourier transform by a filter function
  - Then we will again shift the DFT from center to the corners
  - Last step would be take to inverse discrete Fourier transform, to bring the result back from frequency domain to spatial domain
  - And this step of post processing is optional, just like pre processing , in which we just increase the appearance of image.

FREQUENCY DOMAIN FILTERING FUNDAMENTALS:

Frequency Domain Filters are used for smoothing and sharpening of image by removal of high or low frequency components. Sometimes it is possible to remove very high and very low frequencies.

- Frequency domain filters are different from spatial domain filters as it basically focuses on the frequency of the images.
- It is basically done for two basic operation i.e., Smoothing and Sharpening.
- Filtering in the frequency domain consists of modifying the Fourier transform of an image, then computing the inverse transform to obtain the spatial domain representation of the processed result.
- Thus, given (a padded) digital image, f(x ,y), of size P x Q pixels, the basic filtering equation in which we are interested has the form:

$$g(x,y) = \text{Real}\left\{\Im^{-1}\left[H(u,v)F(u,v)\right]\right\}$$

Where $\Im^{-1}$ is the IDFT , F (u,v) is the DFT of the input image, f(x,y), H(u,v) is a filter transfer function and g(x,y) is the filtered i.e output image.

- Function F,H, and g are arrays of size P x Q, the same as the padded input image.
- The product H(u,v) F(u,v) is formed using element wise multiplication.
- The filter transfer function modifies the transform of the input image to yield the processed output, g(x,y).
- The task of specifying H(u,v) is simplified considerably by using functions that are symmetric about their center, which requires that F(u,v) be centered also.
- STEPS FOR FILTERING IN THE FREQUENCY DOMAIN:

The process of filtering in the frequency domain can be summarized as follows:

1. Given an input image f(x,y) of size M x N, obtain the padding sizes P and Q that is, P=2M and Q=2N.
2. From a padded image $f_p(x,y)$ of size P x Q using zero-,mirror-,or replicate padding.
3. Multiply $f_p(x,y)$ by $(-1)^{x+y}$ to center the Fourier transform on the P x Q frequency rectangle.
4. Compute the DFT, F(u,v), of the image from step 3.
5. Construct a real, symmetric filter transfer function, H(u,v), of size P x Q with center at(P/2,Q/2)
6. From the product G(u,v) = H(u,v) F(u,v) using elementwise multiplication; that is, G(i,k)=H(i,k)F(i,k) for i=0,1,2,...,M-1 and k=0,1,2,...,N-1.
7. Obtain the final filtered images of size P x Q by computing the IDFT of G(u,v):

$$g_p(x,y) = \left(\text{real}\left[\Im^{-1}\{G(u,v)\}\right]\right)(-1)^{x+y}$$

8. Obtain the final filtered result, g(x,y), of the same size as the impute image by extracting the M x N region from the top,left quadrant of $g_p(x,y)$.

2.2.2 Smoothing using lowpass filters

- Low pass filtering i.e.smoothing, is employed to remove high spatial frequency noise from a digital image.

- The low-pass filters usually employ moving window operator which affects one pixel of the image at a time, changing its value by some function of a local region (window) of pixels.
- The operator moves over the image to affect all the pixels in the image.
- Three types of lowpass filters are:
  - IDEAL,
  - BUTTERWORTH, and
  - GAUSSIAN.
- These three categories cover the range from very sharp (ideal) to very smooth (Gaussian) filtering.
- The shape of a Butterworth filter is controlled by a parameter called the filter order.
- For large values of this parameter, the Butterworth filter approaches the ideal filter.
- For lower values, the Butterworth filter is more like a Gaussian filter.
- Thus, the Butterworth filter provides a transition between two "extremes."
- All filtering in this section follows the procedure outlined in the previous section, so all filter transfer functions, H(, u v), are understood to be of size P x Q ; that is, the discrete frequency variables are in the range u =0,1,2,...,P-1 and v = 0,1,2,...,Q-1 where P and Q are the padded sizes.
- IDEAL LOW PASS FILTER
  - Simply cut off all high frequency components that are a specified distance D_0 from the origin of the transform

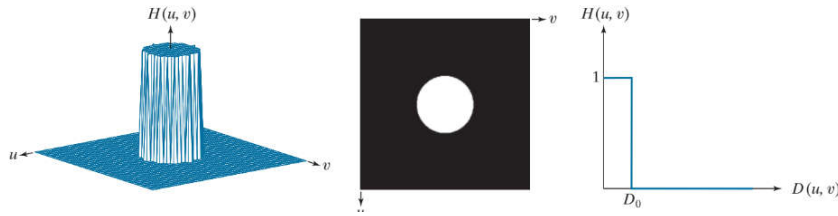


FIGURE (a) Perspective plot of an ideal lowpass-filter transfer function.
(b) Function displayed as an image.
(c) Radial cross section.

  - changing the distance changes the behaviour of the filter.

- The name ideal indicates that all frequencies on or inside a circle of radius D_0 are passed without attenuation,whereas all frequencies outside the circle are completely attenuated (filtered out).
- The ideal lowpass filter transfer function is radially symmetric about the origin.
- This means that it is defined completely by a radial cross section, as above Fig.(c) shows.
- A 2-D representation of the filter is obtained by rotating the cross section 360°.
- The transfer function for the ideal low pass filter can be given as:

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

where $D(u,v)$ is given as:

$$D(u,v)=[(u-M/2)^2+(v-N/2)^2]^{1/2}$$

- For an ILPF cross section, the point of transition between the values H(u, v) = 1 and H(u, v) = 0 is called the cutoff frequency.
- In above Fig.the cutoff frequency is D_0.
- The sharp cutoff frequency of an ILPF cannot be realized with electronic components, although they certainly can be simulated in a computer (subject to the constrain that the fastest possible transition is limited by the distance between pixels).
- The lowpass filters in this chapter are compared by studying their behavior as a function of the same cutoff frequencies.
- One way to establish standard cutoff frequency loci using circles that enclose specified amounts of total image power P_T ,which we obtain by summing the components of the power spectrum of the padded images at each point (u,v),for u = 0,1,2,...,P-1 and v = 0,1,2,...,Q-1 that is,

$$P_T=\sum_{u=0}^{P-1}\sum_{v=0}^{Q-1}P(u,v)$$

- If the DFT has been centered, a circle of radius D_0 with origin at the center of the frequency rectangle encloses □ percent of the power, where

$$\alpha=100\left[\sum_u\sum_v P(u,v)/P_T\right]$$

- and the summation is over values of (u,v) that lie inside the circle or on its boundary.

- Belove given Figures (a) and (b) show a test pattern image and its spectrum.
- The circles superimposed on the spectrum have radii of 10, 30, 60, 160, and 460 pixels, respectively, and enclosed the percentages of total power listed in the figure caption.
- The spectrum falls off rapidly, with close to 87% of the total power being enclosed by a relatively small circle of radius 10.
- The significance of this will become evident in the following example.

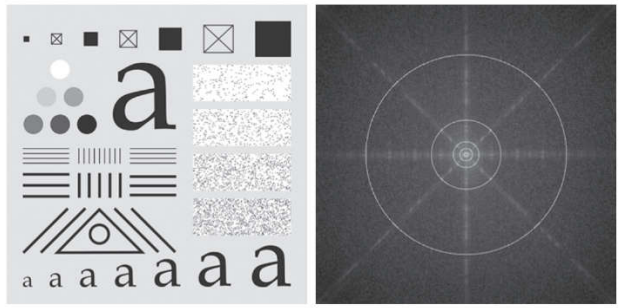
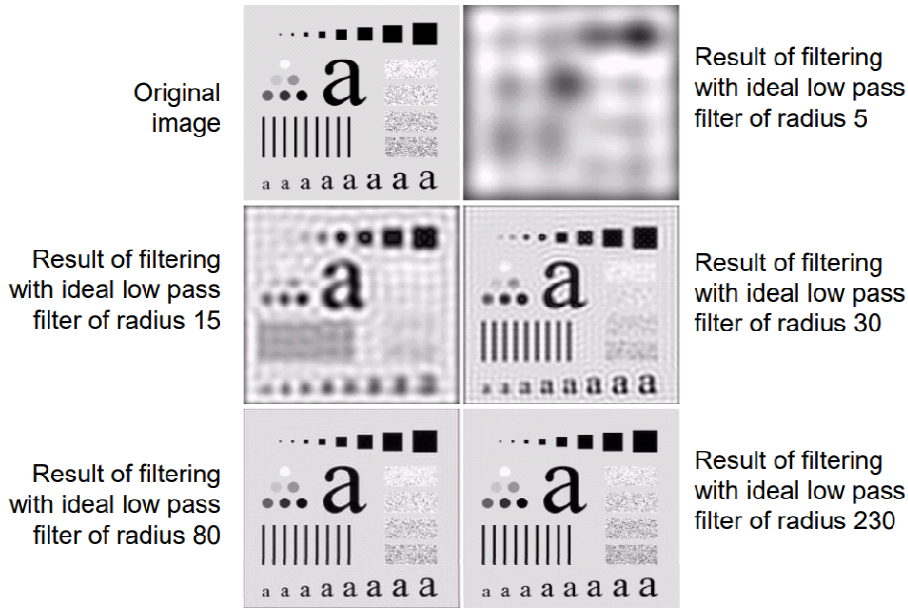


FIGURE (a) Test pattern of size 688 688 × pixels, and

(b) its spectrum. The spectrum is double the image size as a result of padding, but is shown half size to fit. The circles have radii of 10, 30, 60, 160, and 460 pixels with respect to the full-size spectrum. The radii enclose 86.9, 92.8,95.1, 97.6, and 99.4% of the padded image power, respectively.

- Above we show an image, it's Fourier spectrum and a series of ideal low pass filters of radius 5, 15, 30, 80 and 230 superimposed on top of it.



- It is clear from this example that ideal lowpass filtering is not practical.

- However, it is useful to study the behavior of ILPFs as part of our development of filtering concepts.
- Also, as shown in the discussion that follows, some interesting insight is gained by attempting to explain the ringing property of ILPFs in the spatial domain.

- **BUTTERWORTH LOWPASS FILTERS:**

- The transfer function of a Butterworth lowpass filter (BLPF) of order n, with cutoff frequency at a distance D_0 from the center of the frequency rectangle, is defined as

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

- Following Fig. shows a perspective plot, image display, and radial cross sections of the BLPF function.

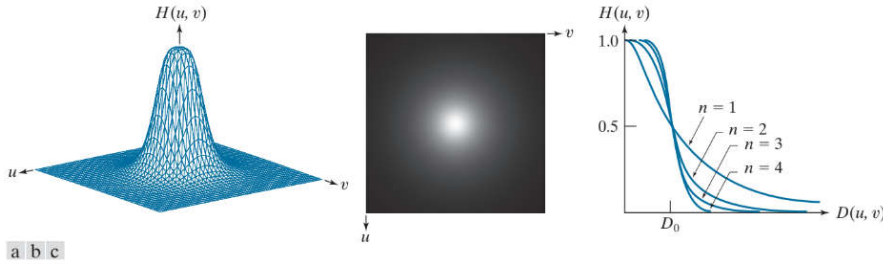


FIGURE (a) Perspective plot of a Butterworth lowpass-filter transfer function.

(b) Function displayed as an image.

(c) Radial cross sections of BLPFs of orders 1 through 4.

- Comparing the cross section plots in given Figs. we see that the BLPF function can be controlled to approach the characteristics of the ILPF using higher values of n, and the GLPF for lower values of n, while providing a smooth transition from low to high frequencies.
- Thus, we can use a BLPF to approach the sharpness of an ILPF function with considerably less ringing.

- **GAUSSIAN LOWPASS FILTERS:**

- Gaussian lowpass filter (GLPF) transfer functions have the form

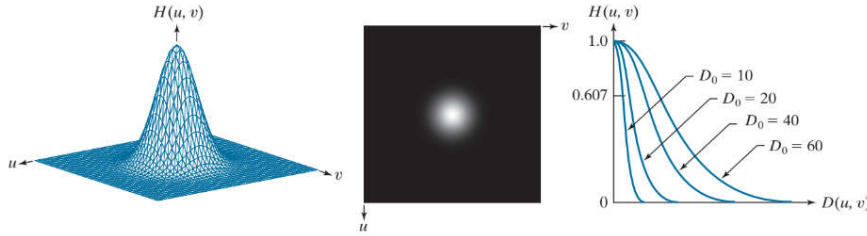
$$H(u,v) = e^{-D^2(u,v)/2\sigma^2}$$

- Unlike our earlier expressions for Gaussian functions, we do not use a multiplying constant here in order to be consistent with the filters discussed in this and later sections, whose highest value is 1.

- As before, σ is a measure of spread about the center.
- By letting $\sigma = D_0$, we can express the Gaussian transfer function in the same notation as other functions in this section:

$$H(u,v) = e^{-D^2(u,v)/2D_0^2}$$

- where D_0 is the cutoff frequency.
- When D (u, v) = D_0 the GLPF transfer function is down to 0.607 of its maximum value of 1.0.
- Following Figure shows a perspective plot, image display, and radial cross sections of a GLPF transfer function.



a b c

FIGURE (a) Perspective plot of a GLPF transfer function.

(b) Function displayed as an image.

(c) Radial cross sections for various values of D0

- Summary Table:

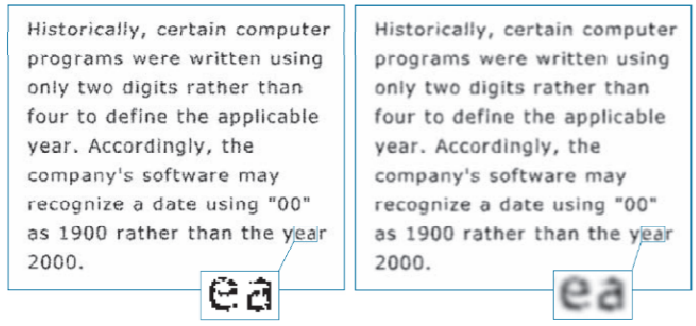
TABLE
Lowpass filter transfer functions. D_0 is the cutoff frequency, and n is the order of the Butterworth filter.

Ideal	Gaussian	Butterworth
$H(u,v) = \begin{cases} 1 & \text{if } D(u,v) \le D_0 \\ 0 & \text{if } D(u,v) > D_0 \end{cases}$	$H(u,v) = e^{-D^2(u,v)/2D_0^2}$	$H(u,v) = \dfrac{1}{1 + [D(u,v)/D_0]^{2n}}$

- **ADDITIONAL EXAMPLES OF LOWPASS FILTERING**
- The magnified section in following Fig.(a) shows that the characters in this document have distorted shapes due to lack of resolution, and many of the characters are broken.

a b

FIGURE
(a) Sample text of low resolution (note the broken characters in the magnified view). (b) Result of filtering with a GLPF, showing that gaps in the broken characters were joined.



- Although humans fill these gaps visually without difficulty, machine recognition systems have real difficulties reading broken characters.
- One approach for handling this problem is to bridge small gaps in the input image by blurring it.
- Figure (b) shows how well characters can be "repaired" by this simple process using a Gaussian lowpass filter with $D_0 = 120$.
- It is typical to follow the type of "repair" just described with additional processing, such as thresholding and thinning, to yield cleaner characters.
- Following figure shows an application of lowpass filtering for producing a smoother, softer-looking result from a sharp original.
- For human faces, the typical objective is to reduce the sharpness of fine skin lines and small blemishes.
- The magnified sections in Figs.(b) and (c) clearly show a significant reduction in fine skin lines around the subject's eyes.
- In fact, the smoothed images look quite soft and pleasing.

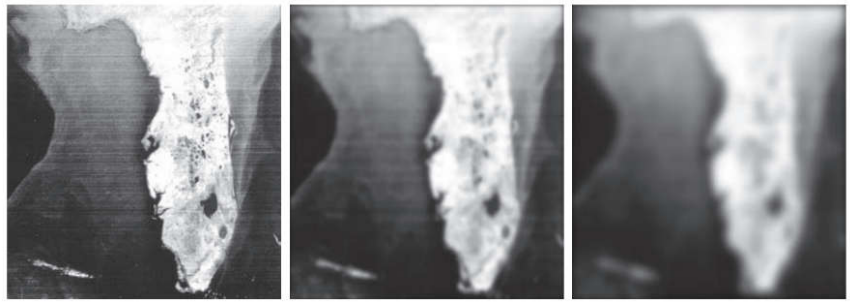
FIGURE 4.49

(a) Original 785x 732 image.

(b) Result of filtering using a GLPF with $D_0 = 150$.

(c) Result of filtering using a GLPF with $D_0 = 130$. Note the reduction in fine skin lines in the magnified sections in (b) and (c).

- Following shows two applications of lowpass filtering on the same image, but with totally different objectives.



a b c

FIGURE(a) 808 x 754 satellite image showing prominent horizontal scan lines.

(b) Result of filtering using a GLPF with $D_0 = 50$.

(c) Result of using a GLPF with D0 = 20. (Original image courtesy of NOAA.)

- Figure (a) is an 808 754 × segment of a very high resolution radiometer (VHRR) image showing part of the Gulf of Mexico (dark) and Florida (light) (note the horizontal sensor scan lines).
- The boundaries between bodies of water were caused by loop currents.
- This image is illustrative of remotely sensed images in which sensors have the tendency to produce pronounced scan lines along the direction in which the scene is being scanned.

2.2.3 Sharpening using high-pass filters

A high-pass filter can be used to make an image appear sharper.

These filters emphasize fine details in the image - the opposite of the low-pass filter.

High-pass filtering works in the same way as low-pass filtering; it just uses a different convolution kernel.

IDEAL, GAUSSIAN, AND BUTTERWORTH HIGHPASS FILTERS FROM LOWPASS FILTERS

- As was the case with kernels in the spatial domain , subtracting a lowpass filter transfer function from 1 yields the corresponding highpass filter transfer function in the frequency domain:

$$H_{HP}(u,v) = 1 - H_{LP}(u,v)$$

where $H_{LP}(u,v)$ is the transfer function of a lowpass filter.

- Thus, it follows from that an ideal highpass filter (IHPF) transfer function is given by

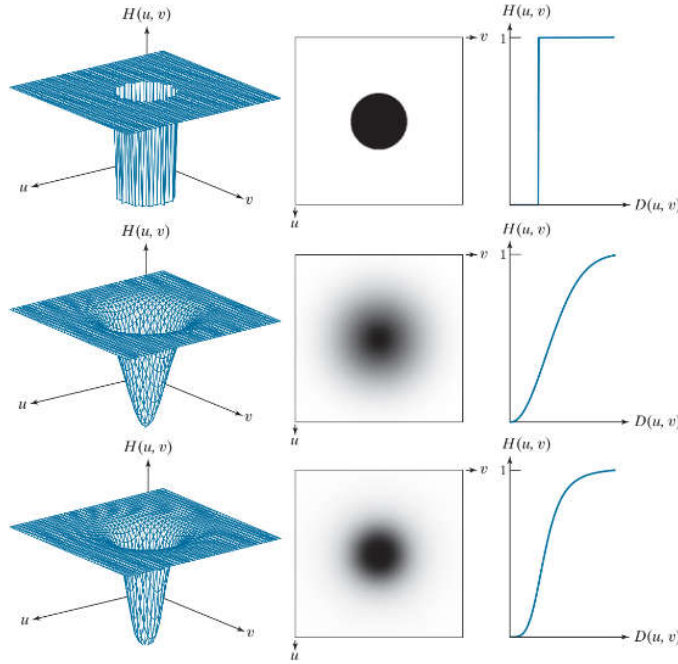
$$H(u,v) = \begin{cases} 0 & \text{if } D(u,v) \le D_0 \\ 1 & \text{if } D(u,v) > D_0 \end{cases}$$

where, as before, D(u, v) is the distance from the center of the PxQ frequency rectangle.

- Similarly, it follows that the transfer function of a Gaussian highpass filter (GHPF) transfer function is given by and, that the transfer function of a Butterworth highpass filter (BHPF) is

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

- Following figure shows 3-D plots, image representations, and radial cross sections for the preceding transfer functions.
- As before, we see that the BHPF transfer function in the third row of the figure represents a transition between the sharpness of the IHPF and the broad smoothness of the GHPF transfer function.



FIGURE

Top row: Perspective plot, image, and, radial cross section of an IHPF transfer function.

Middle and bottom rows: The same sequence for GHPF and BHPF transfer functions.

(The thin image borders were added for clarity. They are not part of the data.)

Summary Table:

TABLE 4.6
Highpass filter transfer functions. D_0 is the cutoff frequency and n is the order of the Butterworth transfer function.

Ideal	Gaussian	Butterworth
$H(u,v) = \begin{cases} 0 & \text{if } D(u,v) \le D_0 \\ 1 & \text{if } D(u,v) > D_0 \end{cases}$	$H(u,v) = 1 - e^{-D^2(u,v)/2D_0^2}$	$H(u,v) = \dfrac{1}{1 + [D_0/D(u,v)]^{2n}}$

Similarities between Low pass filter and High pass filter:

- Both filters are used to remove unwanted frequency components from a signal.
- Both filters have a cut-off frequency, which is the frequency at which the filter begins to attenuate the signal.
- The steepness of the cut-off slope depends on the order of the filter, with higher-order filters having steeper slopes.
- Both filters can introduce phase shifts, which can affect the time-domain characteristics of the signal.
- Both filters are used in a variety of applications including audio processing, image processing, communication systems, and biomedical signal processing.

Difference between Low pass filter and High pass filter:

Low pass filter	High pass filter
• It is used for smoothing the image.	• It is used for sharpening the image.
• It attenuates the high frequency.	• It attenuates the low frequency.
• Low frequency is preserved in it.	• High frequency is preserved in it.
• It allows the frequencies below the cut off frequency to pass through it.	• It allows the frequencies above cut off frequency to pass through it.
• It consists of a resistor followed by a capacitor.	• It consists of a capacitor that is followed by a resistor.
• It helps in removal of aliasing effects.	• It helps in removal of noise.
• G(u, v) = H(u, v) . F(u, v)	• H(u, v) = 1 – H'(u, v)

OBTAINING THE DFT FROM THE CONTINUOUS TRANSFORM OF A SAMPLED FUNCTION

- The Fourier transform of a sampled, band-limited function extending from $-\infty$ to ∞ is a continuous, periodic function that also extends from $-\infty$ to ∞.
- The objective of this section is to derive the DFT of such finite sample sets.

$$\tilde{F}(\mu) = (F \star S)(\mu) = \int_{-\infty}^{\infty} F(\tau) S(\mu - \tau)\, d\tau$$

$$= \frac{1}{\Delta T} \int_{-\infty}^{\infty} F(\tau) \sum_{n=-\infty}^{\infty} \delta\left(\mu - \tau - \frac{n}{\Delta T}\right) d\tau$$

$$= \frac{1}{\Delta T} \sum_{n=-\infty}^{\infty} \int_{-\infty}^{\infty} F(\tau)\, \delta\left(\mu - \tau - \frac{n}{\Delta T}\right) d\tau$$

$$= \frac{1}{\Delta T} \sum_{n=-\infty}^{\infty} F\left(\mu - \frac{n}{\Delta T}\right)$$

- Above equation gives the transform, $\overline{F}(\mu)$, of sampled data in terms of the transform of the original function, but it does not give us an expression for $\overline{F}(\mu)$ in terms of the sampled function $\overline{F}(t)$ itself.
- We find that expression directly from the definition of the transform in equation
- The Fourier transform of a continuous function f(t) of a continuous variable, t, denoted $\Im\{f(t)\}$, is defined by the equation

$$\Im\{f(t)\} = \int_{-\infty}^{\infty} f(t) e^{-j2\pi\mu t}\, dt$$

We get,

$$\overline{F}(\mu) = \int_{-\infty}^{\infty} \overline{f}(t) e^{-j2\pi ut}\, dt$$

- By substituting equation

$$\overline{f}(t) = f(t) s_{\Delta T}(t) = \sum_{n=-\infty}^{\infty} f(t)\delta(t - n\Delta t)$$

for $\overline{F}(t)$,

we obtain,

$$\tilde{F}(\mu) = \int_{-\infty}^{\infty} \tilde{f}(t) e^{-j2\pi\mu t} dt = \int_{-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f(t)\delta(t - n\Delta T) e^{-j2\pi\mu t} dt$$

$$= \sum_{n=-\infty}^{\infty} \int_{-\infty}^{\infty} f(t)\delta(t - n\Delta T) e^{-j2\pi\mu t} dt$$

$$= \sum_{n=-\infty}^{\infty} f_n e^{-j2\pi\mu n\Delta T}$$

- The last step follows from Eq.

$$f_k = \int_{-\infty}^{\infty} f(t)\delta(t - k\Delta T)dt$$
$$= f(k\Delta T)$$

and the sifting property of the impulse.

- Although f_n is a discrete function, its Fourier transform, $\overline{F}(u)$, is continuous and infinitely periodic with period 1/ΔT, as we know from Eq.

$$\tilde{F}(\mu) = (F \star S)(\mu) = \int_{-\infty}^{\infty} F(\tau)S(\mu - \tau)d\tau$$
$$= \frac{1}{\Delta T}\int_{-\infty}^{\infty} F(\tau)\sum_{n=-\infty}^{\infty}\delta\left(\mu - \tau - \frac{n}{\Delta T}\right)d\tau$$
$$= \frac{1}{\Delta T}\sum_{n=-\infty}^{\infty}\int_{-\infty}^{\infty} F(\tau)\delta\left(\mu - \tau - \frac{n}{\Delta T}\right)d\tau$$
$$= \frac{1}{\Delta T}\sum_{n=-\infty}^{\infty} F\left(\mu - \frac{n}{\Delta T}\right)$$

.

- Therefore, all we need to characterize $\overline{F}(u)$ is one period, and sampling one period of this function is the basis for the DFT.

- Suppose that we want to obtain M equally spaced samples of $\overline{F}(u)$ taken over the one period interval from $\mu = 0$ to $\mu = 1/\Delta$ T .

- This is accomplished by taking the samples at the following frequencies:

$$\mu = \frac{m}{M\Delta T} \quad \text{m} = 0,1,2, \ldots\ldots, \text{M} - 1$$

- Substituting this result for m into Eq.

$$\tilde{F}(\mu) = \int_{-\infty}^{\infty}\tilde{f}(t)e^{-j2\pi\mu t}dt = \int_{-\infty}^{\infty}\sum_{n=-\infty}^{\infty} f(t)\delta(t - n\Delta T)e^{-j2\pi\mu t}dt$$
$$= \sum_{n=-\infty}^{\infty}\int_{-\infty}^{\infty} f(t)\delta(t - n\Delta T)e^{-j2\pi\mu t}dt$$
$$= \sum_{n=-\infty}^{\infty} f_n e^{-j2\pi\mu n\Delta T}$$

and letting F_m denote the result yields

$F_m = \sum_{n=0}^{M-1} F_n e^{-j2\pi mn/M}$ m = 0,1,2…. ----------

---------- Eq. A

- This expression is the discrete Fourier transform we are seeking.Given a set $\{f_m\}$ consisting of M samples of f (t), above Eq. yields a set $\{f_m\}$ of M complex values corresponding to the discrete Fourier transform of the input sample set.
- Conversely, given $\{f_m\}$, we can recover the sample set $\{f_m\}$ by using the inverse discrete Fourier transform (IDFT)

$$f_n = \frac{1}{M}\sum_{m=0}^{M-1} F_m e^{j2\pi mn/M} \qquad n = 0,1,2\ldots.$$

---------- Eq. B

- substituting Eq. (B) for f_n into Eq. (A) gives the identity $F_m \equiv F_m$.
- Similarly, substituting Eq. (A) into Eq. (B) for F_m yields $f_n \equiv f_n$.
- This implies that Eqs. (A) and (B) constitute a discrete Fourier transform pair.
- Furthermore, these identities indicate that the forward and inverse Fourier transforms exist for any set of samples whose values are finite.
- Note that neither expression depends explicitly on the sampling interval ΔT, nor on the frequency intervals of Eq.

$$\mu = \frac{m}{M\Delta T} \qquad m = 0,1,2, \ldots.. , M - 1$$

- Therefore, the DFT pair is applicable to any finite set of discrete samples taken uniformly.
- We used m and n in the preceding development to denote discrete variables because it is typical to do so for derivations.
- However, it is more intuitive, especially in two dimensions, to use the notation x and y for image coordinate variables and u and v for frequency variables, where these are understood to be integers.
- Then,
  - Eqs. (A) and (B) become

$$F(u) = \sum_{x=0}^{M-1} F(x) e^{-j2\pi ux/M} \qquad u = 0,1,2\ldots.$$

------ Eq. C

and

$$F(x) = \frac{1}{M}\sum_{u=0}^{M-1} F(u) e^{j2\pi ux/M} \qquad x = 0,1,2\ldots.$$

-------- Eq. D

- where we used functional notation instead of subscripts for simplicity.

- Comparing Eqs. (A) through (D), you can see that $F(u) \equiv F_m$ and $f(x) \equiv f_n$.
- From this point on, we use Eqs. (C) and (D) to denote the 1-D DFT pair.
- As in the continuous case, we often refer to Eq. (C) as the forward DFT of f (x), and to Eq. (D) as the inverse DFT of F (u).
- As before, we use the notation $f(x) \Leftrightarrow F(u)$ to denote a Fourier transform pair.
- Sometimes you will encounter in the literature the 1/M term in front of Eq. (C) instead.
- That does not affect the proof that the two equations form a Fourier transform pair.
- Knowledge that f(x) and F(u) are a transform pair is useful in proving relationships between functions and their transforms.
- For example, you are asked in Problem 4.17 to show that

  $f(x-x_0) \Leftrightarrow F(u)\ e^{-j2\pi ux_0/M}$ is a Fourier transform pair.

- That is, you have to show that the DFT of and, conversely, that the inverse DFT of

  $f(x - x_0)$ is $F(u)\ e^{-j2\pi ux_0/M}$

- Because this is done by substituting directly into Eqs. (C) and (D), and you will have proved already that these two equations constitute a Fourier transform pair , if you prove that one side of "$\Leftrightarrow$" is the DFT (IDFT) of the other, then it must be true the other side is the IDFT (DFT) of the side you just proved.
- It turns out that having the option to prove one side or the other often simplifies proofs significantly.
- This is true also of the 1-D continuous and 2-D continuous and discrete Fourier transform pairs.
- It can be shown that both the forward and inverse discrete transforms are infinitely periodic, with period M.
- That is,

  $F(u) = F(u + kM)$

  and

  $f(x)=f(x + kM)$

  where k is an integer.

- The discrete equivalent of the 1-D convolution is

$$f(x) * h(x) = \sum_{m=0}^{M-1} f(m)h(x-m) \qquad \text{Where } x = 0,1,2,...,M-1 \quad \text{---------- Eq. F}$$

- Because in the preceding formulations the functions are periodic, their convolution also is periodic. Above Equation F gives one period of the periodic convolution.
- For this reason, this equation often is referred to as circular convolution.
- This is a direct result of the periodicity of the DFT and its inverse.
- This is in contrast with the convolution in which values of the displacement, x, were determined by the requirement of sliding one function completely past the other, and were not fixed to the range [0,M-1] as in circular convolution.
- Finally, we point out that the convolution theorem is applicable also to discrete variables.

RELATIONSHIP BETWEEN THE SAMPLING AND FREQUENCY INTERVALS

- If f (x) consists of M samples of a function f (t) taken ΔT units apart, the length of the record comprising the set {f(x)}, x = 0,1,2,...,M-1, is T = MΔT
- The corresponding spacing, Δu, in the frequency domain follows from Eq.

$$u = \frac{m}{M\Delta T} \quad \text{m} = 0,1,2,..,\ M - 1 \ :$$

$$\Delta u = \frac{1}{M\Delta T} = \frac{1}{T}$$

- The entire frequency range spanned by the M components of the DFT is then

$$R = M\Delta u = \frac{1}{\Delta T}$$

- Thus, we see from above Eqs. that the resolution in frequency, Δu, of the DFT depends inversely on the length (duration, if t is time) of the record, T, over which the continuous function, f (t), is sampled; and the range of frequencies spanned by the DFT depends on the sampling interval ΔT.
- Keep in mind these inverse relationships between Δu and ΔT.

2.3 FAST FOURIER TRANSFORMS

It is important to develop a basic understanding of methods by which Fourier transform computations can be simplified and speeded up.

SEPARABILITY OF THE 2-D DFT

- The 2-D DFT is separable into 1-D transforms.
- We can write Eq.as

$$F(u,v) = \sum_{x=0}^{M-1} e^{-j2\pi nx/M} \sum_{y=0}^{N-1} f(x,y) e^{-j2\pi vy/N}$$

$$= \sum_{x=0}^{M-1} F(x,v) e^{-j2\pi ux/M}$$

where

$$F(u,v) = \sum_{v=0}^{N-1} f(x,y) e^{-j2\pi vy/M}$$

- For one value of x, and for v = 0,1,2,...,N-1, we see that F (x ,v) is the 1-D DFT of one row of f(x ,y).
- We conclude that the 2-D DFT of f (x,y) can be obtained by computing the 1-D transform of each row of f (x,y) and then computing the 1-D transform along each column of the result.
- This is an important simplification because we have to deal only with one variable at a time.
- A similar development applies to computing the 2-D IDFT using the 1-D IDFT.
- However, as we show in the following section, we can compute the IDFT using an algorithm designed to compute the forward DFT, so all 2-D Fourier transform computations are reduced to multiple passes of a 1-D algorithm designed for computing the 1-D DFT.

COMPUTING THE IDFT USING A DFT ALGORITHM

- Taking the complex conjugate of both sides of Eq.and multiplying the results by MN yields

$$MNf^*(x,y) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F^*(u,v) e^{-j2\pi(ux/M + vy/N)}$$

- If we substitute F*(u,v) into an algorithm designed to compute the 2-D forward Fourier transform, the result will be MNf^*(x ,y).
- Taking the complex conjugate and dividing this result by MN yields f (x ,y), which is the inverse of F(u, v).

- The key concept to keep in mind is that we simply input F*(u,v)into whatever forward algorithm we have.
- The result will be MNf*(x,y).
- All we have to do with this result to obtain f(x ,y) is to take its complex conjugate and divide it by the constant MN.
- When f (x ,y) is real, as typically is the case, then f^*(x,y)= f(x,y).

THE FAST FOURIER TRANSFORM (FFT)

- The fast Fourier transform (FFT), reduces computations to the order of MN $\log_2$ MN multiplications and additions.
- The computational reductions afforded by the FFT are impressive indeed.
- For example, computing the 2-D FFT of a 2048 X 2048 image would require on the order of 92 million multiplication and additions, which is a significant reduction from the one trillion computations mentioned.
- The algorithm we selected to accomplish this objective is the so-called successive-doubling method, which was the original algorithm that led to the birth of an entire industry.
- This particular algorithm assumes that the number of samples is an integer power of 2, but this is not a general requirement of other approaches (Brigham [1988]).
- We know from the previous section that 2-D DFTs can be implemented by successive passes of the 1-D transform, so we need to focus only on the FFT of one variable.
- In derivations of the FFT, it is customary to express Eq. in the form

$$F(u) = \sum_{x=0}^{M-1} f(x) W_M^{ux}$$

- for u = 0,1,2,....,M-1 , where

$W_M = e^{-j2\pi/M}$

- and M is assumed to be of the form

$M=2^P$

- Where p is a positive integer.
- Then it follows that M can be expressed as

M=2k

- With K being a positive integer and by substituting values

$$F(u) = \sum_{x=0}^{2K-1} f(x) W_{2K}^{ux}$$

$$= \sum_{x=0}^{K-1} f(2x) W_{2K}^{u(2x)} + \sum_{x=0}^{K-1} f(2x+1) W_{2K}^{u(2x+1)}$$

- However, it can be shown using Eq. that $W_{2K}^{2ux} = W_K^{ux}$, so Eq. can be written as

$$F(u) = \sum_{x=0}^{K-1} f(2x)\, W_K^{ux} + \sum_{x=0}^{K-1} f(2x+1)\, W_{2K}^{u(2x+1)}$$

- Defining

$$F_{even}(u) = \sum_{x=0}^{K-1} f(2x)\, W_K^{ux} \quad \text{for } u=0,1,2,...,K-1, \text{ and}$$

$$F_{odd}(u) = \sum_{x=0}^{K-1} f(2x+1)\, W_K^{ux} \quad \text{for } u=0,1,2,...,K-1, \text{ reduces Eq. to}$$

- $F(u) = F_{even}(u) + F_{odd}(u)\, W_K^u$
- Also, because $W_M^{u+K} = W_K^u$ and $W_{2K}^{u+K} = -W_{2K}^u$, it follows that

$$F(u + K) = F_{even}(u) - F_{odd}(u)\, W_{2K}^u$$

- An M-point DFT can be computed by dividing the original expression into two parts, as indicated in above Eqs.
- Computing the first half of F (u) requires evaluation of the two (M/2)-point transforms given in Eqs.
- The resulting values of $F_{even}(u)$ and $F_{odd}(u)$ are then substituted into Eq.to obtain F(u)

  for u =0,1,2,...,(M/2-1).
- The other half then follows directly from Eq. without additional transform evaluations.
- It provides computationally efficient algorithms for evaluating the DFT.
- Direct computation of DFT has large number addition and multiplication operations.
- The DFT has various applications such as linear filtering, correlation analysis, and spectrum analysis.
- Hence an efficient computation of DFT is an important issue in DSP.

There are two different approaches in computing efficient DFT, those are:

1 Divide and Conquer approach

2 DFT as Linear filtering.

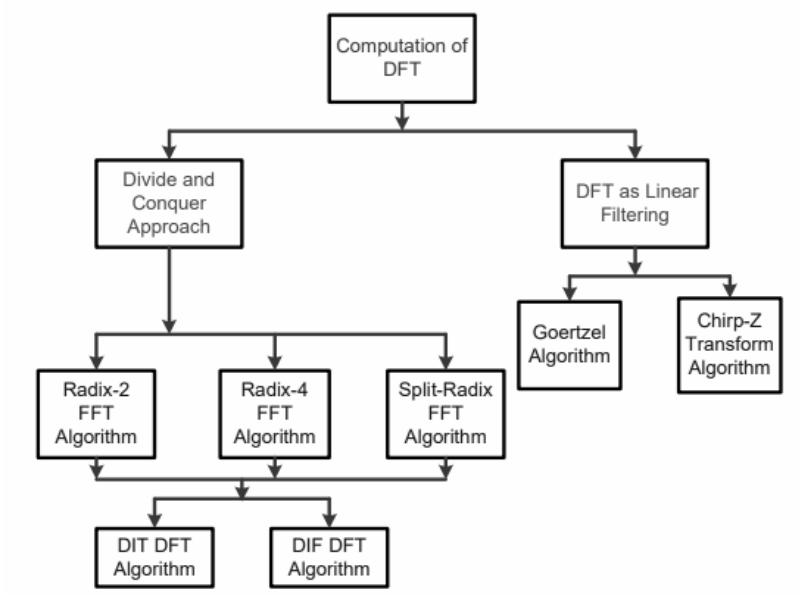


Figure 1: FFT Algorithms

DFT Solved Examples

Example 1

Verify Parseval's theorem of the sequence $x(n) = \frac{1}{4}^n u(n)$

Solution – $\sum_{-\infty}^{\infty} |x_1(n)|^2 = \frac{1}{2\pi} \int_{-\pi}^{\pi} |X_1(e^{j\omega})|^2 d\omega$

L.H.S $\sum_{-\infty}^{\infty} |x_1(n)|^2$

$$= \sum_{-\infty}^{\infty} x(n)x^*(n)$$

$$= \sum_{-\infty}^{\infty} (\frac{1}{4})^{2n} u(n) = \frac{1}{1-\frac{1}{16}} = \frac{16}{15}$$

R.H.S. $X(e^{j\omega}) = \frac{1}{1-\frac{1}{4}e-j\omega} = \frac{1}{1-0.25\cos\omega+j0.25\sin\omega}$

$$\iff X^*(e^{j\omega}) = \frac{1}{1-0.25\cos\omega-j0.25\sin\omega}$$

Calculating, $X(e^{j\omega}).X^*(e^{j\omega})$

$$= \frac{1}{(1-0.25\cos\omega)^2+(0.25\sin\omega)^2} = \frac{1}{1.0625-0.5\cos\omega}$$

$$\frac{1}{2\pi}\int_{-\pi}^{\pi} \frac{1}{1.0625-0.5\cos\omega} d\omega$$

$$\frac{1}{2\pi}\int_{-\pi}^{\pi} \frac{1}{1.0625-0.5\cos\omega} d\omega = 16/15$$

We can see that, LHS = RHS. *HenceProved*

Example 2

Compute the N-point DFT of $x(n) = 3\delta(n)$

Solution – We know that,

$$X(K) = \sum_{n=0}^{N-1} x(n) e^{\frac{j2\Pi kn}{N}}$$

$$= \sum_{n=0}^{N-1} 3\delta(n) e^{\frac{j2\Pi kn}{N}}$$

$$= 3\delta(0) \times e^0 = 1$$

So, $x(k) = 3, 0 \leq k \leq N-1$ ... Ans.

Example 3

Compute the N-point DFT of $x(n) = 7(n - n_0)$

Solution – We know that,

$$X(K) = \sum_{n=0}^{N-1} x(n) e^{\frac{j2\Pi kn}{N}}$$

Substituting the value of xn,

$$\sum_{n=0}^{N-1} 7\delta(n - n_0) e^{-\frac{j2\Pi kn}{N}}$$

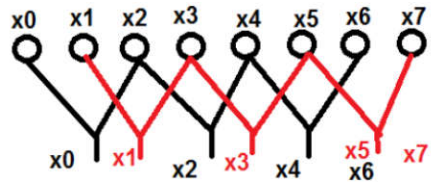
$$= e^{-kj14\Pi kn_0/N}$$ ... Ans

- In DFT methods, we have seen that the computational part is too long.
- We can reduce that through FFT or fast Fourier transform.
- So, we can say FFT is nothing but computation of discrete Fourier transform in an algorithmic format, where the computational part will be reduced.
- The main advantage of having FFT is that through it, we can design the FIR filters.
- Mathematically, the FFT can be written as follows;

$$x[K] = \sum_{n=0}^{N-1} x[n] W_N^{nk}$$

Example 4

Consider eight points named from x_0 to x_7 . We will choose the even terms in one group and the odd terms in the other. Diagrammatic view of the above said has been shown below:



Here, points x_0, x_2, x_4 and x_6 have been grouped into one category and

Similarly, points x_1, x_3, x_5 and x_7 have been put into another category.

Now, we can further make them in a group of two and can proceed with the computation.

Now, let us see how these breaking into further two is helping in computation.

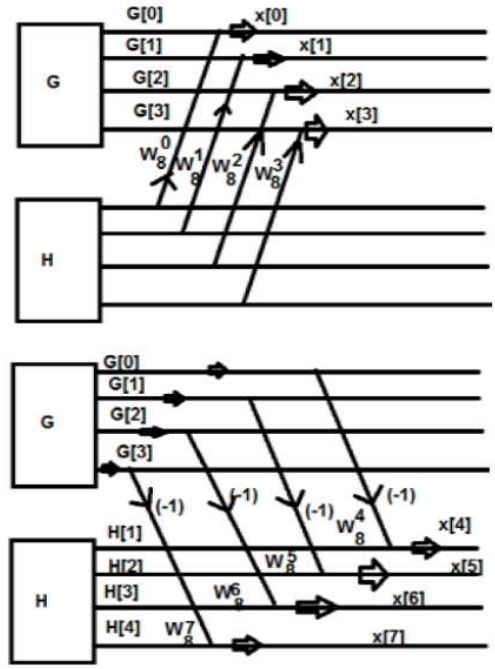
$$x[k] = \sum_{r=0}^{\frac{N}{2}-1} x[2r]W_N^{2rk} + \sum_{r=0}^{\frac{N}{2}-1} x[2r+1]W_N^{(2r+1)k}$$

$$= \sum_{r=0}^{\frac{N}{2}-1} x[2r]W_{N/2}^{rk} + \sum_{r=0}^{\frac{N}{2}-1} x[2r+1]W_{N/2}^{rk} \times W_N^k$$

$$= G[k] + H[k] \times W_N^k$$

Initially, we took an eight-point sequence, but later we broke that one into two parts G[k] and H[k]. G[k] stands for the even part whereas H[k] stands for the odd part.

If we want to realize it through a diagram, then it can be shown as below:



From the above figure, we can see that

$$W_8^4 = -1$$

$$W_8^5 = -W_8^1$$

$$W_8^6 = -W_8^2$$

$$W_8^7 = -W_8^3$$

Similarly, the final values can be written as follows −

$$G[0] - H[0] = x[4]$$

$$G[1] - W_8^1 H[1] = x[5]$$

$$G[2] - W_8^2 H[2] = x[6]$$

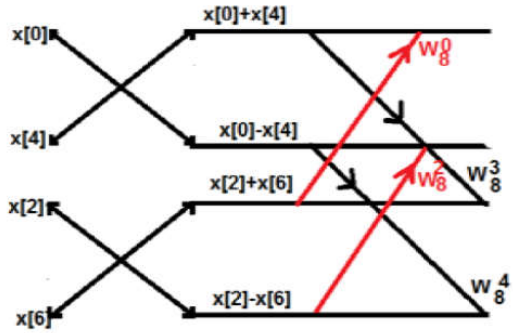
$$G[1] - W_8^3 H[3] = x[7]$$

The above one is a periodic series.

The disadvantage of this system is that K cannot be broken beyond 4 point.

Now Let us break down the above into further.

We will get the structures something like this

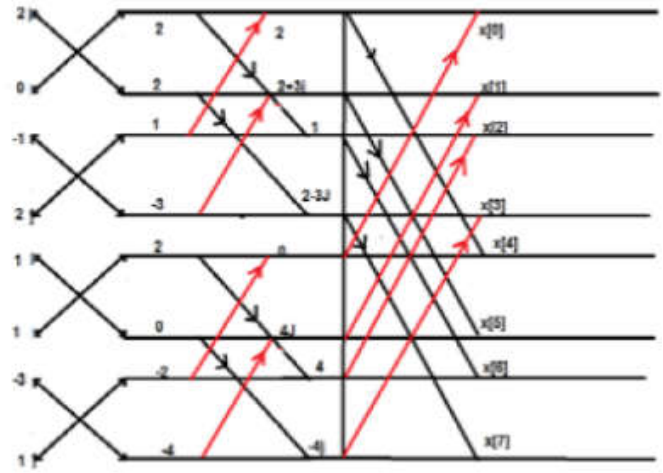


Example 5

Consider the sequence x[n]={ 2,1,-1,-3,0,1,2,1}. Calculate the FFT.

Solution − The given sequence is x[n]={ 2,1,-1,-3,0,1,2,1}

Arrange the terms as shown below;



Example 6

Find the DFT of a sequence x(n)= {1,1,0,0} and find the IDFT of Y(K)= {1,0,1,0}

Let us assume $N = L = 4$.

We have $X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi nk/N} \quad k = 0, 1, \ldots, N-1$

$$X(0) = \sum_{n=0}^{3} x(n) = x(0) + x(1) + x(2) + x(3)$$

$$= 1 + 1 + 0 + 0 = 2$$

$$X(1) = \sum_{n=0}^{3} x(n)e^{-j\pi n/2} = x(0) + x(1)e^{-j\pi/2} + x(2)e^{-j\pi} + x(3)e^{-j3\pi/2}$$

$$= 1 + \cos\frac{\pi}{2} - j\sin\frac{\pi}{2}$$

$$= 1 - j$$

$$X(2) = \sum_{n=0}^{3} x(n)e^{-j\pi n} = x(0) + x(1)e^{-j\pi} + x(2)e^{-j2\pi} + x(3)e^{-j3\pi}$$

$$= 1 + \cos\pi - j\sin\pi$$

$$= 1 - 1 = 0$$

$$X(3) = \sum_{n=0}^{3} x(n)e^{-j3n\pi/2} = x(0) + x(1)e^{-j3\pi/2} + x(2)e^{-j3\pi} + x(3)e^{-j9\pi/2}$$

$$= 1 + \cos\frac{3\pi}{2} - j\sin\frac{3\pi}{2}$$

$$= 1 + j$$

$$X(k) = \{2, 1 - j, 0, 1 + j\}$$

$$y(n) = \frac{1}{N}\sum_{k=0}^{N-1} Y(k)e^{j2\pi nk/N} \quad n = 0, 1, \ldots N-1$$

$$y(0) = \frac{1}{4}\sum_{k=0}^{3} Y(k) \quad n = 0, 1, 2, 3$$

$$= \frac{1}{4}[y(0) + y(1) + y(2) + y(3)]$$

$$= \frac{1}{4}[1 + 0 + 1 + 0]$$

$$= 0.5$$

$$y(1) = \frac{1}{N}\sum_{k=0}^{3} Y(k)e^{j\pi k/2}$$

$$y(1) = \frac{1}{4}\left[Y(0) + Y(1)e^{j\pi/2} + Y(2)e^{j\pi} + Y(3)e^{j3\pi/2}\right]$$

$$= \frac{1}{4}[1 + 0 + \cos\pi + j\sin\pi + 0]$$

$$= \frac{1}{4}[1 + 0 - 1 + 0] = 0$$

$$y(2) = \frac{1}{4}\left[Y(0) + Y(1)e^{j\pi} + Y(2)e^{j2\pi} + Y(3)e^{j3\pi}\right]$$

$$= \frac{1}{4}[1 + 0 + \cos 2\pi + j\sin 2\pi + 0]$$

$$= \frac{1}{4}[1 + 0 + 1 + 0] = 0.5$$

$$y(3) = \frac{1}{4}\left[Y(0) + Y(1)e^{j3\pi/2} + Y(2)e^{j3\pi} + Y(3)e^{j9\pi/2}\right]$$

$$= \frac{1}{4}[1 + 0 + \cos 3\pi + j\sin 3\pi + 0]$$

$$= \frac{1}{4}[1 + 0 + (-1) + 0] = 0$$

$$y(n) = \{0.5, 0, 0.5, 0\}$$

Example 7

Find the DFT of a sequence

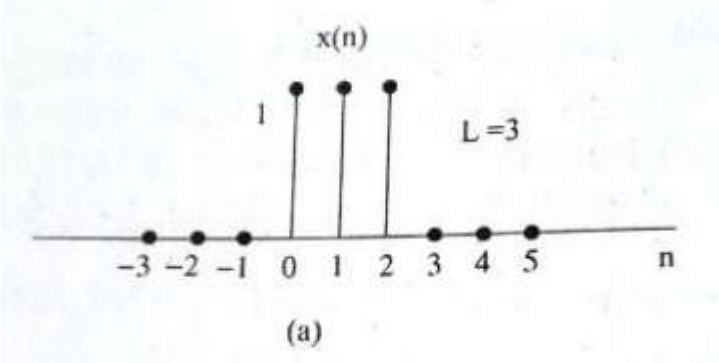
x(n) = 1 for $0 \le n \le 2$

= 0 otherwise

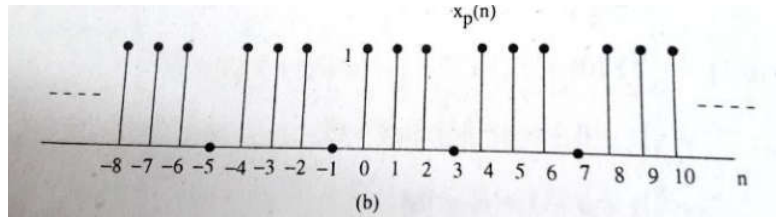
For (i) N=4 (ii) N=8. Plot | X(K) | and ∟X(K)

Solution: i) N= 4

Fig a) Sequence given in problem



b) Periodic extension of the sequence for N=4



For $N = 4$

$$X(k) = \sum_{n=0}^{3} x(n) e^{-j\pi nk/2} \quad k = 0,1,2,3$$

For $k = 0$

$$X(0) = \sum_{n=0}^{3} x(n) = x(0) + x(1) + x(2) + x(3)$$
$$= 3$$

Therefore, $|X(0)| = 3, \angle X(0) = 0$

For $k = 1$

$$X(1) = \sum_{n=0}^{3} x(n) e^{-j\pi n/2}$$
$$= x(0) + x(1)e^{-j\pi/2} + x(2)e^{-j\pi} + x(3)e^{-j3\pi/2}$$
$$= 1\cos\frac{\pi}{2} - j\sin\frac{\pi}{2} + \cos\pi - j\sin\pi + 0$$
$$= 1 - j - 1 = -j$$
$$|X(1)| = 1, \quad \angle X(1) = \frac{-\pi}{2}$$

For $k = 2$

$$X(2) = \sum_{n=0}^{3} x(n) e^{-j\pi n}$$
$$= x(0) + x(1)e^{-j\pi} + x(2)e^{-j2\pi} + x(3)e^{-j3\pi}$$
$$= 1 + \cos\pi - j\sin\pi + \cos 2\pi - j\sin 2\pi + 0$$
$$= 1 - 1 + 1 = 1$$

Therefore,

$$|X(2)| = 1, \quad \angle X(2) = 0$$

For $k = 3$

$$X(3) = \sum_{n=0}^{3} x(n)e^{-j3\pi n/2}$$

$$= x(0) + x(1)e^{-j3\pi/2} + x(2)e^{-j3\pi} + x(3)e^{-j9\pi/2}$$

$$= 1 + \cos\frac{3\pi}{2} - j\sin\frac{3\pi}{2} + \cos 3\pi - j\sin 3\pi + 0$$

$$= 1 + j - 1 = j$$

Therefore, $|X(3)| = 1, \quad \angle X(3) = \frac{\pi}{2}$

$$|X(k)| = \{3,1,1,1\}$$

$$\angle X(k) = \left\{0, -\frac{\pi}{2}, 0, \frac{\pi}{2}\right\}$$

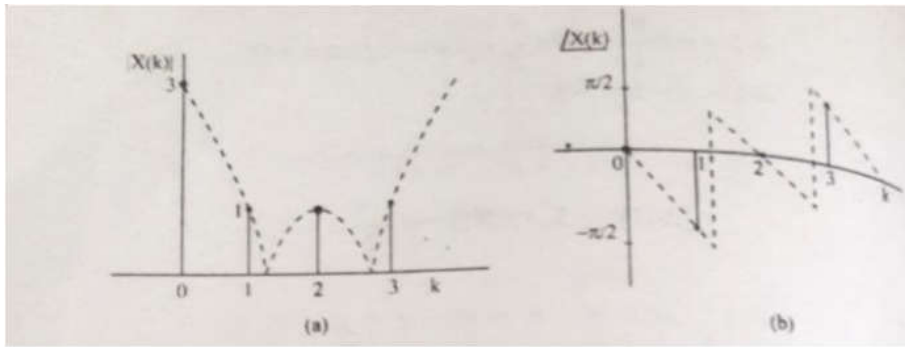


Fig: frequency response of x(n) for N=4

ii) N= 8

For $N = 8$

The periodie extension of $x(n)$ is shown in fig. 3.7 can be obtained by adding five ($\because N - L$ zeros)

$x(0) = x(1) = x(2) = 1$ and $x(n) = 0$ for $3 \le n \le 7$

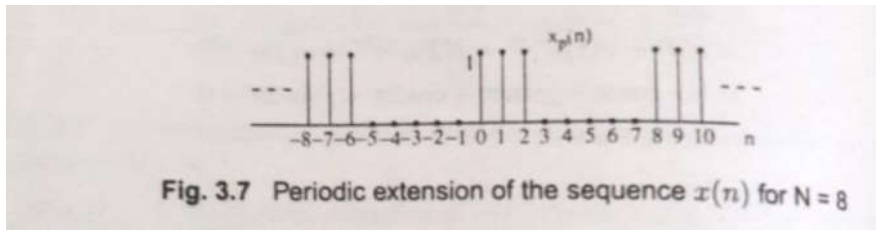


Fig. 3.7 Periodic extension of the sequence $x(n)$ for N = 8

For $N = 8$

$$X(k) = \sum_{n=0}^{7} x(n) e^{-j\pi nk/4} \qquad \kappa = 0,1...7$$

For $k = 0$

$$X(0) = \sum_{n=0}^{7} x(n)$$

$$X(0) = 1+1+1+0+0+0+0+0 = 3$$

Therefore, $|X(0)| = 3 \qquad \angle X(0) = 0$

For $k = 1$

$$\begin{aligned} X(1) &= \sum_{n=0}^{7} x(n) e^{-j\pi n/4} \\ &= x(0) + x(1)e^{-j\pi/4} + x(2)e^{-j\pi/2} \\ &= 1 + 0.707 - j0.707 + 0 - j \\ &= 1.707 - j1.707 \end{aligned}$$

Therefore,

$$|X(1)| = 2.414, \qquad \angle X(1) = \frac{-\pi}{4}$$

For $k = 2$

$$\begin{aligned} X(2) &= \sum_{n=0}^{7} x(n) e^{-j\pi n/2} \\ &= x(0) + x(1)e^{-j\pi/2} + x(2)e^{-j\pi} \\ &= 1 + \cos\frac{\pi}{2} - j\sin\frac{\pi}{2} + \cos\pi - j\sin\pi \\ &= 1 - j - 1 = j \end{aligned}$$

Therefore

$$|X(2)| = 1, \quad \angle X(2) = \frac{-\pi}{2}$$

For $k = 3$

$$\begin{aligned} X(3) &= \sum_{n=0}^{7} x(n) e^{-j3\pi n/4} \\ &= x(0) + x(1)e^{-j3\pi/4} + x(2)e^{-j3\pi/2} \\ &= 1 + \cos\frac{3\pi}{4} - j\sin\frac{3\pi}{4} + \cos\frac{3\pi}{2} - j\sin\frac{3\pi}{2} \\ &= 1 - 0.707 - j0.707 + j \\ &= 0.293 + j0.293 \end{aligned}$$

$Therefore, |X(3)| = 0.414, \quad \angle X(3) = \frac{\pi}{4}$

For $k = 4$

$$X(4) = \sum_{n=0}^{7} x(n)e^{-j\pi n}$$
$$= x(0) + x(1)e^{-j\pi} + x(2)e^{-j2\pi}$$
$$= 1 + \cos\pi - j\sin\pi + \cos 2\pi - j\sin 2\pi$$
$$= 1 - 1 + 1 = 1$$

$Therefore, |X(4)| = 1, \quad \angle X(4) = 0$

For $k = 5$

$$X(5) = \sum_{n=0}^{7} x(n)e^{-j5\pi n/4}$$
$$= x(0) + x(1)e^{-j5\pi/4} + x(2)e^{-j5\pi/2}$$
$$= 1 + \cos\frac{5\pi}{4} - j\sin\frac{5\pi}{4} + \cos\frac{5\pi}{2} - j\sin\frac{5\pi}{2}$$
$$= 1 - 0.707 + j0.707 - j$$
$$= 0.293 - j0.293$$

$$|X(5)| = 0.414, \angle X(5) = -\frac{\pi}{4}$$

For $k = 6$

$$X(6) = \sum_{n=0}^{7} x(n)e^{-j3\pi n/2}$$
$$= x(0) + x(1)e^{-j3\pi/2} + x(2)e^{-j3\pi}$$
$$= 1 + j - 1 = j$$

$$|X(6)| = 1, \quad \angle X(6) = -\frac{\pi}{2}$$

For $k = 7$

$$X(7) = \sum_{n=0}^{7} x(n)e^{-j7\pi n/4}$$
$$= 1 + e^{-j7\pi/4} + e^{-j7\pi/2}$$
$$= 1 + \cos\frac{7\pi}{4} - j\sin\frac{7\pi}{4} + \cos\frac{7\pi}{2} - j\sin\frac{7\pi}{2}$$
$$= 1 + 0.707 + j0.707 + j$$
$$= 1.707 + j1.707$$

$$|X(7)| = 2.414, \quad \angle X(7) = \frac{\pi}{4}$$

$$|X(k)| = \{3,\ 2.414,\ 1,\ 0,\ 414,\ 1,\ 0.414,\ 1,\ 0.414,\ 1,\ 2.414\}$$

$$\angle X(k) = \left\{0,\ -\frac{\pi}{4},\ -\frac{\pi}{2},\ \frac{\pi}{4},\ 0,\ \frac{-\pi}{4},\ \frac{\pi}{2},\ \frac{\pi}{4}\right\}$$

For $k = 7$

$$X(7) = \sum_{n=0}^{7} x(n) e^{-j7\pi n/4}$$

$$= 1 + e^{-j7\pi/4} + e^{-j7\pi/2}$$

$$= 1 + \cos\frac{7\pi}{4} - j\sin\frac{7\pi}{4} + \cos\frac{7\pi}{2} - j\sin\frac{7\pi}{2}$$

$$= 1 + 0.707 + j0.707 + j$$

$$= 1.707 + j1.707$$

$$|X(7)| = 2.414, \quad \angle X(7) = \frac{\pi}{4}$$

$$|X(k)| = \{3,\ 2.414,\ 1,\ 0,\ 414,\ 1,\ 0.414,\ 1,\ 0.414,\ 1,\ 2.414\}$$

$$\angle X(k) = \left\{0,\ -\frac{\pi}{4},\ -\frac{\pi}{2},\ \frac{\pi}{4},\ 0,\ \frac{-\pi}{4},\ \frac{\pi}{2},\ \frac{\pi}{4}\right\}$$

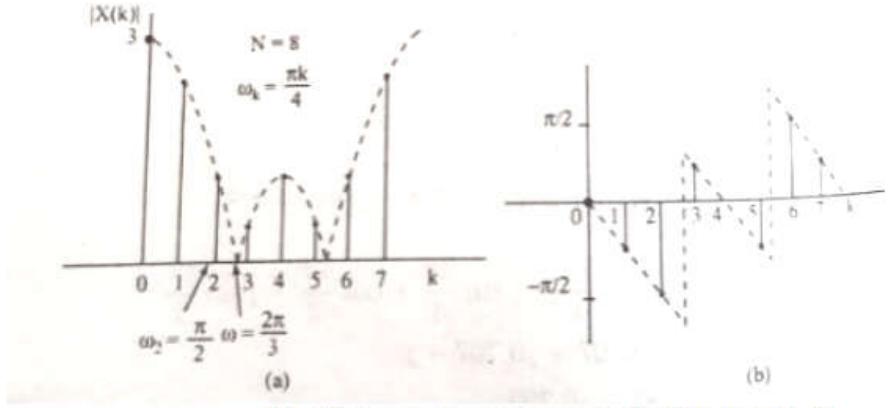


Fig: frequency response of x(n) for N=8

2.4 SUMMARY

- In this chapter we have seen a progression from sampling to the Fourier transform, and then to filtering in the frequency domain.
- Introduction The sampling theorem explained in the context of the frequency domain.
- The same is true of effects such as aliasing.
- The material starts with basic principles, so that any reader with a modest mathematical background would be in a position not only to absorb the material, but also to apply it.
- Summary of DFT definitions and corresponding expressions.

	Name	Expression(s)
1)	Discrete Fourier transform (DFT) of $f(x,y)$	$F(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(ux/M+vy/N)}$
2)	Inverse discrete Fourier transform (IDFT) of $F(u,v)$	$f(x,y) = \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)e^{j2\pi(ux/M+vy/N)}$
3)	Spectrum	$\lvert F(u,v)\rvert = \left[R^2(u,v) + I^2(u,v)\right]^{1/2}$ $R = \text{Real}(F)$; $I = \text{Imag}(F)$
4)	Phase angle	$\phi(u,v) = \tan^{-1}\left[\frac{I(u,v)}{R(u,v)}\right]$
5)	Polar representation	$F(u,v) = \lvert F(u,v)\rvert e^{j\phi(u,v)}$
6)	Power spectrum	$P(u,v) = \lvert F(u,v)\rvert^2$
7)	Average value	$\bar{f} = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y) = \frac{1}{MN}F(0,0)$
8)	Periodicity (k_1 and k_2 are integers)	$F(u,v) = F(u+k_1M,v) = F(u,v+k_2N)$ $= F(u+k_1,v+k_2N)$ $f(x,y) = f(x+k_1M,y) = f(x,y+k_2N)$ $= f(x+k_1M,y+k_2N)$
9)	Convolution	$(f \star h)(x,y) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f(m,n)h(x-m,y-n)$
10)	Correlation	$(f \star h)(x,y) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f^*(m,n)h(x+m,y+n)$
11)	Separability	The 2-D DFT can be computed by computing 1-D DFT transforms along the rows (columns) of the image, followed by 1-D transforms along the columns (rows) of the result. See Section 4.11.
12)	Obtaining the IDFT using a DFT algorithm	$MNf^*(x,y) = \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F^*(u,v)e^{-j2\pi(ux/M+vy/N)}$ This equation indicates that inputting $F^*(u,v)$ into an algorithm that computes the forward transform (right side of above equation) yields $MNf^*(x,y)$. Taking the complex conjugate and dividing by MN gives the desired inverse. See

2.5 EXERCISE

Answer the following:

1. Explain the process of obtaining the Discrete Fourier transform from the continuous transform of a sampled function.
2. Explain the properties of the 2D Discrete Fourier transform.
3. Explain the following with relevant equations
   a. The 2D discrete Fourier transform and its inverse.
   b. The 2D continuous Fourier transform pair

4. Explain Image smoothing and Image sharpening in frequency domain.
5. Explain the steps for filtering in frequency domain in detail.
6. Write a short note on Sampling and the Fourier Transform of Sampled Functions.
7. Explain Sharpening in the Frequency Domain Filters using highpass filter.
8. Explain convolution.
9. Explain smoothing lowpass for
    a. IDEAL,
    b. BUTTERWORTH, and
    c. GAUSSIAN.
10. Write a short note on FFT.

2.6 REFERENCES

Digital Image Processing by Rafael Gonzalez & Richard Woods, Pearson; 4th edition.pdf

https://sbme-tutorials.github.io/2018/cv/notes/3_week3.html

https://www.cis.rit.edu/class/simg782/lectures/lecture_14/lec782_05_14.pdf

https://universe.bits-pilani.ac.in/uploads/JNKDUBAI/ImageProcessing7-FrequencyFiltering.pdf

https://www.tutorialspoint.com/digital_signal_processing/dsp_discrete_fourier_transform_solved_examples.htm

https://faculty.nps.edu/rcristi/EC3400online/homework/solutions/Solutions_Chapter3.pdf

https://www.sathyabama.ac.in/sites/default/files/course-material/2020-10/UNIT3_1.pdf

❖❖❖❖

3

IMAGE PROCESSING FUNDAMENTALS AND PIXEL TRANSFORMATION-I

Unit Structure:

3.0 OBJECTIVES

This chapter provides an overview of image processing fundamental, components of an image processing system, fundamental steps in digital image processing, tools and libraries available for image processing, image types and files formats, various application domains where image processing can be highly useful, basic intensity transformation techniques in spatial domain which includes image negative, log transformation and power law transformation and contrast stretching technique to increase the range of intensity levels in low contrast images.

3.1 DEFINITION

What is Image Processing?

Image processing refers to the manipulation and analysis of digital images using various algorithms and techniques to extract useful information or enhance certain aspects of the image. It involves acquiring, processing, analyzing, and interpreting images to improve their quality, extract relevant features, or perform specific tasks such as object detection, image restoration, image segmentation and compression.

Image processing can be categorized into two main types:

1. **Analog Image Processing:** This involves processing images that are represented in analog form, such as photographs or printed images. Analog image processing techniques include filtering, sharpening, and noise reduction using traditional methods like optical filters or chemical processes.

2. **Digital Image Processing:** This deals with processing digital images that are stored in a computer or a digital device. Digital image processing techniques use algorithms and mathematical operations to manipulate images, such as image enhancement, image restoration, image compression, object detection, and pattern recognition.

Why do we need to process an image?

It is motivated by three major applications-

1. Improvement of pictorial information for human perception -We want to enhance the quality of image so that the image will have a better look.

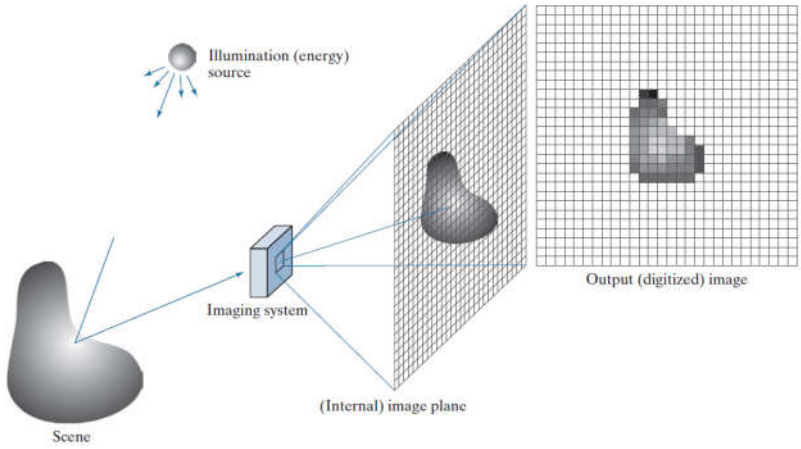
2. Image processing for automated machine applications - Quality control in assembly automation.

3. Efficient Storage and transmission - Disk space for storing the image can be reduced. Process the image or video so that it can be transmitted over a low bandwidth communication channel.

Digital Image Representation as a Matrix-

A digital image is a representation of a two-dimensional image as a finite set of digital values, called picture **elements** or **pixels.**

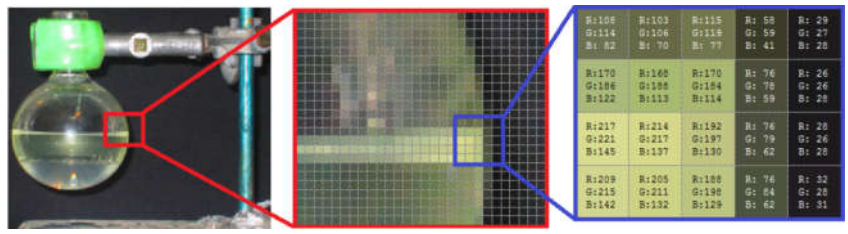
An image is defined as a two-dimensional function, **f(x,y)**, where **x** and **y** are spatial coordinates, and the amplitude of **f**at any pair of coordinates (x,y) is called the **intensity** of that image at that point. It can be considered as a matrix whose row and column indices specify a point in the imageand the element value identifies gray level value at that point.



There are infinite no of points in both the directions and intensity value is **continuousbetween 0 and 1**at every point. It is not possible to store these continuous values in digital form. Instead of storing all the intensity values at all possible points in the image, we try to take **sample** of the image.

Each of these sample values are quantized and the quantization is done using 8-bit (0 to 255) for **gray** scale images and 24-bit for **colored** images (8-bit for each channel RGB).

8-bit Grey scale image. Intensity values ranges from 0-255



24-bit RGB Image

3.2 OVERLAPPING FIELDS WITH IMAGE PROCESSING

The continuum from image processing to computer vision can be broken up into low-level, mid-level and high-level processes.

1. **Low level Processes:**

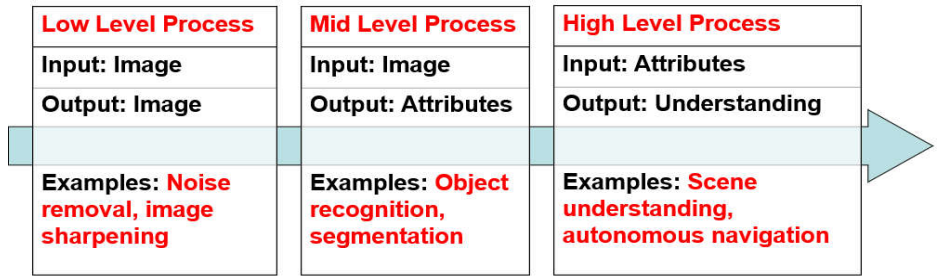
- Input and output are images
- Tasks: Primitive operations, such as, image processing to reduce noise, contrast enhancement and image sharpening

2. Mid-level Processes:

- Inputs are images. Outputs are attributes extracted from those images (edges, contours, identity of individual objects)
- Tasks: Segmentation (partitioning an image into regions or objects), description of those segmented objects to reduce them to a form suitable for computer processing, classifications of objects.

3. High-Level Processes:

- Image analysis and computer vision



3.3 COMPONENTS OF AN IMAGE PROCESSING SYSTEM

Image Sensors:

With reference to sensing, two elements are required to acquire digital images.

- The first is the physical device that is sensitive to the energy radiated by the object we wish to image (Sensor).
- The second, called a digitizer, is a device for converting the output of the physical sensing device into digital form.

Specialized image processing hardware:

It usually consists of the digitizer plus hardware that performs other primitive operations such as arithmetic and logical operations (ALU). Eg. Noise reduction. This type of hardware sometimes is called a front-end subsystem.

Computer:

The computer in an image processing system is a general-purpose computer and can range from a PC to a supercomputer. In dedicated applications, sometimes specially designed computers are used to achieve a required level of performance.

Image Processing Software:

Software for image processing consists of specialized modules that perform specific tasks. A well-designed package also includes the capability for the user to write code that, as a minimum, utilizes the specialized modules

Mass Storage Capability:

Mass storage capability is a must in a image processing applications. And image of sized 1024 * 1024 pixels requires one megabyte of storage space if the image is not compressed. Digital storage for image processing applications falls into three principal categories:

1. Short-term storage for use during processing.
2. On line storage for relatively fast recall.
3. Archival storage, characterized by infrequent access
4. Image Displays –

Image Display: The displays in use today are mainly color (preferably flat screen) TV monitors. Monitors are driven by the outputs of the image and graphics display cards that are an integral part of a computer system.

Hardcopy Devices: Devices used for recording images include laser printers, film cameras, heat-sensitive devices, inkjet units and digital units, such as optical and CD-Rom disks.

Networking: Networking is almost a default function in any computer system, in use today. Because of the large amount of data inherent in image processing applications the key consideration in image transmission is bandwidth.

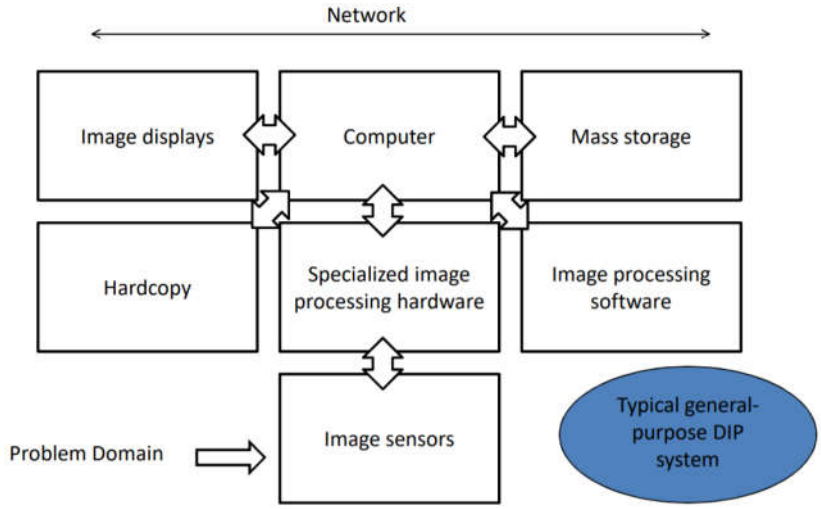


Fig: Components of a general- purpose image processing system

3.4 FUNDAMENTAL STEPS IN DIGITAL IMAGE PROCESSING

1. Image Acquisition:

Image acquisition could be as simple as being given an image that is already in digital form. Generally, the image acquisition stage involves pre-processing, such as scaling etc.

2. Image Enhancement:
Basically, the idea behind enhancement techniques is to bring out detail that is obscured, or simply to highlight certain features of interest in an image. Such as, changing brightness & contrast etc.

3. Image Restoration:
Unlike enhancement, which is subjective, image restoration is objective. Restoration techniques tend to be based on mathematical or probabilistic models of image degradation.

4. Color Image Processing:
It deals with pseudo color and full color image processing color models are applicable to digital image processing.

5. Wavelets and Multi-Resolution Processing:
It is foundation of representing images in various degrees of resolution. It is used for image data compression.

6. Compression:
Compression deals with techniques for reducing the storage required to save an image or the bandwidth to transmit it.

7. Morphological Processing:
Morphological processing deals with tools for extracting image components that are useful in the representation and description of shape.

8. Segmentation:
Segmentation procedures partition an image into its constituent parts or objects.

9. Representation and Description:
Representation and description almost always follow the output of a segmentation stage, which usually is raw pixel data, constituting either the boundary of a region or all the points in the region itself.

10. Object Detection and Recognition:
It is a process that assigns a label to an object based on its descriptor.

11. Knowledge Base:
Knowledge may be as simple as detailing regions of an image where the information of interest is known to be located, thus limiting the search that has to be conducted in seeking that information.

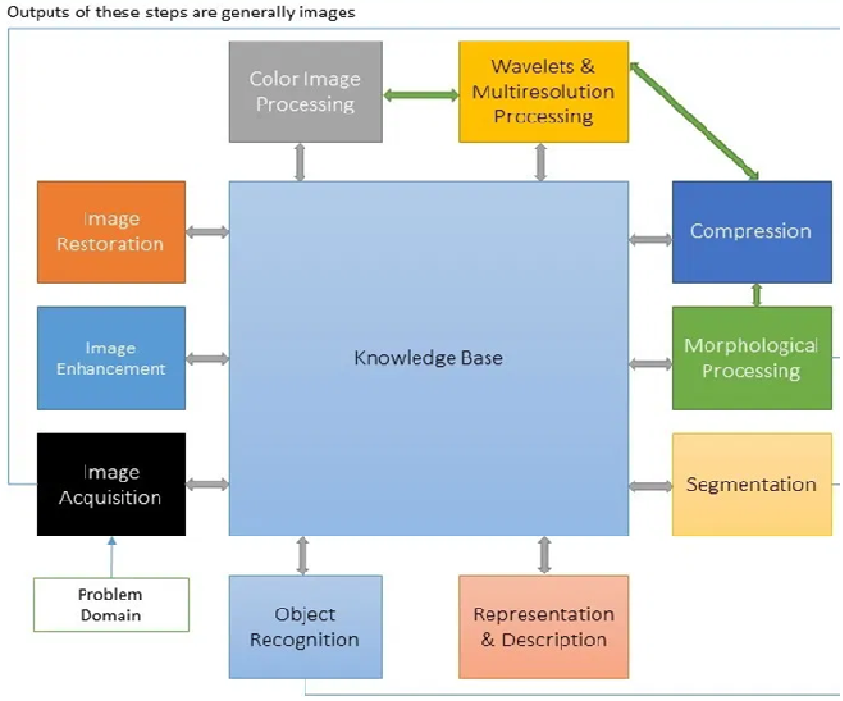


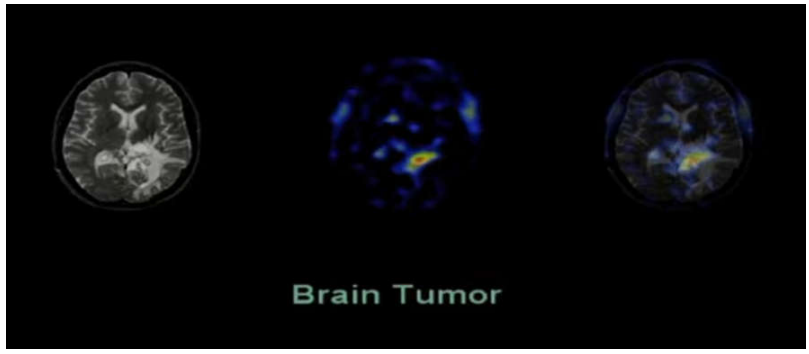
Figure: Steps in Digital Image Processing

3.5 APPLICATION OF IMAGE PROCESSING

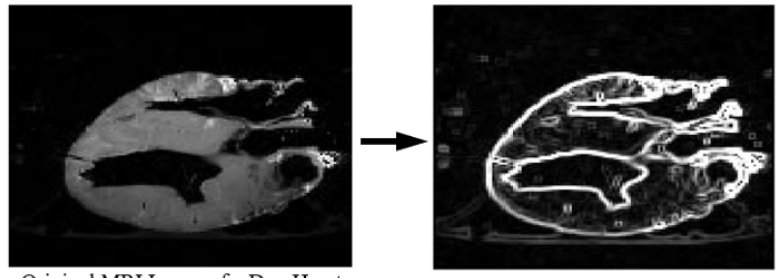
Medical Technology:

Image processing has been extensively used in medical research and has enabled more efficient and accurate treatment plans. Since medical usage calls for highly trained image processors, these applications require significant implementation and evaluation before they can be accepted for use.

For example, it can be used for the early detection of tumor /breast cancer. It can detect the exact location and size of the tumour- help the doctor to plan the operations



Left hand image is Original CT Scan Image of human brain. Right hand images are the processed images. Region of yellow and red tells the **presence of tumour** in the brain.



Original MRI Image of a Dog Heart

Edge Detection Image

A slice from MRI scan of canine heart and find boundaries between types of tissue. A suitable filter is used to highlight edges.

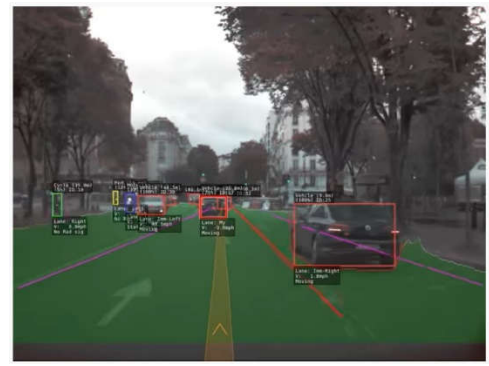
Image Reconstruction

Image processing can be used to recover and fill in the missing or corrupt parts of an image. This involves using image processing systems that have been trained extensively with existing photo datasets to create newer versions of old and damaged photos.

Machine Vision

One of the most interesting and useful applications of Image Processing is in Computer Vision. Computer Vision is used to make the computer see, identify things, and process the whole environment as a whole.

An important use of computer vision is self-driving cars, drones etc. Computer Vision helps in obstacle detection, path recognition, and understanding the environment.



Traffic Sensing Technologies:

In the case of traffic sensors, we use a video image processing system or VIPS. This consists of a) an image capturing system b) a telecommunication system and c) an image processing system. When capturing video, a VIPS has several detection zones which output an “on” signal whenever a vehicle enters the zone, and then output an “off” signal whenever the vehicle exits the detection zone. These detection zones can be set up for multiple lanes and can be used to sense the traffic in a particular station.

Remote Sensing and Satellite Imaging:

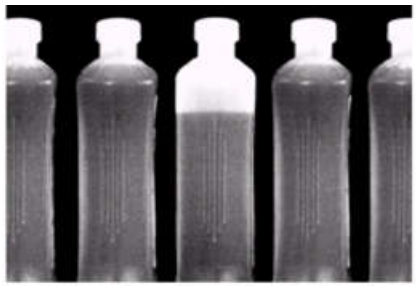
Digital image processing techniques are used extensively to manipulate satellite images for terrain classification, any change detection and analysis of geological features, meteorology, urban planningand disaster monitoring etc.

Image processing techniques are used to enhance, analyze, and interpret remote sensing data to derive insights about the Earth's surface, its features, and changes over time.

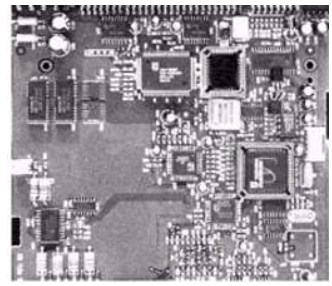
Industrial Inspection:

Industrial Vision systems are used in all kinds of industries for quality check and inspection.

For eg, whether a bottle is filled up to the specified level or not. Machine inspection is used to determine that all components are present and that all solder joints are acceptable.



Bottle level check



PCB Check

Biometrics:

Image processing plays a crucial role in biometric systems for tasks such as face recognition, fingerprint recognition, iris recognition, and hand geometry analysis.

Security and Surveillance:

Image processing techniques are employed in security and surveillance systems for tasks such as video analytics, motion detection, object recognition, and tracking of suspicious activities.

3.6 IMAGE PROCESSING PIPELINE

The following steps describe the basic steps in the image processing pipeline.

Acquisition and storage: The image needs to be captured (using a camera, for example) and stored on some device (such as a hard disk) as a file (for example, a JPEG file).

Load into memory and save to disk: The image needs to be read from the diskinto memory and stored using some data structure (for example, numpy, ndarray).

Manipulation, enhancement, and restoration: We need to run some pre-processingalgorithmsto do the following:

- Run a few transformations on the image; for example, grayscale conversion
- Enhance the quality of the image (filtering; for example, deblurring)
- Restore the image from noise degradation

Segmentation: The image needs to be segmented in order to extract the objects of interest.

Information extraction/representation: The image needs to be represented in some alternative form; for example, one of the following:

- Some hand-crafted feature-descriptor can be computed
- Some features can be automatically learned from the image (for example, the weights and bias values learned in the hidden layers of a neural net with deep learning)
- The image is going to be represented using that alternative representation

Image understanding/interpretation: This representation will be used to understand the image better with the following:

- Image classification (for example, whether an image contains a human object or not)
- Object recognition (for example, finding the location of the car objects in an image with a bounding box)

3.7 TOOLS AND LIBRARIES FOR IMAGE PROCESSING

Installing some image processing libraries in Python

In Python, there are many libraries that we can use for image processing. The ones we are going to use are: NumPy, SciPy, scikit-image, PIL (Pillow), OpenCV, scikit-learn, SimpleITK, and Matplotlib.

The matplotliblibrary will primarily be used for display purposes, whereas numpy will be used for storing an image.

The scikit-learn library will be used for building machine-learning models for image processing, and scipy will be used mainly for image enhancements.

The scikit-image, mahotas, and opencv libraries will be used for different image processing algorithms.

The following code block shows how the libraries that we are going to use can be downloaded and installed with pip from a Python prompt (interactive mode):

```
>>> pip install numpy

>>> pip install opencv-python

>>> pip install scipy

>>> pip install scikit-image

>>> pip install scikit-learn
```

>>> pip install pillow

>>> pip install SimpleITK

>>> pip install matplotlib

3.8 IMAGE TYPES

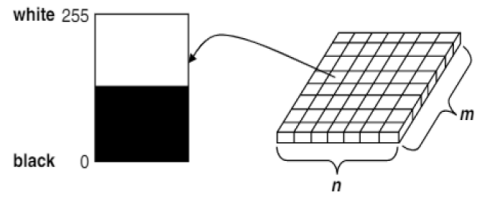
Images can be broadly classified under four categories: (i) Black and qhite or binary images, (ii) grayscale images, (iii) colour images, (iv) multispectral images.

(i) Binary Images:

Binary images take only two values, either '0' or '1'. The brightness graduation can not be differentiated in binary images.

A binary image is referred as a 1-bit image because it takes only 1 binary digit to represent each pixel.

A gray-scale image can be converted to a black-and-white or binary image by the thresholding operation.

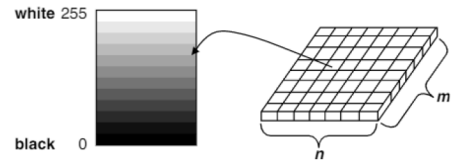


Binary image representation

(ii) Gray-scale Images:

Each pixel value in a gray-scale image corresponds to the amount or quantity of light. The brightness graduation can be differentiated in a gray-scale image.

An 8-bit image will have a brightness variation from 0 to 255 where '0' represents black and '255' represents white



(iii) Color Images

Color images has three values per pixel and they measure the intensity and chrominance of light. Each pixel is a vector of color components.

Color images can be modelled as three-band monochrome image data, where each band of data corresponds to a different color. The actual information stored in the digital image data is the brightness information in each spectral band.

Common color spaces are RGB (Red, Green, Blue), HSI (Hue Saturation and Intensity) CMYK (Cyan, Magenta, Yellow, Black).

R=255, G=0, B=0

R=0, G=255, B=0

R=0, G=0, B=255

(iv) Multispectral Images:

Multispectral images are images of the same object taken in different bands of visible or infrared regions of the electromagnetic spectrum. This includes infrared, ultraviolet and other bands in electromagnetic spectrum.

Multi spectral images typically contain information outside the normal human perceptual range. The information available in multispectral image is not visible to a human observer. However, the information is often represented in visual form by mapping the different spectral bands to RGB components.

Images acquired for remote-sensing applications are generally multi-spectral in nature. They are used by scientists on the earth to study dynamics and processes occurring in the earth surface.

3.9 IMAGE FILE FORMATS

A digital image is often encoded in the form of binary files for the purpose of storage and transmission. Different file formats compress the image data by different amounts.

Common image file formats are GIF (Graphical Interchange Formats), JPEG (Joint Photographic Expert Group), PNG (Portable Network Graphics), TIFF (Tagged Image File Format), PSD file format and EPS.

GIF File Format-

- The GIF file format uses lossless compression scheme. As a result, the quality of the image is pre- served.
- GIF interlaced images can be displayed as low-resolution images initially and then develop clarity and detail gradually.
- GIF images can be used to create simple animations.
- GIF 89a images allow for one transparent colour.

Advantages of GIF File Format - GIF uses lossless compression algorithm that provides up to 4:1 compression of images which is the most widely supported graphics format on the Web.GIF supports transparency and interlacing

Limitations of GIF File Format - GIF file format supports a maximum of 256 colours. Due to this particular limitation, complex images may lose some detail when translated into GIF

JPEG Files -

- JPEG is not actually a file type. JPEG is the most important current standard for image compression. JPEG standard was created by a working group of the International Organisation for Standardisation (ISO). This format provides the most dramatic compression option for photographic images. JPEG compression is used within the JFIF file format that uses the file extension (jpg).
- This format is useful when the storage space is at a premium. JPEG pictures store a single raster image in 24-bit colour.

JPEG is a platform-independent format that supports the highest levels of compression; however, this compression is lossy. JPEG images are not interlaced; however, progressive JPEG images support interlacing.

Advantage of JPEG File Format- The strength of JPEG file format is its ability to compress larger image files. Due to this compression, the image data can be stored effectively and transmitted efficiently from one place to another

Limitations of JPEG File Format- JPEG in its base version does not support multiple layers, high dynamic range. Hence JPEG will not be a wise choice if one is interested in maintaining high quality pictures.

PNG Files -

- PNG stands for Portable Network Graphics. PNG is a bitmapped image format that employs **lossless data compression**. PNG was created to improve and replace the GIF format.
- The PNG file format is regarded and was made as a free and open-source successor to the GIF file format. The PNG file format supports true colour (16 million colours), whereas the GIF file format only allows 256 colours.
- The **lossless** PNG format is best suited for editing pictures, whereas the lossy formats like JPG are best for the final distribution of photographic-type images due to smaller file size.
- Yet many earlier browsers do not support the PNG file format, however with the release of Internet Explorer 7 all popular modern browsers fully support PNG. Special features of PNG files include support for up to **48 bits of colour information**.

TIFF Files -

- TIFF stands for Tagged Image File Format and was developed by the Aldus Corporation in the 1980s. It was later supported by Microsoft.
- TIFF files are often used with scanned images. Since a TIFF file does not compress an image file, hence images are often large but the quality is preserved.
- It uses a filename extension of TIFF of TIF. The TIFF format is often used to exchange files between applications and computer platforms.
- Within TIFF, a lossless compression routine known as LZW is available. This reduces the size of the stored file without perceptible loss in quality.
- The goals of the TIFF specification include extensibility, portability, and revisability.

Advantages of TIFF File Format is that it can support any range of image resolution, size, and colour depth and different compression techniques.
Disadvantage of TIFF file format is its large file size that limits its usage in web applications.

3.10 INTENSITY TRANSFORMATIONS

3.10.1Introduction –

All the image processing techniques discussed in this section are implemented in the spatial domain (plane containing the pixels of an image). Spatial domain techniques operate directly on the pixels of an image. In frequency domain, operations are performed on the Fourier transform of an image, rather than on the image itself.

Generally, spatial domain techniques are more efficient computationally and require less processing resources to implement.

3.10.2 Image Transformations Basics –

The spatial domain processes can be denoted by the expression-

$g(x, y) = T[f(x, y)]$

Where f(x,y) is the input image, g(x,y) is the output image, and T is an operator on f defined over a neighborhood of point (x, y). The point (x, y) shown is an arbitrary location in the image, and the small region shown containing the point is a neighborhood of (x, y),

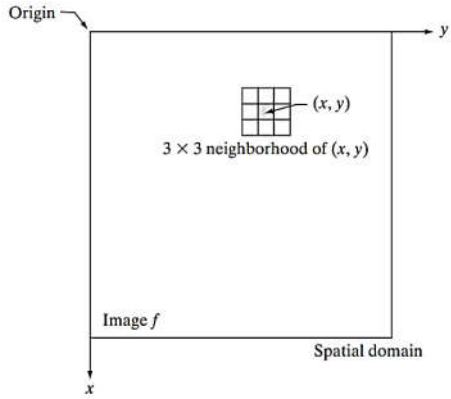


Figure: A 3X3 neighborhood about a Point (x, y) in an image in the spatial domain. The neighborhood is moved from pixel to pixel in the image to generate an output image.

3.10.3 Spatial filtering-

The process consists of moving the origin of the neighborhood from pixel to pixel and applying the operator T to the pixels in the neighborhood to yield the output at that location.Typically, the process starts at the top left of the input image and proceeds pixel by pixel in a horizontal scan, one row at a time.

When the origin of the neighborhood is at the border of the image, part of the neighborhood will reside outside the image. The procedure is either to ignore the outside neighbors in the computations specified by T, or to pad the image with a border of 0's or some other specified intensity values.

The procedure just described is called spatial filtering, in which the neighborhood, along with a predefined operation, is called a spatial filter (also referred to as a spatial mask, kernel, template, or window).

The smallest possible neighborhood is of size 1X1. In this case, **g** depends only on the value of **f** at a single point (x, y) and T in equation becomes an intensitytransformation function of the form – s = T(r)

where, '**s**' and '**r**'are variables denoting the intensity of g and f at any point (x, y) respectively.

3.11 SOME BASIC INTENSITY TRANSFORMATION FUNCTIONS

3.11.1 Negative Images

The negative of an image with intensity levels in the range [0, L-1] is obtained by using the negative transformation which is given by the expression-

s = L - 1 – r

Reversing the intensity levels of an image in this manner produces the equivalent of a photographic negative.

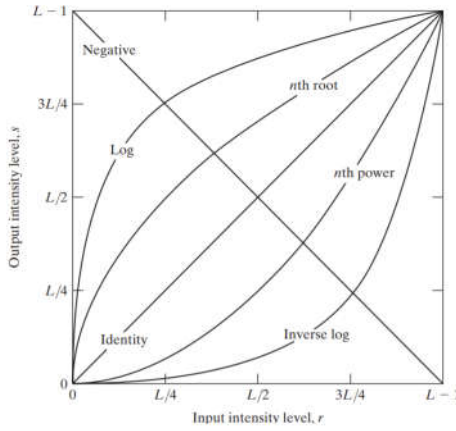
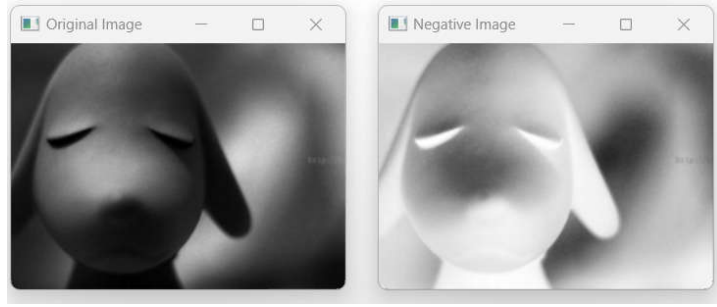


Figure: Shows Some basic intensity transformation functions. All curves were scaled to fit in the range shown.

This type of processing is particularly suited for enhancing white or gray detail embedded in dark regions of an image, especially when the black areas are dominant in size.



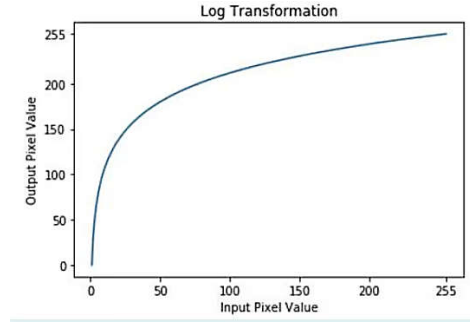
Original Image and its Negative image

3.11.2 Log Transform -

The general form of the log transformation is-

s = c log(1 + r)

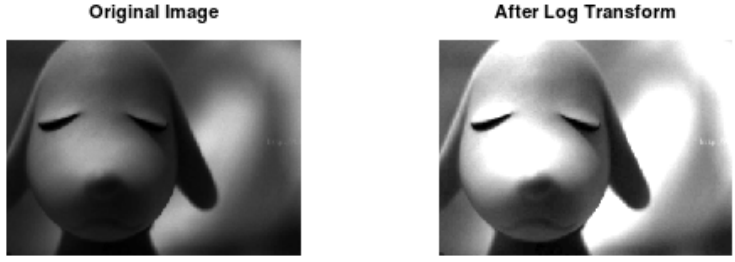
where '**s**' and '**r**' are the pixel values of the output and input image and **c** is a constant, and it is assumed that r ≥ 0. The value 1 is added to each of the pixel value of the input image because if there is a pixel intensity of 0 in the image, then log(0) is equal to infinity.



The shape of the log curve shows that this transformation maps a narrow range of low intensity values in the input into a wider range of output levels. The opposite is true of higher values of input levels.

We use a transformation of this type to expand the values of dark pixels in an image while compressing the higher-level values**.** The opposite is true of the inverse log transformation.

The log function has the important characteristic that it compresses the dynamic range of images with large variations in pixel values.



3.11.3 Power-law Transform-

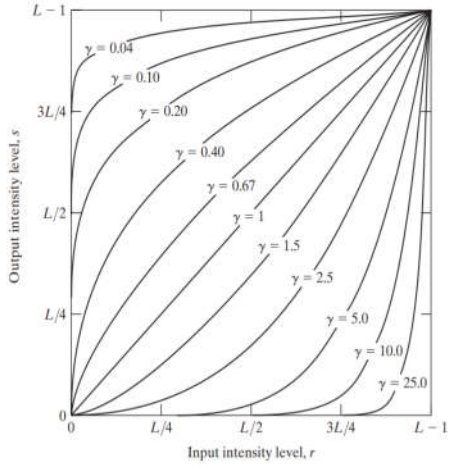
Power-law transformations have the basic form-

$s = cr^{\gamma}$

where c and γ are positive constants.

Power-law curves with fractional values of γ map a narrow range of dark input values into a wider range of output values, with the opposite being true for higher values of input levels.

Curves generated with values of γ >1 have exactly the opposite effect as those generated with values of γ <1. It reduces to the identity transformation when c= γ=1

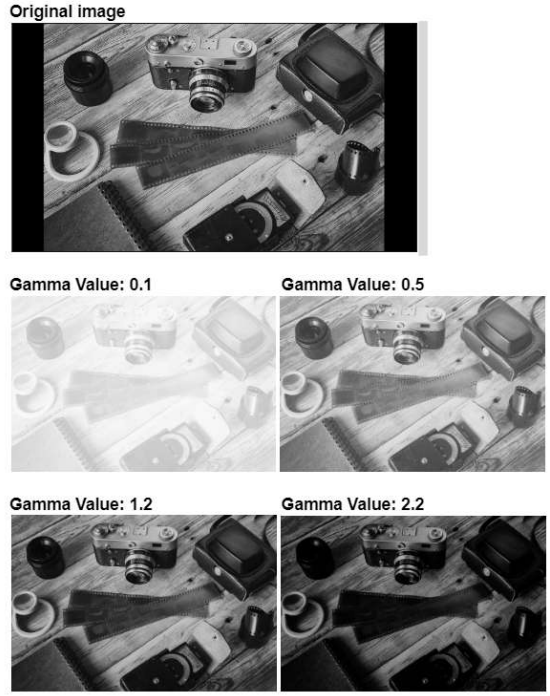


Plots of the equation $s = cr^{\gamma}$ for various values of γ (c=1 in all cases)

A variety of devices used for image capture, printing, and display respond according to a power law. By convention, the exponent in the power-law equation is referred to as gamma. The process used to correct these power-law response **phenomena is called gamma correction**.

For example, cathode ray tube (CRT) devices have an intensity-to-voltage response that is a power function, with exponents varying from approximately 1.8 to 2.5

With reference to the curve for γ=2.2, we see that such display systems would tend to produce images that are darker than intended.



3.12 PIECEWISE-LINEAR TRANSFORMATION FUNCTIONS

3.13.1Contrast Stretching-

One of the simplest piecewise linear functions is a **contrast-stretching** transformation. Low-contrast images can result from poor illumination, lack of dynamic range in the imaging sensor, or even the wrong setting of a lens aperture during image acquisition.

Contraststretchingis a process that expands the range of intensity levels in an image so that it spans the full intensity range of the recording medium or display device.

- The locations of points (r_1, s_1) and (r_2, s_2) control the shape of the transformation function.
- If $r_1 = s_1$ and $r_2 = s_2$ the transformation is a linear function that produces no changes in intensity levels.
- If $r_1 = r_2$, $s_1 = 0$ and $s_2 = L-1$ the transformation becomes a **thresholding function**that creates a binary image.
- Intermediate values of (r_1, s_1) and (r_2, s_2) produce various degrees of spread in the intensity levels of the output image, thus affecting its contrast.

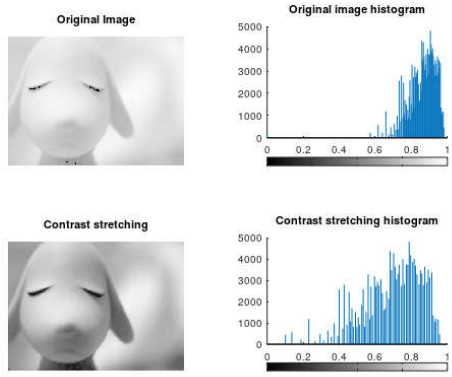
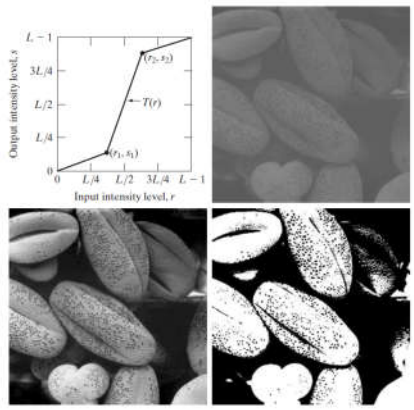


Figure shows the Contrast stretching Transformation of High Contrast Image along with the image histogram.



a b
c d

Contrast stretching.

(a) Form of transformation function. (b) A low-contrast image.

(c) Result of contrast stretching.

(d) Result of thresholding.

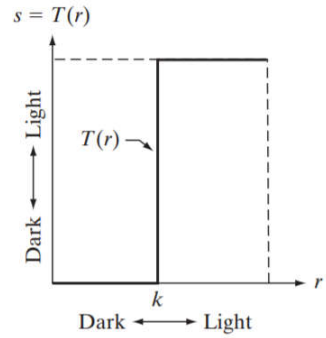
(Original image courtesy of Dr.

Roger Heady, Research School ofBiological Sciences, Australian National University, Canberra,

Australia.)

Thresholding function –

- In the limiting case, T(r) produces a two-level (binary) image.
- A mapping of this form is called a thresholding function.



3.13 SUMMARY

In this chapter, we learnt about the fundamentals about Image processing. Before understanding image processing, we discussed about the representation of images in digital form (matrix form). Different types of images and image file formats are discussed. Intensity levels of binary images, gray-scale images and colored images are briefly explained in the

chapter. Wide area of applications of Iage processing is explained. Components and steps in digital image processing system is discussed with diagrams.

Basic intensity transformation functions like log transformation and power low transformation, contrast stretching and thresholding are demonstrated using the transformed output images and function curves.

3.14 EXERCISE QUESTIONS

1. What is Image processing? Which are the overlapping fields with Image Processing?
2. What is the motivation behind the need of image processing?
3. Discuss the applications of digital image processing in various fields.
4. Explain the components of an image processing system.
5. Explain the fundamental steps in Digital Image Processing.
6. Write a short note on Image Processing Pipeline.
7. Discuss Tools and Libraries for Image Processing.
8. Write a note on various image types.
9. Discuss different Image file format in brief.
10. Explain briefly the following basic image intensity transformations functions-

    (i) Negative Image,
    (ii) Log Transformation and
    (iii) Power Law Transformation.

3.15 REFERENCES

1. https://www.researchgate.net/publication.
2. Digital Image Processing by Rafael Gonzalez & Richard Woods, Pearson; 4th edition, pdf.
3. Digital Image Processing by S. Jayaraman, Tata McGraw Hill Publication, pdf.
4. Images used are processed using the tools- OpenCV (Python) and GNU Octave (compatible with MATLAB).

❖❖❖❖

4

IMAGE PROCESSING FUNDAMENTALS AND PIXEL TRANSFORMATION-II

Unit Structure:

4.0 OBJECTIVE

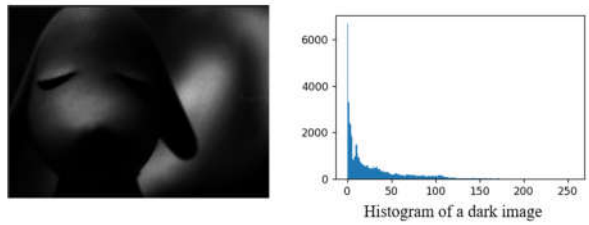
This chapter provides an overview of image histogram processing. The objective behind histogram processing is to improve the contrast of the image by stretching the image intensity range. This chapter also includes the mechanics of linear and non-linear low pass (smoothing/blur) filters. First order and second order derivatives are discussed to understand the high pass (sharpening)filters. Various smoothing and sharping filters are explained with examples.

4.1 DEFINITIONS

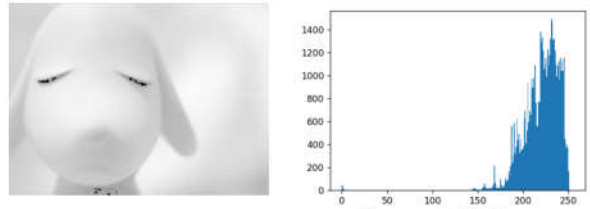
Image Histogram:

Image histogram shows the graphical representation of intensity distribution of all the pixels in an image. We can consider histogram as a graph or plot, which gives an overall idea about the intensity distribution of an image.

It is a plot with pixel values (ranging from 0 to 255 in Gray-scale images) in X-axis and corresponding number of pixels in the image on Y-axis.



Histogram of a dark image

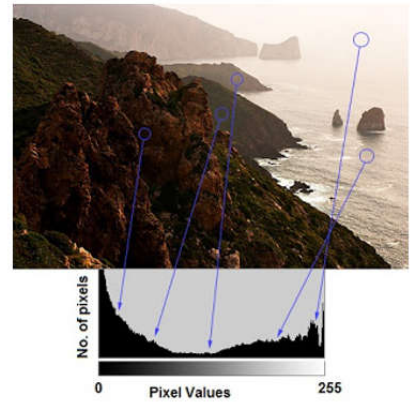


Histogram of a Light image

Above histogram is plotted using hist() function in OpenCV-python

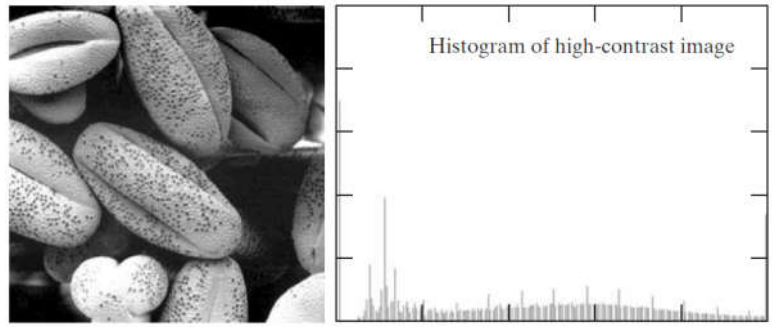
Histogram of an image gives idea about contrast, brightness, intensity distribution etc of that image.
All image processing tools provides features to display image histogram.

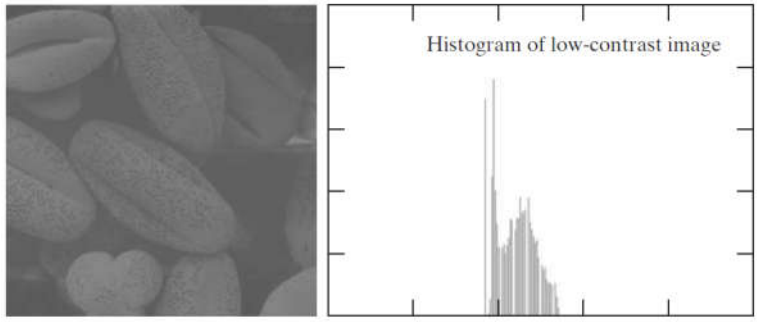
Figure shows the image and its histogram.
Image Ref- https://docs.opencv.org/



4.2 HISTOGRAM PROCESSING

The histogram of a digital image with intensity levels in the range [0, L-1] is a discrete function $\mathbf{h(r_k) = n_k}$where $\mathbf{r_k}$is the $\mathbf{k^{th}}$intensity value and $\mathbf{n_k}$is the number of pixels in the image with intensity $\mathbf{r_k}$.





The horizontal axis of the histograms are values of $r_k$and the vertical axis are values of

$h(r_k) = n_k$ or $p(r_k) = r_k$ /MN if the values are normalized, where**, M and N are the row and column** dimensions of the image.

We see that the components of the histogram in the **high-contrast** image cover a wide range of the intensity scale and, further, that the distribution of pixels is not too far from uniform.Intuitively, it is reasonable to conclude that an image whose pixels tend to occupy the entire range of possible intensity levels and, in addition, tend to be distributed uniformly, will have an appearance of high contrast and will exhibit a large variety of gray tones.

It is possible to develop a transformation function that can achieve this effect automatically, using only the histogram of an input image.

4.3 HISTOGRAM EQUALIZATION

Histogram Equalization is a computer image processing technique used to **improve contrast** in images. It accomplishes this by effectively spreading out the most frequent intensity values, i.e. stretching out the intensity range of the image.

This method usually increases the global contrast of images and allows for areas of lower local contrast to gain a higher contrast.

It is common practice to **normalize** a histogram by dividing each of its components by the total number of pixels in the image, denoted by the product MN, where**, M and N are the row and column** dimensions of the image.

Thus, a normalized histogram is given by $\mathbf{p(r_k) = r_k /MN}$, for k = 0, 1, 2, … , L - 1.

$p(r_k)$ is an estimate of the probability of occurrence of intensity level r_k in an image. The sum of all components of a normalized histogram is equal to 1.

Assuming initially continuous intensity values, let the variable r denote the intensitiesof an image to be processed. As usual, we assume that r is

in the range [0, L − 1], with r = 0 representing black and r = L − 1 representing white.

The discrete form of the Histogram Equalization transformation is-

$$s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j) \quad k = 0,1,2,\ldots,L-1$$

The transformation (mapping) $T(r_k)$ in this equation is called a histogram equalization or histogram linearization transformation

A simple illustration of the mechanics of histogram equalization:

Suppose that a 3-bit image ($L=2^3=8$) of size 64X64 pixels (MN=4096) has the intensity distribution shown in the Table where the intensity levels are integers in the range [0, L - 1] = [0, 7].

r_k	n_k	$p_r(r_k) = n_k/MN$
$r_0 = 0$	790	0.19
$r_1 = 1$	1023	0.25
$r_2 = 2$	850	0.21
$r_3 = 3$	656	0.16
$r_4 = 4$	329	0.08
$r_5 = 5$	245	0.06
$r_6 = 6$	122	0.03
$r_7 = 7$	81	0.02

Values of the histogram equalization transformation function are obtained as follows–

$$s_0 = T(r_0) = 7\sum_{j=0}^{0} p_r(r_j) = 7p_r(r_0) = 1.33$$

$$s_1 = T(r_1) = 7\sum_{j=0}^{1} p_r(r_j) = 7p_r(r_0) + 7p_r(r_1) = 3.08$$

$S_2 = T(r_2) = T(r_1) + 7*p_r(r_2) = 3.08 + 7*(0.21) = 4.55$

Similarly, $S_3 = 5.67$, $S_4 = 6.23$, $S_5 = 6.65$, $S_6 = 6.86$, and $S_7 = 7.00$

At this point, the **s**values still have fractions because they were generated by summing probability values, so we round them to the nearest integer:

$s_0 = 1.33 \rightarrow 1$ $\quad s_2 = 4.55 \rightarrow 5$ $\quad s_4 = 6.23 \rightarrow 6$ $\quad s_6 = 6.86 \rightarrow 7$

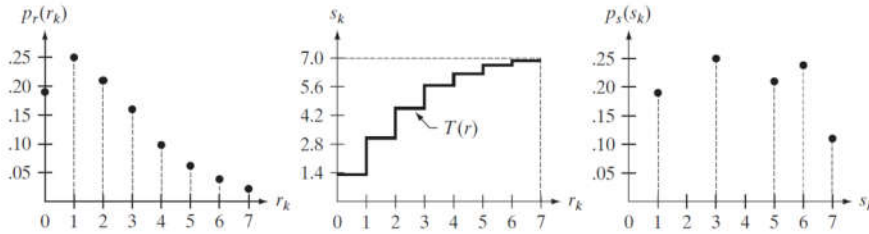
$s_1 = 3.08 \rightarrow 3$ $\quad s_3 = 5.67 \rightarrow 6$ $\quad s_5 = 6.65 \rightarrow 7$ $\quad s_7 = 7.00 \rightarrow 7$

These are the values of the equalized histogram. Observe that there are only five distinct intensity levels. Because $r_0=0$ was mapped to $s_0 = 1$. there are 790 pixels in the histogram equalized image with this value.

Also, there are 1023 pixels with a value of $s_1 = 3$ and 850 pixels with a value of $s_2 = 5$.

However, both $r_3$and r_4 were mapped to the same value, 6, so there are (656 + 329) = 985 pixels in the equalized image with this value.

Similarly, there are (245 + 122 + 81) = 448 pixels with a value of 7 in the histogram equalized image. Dividing these numbers by MN = 4096 yielded the **equalized histogram.**



a b c

Illustration of Histogram equalization of a 3-bit (8 intensity levels) image. (a) Original histogram. (b) Transformation function. (c) Equalized histogram.

Solution:

Gray level	nk	PDF	CDF	(L-1)*CDF	ROUND OFF
0	790	0.19	0.19	1.33	1
1	1023	0.25	0.44	3.08	3
2	850	0.21	0.65	4.55	5
3	656	0.16	0.81	5.67	6
4	329	0.08	0.89	6.23	6
5	245	0.06	0.95	6.65	7
6	122	0.03	0.98	6.86	7
7	81	0.0.2	1	7	7
	4096				

4.4 HISTOGRAM MATCHING

As explained in the last section, histogram equalization produces a transformation function that seeks to generate an output image with a uniform histogram. When automatic enhancement is desired, this is a good approach to consider because the results from this technique are predictable and the method is simple to implement.

However, there are applications in which histogram equalization is not suitable. In particular, it is useful sometimes to be able to specify the shape of the histogram that we wish the processed image to have. The method used to generate images that have a specified histogram is called **histogram matching or histogram specification.**

4.5 MECHANICSOF SPATIAL FILTERING

"Filtering" refers to accepting (passing) or rejecting certain frequency components.

For example, a filter that passes low frequencies is called a **lowpass**filter. The net effect produced by a lowpass filter is to blur (smooth) an image.

We can accomplish a similar smoothing directly on the image itself by using spatial filters (also called spatial masks, kernels, templates, and windows).

A spatial filter consists of two things-

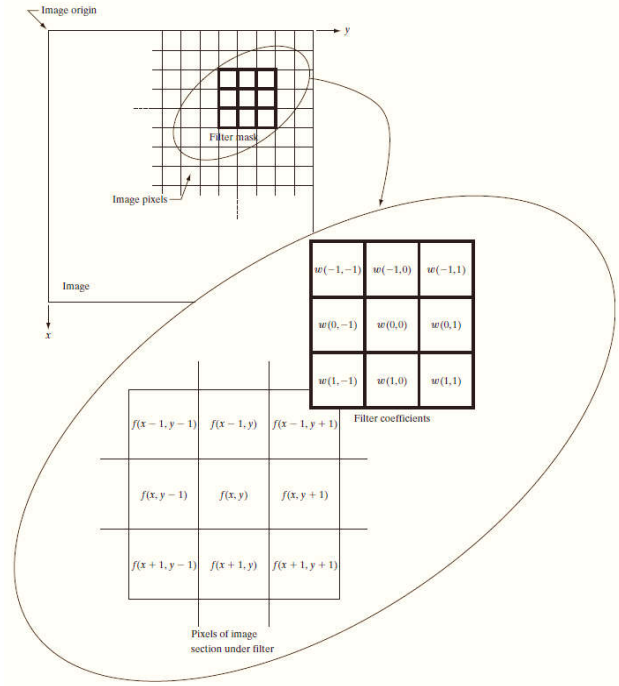
- A neighborhood, (typically a small rectangle)
- A predefined operation that is performed on the image pixels encompassed by the neighborhood.

Filtering creates a new pixel with coordinates equal to the coordinates of the center of the neighborhood, and whose value is the result of the filtering operation.

A processed or filtered image is generated as the center of the filter visits each pixel in the input image.

At any point (x, y) in the image, the response, g(x,y) , of the filter is the sum of products of the filter coefficients and the image pixels encompassedby the filter:

$$g(x, y) = w(-1, -1) * f(x - 1, y - 1) + w(-1, 0) * f(x - 1, y) + \ldots\ldots + w(0, 0) * f(x, y) + w(1, 1) * f(x + 1, y + 1)$$



Observe that the center coefficient of the filter, w(0,0), aligns with the pixel at location (x, y).

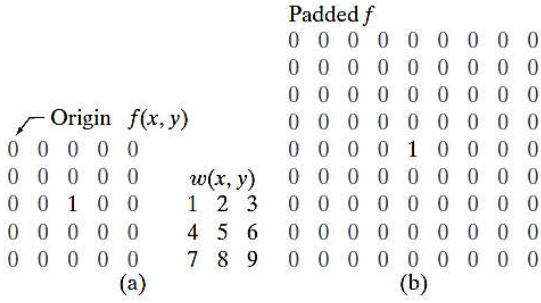
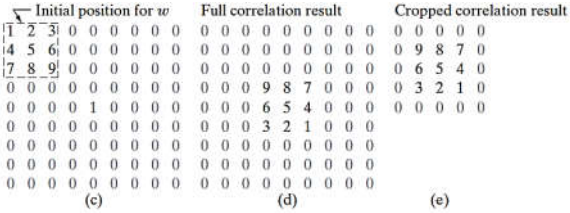
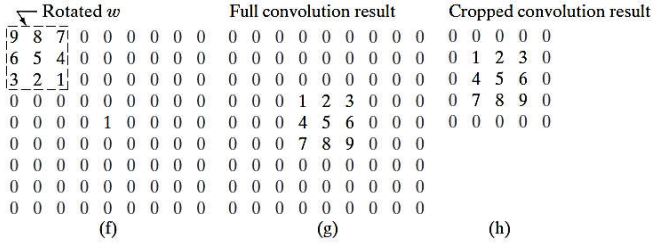
For a mask of size **m x n** we assume that m=2a+1 and n=2b+1 where a and b are positive integers.

Spatial Correlation and Convolution-

There are two closely related concepts that must be understood clearly when performing linear spatial filtering. One is correlation and the other is convolution.

Correlation is the process of moving a filter mask over the image and computing the sum of products at each location, exactly as explained in the previous section.

The mechanics of **convolution** are the same, except that the filter is first rotated by 180°. For example-

	- For a filter of size ***m x n*** we pad the image with a minimum of *m-1* rows of 0's at the top and bottom and *n-1* columns of 0's on the left and right. **m** and **n** are equal to 3, so we pad f with two rows of 0's above and below and two columns of 0's to the left and right
	Spatial Correlation
	Spatial Convolution

4.6 IMAGE SMOOTHING (LOW PASS) FILTER

Smoothing filters are used for blurring and for noise reduction. Blurring is used in preprocessing tasks, such as removal of small details from an image prior to (large) object extraction,

Linear Filers- Average filter

The output (response) of a smoothinglinear spatial filter is simply the average of the pixels contained in the neighborhood of the filter mask. These filters sometimes are called **averaging filters.** They also are referred to a **lowpass filters**.

Replace the value of every pixel in an image by the **average** of the intensity levels in the neighborhood defined by the filter mask, this process results in an image with reduced "sharp" transitions in intensities.

Because random noise typically consists of sharp transitions in intensity levels, the most obvious application of smoothing is **noise reduction**.

However, edges (which almost always are desirable features of an image) also are characterized by sharp intensity transitions, so averaging filters have the undesirable **side-effect** that they **bluredges.**

Let us first understand the filter/mask/kernel-

BOX FILTER KERNELS-

(all co-efficient are equal)

The simplest, separable lowpass filter kernel is the box kernel, whose coefficients have the same value (typically 1). The name "box kernel" comes from a constant kernel resembling a box when viewed in 3-D.

$$\frac{1}{9} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

WEIGHTED AVERAGE FILTER-

The constant multiplier in front of mask is equal to 1 divided by the sum of the values of its coefficients, as is required to compute an average.

A 3x3 smoothing (averaging) filter masks.

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

An Example of Average Filter-

2D Average filtering example using a 3 x 3 sampling window:
Keeping border values unchanged

Average = round(1+4+0+2+2+4+1+0+1)/9 = 2

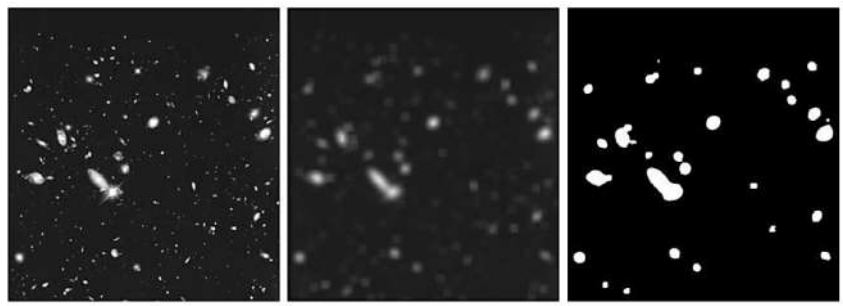
Input

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

Output

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

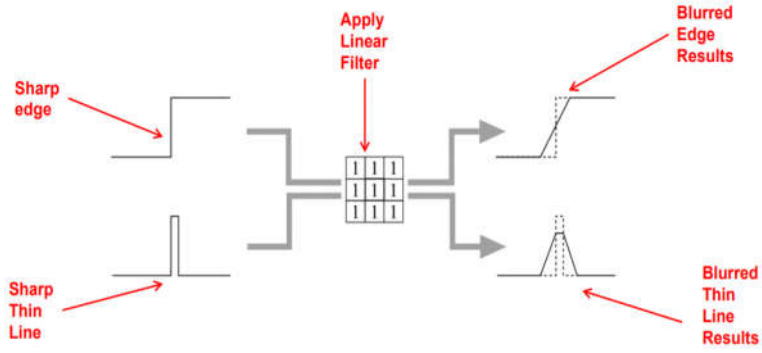
- An important application of spatial averaging is to blur an image for the purpose of getting a gross representation of objects of interest, such that the intensity of smaller objects blends with the background and larger objects become "bloblike" and easy to detect.
- The size of the mask establishes the relative size of the objects that will be blended with the background.



a b c

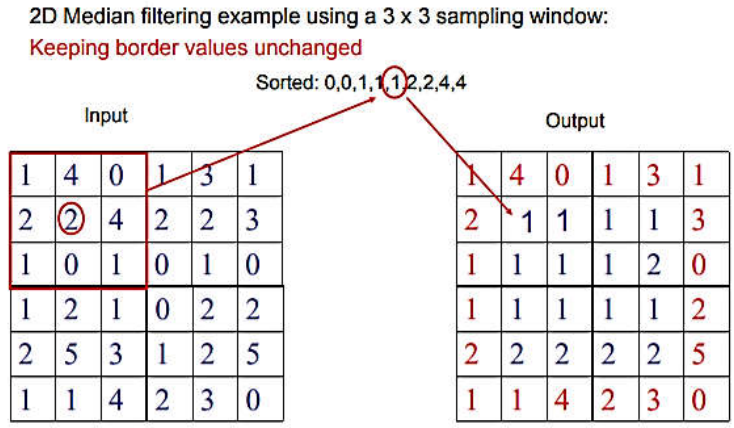
FIGURE 3.34 (a) Image of size 528 × 485 pixels from the Hubble Space Telescope. (b) Image filtered with a 15 × 15 averaging mask. (c) Result of thresholding (b). (Original image courtesy of NASA.)

- Linear filters blur all image structures points, edges and lines, reduction of image quality
- Linear filters thus not used a lot for removing noise



4.7 SMOOTHING ORDER-STATISTIC (NON-LINEAR) FILTERS

Order-statistic filters are **nonlinear** spatial filters whose response is based on ordering (ranking) the pixels contained in the image area encompassed by the filter, and then replacing the value of the center pixel with the value determined by the ranking result.



Nonlinear filters – Median Filters

- The best-known filter in this category is the median filter, which, as its name implies, replaces the value of a pixel by the median of the intensity values in the neighborhood of that pixel (the original value of the pixel is included in the computation of the median).

- Median filters are quite popular because, for certain types of random noise, they provide excellent noise-reduction capabilities, with considerably less blurring than linear smoothing filters of similar size.

- Median filters are particularly effective in the presence of impulse noise, also called salt-and-pepper noise because of its appearance as white and black dots superimposed on an image.
- The median, of a set of values is such that half the values in the set are less than or equal to the median and half are greater than or equal to the median.

- Although the median filter is by far the most useful order-statistic filter in image processing, it is by no means the only one. The median represents the 50th percentile of a ranked set of numbers,

- Using the 100^{th} percentile results in the so-called **max filter**, which is useful for finding the brightest points in an image.

- The 0^{th} percentile filter is the **min filter**, used for the opposite purpose.

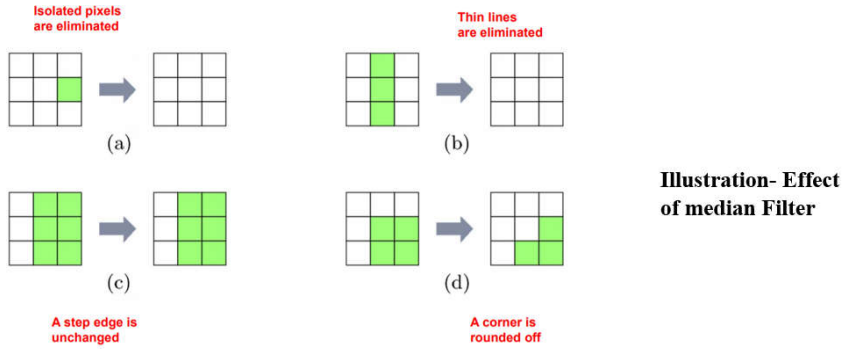


Illustration- Effect of median Filter

4.8 SHARPENING FILTERS (HIGH PASS FILTERS)

Sharpeningfilters highlight fine detail in an image or enhance detail that has been blurred. Averaging technique used for smoothingis same as integration while sharpening can be achieved by differentiation. Differentiation enhances **edges** and **discontinuities** (including noise) and deemphasizes slow varying gray-scalevalues.

**Strength of response of a derivative operator is proportional to the degree of discontinuity of image at the point where the operator is applied.

Derivatives: (1st orderand 2nd order)

Derivative of a digital function is defined in terms of differences. There are various ways to define these differences. However, we require that any definition we use for a **first derivative-**

1. must be zero in areas of constant intensity;
2. must be nonzero at the onset of an intensity step or ramp; and
3. must be nonzero along ramps.

Similarly, any definition of a second **derivative-**

1. must be zero in constant areas;
2. must be nonzero at the onset and end of an intensity step or ramp; and
3. must be zero along ramps of constant slope.

Basic definition of **first-order** derivative of a one-dimensional function given by the difference-

$$\partial f/\partial x = f(x+1) - f(x)$$

Second-order derivative is defined by the difference-

$$\partial^2 f/\partial x^2 = f(x+1) + f(x-1) - 2*f(x)$$

Explanation of second order derivative:

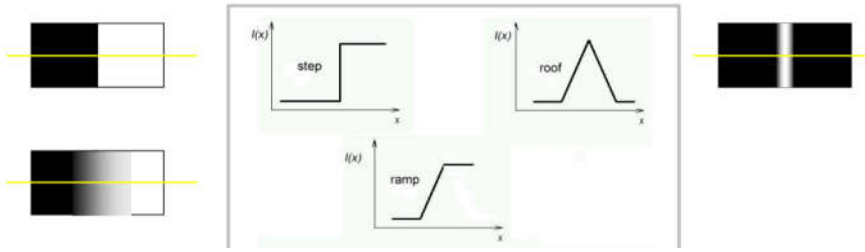
$\partial^2 f/\partial x^2 = [\, f(x+1) - f(x) \,] - [\, f(x) - f(x-1) \,]$

$\partial^2 f/\partial x^2 = [\, f(x+1) - f(x) - f(x) + f(x-1) \,]$

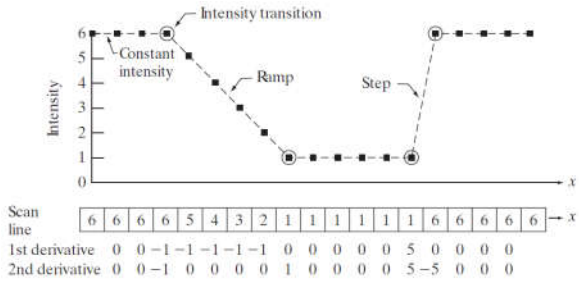
$\partial^2 f/\partial x^2 = [\, f(x+1) - 2f(x) + f(x-1) \,]$

Let's first understand the **different types of Edges:**

Edge Definition-Edge is a boundary between two regions with relatively distinct gray level properties. Edges are pixels where the brightness function changes abruptly.



4.9 ILLUSTRATION OF THE FIRST AND SECOND DERIVATIVES OF A 1-D DIGITAL FUNCTION-EXAMPLE



As the figure shows, the scan line contains **an intensity ramp, three sections of constant intensity, and an intensity step.**

The circles indicate the onset or end of intensity transitions.

A section of a horizontal intensity profile from an image

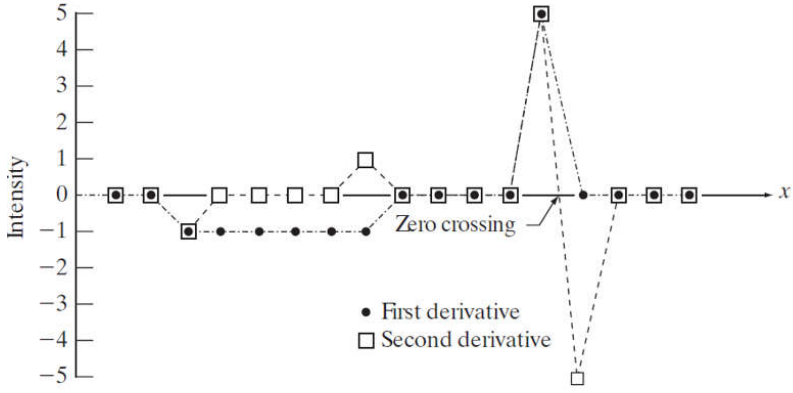
When computing the first derivative at a location **x**, subtract the value of the function at that location from the next point. So, this is a "look-ahead" operation.

In this method we take the 1st order derivative of the intensity value across the image and find points where the derivative is maximum then the edge could be located. Similarly, to compute the second derivative at **x**, we use the previous and the next points in the computation.

To avoid a situation in which the previous or next points are outside the range of the scan line, we show derivative computations in Figure, from the second through the penultimate points in the sequence.

Observation from the figure above-

1. First, we encounter an area of constant intensity and, as Figures show, both derivatives are zerothere, so condition (1) is satisfied for both.
2. Next, we encounter an intensity ramp followed by a step, and we note that the first-order derivative is nonzero at the onset of the ramp and the step. similarly, the second derivative is nonzero at the onset and end of both the ramp and the step; therefore, property (2) is satisfied for both derivatives.
3. We see that property (3) is satisfied also for both derivatives because the first derivative is nonzero and the second is zero along the ramp.



Scan line	6	6	6	6	5	4	3	2	1	1	1	1	1	1	6	6	6	6	6	→ x
1st derivative	0	0	−1	−1	−1	−1	−1	−1	0	0	0	0	0	5	0	0	0	0		
2nd derivative	0	0	−1	0	0	0	0	1	0	0	0	0	5	−5	0	0	0			

Note that the sign of the second derivative changes at the onset and end of a step or ramp.

The 2nd derivative of an image - where **the image highlights regions of rapid intensity change** and is therefore often used for **edge detection-zero crossing edge detectors**.

Using the Second Derivative for Image Sharpening-

The approach basically consists of defining a discrete formulation of the second-order derivative and then constructing a **filter mask** based on that formulation.

We are interested in **isotropic**filters, whose response is independent of the direction of the discontinuities in the image to which the filter is applied.In other words, isotropic filters are **rotation invariant**, in the sense that rotating the image and then applying the filter gives the same result as applying the filter to the image first and then rotating the result.

4.10 IMAGE SHARPENING FILTER -THE LAPLACIAN

The simplest isotropic derivative operator is the Laplacian, which, for a function (image) f (x,y) of two variables, is defined as-

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

Since derivative of any order is a linear operation, Laplacian is a linear operator. Discrete Laplacian operator Must satisfy the properties of second derivative.

Partial order derivative in x-direction-

$\partial^2 f/\partial x^2 = f(x + 1, y) + f(x - 1, y) – 2*f(x,y)$ ……eq1

Partial order derivative in y-direction-

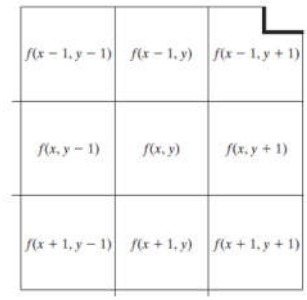
$\partial^2 f/\partial y^2 = f(x, y + 1) + f(x , y - 1) – 2*f(x,y)$…… eq 2

Discrete Laplacian in two dimensions is given by taking the sum of partial equations 1 and2 -

$\nabla^2 f(x, y) = f(x + 1, y) + f(x - 1, y) + f(x, y + 1) + f(x, y - 1) - 4f(x, y)$

The mask is given by –

0	1	0
1	-4	1
0	1	0



- The mask gives **isotropic** result in increments of 90°
- Because the Laplacian is a derivative operator, its highlights intensity discontinuities in an image and deemphasizes regions with slowly varying intensity levels.
- This will tend to produce images that have grayish edge lines and other discontinuities, all superimposed on a dark, featureless background.

4.11 USING FIRST-ORDER DERIVATIVES FOR (EDGE DETECTION) IMAGE SHARPENING - THE GRADIENT

First derivatives in image processing are implemented using the magnitude of the gradient. For a function f(x,y), the gradient offat coordinates (x, y) is defined as the two-dimensional column **vector**-

$$\nabla f \equiv \text{grad}(f) \equiv \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

This vector has the important geometrical property that it points in the direction of the greatest rate of change of fat location (x, y).

The **magnitude**(length) of vector ∇f denoted as M(x, y), where M(x,y) is the value at (x, y) of the rate of change in the direction of the gradient vector.

$$M(x, y) = \text{mag}(\nabla f) = \sqrt{g_x^2 + g_y^2}$$

Note that M(x, y) is an image of the same size as the original. It is common practice to refer to this image as the **gradient image**. In some implementations, it is more suitable computationally to approximate the squares and square root operations by absolute values:

$$M(x, y) \approx |g_x| + |g_y|$$

It is simpler to compute and preserves relative changes in gray levels. It does not preserve isotropic feature property.

Digital approximations to compute appropriate filtermasks - use the following notation to denote intensities in a 3 X 3 region

Z_1	Z_2	Z_3
Z_4	Z_4	Z_5
Z_6	Z_7	Z_8

- Simplest approximations is given by

  $g_x = f(x, y + 1) - f(x, y)$

  $g_y = f(x + 1, y) - f(x, y)$

- Robert's definition, based on cross differences

  $g_x = f(x + 1, y + 1) - f(x, y)$

  $g_y = f(x, y + 1) - f(x + 1, y)$

- Compute the gradient as-

$$M(x, y) = \sqrt{[f(x + 1, y + 1) - f(x, y)]^2 + [f(x, y + 1) - f(x + 1, y)]^2}$$

- Using absolute values, the gradient is given by –

$$M(x,y) \approx |f(x+1,y+1) - f(x,y)| + |f(x,y+1) - f(x+1,y)|$$

- Implemented with the mask (**Robert'scross-gradient operator**)-

-1	0
0	1

0	-1
1	0

- Even-sized masks are different to implement due to lack of centre of symmetry.

- An approximation using absolute values at point f(x; y) using a 3 X 3 mask is given by **Sobel operators.**

$$\begin{aligned} M(x,y) \approx\ & |(f(x-1,y+1) + 2f(x,y+1) + f(x+1,y+1)) - \\ & (f(x-1,y-1) + 2f(x,y-1) + f(x+1,y-1))| + \\ & |(f(x+1,y-1) + 2f(x+1,y) + f(x+1,y+1)) - \\ & (f(x-1,y-1) + 2f(x-1,y) + f(x-1,y+1))| \end{aligned}$$

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

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

- Difference in first and third row in the left mask gives partial derivative in the vertical direction.

- Difference in first and third column in the right mask gives partial derivative in the horizontal direction.

- Mask gives gradient in x and y directions, and coefficients sum to zero indicating no change in constant gray-level areas

- Used for **edge detection.**

4.12 SUMMARY

In this chapter we learned about the image histogram and its transformation. Histogram equalization can be used to improve the contrast of the image by stretching out the intensity range of the image. We discussed the low pass and high pass filters in spatial domain of the image. Smoothing Linear (Average filter) and non-linear filters (Median Filters) are discussed and their impact was shown for noise reduction. Mechanics of Sharpening filters is illustrated using first order and second order derivatives of a 1-D function using a line profile. Mathematics of Laplacian filter and gradient operators are discussed.

4.13 EXERCISE QUESTIONS

1. What is image histogram? Discuss the histogram pattens of low contrast and high contrast images.

2. Histogram of an image with 8 quantization levels is given below. Perform histogram equalization. Draw original and equalized histogram.

Grey Level	0	1	2	3	4	5	6	7
No of Pixel	220	70	50	150	130	70	150	160

3. What is median filter used for? Apply median filter on the below image keeping border values unchanged-

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

4. What is low pass filtering? Discuss the advantage of weighted average filters in the spatial domain.

5. How does Weighted average filter reduce the effect of blurring as compared to average filter. Apply the below filter to the given image-

Input Image

40	40	200	40	40
40	40	40	40	40
40	40	100	40	40
40	0	40	40	40
40	40	0	40	40
40	40	40	40	40

Kernel

1/16

1	2	1
2	4	2
1	2	1

6. Discuss the requirements to be satisfied for 1st order derivatives and 2nd order derivates in constant intensities, at ramp and at step edged of an image.

7. Write a note on high pass filters and formula for calculating 1st and second order derivatives.

8. Calculate the 1st and 2nd order derivatives for the below given profile of a line.

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

   Show the zero-crossing region in the graph.

9. Use diagrams to define a **Step, a Ramp and a Roof** in edge detection. Also discuss the impact of applying a median filter on isolated pixels and thin lines.

10. Write a note on Laplacian filter.

11. Using First-Order Derivatives for derive The Gradient Filter for edge detection.

4.14 REFERENCES

1. https://towardsdatascience.com/histogram-equalization-5d1013626e64
2. https://docs.opencv.org/
3. Digital Image Processing by Rafael Gonzalez & Richard Woods, Pearson; 4th edition, pdf.
4. Digital Image Processing by S. Jayaraman, Tata Mc-Graw Hill Publication, pdf.
5. Images used are processed using the tools- OpenCV (Python) and GNU Octave (compatible with MATLAB).

5

STRUCTURAL AND MORPHOLOGICAL OPERATIONS

Unit Structure :

5.0 OBJECTIVES

After going through this unit, you will be able to:

- Learn dilation, erosion, opening, and closing for morphological operations.
- Master edge detection with Sobel, Prewitt, Robert, and Canny techniques.
- Explore LoG and DoG filters for edge detection at different scales.
- Understand Gaussian and Laplacian pyramids for multi-scale image representation.
- Apply morphological operations like dilation, erosion, opening, and closing for image manipulation and feature extraction.

5.1 EDGE DETECTION

Our eyes naturally focus on edges in an image, as they often reveal important details about the scene. In image processing, edge detection plays a similar role, identifying sharp changes in brightness that can tell us a lot about the world captured in the image.

Why are edges important?

These sudden brightness shifts often correspond to:

- Differences in depth (nearby vs. faraway objects)
- Changes in surface orientation (flat vs. curved surfaces)
- Variations in materials (wood vs. metal)
- Shifts in lighting conditions (bright sun vs. shadow)

Ideally, edge detection would result in a clean map highlighting object boundaries, surface markings, and other significant changes in the image. This filtered image would contain less data, focusing on the essential structure while filtering out less relevant details.

Benefits of Clean Edges:

- **Reduced data:** Less data to process means faster analysis and potentially lower storage requirements.
- **Simplified interpretation:** Clear edges make it easier to understand the image content.

Challenges of Real-World Images:

Unfortunately, real-world images are rarely perfect. Edge detection can suffer from:

- **Fragmentation:** Edges may be broken into disconnected pieces, making it difficult to understand the whole picture.
- **Missing segments:** Important edges might be entirely missed.
- **False edges:** Edges may appear where there are no real changes in the scene, creating confusion.

If successful edge detection occurs, the subsequent task of interpreting the information within the original image can be significantly simplified. However, achieving ideal edges from real-life images of moderate complexity is not always feasible. Extracted edges from such images often suffer from fragmentation, where the edge curves are disjointed, leading to missing edge segments and false edges that do not correspond to meaningful features in the image. This complication further challenges the interpretation of the image data.

Edge detection serves as a fundamental step in various fields including image processing, image analysis, pattern recognition, and computer vision techniques. In recent years, considerable and successful research has also been conducted on computer vision methods that do not explicitly rely on edge detection as a pre-processing step.

5.2 EDGE PROPERTIES

Edge properties refer to the characteristics that define an edge detected in an image. These properties can be used to further analyze the edges and extract more information from the image. Here are some key edge properties:

1. Strength/Magnitude:
- Represents the intensity of the change in pixel intensity at the edge. Higher values indicate a more significant difference in brightness between neighboring pixels.

2. Orientation:
- Indicates the direction of the edge, typically measured in degrees (0-180) or radians (0-π). This helps differentiate between horizontal, vertical, or diagonal edges.

3. Location:
- Specifies the coordinates (x,y) of each edge pixel in the image. This allows for precise localization of edges and potential object boundaries.

4. Type:
- Depending on the edge detection algorithm used, the edge type might be categorized as a step edge (sharp transition), a ramp edge (gradual change), or a roof edge (double intensity change).

5. Connectivity:
- Describes how edge pixels are connected. Edges can be isolated points, short segments, or continuous curves outlining objects.

6. Curvature:
- Represents the degree to which the edge bends or curves. This can be helpful in identifying shapes and distinguishing between smooth and sharp corners.

7. Color:
- In color images, the edge may have a specific color profile that can be informative. For example, an edge between a red object and a blue background might have a combined color representing the transition.

By analyzing these edge properties, we can gain a deeper understanding of the image content. For example, strong edges with specific orientations might indicate object boundaries, while weak, fragmented edges could be noise or irrelevant details. Additionally, edge properties can be used for tasks like:

- **Image segmentation:** Grouping pixels with similar edge properties to isolate objects.
- **Shape recognition:** Analyzing edge curvature and orientation to identify shapes in the image.

- **Motion detection:** Tracking changes in edge properties over time to detect moving objects.

Understanding edge properties allows us to exploit edge detection for a wider range of image processing and computer vision applications.

5.3 SIMPLE EDGE MODEL

In image processing, a simple edge model is often used to understand the fundamental concept of edges and how edge detection algorithms work. This model provides a theoretical basis for more complex real-world scenarios.

There are two main approaches to simple edge models:

5.3.1. Step Edge Model:

This model assumes an ideal edge as a sudden and sharp transition in intensity between two constant regions. Imagine a horizontal line in a digital image where all pixels to the left of the line have a constant value (e.g., black) and all pixels to the right have another constant value (e.g., white). This creates a perfect step change in brightness, representing the simplest edge possible.

5.3.2. Ramp Edge Model:

This model acknowledges that edges in real images are rarely perfect step changes. Instead, the intensity might gradually transition from one level to another over a few pixels. This creates a ramp-like structure, where the brightness value changes progressively across the edge region.

Both models are abstractions, but they help us understand the core concept of edges and how edge detection algorithms function. These algorithms typically involve applying filters or mathematical operations to the image that aim to identify these sudden changes (step model) or significant intensity variations (ramp model).

Here are some key points to remember about simple edge models:

- They are theoretical constructs and don't perfectly represent real-world edges.
- They provide a foundation for understanding how edge detection works.
- Real image edges often exhibit a combination of step and ramp characteristics.

By understanding these simple models, we can move on to exploring more sophisticated edge detection techniques that can handle the complexities of real-world images.

5.4 EDGE DETECTION TECHNIQUES

Edge detection techniques play a crucial role in identifying boundaries and transitions within images, enabling various image processing tasks. Here, we delve into several prominent edge detection methods, including Sobel, Canny, Prewitt, and Robert operators, as well as LoG (Laplacian of Gaussian) and DoG (Difference of Gaussians) filters.

5.4.1. Sobel

The Sobel operator is a widely-used method for detecting edges in digital images. It is based on the computation of image gradients to identify areas of rapid intensity change, which typically correspond to edges or boundaries between objects. The Sobel operator is particularly effective due to its simplicity and computational efficiency. Here, we delve into the principles behind the Sobel operator and its application in edge detection.

The Sobel edge detection technique, which is commonly used in digital image processing. The Sobel operator helps identify edges by estimating the gradient magnitude and direction at each pixel in a grayscale image.

The Sobel algorithm is explained in detail below:

1. **Converting the Image into Grayscale**:
- Before applying edge detection, we convert the original color image into grayscale. Grayscale images have a single intensity channel, making them suitable for edge analysis.

2. **Sobel Filters**:
- The Sobel operator involves convolving the image with two filters: one in the x-direction and another in the y-direction.
- These filters are derivative filters, meaning they compute the first derivative of the image intensity along the x and y axes.

3. **Sobel-x Filter(Horizontal Sobel Kernel)**:
- To obtain the Sobel derivative along the x-direction, we perform an outer product between a 1D Gaussian filter and the x derivative.
- The Gaussian filter helps reduce noise, resulting in a smoother gradient computation.
- The Sobel-x filter looks like this:

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

4. **Sobel-y Filter(Vertical Sobel Kernel)**:
- Similarly, the Sobel-y filter is obtained by performing an outer product between the y derivative and a 1D Gaussian filter.
- The Sobel-y filter looks like this:

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

5. **Gradient Magnitude and Direction**:

- After convolving the grayscale image with both Sobel filters, we calculate the gradient magnitude and direction for each pixel.
- The gradient magnitude (G) can be computed as:

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

- Where (G_x) and (G_y) represent the gradients in the x and y directions, respectively.

6. **Edge Emphasis**:
   - Pixels with high gradient magnitude form edges in the image.
   - The Sobel operator emphasizes these edges, making them stand out.

Applications:

- **Edge Detection:** The Sobel operator is primarily used for edge detection in various image processing applications, such as object recognition, image segmentation, and feature extraction.
- **Image Analysis:** It is also employed in image analysis tasks where the detection of boundaries and transitions between regions is essential for understanding image content.
- **Real-Time Processing:** Due to its computational efficiency, the Sobel operator is suitable for real-time processing applications, including video processing and computer vision tasks.

Limitations:

- **Sensitive to noise:** Noise in the image can lead to spurious edges in the output. Pre-processing with noise reduction techniques might be necessary.
- **Not scale-specific:** The Sobel operator might struggle with edges at different scales in the image. Techniques like Difference of Gaussians (DoG) can address this limitation.

Example

Let's say we're working with a basic 3x3 grayscale image where the pixel intensities vary:

image

50	100	150
75	120	175
100	150	200

We'll apply the Sobel operator to detect edges in this image.

Horizontal Sobel Kernel:

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

Vertical Sobel Kernel:

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

Convolution:

We apply the horizontal and vertical Sobel kernels separately to the image using convolution:

For the horizontal gradient (gx):

gx = (-1*50) + (0*100) + (1*150) + (-2*75) + (0*120) + (2*175) + (-1*100) + (0*150) + (1*200)

= -50 + 0 + 150 - 150 + 0 + 350 - 100 + 0 + 200

= -50 + 0 + 150 - 150 + 0 + 350 - 100 + 0 + 200

= 300

For the vertical gradient (gy):

gy = (-1*50) + (-2*100) + (-1*150) + (0*75) + (0*120) + (0*175) + (1*100) + (2*150) + (1*200)

= -50 - 200 - 150 + 0 + 0 + 0 + 100 + 300 + 200

= -400 + 0 + 0 + 400

= 0

Gradient Magnitude:

The gradient magnitude is calculated using the formula:

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

= sqrt (G_x^2 + G_y^2)

= sqrt (300^2 + 0^2)

= sqrt (90000)

= 300

Thresholding:

We compare the gradient magnitude (300) to a predefined threshold (let's say 200). Since the magnitude exceeds the threshold, we consider this pixel as part of an edge.

Output:

We mark the corresponding pixel in the output image as an edge pixel.

5.4.2. Canny

The Canny edge detector is a widely recognized and powerful technique for identifying significant edges in images. It builds upon the foundation of simpler methods like Sobel or Prewitt but incorporates additional steps to achieve more robust and well-defined edges. The Canny algorithm provides accurate results by considering multiple stages of processing.

The Canny edge detector involves several key steps:

1. **Gaussian Smoothing**:
   - Before detecting edges, the Canny algorithm applies Gaussian blurring to the image. This helps reduce noise and smooths out pixel intensity variations.
   - The Gaussian filter is defined by the following equation:

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

   - Here, (x) and (y) represent the pixel coordinates, and controls the blurring strength.

2. **Gradient Calculation**:
   - Compute the gradient magnitude and direction using Sobel operators. These operators estimate the intensity changes along the x and y axes.
   - The gradient magnitude (G) is given by:

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

   - Where (G_x) and (G_y) are the gradients in the x and y directions, respectively.

3. **Non-Maximum Suppression**:
   - Suppress non-maximum gradient values to keep only the local maxima. This step ensures that only thin edges remain.
   - Compare the gradient magnitude with its neighbours along the gradient direction. If it's the maximum, retain it; otherwise, set it to zero.

4. **Double Thresholding**:
   - Set two thresholds: a high threshold (HT) and a low threshold (LT).
   - Pixels with gradient magnitude above (HT) are considered strong edges.
   - Pixels with gradient magnitude between (LT) and (HT) are considered weak edges.
   - Pixels below (LT) are suppressed (set to zero).

5. **Edge Tracking by Hysteresis**:
   - Connect weak edges to strong edges if they form continuous contours.
   - Follow the edges based on connectivity.

Applications:

- Robust Edge Detection: Canny effectively suppresses noise and identifies well-defined edges.
- Good Localization: The edges are accurately positioned relative to the actual intensity changes in the image.
- Single Edge Response: Each edge point is detected only once, avoiding multiple detections for the same edge.

Limitations:

- Computational Cost: Compared to simpler methods like Sobel, Canny involves more steps and can be computationally more expensive.
- Parameter Tuning: The choice of thresholds (HT and LT) can impact the results. Selecting appropriate thresholds depends on the specific image and desired edge characteristics.

Example:

1. Grayscale Conversion: Convert the image to black and white.

Imagine we have a simple 5x5 image with varying shades of gray:

100 100 100 150 200
100 100 100 150 200
100 100 100 150 200
100 100 100 150 200
100 100 100 150 200

2. Blur the Image: Smooth out the image to reduce noise.

After blurring, the image remains the same:

100 100 100 150 200
100 100 100 150 200
100 100 100 150 200
100 100 100 150 200
100 100 100 150 200

3. Find Edges: Look for areas of rapid intensity change.

The algorithm detects edges where there's a sudden change in intensity:

0 0 50 100 100
0 0 50 100 100
0 0 50 100 100
0 0 50 100 100
0 0 50 100 100

4. Thinning: Ensure edges are only one pixel wide.

Thin the edges to keep only the strongest ones:

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

5. Thresholding: Determine which edges are significant.

Set a threshold to distinguish strong and weak edges:

0 0 0 255 255
0 0 0 255 255
0 0 0 255 255
0 0 0 255 255
0 0 0 255 255

6. Edge Tracking: Connect weak edges to strong ones.

Ensure weak edges that are part of a strong edge are retained:

0 0 0 255 255
0 0 0 255 255
0 0 0 255 255
0 0 0 255 255
0 0 0 255 255

5.4.3. Prewitt

The Prewitt operator is a fundamental technique in image processing used for edge detection. It's a simple yet effective way to identify areas in an image where there's a sudden change in pixel intensity, which often corresponds to the edges of objects.

Here are the steps involved:

1. Read the Image:

First, read the image you want to process. If it’s a colored image, convert it to grayscale.

2. Convert to Grayscale:

If your image is in color, convert it to grayscale. The Prewitt operator works with grayscale images.

3. Apply Prewitt Masks:

The Prewitt operator provides two masks: one for detecting edges in the horizontal direction (along the x-axis) and another for detecting edges in the vertical direction (along the y-axis).

The Prewitt masks are as follows:

$$\text{Prewitt Operator [X-axis]} = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\text{Prewitt Operator [Y-axis]} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

4. Convolution:

- Convolve the image with both the Prewitt masks separately. This involves sliding the masks over the image and computing the weighted sum of pixel intensities.
- The result of convolution along the x-axis (kx) and y-axis (ky) represents the gradient components.

5. Edge Magnitude:

- Calculate the edge magnitude (ked) by combining the gradient components:

$$ked = \sqrt{kx^2 + ky^2}$$

6. Display the Results:

- Display the original grayscale image, the edge detection along the x-axis (abs(kx)), the edge detection along the y-axis (abs(ky)), and the full edge detection (abs(ked)).

Applications of Prewitt Operator:

- **Medical Imaging:** Can be used to identify boundaries of organs and tissues in X-ray or MRI scans.
- **Object Recognition:** Helps locate object outlines in images for further analysis.
- **Motion Detection:** Useful in identifying areas of significant intensity changes which could indicate movement.

Limitations

- Noise: Prone to false edges due to image noise.
- Accuracy: Less precise in pinpointing exact edge location and direction, especially for diagonals.
- Weak Edges: May miss subtle changes in intensity.

Example

Imagine you have a grayscale image of a black cat sitting on a white chair. The Prewitt operator would be helpful in identifying the edges where the black fur meets the white chair.

Here's how it works:

1. **Masks:** The Prewitt operator uses two 3x3 masks, one for horizontal edges and one for vertical edges.
   - **Horizontal Mask:** Emphasizes changes in intensity from left to right.

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

   - **Vertical Mask:** Emphasizes changes in intensity from top to bottom.

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

2. **Convolution:** It slides these masks across the image pixel by pixel. At each position, it multiplies the mask values with the corresponding pixel intensities in the image and sums the products. This gives a new value representing the "edge strength" for that pixel.

3. **Edge Detection:**

- **Magnitude:** We can calculate the overall edge strength by taking the absolute value of the results from both horizontal and vertical convolutions. This highlights strong edges (areas with high intensity change).
- **Direction (Optional):** We can also calculate the direction of the edge (horizontal, vertical, or diagonal) based on the signs of the mask values in the convolution.

The final result would be a new image highlighting the edges of the cat (where fur meets chair) with varying intensity based on how strong the change in grayscale value is at that point.

5.4.4. Robert edge detection techniques

The Roberts operator is a simple and quick-to-compute method for measuring a 2-D spatial gradient on an image. It highlights strong spatial gradient regions, which often correspond to edges.

1. **Roberts Edge Detection:**

- The Roberts operator calculates the gradient intensity in discrete differentiation. It approximates the gradient by computing the sum of squares of differences between diagonally adjacent pixels.
- The operator uses a pair of 2x2 convolution masks, one of which is merely the other rotated by 90 degrees. This is similar to the Sobel operator.
- The masks are designed to respond maximally to edges running at a 45° angle to the pixel grid.
- The masks can be applied to the input grayscale image independently to produce separate gradient component measurements in each orientation.
- These components can then be combined to determine the absolute magnitude and orientation of the gradient at each location.

2. **Roberts Masks:**

- The two Roberts masks are as follows:

$$\text{Roberts Operator [X-axis]:} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

$$\text{Roberts Operator [Y-axis]:} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

3. **Steps to Apply Roberts Edge Detection:**

- Read the image and convert it to grayscale.
- Initialize the Roberts cross operator masks.
- Convolve the image with both the X-axis and Y-axis masks.
- Calculate the gradient magnitude using:

$$G=\sqrt{horizontal^2+vertical^2}$$

Applications:

- **Simple and Fast:** The Roberts operator is very easy to understand and implement due to its small kernel size (2x2) and simple calculations.

Limitations:

- **Overly Sensitive:** It can be sensitive to noise in the image, leading to the detection of false edges.

- **Less Precise:** While good at detecting strong edges, it may not be as accurate in pinpointing the exact location of the edge compared to more sophisticated methods.
- **Diagonal Focus:** The Roberts operator is primarily sensitive to edges along diagonal directions (because it checks diagonals).

Example

Consider a tiny 2x2 grayscale image:

[100, 150]

[120, 180]

We'll apply the Robert operators to compute the horizontal and vertical gradients, then calculate the gradient magnitude.

1. Horizontal gradient:

[-1, 0]		[100, 150]		[-1*100 + 0*120, -1*150 + 0*180]		[-100, -150]
[0, 1]	*	[120, 180]	=	[0*100 + 1*120, 0*150 + 1*180]	=	[120, 180]
[0, 0]		[0, 0]				

2. Vertical gradient:

[0, -1]		[100, 150]		[0*100 + -1*120, 0*150 + -1*180]		[-120, -180]
[1, 0]	*	[120, 180]	=	[1*100 + 0*120, 1*150 + 0*180]	=	[100, 150]
[0, 0]		[0, 0]				

3. Compute gradient magnitude:

gradient_magnitude = sqrt((-100)^2 + (-150)^2 + (-120)^2 + (-180)^2)

= sqrt(10000 + 22500 + 14400 + 32400)

= sqrt(79300)

≈ 281.39

The gradient magnitude for this simplified example is approximately 281.39.

5.4.5. LoG filters

Log filters are tools used to narrow down the content of log files, which are digital records of events that occur within a system or application. There are many reasons why you might want to filter log files. The term "LoG filters" can refer to two different concepts depending on the context:

- **Laplacian of Gaussian (LoG) in Image Processing:** The LoG filter is used in image processing for edge detection. It involves applying a Gaussian blur to an image before calculating the Laplacian. This helps to reduce noise which can be amplified when calculating the second derivative, which is what the Laplacian does. The LoG operator takes a single grayscale image as input and produces another grayscale image as output. The 2-D LoG function centered on zero and with Gaussian standard deviation σ has the form:

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

This filter highlights regions of rapid intensity change and is often used for edge detection.

- **Logging Filters in Computing:** In the context of computing and cloud services, LoG filters could refer to filters used in logging to query and filter data. For example, Google Cloud's Logging query language allows you to write filters to create sinks and log-based metrics. A query is a Boolean expression that specifies a subset of all the log entries in your selected Google Cloud resource.

Application:

- Effective for edge detection.
- Enhances features and textures.
- Useful for blob detection.
- Enables scale-space analysis.

Limitations:

- Can be computationally intensive.
- Sensitive to parameter choices.
- Challenges in precise edge localization.
- Susceptible to noise amplification.
- Complexity in selecting the appropriate scale.

Example

Consider a small 3x3 grayscale image:

[100, 150, 200]
[120, 180, 220]
[90, 140, 190]

We'll apply the LoG filter with a Gaussian kernel of size 3x3 and sigma (standard deviation) of 1.

1. Apply Gaussian Smoothing:

We'll convolve the image with a 3x3 Gaussian kernel.

Gaussian Kernel:

1/16 2/16 1/16
2/16 4/16 2/16
1/16 2/16 1/16

Smoothed Image:

[147.8125, 183.5, 198.1875]

[163.75, 190.25, 206.25]

[140.1875, 167.75, 183.8125]

2. Apply Laplacian Operator:

We'll apply the Laplacian operator to the smoothed image.

Laplacian Kernel:

0 1 0

1 -4 1

0 1 0

LoG Image:

[-17.9375, -5.9375, 9.3125]

[-7.4375, -14.1875, -1.4375]

[4.3125, -1.4375, -7.9375]

The resulting LoG-filtered image for this numerical example is:

[-17.9375, -5.9375, 9.3125]

[-7.4375, -14.1875, -1.4375]

[4.3125, -1.4375, -7.9375]

This demonstrates the application of the Laplacian of Gaussian filter to a small numerical example. In practice, larger images and different sigma values would be used for more meaningful results.

5.4.6. DoG filters

The term “DoG filters” refers to Difference of Gaussians, which is a technique used in image processing, particularly in edge detection and feature extraction. The DoG filter is a band-pass filter that highlights edges and details in an image by subtracting one blurred version of an original image from another, less blurred version of the original.

Here’s how the DoG filter works:

Create two blurred images: First, you create two blurred versions of the original image using Gaussian blurring with different standard deviations (σ). The first image is blurred with a smaller σ (narrower Gaussian), and the second image is blurred with a larger σ (wider Gaussian).

Subtract the blurred images: You then subtract the wider-blurred image from the narrower-blurred image. This subtraction process results in a new image that emphasizes the edges that are present in the spatial range between the two levels of blurring.

Mathematical Representation: The DoG function (D(x, y, σ)) can be represented as the difference between two Gaussian blurred images (G(x, y, kσ) and (G(x, y, σ):

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

where (k) is a constant that determines the difference in blurring between the two images.

Application:

- Effective for edge detection.
- Enhances features and textures.
- Enables scale-space analysis.
- Useful for blob detection.

Limitations:

- Parameter sensitivity.
- Limited edge localization.
- Noise sensitivity.
- Computational overhead (though less than LoG).

Example:

Suppose we have a 3x3 grayscale image:

[100, 150, 200]

[120, 180, 220]

[90, 140, 190]

We'll apply two Gaussian filters with different standard deviations (σ) and compute their difference.

1. Gaussian with σ1 = 1:

[0.0751, 0.1238, 0.0751]

[0.1238, 0.2042, 0.1238]

[0.0751, 0.1238, 0.0751]

2. Gaussian with σ2 = 2:

[0.028, 0.05, 0.028]

[0.05, 0.091, 0.05]

[0.028, 0.05, 0.028]

3. Difference (DoG):

[0.0471, 0.0738, 0.0471]

[0.0738, 0.1132, 0.0738]

[0.0471, 0.0738, 0.0471]

In this example, we perform the following steps:

- **Convolution:** We apply two Gaussian filters with different standard deviations to the original image.
- **Subtraction:** We subtract the resulting blurred images to obtain the difference (DoG) image.

5.5 IMAGE PYRAMIDS

Image pyramids are a multi-level approach in image processing that allows for multi-scale representation and analysis of images. Types are:

5.5.1 Gaussian Pyramid

Definition: A Gaussian pyramid is a hierarchical representation of an image that captures its content at multiple scales by successively applying Gaussian blurring and down sampling.

Construction:
- The process begins with the original image, which is convolved with a Gaussian kernel to blur it.
- Subsequently, the blurred image is subsampled to reduce its size.
- This process is repeated iteratively, creating a pyramid of images, each with reduced resolution but capturing the image content at different scales.

Purpose:
- Gaussian pyramids are used in various applications such as image blending, texture analysis, and scale-invariant feature detection.
- They provide an efficient and compact representation of images at multiple resolutions, enabling computationally efficient processing at different scales.

Example

Consider an original image of size 256x256 pixels. We can construct a Gaussian pyramid with, for example, 4 levels. At each level, we successively blur the image using Gaussian filtering and then downsample it by a factor of 2. This results in a pyramid with images at decreasing resolutions, capturing the image content at different scales. Here's an example:

Level 0: Original Image (256x256)

Level 1: Blurred &Downsampled (128x128)

Level 2: Further Blurred &Downsampled (64x64)

Level 3: Further Blurred &Downsampled (32x32)

Level 4: Further Blurred &Downsampled (16x16)

5.5.2 Laplacian Pyramid

Definition: Derived from the Gaussian pyramid, the Laplacian pyramid represents the details of an image at different scales.

Construction:

- The Laplacian pyramid is constructed by taking the difference between each level of the Gaussian pyramid and the up sampled version of the next level.
- This difference operation retains the high-frequency details discarded during Gaussian blurring.

Purpose:

- Laplacian pyramids are commonly used in image compression, as they efficiently capture the information required for image reconstruction while discarding redundant details.
- They are also employed in tasks like edge detection, texture synthesis, and image sharpening.

Benefits:

- The Laplacian pyramid provides a compact representation of image details, making it useful for various image processing tasks requiring multi-scale analysis.

Example

Using the Gaussian pyramid constructed above, we can derive the Laplacian pyramid. For each level of the Gaussian pyramid (except the last level), we up sample the next level and subtract it from the current level. This results in a pyramid capturing the details at different scales. Here's an example:

Level 0: Original Image (256x256)

Level 1: Details (128x128)

Level 2: Details (64x64)

Level 3: Details (32x32)

5.5.3 Morphological pyramid

Definition: A morphological pyramid is a variant of the Laplacian pyramid used specifically in morphological image processing.

Construction:

- Like the Laplacian pyramid, it is built by taking the difference between levels of the Gaussian pyramid.
- However, instead of the difference operation, morphological operations such as erosion and dilation are applied.

Purpose:

- Morphological pyramids are used in tasks like image segmentation, shape analysis, and feature extraction in morphological image processing.
- They enable the representation of morphological features at different scales, facilitating the analysis and processing of complex structures in images.

Example

For a morphological pyramid, suppose we have an image containing binary objects (e.g., shapes). We can use morphological operations such as erosion and dilation to create a pyramid representing the morphological features at different scales. Here's an example:

Level 0: Original Binary Image (256x256)

Level 1: Eroded & Dilated (128x128)

Level 2: Further Eroded & Dilated (64x64)

Level 3: Further Eroded & Dilated (32x32)

5.6 SUMMARY

In image processing, edge detection is crucial for identifying boundaries between objects or regions. Various techniques, such as Sobel, Canny, and Prewitt operators, utilize gradient information to detect edges accurately. Simple edge models like step and ramp edges offer idealized representations for analysis. Advanced methods like LoG and DoG filters combine Gaussian blurring with differentiating operators to detect edges at multiple scales. Additionally, image pyramids, including Gaussian, Laplacian, and morphological pyramids, facilitate multi-scale analysis, aiding tasks such as image compression and feature extraction. These techniques are fundamental in computer vision applications, offering insights into image structures and facilitating further processing.

5.7 REFERENCE FOR FURTHER READING

- Gonzalez, R. C., Woods, R. E., & Eddins, S. L. (2018). Digital Image Processing Using MATLAB. Gatesmark Publishing.
- Sonka, M., Hlavac, V., & Boyle, R. (2014). Image Processing, Analysis, and Machine Vision. Cengage Learning.

- Jain, A. K. (1989). Fundamentals of Digital Image Processing. Prentice Hall.
- Burger, W., & Burge, M. J. (2009). Principles of Digital Image Processing: Fundamental Techniques. Springer Science & Business Media.
- Szeliski, R. (2010). Computer Vision: Algorithms and Applications. Springer Science & Business Media.

Websites

- https://docs.opencv.org/
- https://www.pyimagesearch.com/
- https://towardsdatascience.com/tagged/image-processing
- https://www.mathworks.com/help/images/

6

IMAGE PROCESSING

Unit Structure :

6.0 OBJECTIVES

After completing this unit, you will be proficient in a range of fundamental image processing techniques. You will be able to effectively utilize erosion and dilation to enhance boundaries and fill gaps, implement opening and closing operations for noise reduction and object preservation, employ Hit-or-Miss Transformation for pattern detection, and skeletonize images to thin object structures while maintaining connectivity. Additionally, you will have the skills to compute convex hulls for understanding object shapes, remove small objects while preserving essential features, and utilize white and black top-hat transformations for feature extraction. You will also be adept at extracting boundaries for further analysis and applying grayscale operations for contrast enhancement and filtering, thereby enhancing your capabilities in image processing.

Set theory notation is a powerful tool in image processing, particularly for defining fundamental morphological operations like erosion and dilation. Here's a breakdown of how it's used:

Basics of set theory

Image as a Set:

- We consider a digital image as a set of pixels. Each pixel is represented by its coordinates (x, y) and has a specific intensity value.

- In binary images, the intensity values are typically 0 (black) and 1 (white). So, the image becomes a set of coordinates representing foreground pixels (usually white).

Basic Set Operations:

- **Union (U):** This combines pixels that belong to either set A or set B or both. In image processing, it could represent combining two separate objects in an image.
- **Intersection (∩):** This includes pixels that belong to both set A and set B. It's useful for finding overlapping regions between objects.
- **Difference (\):** This includes pixels that are in set A but not in set B. In image processing, it can be used to isolate objects by removing overlapping areas.
- **Complement (A^c):** This represents all the pixels that are not in set A. In a binary image, it would be all the black pixels if A represents the white foreground objects.

Morphological Operations:

- **Erosion (⊖):** As discussed earlier, erosion is defined using a subset relationship. A pixel is included in the eroded image (A ⊖ B) only if the structuring element B, shifted to that pixel's location, is entirely contained within the original image (A).
- **Dilation (⊕):** Dilation, the opposite of erosion, expands objects by incorporating pixels from the background that are close to the object boundaries. It can be defined using set complement and union operations on the structuring element and the original image.

6.1 EROSION

Erosion is a fundamental morphological operation in image processing that is used to shrink or thin objects in a binary image. It involves a process where the boundaries of objects are eroded away based on a predefined structuring element. This structuring element, which can be of various shapes like a disk or a square, moves over the image, and at each position, it compares its shape with the object's pixels in the image. If the structuring element fits within the object, the original pixel value is retained; otherwise, it is set to zero, effectively eroding the boundary of the object.

To summarize, erosion results in:

- **Shrinks objects:** Erosion nibbles away at the edges of foreground objects (usually white pixels) in a binary image.
- **Enlarges holes:** As objects shrink, any holes within them conversely become larger.
- **Uses a structuring element:** The effect of erosion depends on a small binary shape called a structuring element (or kernel). This element

defines the neighborhood around a pixel that's considered during erosion. Common structuring elements include squares, circles, and diamonds.

The erosion ofa binary image Aby a structuring element B is defined as:

$$A \ominus B = \{z \mid (Bz \subseteq A)\}$$

where:

- $A \ominus B$ represents the erosion of image A by structuring element B.
- z represents a pixel location in the image.
- $(Bz \subseteq A)$ is a set operation that checks if the shifted version of B, denoted by Bz, is completely contained within the original image A.

A pixel location (z) is considered part of the eroded image ($A \ominus B$) only if the structuring element B, when centered at that location (z), fits entirely within the foreground region of the original image (A).

Applications of erosion:

- **Noise reduction:** Erosion can help remove small isolated pixels that might be caused by noise in the image.
- **Object separation:** By selectively eroding objects, you can isolate them from touching or overlapping objects.
- **Shape analysis:** Erosion can be used to analyze the shape of objects in an image by measuring how much they shrink under erosion.

Example

Imagine a binary image where "1" represents the object and "0" represents the background:

0 0 0 0 0
0 1 1 1 0
0 1 1 1 0
0 1 1 1 0
0 0 0 0 0

Let's use a simple 3x3 square structuring element:

1 1 1
1 1 1
1 1 1

To create erosion, we slide the structural element over the image. If all of the pixels under the structural element are "1", the center pixel remains "1"; otherwise, it is "0".

Here is the outcome of erosion:

0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0

In this simplified example, the objects in the binary image have shrunk, or eroded, because we've eliminated pixels from their edges.

6.2 DILATION

Dilation, alongside erosion, is a fundamental operation in image processing, particularly within the realm of mathematical morphology. In contrast to erosion, which shrinks objects, dilation expands the boundaries of objects in a binary image.

To summarize, dilation results in:

- Increases object size: Dilation adds pixels to the edges of foreground objects (usually white pixels).
- Reduces holes: As objects grow, small holes within them tend to disappear.
- Uses a structuring element: Similar to erosion, dilation relies on a small binary shape called a structuring element (or kernel) to define the neighborhood around a pixel. Common structuring elements include squares, circles, and diamonds.

The dilation of a binary image A by a structuring element B is defined as:

The dilation of a binary image (A) by a structuring element (B) is a fundamental operation in image processing, particularly in the field of mathematical morphology. It is defined as the set of all displacements (z) such that (B) and (A) overlap by at least one element when (B) is translated by (z). In simpler terms, it involves expanding the pixels of image (A) according to the shape defined by (B).

$$A \oplus B = \{z \mid (\hat{B})_z \cap A \neq \emptyset\}$$

Where

$(\hat{B})_z$ represents the translation of (B) by the vector (z), and the operation checks for an overlap between (A) and (B). If there is at least one common element, the pixel in the resulting image at position (z) is set to 1 (or the maximum value of the pixels in the neighborhood for grayscale images).

Applications of dilation:

- **Object thickening:** Dilation can be used to thicken lines or strengthen weak edges of objects.
- **Bridging gaps:** If objects are partially touching or have small gaps between them, dilation can help connect them.
- **Noise reduction:** Dilation can be useful for filling in small holes caused by image noise, although it might also enlarge existing objects slightly.

Example

Consider a binary image where "1" represents the object and "0" represents the background:

0 0 0 0 0
0 1 0 1 0
0 0 0 0 0

Let's use a simple 3x3 square structuring element:

1 1 1
1 1 1
1 1 1

To perform dilation, we slide this structuring element over the image. If any part of the structuring element overlaps with a foreground pixel (marked as "1") in the image, we set the center pixel of the structuring element to "1".

Here's the result of dilation:

1 1 1 1 1
1 1 1 1 1
1 1 1 1 1

In this simplified example, the objects in the binary image have been expanded or thickened as we've added pixels to their boundaries using the 3x3 square structuring element.

Duality

Erosion and dilation are duals of each other with respect to set complementationand reflection. This duality is a powerful concept that helps in understanding and applying these operations effectively in image processing and analysis.

The duality principle can be expressed as:

$$(A \ominus B)^c = A^c \oplus \hat{B}$$

$$(A \oplus B)^c = A^c \ominus \hat{B}$$

These operations are used in various image processing tasks such as noise reduction, shape analysis, and feature extraction

6.3 OPENING AND CLOSING

Opening and closing are image processing techniques built upon the foundation of erosion and dilation. They are often used sequentially to achieve specific image manipulations.

Opening:

- **Function:** Removes small objects in the foreground (usually white pixels) while preserving larger ones.

- **Process:** Performs erosion followed by dilation using the same structuring element.
- **Analogy:** Imagine opening a box with a small key. Small features (dust particles) get removed during "erosion," while the desired object (larger than the key) remains after "dilation."
- **Applications:** Noise reduction, object separation (by selectively removing small objects touching larger ones).
- Opening is performed by applying erosion followed by dilation on an image.
- It's useful for removing small objects or noise from the foreground of an image while preserving the larger structures.
- The opening operation can be represented as $A \circ B = (A \ominus B) \oplus B$, where A is the input image and B is the structuring element.
- Opening is particularly effective in cases where objects of interest are smaller than the structuring element and need to be removed.

Closing:

- **Function:** Fills small holes in the foreground objects and connects narrow breaks between them.
- **Process:** Performs dilation followed by erosion using the same structuring element.
- **Analogy:** Imagine filling small cracks in a wall with putty (dilation) and then smoothing the surface (erosion) to create a more uniform look.
- **Applications:** Improves object segmentation, reduces speckles in images.
- Closing is performed by applying dilation followed by erosion on an image.
- It's useful for closing small gaps or holes in the foreground of an image while preserving the overall shape of objects.
- The closing operation can be represented as $A \cdot B = (A \oplus B) \ominus B$, where A is the input image and B is the structuring element.
- Closing is effective in cases where objects of interest are larger than the structuring element and need to be filled or connected.

Differences:

- **Target:** Opening targets small objects, while closing targets small holes and gaps.

- **Order:** Opening is erosion followed by dilation, whereas closing is dilation followed by erosion.

Combined Usage:

- Often used together for image pre-processing before further analysis.
- The choice of structuring element significantly impacts the outcome.
- By combining opening and closing, you can achieve more sophisticated image manipulations.

Example

Imagine a 1D "image" with values representing foreground (1) and background (0):

Original Image (A): [1, 0, 1, 1, 0, 1, 0, 1, 0]

Structuring Element (B): [1, 1] (assuming a small rectangular structuring element)

Opening (Erosion followed by Dilation):

1. Erosion: Iterate over each element in A. If both elements in B (shifted to that position) are not 1 (foreground), the element in A becomes 0 (background).

Eroded Image: [1, 0, 0, 1, 0, 1, 0, 1, 0] (Notice isolated foreground elements are removed)

2. Dilation: Iterate over each element in the eroded image. If at least one element in B (shifted to that position) is 1 (foreground), the element in the eroded image becomes 1 (foreground).

Opened Image: [1, 0, 1, 1, 0, 1, 0, 1, 0] (Small foreground elements are gone, larger ones remain)

Closing (Dilation followed by Erosion):

1. Dilation: Following the same principle as above.

Dilated Image: [1, 1, 1, 1, 1, 1, 1, 1, 0] (Small gaps are filled)

2. Erosion: Similar to the erosion step in opening.

Closed Image: [1, 1, 1, 1, 0, 1, 0, 1, 0] (Small protrusions are removed, but gaps are filled)

This simplified example demonstrates how opening removes small foreground elements (noise) and closing fills small gaps/holes. Remember, in real images, these operations would be applied to a 2D array of pixels, and the size and shape of the structuring element would significantly impact the results.

Morphological opening and closing has the following properties:

1. **Idempotence:**

- Performing opening or closing on an image multiple times with the same structuring element yields the same result as performing it once.
- Mathematically, $(A \circ B) \circ B = A \circ B$ and $(A \cdot B) \cdot B = A \cdot B$.

2. **Extensivity:**

- Morphological opening and closing are extensive operations, meaning the resulting image always contains the input image.
- Mathematically, $A \subseteq A \circ B$ and $A \subseteq A \cdot B$

3. **Non-Increasing:**

- Morphological opening is non-increasing, meaning the size of objects in the resulting image is never greater than the size of objects in the input image.
- Similarly, morphological closing is also non-increasing.

4. **Commutativity with Inclusion:**

- Morphological opening and closing commute with inclusion, meaning if one image is a subset of another, the same holds true for the opening or closing of those images.
- Mathematically, if $A \subseteq C$, then $A \circ B \subseteq C \circ B$ and $A \cdot B \subseteq C \cdot B$

5. **Associativity:**

- Opening and closing are associative operations, meaning the order of applying these operations does not affect the final result.
- Mathematically, $(A \circ B) \circ C = A \circ (B \circ C)$ and $(A \cdot B) \cdot C = A \cdot (B \cdot C)$

6.4 HIT-OR-MISS TRANSFORMATION

The Hit-or-Miss Transformation is a morphological operation used in image processing to detect specific patterns within a binary image. It operates by employing two structuring elements that are disjoint, meaning they do not overlap. The transformation looks for places where the first structuring element fits within the foreground of the image, while the second structuring element fits within the background.

Mathematically, if (A) is the binary image and (B = (C, D)) is the composite structuring element where (C) and (D) are disjoint, the Hit-or-Miss Transformation of (A) by (B) is defined as:

$$A \otimes B = (A \ominus C) \cap (A^c \ominus D)$$

Here, ($A \ominus C$) represents the erosion of image (A) by structuring element (C), and ($A^c \ominus D$) represents the erosion of the complement of image (A) by structuring element (D).

Applications:

- **Shape Detection:** It can identify specific shapes or objects in an image based on their unique pixel configurations.
- **Line End-Point Detection:** By defining appropriate structuring elements, you can locate the endpoints of lines in an image.
- **Feature Extraction:** It can be used to extract specific features from an image that might be relevant for further analysis.
- **Pattern Recognition:** It can be a building block for more complex pattern recognition algorithms in image processing tasks.

Example

Imagine a small 3x3 binary image (A):

A = [[0, 1, 0],[0, 1, 1],[0, 0, 0]]

find all locations with a vertical line segment (two foreground pixels stacked on top of each other).

Structuring Elements:

- B1 (Hit Element): A vertical line segment with two foreground pixels (1s).

B1 = [[1, 0],[1, 0]]

- B2 (Miss Element): A single foreground pixel (1) in the center, representing a position that should not have a foreground pixel in the original image for a hit.

B2 = [[0, 1, 0],[0, 1, 0]]

Process:

We'll iterate over each pixel in the image (A) and apply the following logic at each position (i, j):

1. **Shift B1 and B2 to the current location (i, j):**
   - Shifted B1: Overwrite the corresponding pixels in A starting from (i, j) with the values from B1.
   - Shifted B2: Overwrite the corresponding pixels in A starting from (i, j) with the values from B2.
2. **Check for hit conditions:**
   - Hit condition 1 (B1): All pixels overlapped by the shifted B1 in A must be 1 (foreground).

- Hit condition 2 (B2): No pixels overlapped by the shifted B2 in A should be 1 (foreground).

3. Output Image:

- If both hit conditions are satisfied (vertical line segment found), set the corresponding pixel (i, j) in the output image to 1 (foreground).
- Otherwise, leave the output pixel (i, j) as 0 (background).

Iterating over the image (A):

- Top-left corner (0, 0): Shifted B1 doesn't fit (background pixel above), so it's a miss. Output (0, 0) remains 0.
- Other locations can be analyzed similarly.

Output Image:

After checking all positions, the output image will have a 1 (foreground) only at location (1, 1) because that's the only place where the vertical line segment is found (two foreground pixels stacked). All other output pixels will be 0 (background).

6.5 SKELETONIZING

Skeletonizing in image processing refers to the process of reducing a shape to its basic form, which is its skeleton. This is done by successively eroding the boundary of the shape until only the 'skeleton' remains. The skeleton consists of a subset of the medial axis points of the shape and represents the general form of the shape in a simplified manner.

In the context of binary images, skeletonizing is used to thin objects to their minimal representation without changing the essential structure or connectivity. This can be particularly useful for feature extraction, pattern recognition, and image analysis tasks.

Mathematically, the skeleton (S) of a shape (A) in a binary image can be defined as:

$$S = \bigcap_{i=1}^{n} (A \ominus B_i)$$

Here, ($A \ominus B_i$) represents the erosion of the shape (A) by a structuring element (B_i), and the intersection of all such erosions gives the skeleton of the shape. The structuring elements (B_i) are chosen to preserve the connectivity of the shape while reducing its boundary.

What skeletonizing does:

- **Reduces object thickness:** It iteratively removes pixels from the boundaries of foreground objects (usually white pixels) until a one-pixel-wide skeleton remains.
- **Maintains connectivity:** Unlike simple erosion, skeletonization aims to preserve the connectedness of the object, ensuring it remains as a single entity.

Applications:

- **Object recognition:** Skeletons can be used for object recognition tasks as they represent the core structure of the object.
- **Shape analysis:** By analyzing the skeleton, you can measure properties like the object's length, branching points, or overall form.
- **Medical imaging:** In medical image analysis, skeletonizing can be used to extract the centerline of blood vessels or bone structures.

Example

Imagine a small 5x5 binary image (A) representing a simple shape:

```
A = [[0, 1, 1, 1, 0],
     [0, 1, 1, 1, 0],
     [0, 1, 1, 1, 0],
     [0, 1, 1, 1, 0],
     [0, 0, 0, 0, 0]]
```

This represents a square with a thickness of 3 pixels.

Skeletonization Process (Simplified):

We'll assume a basic iterative approach where we remove pixels from the object's boundaries that are "safe" to remove (won't break connectivity).

Iteration 1:

1. Identify pixels on the object's boundary (all foreground pixels touching a background pixel).
2. Remove pixels that have exactly two foreground neighbors (these are safe to remove as they won't disconnect the object).

Result after Iteration 1:

The image will look similar to the original, but some outermost layer pixels might be removed, depending on the specific removal criteria.

Iterations 2 and beyond:

1. Repeat the process of identifying and removing safe boundary pixels.
2. Stop when no more safe pixels can be removed without compromising the one-pixel width or connectivity of the object.

Final Skeleton:

After several iterations, the resulting image might look like:

Skeleton = [0, 1, 0, 1, 0],
[0, 0, 1, 0, 0],
[0, 1, 0, 1, 0],
[0, 0, 1, 0, 0],
[0, 0, 0, 0, 0]]

This represents a one-pixel-wide skeleton that captures the essential shape of the square.

6.6 COMPUTING THE CONVEX HULL

In image processing, computing the convex hull is a crucial step for various applications such as shape analysis, pattern recognition, and object detection. The convex hull in this context refers to the smallest convex shape that can enclose all the points (or pixels) of the object of interest within an image.

Here's the conventional method for computing the convex hull in image processing.:

- **Image Preprocessing:** The image is preprocessed to detect edges or points that define the shape of the object. This might involve thresholding, filtering, and edge detection.
- **Point Collection:** The detected points or edge pixels are collected as a set of coordinates that will be used to compute the convex hull.
- **Convex Hull Algorithms:** One of the convex hull algorithms is applied to these points. Common algorithms used in image processing include:
  - **Graham Scan:** Efficient for larger datasets, it sorts the points and constructs the hull using a stack of candidate vertices1.
  - **Jarvis's March (Gift Wrapping):** Suitable for smaller sets of points, it wraps around the points by selecting the most counterclockwise point relative to the current point2.
  - **QuickHull:** Similar to QuickSort, it uses a divide and conquer strategy to find the convex hull3.
- **Convex Hull Representation:** The resulting convex hull is often represented as a polygon for 2D images or a polyhedron for 3D images. This representation can be used for further analysis or processing tasks.

Applications:

- **Image Registration and Retrieval:** Matching and searching images based on shape.
- **Image Classification:** Categorizing images based on the convexity of shapes.

- **Shape Detection and Extraction:** Identifying and isolating specific shapes within an image.

Convex Hull Algorithms

1. **Graham Scan:**

- **Sorting Points:** First, it sorts the points based on their polar angles relative to a reference point. This sorting step ensures that the points are ordered in a counterclockwise direction.
- **Constructing Hull:** Starting from the point with the smallest polar angle (the reference point), it iterates through the sorted points. For each point, it checks whether the angle formed by the last two points in the hull and the current point makes a left turn. If it does, the point is added to the hull. If it makes a right turn, the last point is removed from the hull until a left turn is achieved.
- **Efficiency:** Graham Scan is efficient for larger datasets because its time complexity is
  O(nlogn), mainly dominated by the sorting step.

2. **Jarvis's March (Gift Wrapping):**

- **Selection of Initial Point:** It starts by selecting the leftmost point as the initial point of the convex hull.
- **Selecting Next Point:** Iteratively, it selects the point with the smallest counterclockwise angle relative to the current point. This is done by considering the cross product of vectors formed by the current point and each candidate point.
- **Wrapping Around Points:** The algorithm wraps around the points until it reaches the initial point again, forming the convex hull.
- **Suitability:** Jarvis's March is suitable for smaller sets of points because its time complexity is $O(n\square)$, where $\square$h is the number of points on the convex hull. For larger datasets, its time complexity can be $O(n2)$, making it less efficient.

3. **QuickHull:**

- **Divide and Conquer:** QuickHull follows a divide and conquer strategy similar to QuickSort. It recursively divides the set of points into subsets based on their relationship with a line connecting the two outermost points (extreme points).
- **Finding Extreme Points:** It first finds the extreme points (leftmost and rightmost points) of the set.
- **Partitioning:** It partitions the points into two subsets based on which side of the line they lie. Points lying on the left side are included in the left subset, and points lying on the right side are included in the right subset.
- **Recursive Hull Construction:** It recursively constructs the convex hull for each subset until no further subdivision is possible.

- **Merging:** Finally, it merges the convex hulls of the two subsets to form the convex hull of the entire set of points.
- **Efficiency:** QuickHull is efficient and has an average-case time complexity of $O(n\log n)$ and a worst-case time complexity of $O(n2)$. It is suitable for both small and large datasets.

Example

Consider an image:

Points: [(1, 2), (3, 5), (6, 4), (7, 2), (4, 1), (2, 1)]

Now, let's apply each of the convex hull algorithms mentioned:

Graham Scan:

It sorts the points based on their angles and constructs the convex hull.

Result: [(1, 2), (2, 1), (7, 2), (6, 4), (3, 5)]

Jarvis's March (Gift Wrapping):

It iteratively selects the next point with the smallest angle to enclose the points.

Result: [(1, 2), (2, 1), (7, 2), (6, 4), (3, 5)]

QuickHull:

It divides the points into subsets and recursively finds the convex hull.

Result: [(1, 2), (2, 1), (7, 2), (6, 4), (3, 5)]

6.7 REMOVING SMALL OBJECTS

In image processing, morphology is a collection of techniques used to analyze and manipulate the shapes of objects in an image. One common morphological operation is removing small objects, which helps clean up images by eliminating unwanted noise or insignificant details.

Here's how it works:

1. **Binary Image :** This technique typically works on binary images, where each pixel is either black (foreground, representing the object) or white (background).
2. **Structuring Element :** We use a small binary shape called a structuring element to probe the image. It's like a tiny stamp that we use to scan the image one pixel at a time. Common structuring elements include squares, circles, or diamonds.
3. **Erosion :** The core operation for removing small objects is often erosion. Imagine using sandpaper on the image (erosion) – it removes

pixels from the edges of objects. In this case, we erode the foreground objects (black pixels) in the image.

4. **Size Threshold :** We define a size threshold for objects. Objects with fewer pixels than the threshold after erosion are considered "small" and unwanted.

5. **Removing Small Objects :** There are two main approaches to remove small objects:

- **Keeping only the eroded image:** This approach discards the original image and keeps only the eroded version, where small objects have vanished.

- **Object Comparison:** We compare the original image with the eroded image. Pixels that were black in the original image but became white after erosion (because they belonged to small objects) are set to white in the original image, effectively removing them.

This process helps eliminate small specks, noise, or isolated dots that might not be relevant to the main objects in the image.

By understanding how removing small objects works in morphology, you can improve the quality of your images and prepare them for further analysis tasks in computer vision or image processing applications.

6.8 WHITE AND BLACK TOP- HATS

White top-hat and black top-hat are morphological operations used in image processing for highlighting specific types of structures or features within an image. Here's a brief overview of each:

- **White Top-Hat:**

The white top-hat operation is the difference between the original image and its opening.

Opening is an erosion followed by a dilation operation. It removes bright regions smaller than the structuring element.White top-hat highlights small bright structures or details in the image that are smaller than the structuring element used in the opening operation.It is useful for enhancing features such as small objects, bright spots, or textures against a relatively bright background.

- **Black Top-Hat:**

The black top-hat operation is the difference between the closing of the original image and the original image itself.Closing is a dilation followed by an erosion operation. It removes dark regions smaller than the structuring element.Black top-hat highlights small dark structures or details in the image that are smaller than the structuring element used in the closing operation.It is useful for enhancing features such as small dark objects, dark spots, or textures against a relatively dark background.

Both white top-hat and black top-hat operations are valuable tools in image processing for extracting and enhancing subtle features that may be difficult to discern in the original image. They are commonly used in tasks such as image enhancement, feature extraction, and texture analysis.

The equations for the white top-hat and black top-hat operations can be expressed as follows:

White Top-Hat:

$$WTH(A)=A-(A\circ B)$$

where:

A is the original image.

$\circ$ represents the opening operation.

B is the structuring element.

Black Top-Hat:

$$BTH(A)=(A\cdot B)-A$$

where:

A is the original image.

represents the closing operation.

B is the structuring element.

In these equations:

WTH(A) represents the white top-hat of image

BTH(A) represents the black top-hat of image

$A\circ B$ denotes the opening of image A using structuring element B.

$A\cdot B$ denotes the closing of image A using structuring element B.

Example

Suppose we have the following binary image A, where "1" represents the object and "0" represents the background:

$$A=\begin{bmatrix}0&0&0&0&0\\0&1&1&1&0\\0&1&1&1&0\\0&1&1&1&0\\0&0&0&0&0\end{bmatrix}$$

We'll use a simple 3x3 square structuring element B:

$$B = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Now, let's compute the white top-hat (WTH) and black top-hat (BTH) of the given binary image.

Sure, let's compute the white top-hat (WTH) and black top-hat (BTH) of the given binary image A.

Given the binary image A and the structuring element B:

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

We'll perform erosion and dilation operations to compute the white top-hat and black top-hat. Let's start by computing erosion A⊖B and dilation A⊕B.

Let's perform erosion and dilation on the given binary image A with the structuring element B:

Erosion (A ⊖ B):Erosion removes pixels from the boundary of objects in the image.

For each pixel, if all the pixels in the 3x3 neighborhood around it are 1 (object), the pixel remains 1; otherwise, it becomes 0.

Dilation (A ⊕ B):Dilation adds pixels to the boundary of objects in the image.

For each pixel, if at least one pixel in the 3x3 neighborhood around it is 1 (object), the pixel becomes 1; otherwise, it remains 0.

Let's apply erosion and dilation operations to the given binary image A:

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Erosion (A ⊖ B):

The result after erosion will remove one layer of pixels from the boundary of the objects in the image.

$$A \ominus B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Dilation (A ⊕ B):

The result after dilation will add one layer of pixels to the boundary of the objects in the image.

$$A \oplus B = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Now, let's compute the white top-hat and black top-hat using these results.

Given the results of erosion (A⊖B) and dilation (A⊕B), we can now compute the white top-hat (WTH) and black top-hat (BTH) of the image A.

White Top-Hat (WTH):

The white top-hat operation is obtained by subtracting the result of erosion (A⊖B) from the original image (A)

WTH(A)=A−(A⊖B)

Subtracting the erosion result from the original image:

$$\text{WTH}(A) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The white top-hat (WTH) of the image

A

A is the same as the original image since erosion did not remove any pixels.

Black Top-Hat (BTH):

The black top-hat operation is obtained by subtracting the original image (A) from the result of dilation (A⊕B)

BTH(A)=(A⊕B)−A

Subtracting the original image from the dilation result:

$$\mathrm{BTH}(A) = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

The black top-hat (BTH) of the image A highlights the boundary of the object, which was added by the dilation operation.

6.9 EXTRACTING THE BOUNDARY

Extracting the boundary of objects in an image using morphology involves using morphological operations to highlight the edges or contours of objects. Here's a general approach to extracting the boundary:

- **Erosion :** Perform erosion on the binary image using a structuring element.Erosion shrinks the objects in the image, effectively removing the object's boundary pixels.
- **Difference :** Compute the difference between the original binary image and the eroded image.This operation effectively isolates the boundary pixels of the objects, as they are the pixels that were removed by erosion.
- **Output :** The resulting binary image will contain only the boundary pixels of the objects, highlighting their contours or edges.This approach effectively extracts the boundary of objects in the image, allowing for further analysis or processing focused on object edges or contours.

The boundary is expressed as:

Boundary(A)=A−($A⊖B$)

Where:

- Arepresents the original binary image.
- Brepresents the structuring element used for erosion.
- Boundary(A)represents the resulting binary image containing only the boundary pixels of the objects.

Example

Consider binary image A, where "1" represents the object and "0" represents the background:

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Let's use a simple 3x3 square structuring element B for erosion:

$$B = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Now, let's compute the boundary of objects in this image A using morphology.

To extract the boundary of objects in the binary image A, we'll follow these steps:

1. **Erosion ($A \ominus B$):**

- Perform erosion on the original binary image Ausing the structuring element B.

2. **Difference:**

- Compute the difference between the original image $A$and the eroded image $A \ominus B$

Let's apply these steps to our example:

1. Erosion ($A \ominus B$):

- Erosion removes pixels from the boundary of objects in the image. For each pixel, if all the pixels in the 3x3 neighbourhood around it are 1 (object), the pixel remains 1; otherwise, it becomes 0.

$$A \ominus B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

2. **Difference:**

- Compute the difference between the original image A and the eroded image $A \ominus B$ to obtain the boundary.

$$\text{Boundary}(A) = A - (A \ominus B)$$

$$= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

6.10 GRAYSCALE OPERATIONS

Grayscale morphology operations are used for processing grayscale images, where each pixel represents a shade of gray. These operations involve applying morphological operations to grayscale images to achieve various tasks such as smoothing, edge detection, and feature extraction. Here are some common grayscale morphology operations:

- **Erosion:**

In grayscale erosion, the minimum pixel value within the neighborhood defined by the structuring element is assigned to the center pixel.Erosion reduces the intensity of bright regions and is useful for removing small bright spots or thinning object boundaries.

- **Dilation:**

In grayscale dilation, the maximum pixel value within the neighborhood defined by the structuring element is assigned to the center pixel.Dilation increases the intensity of bright regions and is useful for filling gaps or thickening object boundaries.

- **Opening:**

Opening is the combination of erosion followed by dilation.It helps in removing small bright regions while preserving the larger structures and is useful for smoothing or filtering images.

- **Closing:**

Closing is the combination of dilation followed by erosion.It helps in filling small dark gaps or holes while preserving the larger structures and is useful for image restoration or enhancing object boundaries.

- **Gradient:**

The gradient of a grayscale image is computed as the difference between dilation and erosion.It highlights the boundaries or edges of objects in the image and is useful for edge detection or feature extraction.

- **Top-Hat Transform:**

The top-hat transform is the difference between the original image and its opening.It extracts small-scale features or details from the image and is useful for image enhancement or background subtraction.

These grayscale morphology operations are fundamental tools in image processing for analyzing and manipulating grayscale images to extract meaningful information or enhance image quality.

Some common grayscale morphology operations:

1. Grayscale Erosion: $(A \ominus B)(x,y)=\min_{(i,j)\in B}\{A(x+i,y+j)\}$
2. Grayscale Dilation: $(A \oplus B)(x,y)=\max_{(i,j)\in B}\{A(x-i,y-j)\}$
3. Grayscale Opening: $A \circ B=(A \ominus B) \oplus B$
4. Grayscale Closing: $A \cdot B=(A \oplus B) \ominus B$
5. Grayscale Gradient:Gradient$(A)=(A \oplus B)-(A \ominus B)$

In these formulas:

- A represents the grayscale image.
- B represents the structuring element.
- $(A \ominus B)(x,y)$denotes the erosion of image A at pixel (x,y).
- $(A \oplus B)(x,y)$denotes the dilation of image A at pixel (x,y).
- $A \circ B$ represents the grayscale opening of image A.
- $A \cdot B$ represents the grayscale closing of image A.
- Gradient(A) represents the gradient of image A.

6.11 SUMMARY

Morphological operations are fundamental techniques in image processing, each serving distinct purposes. Erosion and dilation are basic operations for shrinking and expanding object boundaries, while opening and closing are compound operations used for noise reduction and gap filling, respectively. Hit-or-Miss transformation facilitates pattern matching, and skeletonizing reduces object thickness to topological skeletons. Computing the convex hull encloses objects within the smallest convex shape, while removing small objects enhances segmentation results. White and black top-hats highlight specific features, boundary extraction isolates object contours, and grayscale operations are essential for processing grayscale images. These operations find extensive application in various fields like image analysis, pattern recognition, and feature extraction, offering a powerful toolkit for image enhancement and understanding.

6.12 REFERENCE FOR FURTHER READING

- Gonzalez, R. C., & Woods, R. E. (2008). Digital Image Processing (3rd ed.). Prentice Hall.
- Serra, J. (1982). Image Analysis and Mathematical Morphology. Academic Press.

- Soille, P. (2003). Morphological Image Analysis: Principles and Applications. Springer.
- Haralick, R. M., & Shapiro, L. G. (1992). Computer and Robot Vision: Volume I. Addison-Wesley.
- Dougherty, G. (2003). Digital Image Processing for Medical Applications. Cambridge University Press.

Web references

- OpenCV Documentation: https://docs.opencv.org/
- Scikit-image Documentation: https://scikit-image.org/docs/stable/
- MATLAB Documentation - Image Processing Toolbox: https://www.mathworks.com/help/images/index.html

❖❖❖❖

7

ADVANCED IMAGE PROCESSING OPERATIONS

Unit Structure :

7.0 OBJECTIVES

- To get familiar with various advance image processing operations
- To study the analogy between feature detector and descriptors
- To gain detail insights about the principal components along with different detectors associated with image processing techniques

7.1 INTRODUCTION

In the vast realm of computer vision, unlocking the intricacies of visual data is paramount for machines to perceive and comprehend the world around them. Central to this endeavor is the extraction and description of image features, a fundamental process that underpins numerous applications ranging from object recognition to image matching and beyond. This chapter delves into the intricacies of this critical domain, exploring various techniques and methodologies employed in the extraction and characterization of image features.

From fundamental concepts to advanced techniques, each section sheds light on a specific aspect of this multifaceted domain, offering insights and practical knowledge to readers keen on mastering the art of image analysis. Additionally, this chapter also explores the role of feature descriptors in capturing the salient characteristics of image boundaries, paving the way for robust and efficient feature representation.

This chapter elucidates the principles of PCA and its applications in extracting informative features from high-dimensional data. Through insightful discussions and illustrative examples, readers gain a deeper understanding of how PCA facilitates the extraction of essential visual features while mitigating the curse of dimensionality.

7.2 EXTRACTING IMAGE FEATURES AND DESCRIPTORS: FEATURE DETECTOR VERSUS DESCRIPTORS

Feature detectors and descriptors are essential components in computer vision and image processing tasks, particularly in tasks like object recognition, image matching, and tracking. While they are closely related, they serve different purposes in analyzing and understanding images.

1. Feature Detectors:

- Feature detectors are algorithms designed to identify distinctive points or regions in an image that are salient or informative. These points are often referred to as keypoints or interest points.
- They locate areas with unique characteristics such as corners, edges, or blobs that are likely to be recognizable across different views of an object or scene.
- Common feature detectors include Harris corner detector, FAST (Features from Accelerated Segment Test), and SIFT (Scale-Invariant Feature Transform).

2. Descriptors:

- Descriptors are complementary to feature detectors. Once keypoints are detected, descriptors are used to describe the local appearance of

these keypoints in a way that is invariant to changes in illumination, viewpoint, and scale.

- Descriptors encode information about the local image region surrounding a keypoint into a compact numerical vector.
- The purpose of descriptors is to enable matching of keypoints across different images or frames, facilitating tasks like object recognition, image alignment, and 3D reconstruction.
- Popular descriptors include SIFT descriptors, SURF (Speeded-Up Robust Features), and ORB (Oriented FAST and Rotated BRIEF).

In summary, feature detectors identify distinctive points or regions in images, while descriptors encode information about these keypoints for further processing such as matching and recognition. Together, they form the backbone of many computer vision applications, allowing machines to understand and interpret visual data.

7.3 BOUNDARY PROCESSING AND FEATURE DESCRIPTOR

Boundary processing and feature descriptors play crucial roles in extracting meaningful information from images, enabling machines to interpret and understand visual data. Let's dive into each of these topics:

Boundary Processing:

In the realm of image processing, boundaries delineate the transitions between different regions within an image. Detecting and analyzing these boundaries are fundamental tasks as they often correspond to important structures or objects of interest. Boundary processing involves techniques aimed at enhancing, detecting, and representing these boundary regions.

- **Enhancement :** Boundary enhancement techniques aim to accentuate the edges or transitions between different regions within an image. Common methods include gradient-based approaches, such as the Sobel or Prewitt operators, which compute the gradient magnitude to highlight regions of rapid intensity change.
- **Detection :** Boundary detection algorithms identify and localize the boundaries within an image. Popular techniques include the Canny edge detector, which utilizes gradient information and non-maximum suppression to accurately locate edges while minimizing noise and spurious detections.
- **Representation :** Once boundaries are detected, they need to be represented in a suitable format for further processing. This may involve encoding boundary points into a chain code, polygonal approximation, or parametric representation such as line segments or curves.

Feature Descriptors:

Feature descriptors provide compact representations of local image regions around keypoints or interest points. These descriptors encode

distinctive characteristics of these regions, enabling robust matching and recognition across different images. Feature descriptors should possess several key properties:

- **Distinctiveness :** Descriptors should capture unique information about the local image region to facilitate reliable matching between keypoints.
- **Robustness :** Descriptors should be invariant or resilient to common image transformations such as changes in viewpoint, illumination, and scale. This ensures that the descriptors remain effective under varying conditions.
- **Compactness :** Descriptors should be compact, meaning they should encode relevant information concisely while minimizing computational overhead.
- **Efficiency :** Descriptors should be computationally efficient to allow for real-time processing in applications such as video analysis or robotics.

Some popular feature descriptors include:

- **SIFT (Scale-Invariant Feature Transform) :** SIFT descriptors capture information about gradient orientations in local image patches, providing robustness to scale and rotation changes.
- **SURF (Speeded-Up Robust Features) :** SURF descriptors utilize Haar wavelets to efficiently compute feature descriptors, offering scalability and robustness to image transformations.
- **ORB (Oriented FAST and Rotated BRIEF) :** ORB descriptors combine the FAST keypoint detector with the BRIEF descriptor, providing a fast and efficient solution for feature extraction and matching.

Here's a table summarizing the key differences:

Table 1: Key differences between Boundary Processing and Feature Descriptor

Aspect	Boundary Processing	Feature Descriptor
Focus	Edges and contours	Specific, localized regions
Process	Edge detection, boundary tracing	Extracting mathematical representation (gradients, histograms, etc.)
Applications	Object segmentation, shape analysis, object recognition	Object recognition, image retrieval, image registration
Relationship	Can provide starting points for feature descriptors	Not limited to boundaries

In summary, boundary processing and feature descriptors are essential components in image analysis and computer vision tasks. Boundary

processing facilitates the detection and representation of image boundaries, while feature descriptors encode distinctive information about local image regions, enabling robust image matching and recognition. Together, these techniques form the foundation for various applications in fields such as object recognition, image registration, and scene understanding.

7.4 PRINCIPAL COMPONENTS

Principal Component Analysis (PCA) is a powerful statistical technique used for dimensionality reduction and data visualization. It's particularly useful in fields like computer vision for feature extraction and data compression. PCA is a dimensionality reduction technique that simplifies data by reducing the number of features while retaining most of the information. It does this by identifying the underlying patterns in the data and creating new features, called principal components, that capture the most significant variations in the data.These new features are uncorrelated, meaning they don't contain redundant information. This makes them easier to analyze and interpret, especially for machine learning algorithms.

Here's a simple analogy: imagine you have a dataset of customer data, including features like age, income, and spending habits. These features might be correlated, meaning that customers with higher incomes tend to spend more. PCA can identify this underlying pattern and create a new feature, "purchasing power," that combines age and income into a single variable. This new feature would capture the most significant variation in spending habits, making it easier to analyze customer behavior.

7.4.1 Introduction to Principal Component Analysis (PCA):

Principal Component Analysis (PCA) is a mathematical technique used to reduce the dimensionality of data while preserving its essential structure. It accomplishes this by transforming the original data into a new coordinate system, where the axes (principal components) are orthogonal and ordered by the amount of variance they explain. This transformation enables the most significant variations in the data to be captured by a smaller number of components, facilitating easier visualization, analysis, and interpretation of high-dimensional datasets.

7.4.2 Key Concepts:

1. Principal Components : Principal components are the new orthogonal axes obtained after the transformation. These components are ordered by the amount of variance they explain in the original data. The first principal component captures the most significant variance, followed by subsequent components in decreasing order of importance.

2. Dimensionality Reduction : PCA reduces the dimensionality of data by projecting it onto a lower-dimensional subspace defined by the principal components. This reduction simplifies the data while retaining as much of the original variability as possible.

3. Variance Maximization : PCA seeks to maximize the variance of the data along each principal component. By capturing the directions of maximum variance, PCA ensures that the transformed data retains as much information as possible from the original dataset.

4. Eigenvalues and Eigenvectors : PCA is based on the eigende composition of the covariance matrix of the original data. The eigenvalues represent the amount of variance explained by each principal component, while the eigenvectors represent the directions (or axes) of maximum variance.

5. Projection : After determining the principal components, PCA projects the original data onto these components to obtain the transformed dataset. This projection preserves the most significant information in the data while reducing its dimensionality.

7.4.3 PCA Example

Let's say we have a data set of dimension 300 (n) × 50 (p). n represents the number of observations, and p represents the number of predictors. Since we have a large p = 50, there can be p(p-1)/2 scatter plots, i.e., more than 1000 plots possible to analyze the variable relationship. Wouldn't it be a tedious job to perform exploratory analysis on this data?

In this case, it would be a lucid approach to select a subset of p (p << 50) predictor which captures so much information, followed by plotting the observation in the resultant low-dimensional space.

The image below shows the transformation of high-dimensional data (3 dimension) to low-dimensional data (2 dimension) using PCA. Not to forget, each resultant dimension is a linear combination of p features.

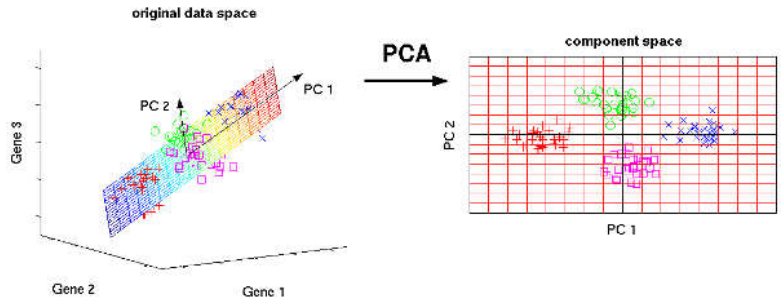


Figure 1: Transformation of high-dimensional data (3D) to low-dimensional data (2D)

7.4.4 What Are Principal Components?

A principal component (PCA) is a normalized linear combination of the original features in a data set. In the image above, PC1 and PC2 are the principal components. Let's say we have a set of predictors as X^1, X^2...,X^p

The principal component can be written as:

$Z^1 = \Phi^{11}X^1 + \Phi^{21}X^2 + \Phi^{31}X^3 + ... + \Phi^{p1}X^p$

where,

- Z^1 is the first principal component
- Φ^{p1} is the loading vector comprising loadings (Φ^1, Φ^2..) of the first principal component. The loadings are constrained to a sum of squares equals to 1. This is because a large magnitude of loadings may lead to a large variance. It also defines the direction of the principal component (Z^1), along which data varies the most. It results in a line in p dimensional space, which is closest to the n observations. Closeness is measured using average squared euclidean distance.
- $X^1..X^p$ are normalized predictors. Normalized predictors have mean values equal to zero and standard deviations equal to one.

First Principal Component:

The first principal component is a linear combination of original predictor variables that captures the data set's maximum variance. It determines the direction of highest variability in the data. Larger the variability captured in the first component, larger the information captured by component. No other component can have variability higher than first principal component.

The first principal component results in a line that is closest to the data, i.e., it minimizes the sum of squared distance between a data point and the line.

Similarly, we can compute the second principal component also.

Second Principal Component (Z^2):

The second principal component is also a linear combination of original predictors, which captures the remaining variance in the data set and is uncorrelated with Z^1. In other words, the correlation between first and second components should be zero. It can be represented as:

$Z^2 = \Phi^{12}X^1 + \Phi^{22}X^2 + \Phi^{32}X^3 + ... + \Phi^{p2}X^p$

If the two components are uncorrelated, their directions should be orthogonal (image below). This image is based on simulated data with 2 predictors. Notice the direction of the components; as expected, they are orthogonal. This suggests the correlation b/w these components are zero.

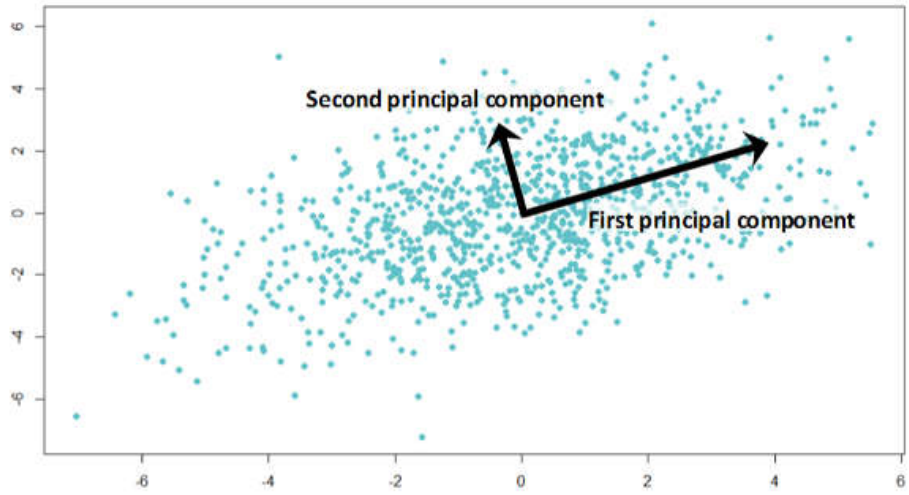


Figure 2: Two PCA components possessing orthogonality

All succeeding principal component follows a similar concept, i.e., they capture the remaining variation without being correlated with the previous component. In general, for n × p dimensional data, min(n-1, p) principal component can be constructed.

The directions of these components are identified unsupervised; i.e., the response variable(Y) is not used to determine the component direction. Therefore, it is an unsupervised approach.

7.4.5 How Principal Component Analysis (PCA) work?

1. Standardize the Data:

- If the features of your dataset are on different scales, it's essential to standardize them (subtract the mean and divide by the standard deviation).

2. Compute the Covariance Matrix:

- Calculate the covariance matrix for the standardized dataset.

3. Compute Eigenvectors and Eigenvalues:

- Find the eigenvectors and eigenvalues of the covariance matrix. The eigenvectors represent the directions of maximum variance, and the corresponding eigenvalues indicate the magnitude of variance along those directions.

4. Sort Eigenvectors by Eigenvalues:

- Sort the eigenvectors based on their corresponding eigenvalues in descending order.

5. Choose Principal Components:

- Select the top k eigenvectors (principal components) where k is the desired dimensionality of the reduced dataset.

6. Transform the Data:

- Multiply the original standardized data by the selected principal components to obtain the new, lower-dimensional representation of the data.

7.4.6 Applications of PCA

1. **Dimensionality Reduction :** PCA is widely used to reduce the dimensionality of high-dimensional datasets, making them more manageable for analysis and visualization without losing critical information.

2. **Feature Extraction :** In computer vision and pattern recognition, PCA can be used to extract relevant features from image datasets, allowing for more efficient and effective image analysis tasks such as object recognition and classification.

3. **Data Compression :** PCA can be employed for data compression by representing the original data using a smaller number of principal components. This compression reduces storage requirements and speeds up computational processes while preserving the essential structure of the data.

4. **Noise Reduction :** PCA can help remove noise or irrelevant variability from datasets by focusing on the principal components that capture the most significant sources of variation, thus enhancing the signal-to-noise ratio.

In summary, Principal Component Analysis (PCA) is a versatile technique with applications across various domains, including computer vision, data analysis, and machine learning. By transforming high-dimensional data into a lower-dimensional space while preserving essential information, PCA enables more efficient analysis, visualization, and interpretation of complex datasets.

7.5 HARRIS CORNER DETECTOR

The Harris Corner Detector is a popular method in computer vision used for detecting corners in an image. It was introduced by Chris Harris and Mike Stephens in 1988. The basic idea behind the Harris Corner Detector is to identify points in an image where there are significant variations in intensity in all directions. These points are typically found at corners, junctions, and edges.

Here's a brief overview of how it works:

1. **Gradient Calculation :** The first step involves computing the gradient of the image. This is typically done using a gradient operator like the Sobel operator.

2. **Structure Tensor :** For each pixel in the image, a structure tensor is calculated. The structure tensor represents the local image structure around the pixel. It is essentially a covariance matrix of the image gradients within a local neighborhood of the pixel.

3. **Corner Response Function :** The corner response function is computed using the eigenvalues of the structure tensor. This function measures the likelihood of a point being a corner. High values of the corner response function indicate strong corners.

4. **Thresholding and Non-maximum Suppression :** After computing the corner response function, a threshold is applied to identify strong corners. Additionally, non-maximum suppression is often performed to select only the local maxima as corner points.

We will now discuss about a corner detection method. Fig. 2 depicts the concept of the Harris-Stephens (HS) corner detector. The fundamental strategy is as follows: To find the corners of an image, drag a tiny window over it. It is intended for the detector window to compute variations in intensity. There are three situations which fascinate us: The conditions that occur when the window spans a boundary between two regions, as in location B, are (1) areas of zero (or small) intensity changes in all directions; (2) areas of changes in one direction but no (or small) changes in the orthogonal direction; and (3) areas of significant changes in all directions. Location C is an example of an area that contains a corner or isolated points. A mathematical model that aims to distinguish between these three conditions is called the HS corner detector.

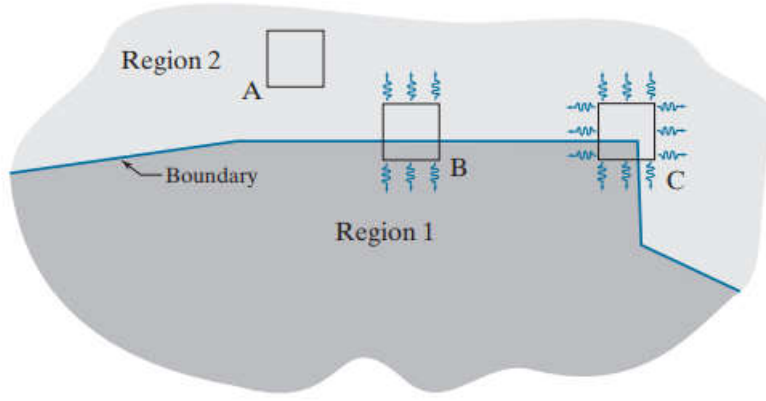


Figure 2: Working of Harris-Stephens corner detector

An image is represented by f, and a patch of the image defined by the values of (s,t) is represented by f (s,t). f (s+x, t+y) gives the patch of the

same size that has been shifted by (x, y). Next, the two patches' weighted total of squared differences is obtained as follows:

$$C(x,y) = \sum_s \sum_t w(s,t)\left[f(s+x,t+y) - f(s,t)\right]^2$$

where w(s,t) = weighting function

One can approximate the shifted patch using the linear terms in a Taylor expansion as given below

$$f(s+x,t+y) \approx f(s,t) + xf_x(s,t) + yf_y(s,t)$$

where $f_x(s,t) = \partial f/\partial x$ and $f_y(s,t) = \partial f/\partial y$, both evaluated at (s,t).

We can rewrite the equation as

$$C(x,y) = \sum_s \sum_t w(s,t)\left[xf_x(s,t) + yf_y(s,t)\right]^2$$

This equation can written in matrix form as

$$C(x,y) = \begin{bmatrix} x & y \end{bmatrix} \mathbf{M} \begin{bmatrix} x \\ y \end{bmatrix}$$

where

$$\mathbf{M} = \sum_s \sum_t w(s,t)\mathbf{A}$$

and

$$\mathbf{A} = \begin{bmatrix} f_x^2 & f_x f_y \\ f_x f_y & f_y^2 \end{bmatrix}$$

The Harris matrix is an alternative designation for Matrix M. It is acknowledged that the terms are assessed at (s,t). M is symmetric if and only if A is symmetric if w(s, t) is isotropic. The weighting function w(s, t) utilized in the HS detector often has one of two forms: either it is an exponential function of the form or it is 1 inside the patch and 0 outside (i.e., it resembles a box lowpass filter kernel).

$$w(s,t) = e^{-(s^2+t^2)/2\sigma^2}$$

The regions A, B, and C in Fig. 2 are represented by the little imagine patches in Figs. 3(a) through (c).In Fig. 3 (d), we show values of (f_x,f_y) computed using the derivative kernels $w_y = -[101]$ and $w_x = w_y^T$. . Figure (e) shows the derivatives of the patch containing the edge. Here, the spread is greater along the x-axis, and about nearly the same as Fig. a in

the y-axis. The derivatives of the patch enclosing the corner are displayed in Fig. (f). In this case, the data is dispersed in both directions, producing two huge eigenvalues and a significantly bigger fitting ellipse that is almost circular.

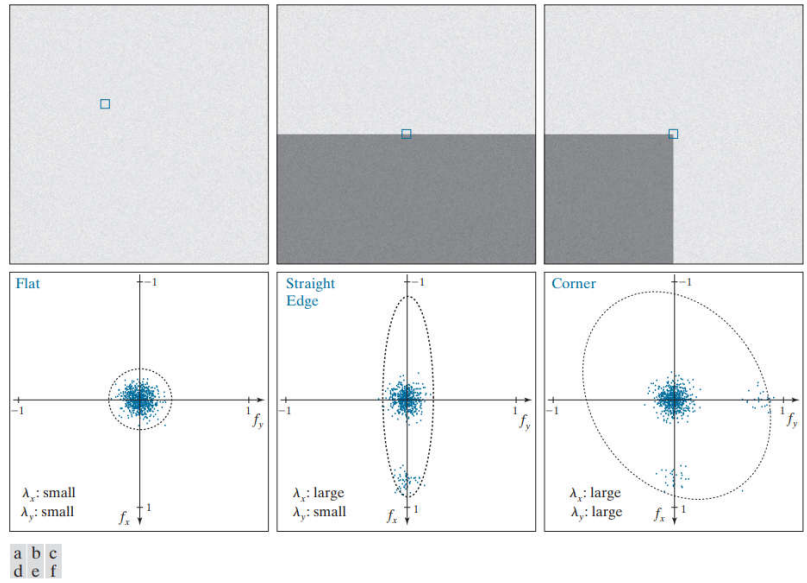


Figure 3

Because a square matrix's determinant equals the product of its eigenvalues and its trace equals the sum of its eigenvalues, the HS detector uses a measure of corner response. The definition of the measure is

$$R = \lambda_x \lambda_y - k(\lambda_x + \lambda_y)^2$$
$$= \det(\mathbf{M}) - k\,\mathrm{trace}^2(\mathbf{M})$$

where the constant is k. When both eigenvalues are large, Measure R indicates the presence of a corner; when one eigenvalue is large and the other small, it indicates the presence of an edge; and when both eigenvalues are small, Measure R's absolute value is small, indicating that the image patch under consideration is flat.

The Harris Corner Detector has been widely used in various applications such as feature detection, image alignment, object recognition, and tracking. However, it's worth noting that it has some limitations, such as sensitivity to noise and variations in lighting conditions. As a result, it's often used in combination with other techniques for more robust feature detection.

7.6 BLOB DETECTOR

A blob detector is a computer vision algorithm used to detect regions in an image that have similar intensity or color. Unlike corner detectors, which focus on detecting sharp changes in intensity, blob detectors are designed to identify regions that are uniform and stand out from their surroundings.

What is a Blob?

A region of interest or a connected component in an image or video that shares some common properties, such color, texture, intensity, or form, is referred to as a blob.

A portion of an image that appears to differ in intensity or color from its surroundings is referred to as a "blob" in image processing. Blobs can represent a wide range of properties, including edges, corners, and objects. They can also appear in different sizes and shapes. Numerous characteristics, including area, perimeter, centroid, bounding box, orientation, circularity, convexity, and eccentricity, can be assigned to blobs. The blobs can be distinguished from other visual components and described and characterized using these attributes.

What is Blob detection?

The technique of locating and detecting blobs in an image is called blob detection. Finding areas of an image that differ noticeably from their surrounds and indicate a major feature or object is the aim of blob detection.The Difference of Gaussian (DoG) approach, Laplacian of Gaussian (LoG) filtering, and thresholding are some of the methods available for blob detection. In order to find areas of a picture that have high contrast or intensity in relation to their surrounds, these techniques convolve the image using different filters. Blobs can be utilized for a variety of activities, including item identification, tracking, and recognition, once they have been identified.Blobs, for instance, can be used to represent particular characteristics of an object, such edges or corners, in object recognition. These features can then be compared with features in other photos to identify the object.

Numerous uses for OpenCV's blob detection can be found in image processing, robotics, and computer vision. Object recognition, gesture recognition, facial recognition, tracking moving objects, region of interest identification, anomaly or event detection are a few examples. Among other domains, blob detection finds application in satellite imaging, surveillance systems, and biomedical imaging.

Necessity of Blob Detection

In OpenCV, blob detection is required for a number of reasons, including:

- Blob detection is a technique that aids in the identification of items inside an image. We can distinguish objects from the background and ascertain their size, shape, and location within the image by identifying and localizing blobs.
- **Feature Extraction :** To extract features from an image, use blob detection. items can be categorized or matched to items in other photos using these attributes.
- **Tracking :** Blob detection facilitates tracking an object's motion over time. The ability to track and identify blobs allows us to ascertain an

object's direction and speed, which is helpful for robotics and autonomous driving applications.

- **Segmentation :** An image can be divided into distinct areas according to their texture or color using blob detection. Finding areas of interest in an image and isolating them from the backdrop are two benefits of segmentation.

7.7 HISTOGRAM OF ORIENTED GRADIENTS

Histogram of Oriented Gradients (HOG) is a feature descriptor used in computer vision and image processing for object detection and recognition. It's particularly popular in tasks like pedestrian detection, face detection, and gesture recognition.

Here's a simplified explanation of how HOG works:

1. **Gradient Calculation :** HOG works by first calculating the gradient (intensity gradient) of pixel values in the image. Typically, this is done by applying Sobel filters in both the horizontal and vertical directions to compute the gradient magnitude and direction.

2. **Gradient Orientation Binning :** The image is divided into small connected regions called cells. For each pixel in each cell, the gradient orientation is calculated, usually quantized into discrete bins. This step essentially quantizes the gradient direction into a finite set of possible orientations.

3. **Histogram Formation :** For each cell, a histogram of gradient orientations is constructed. Each pixel contributes to the histogram of the cell based on the orientation of its gradient. Usually, the histogram bins correspond to a range of gradient orientations.

4. **Block Normalization :** To enhance the invariance to changes in illumination and contrast, neighboring cells are grouped into larger blocks. Normalization is applied to the histograms within each block. This normalization can be L1-norm, L2-norm, or block normalization techniques like Block-wise Contrast Normalization (BCN) or Mutual Orientation Histograms (MOH).

5. **Descriptor Formation :** Finally, the normalized histograms from all the blocks are concatenated to form the final feature vector, which represents the image. This feature vector is then used for various tasks such as classification or object detection.

HOG's popularity stems from its:

- **Simplicity**: The underlying calculations are efficient and easy to implement.

- **Effectiveness**: It effectively captures local object shapes through gradients, making it useful for various detection tasks.

However, it's important to consider HOG's limitations:

- **Complexity**: While good for basic shapes, HOG might struggle with objects undergoing significant pose variations or complex appearances.

HOG has been widely used due to its simplicity and effectiveness, especially when combined with machine learning algorithms like Support Vector Machines (SVMs) or neural networks. However, it's worth noting that HOG alone might not be robust to complex variations in appearance and pose, so it's often used as part of a larger pipeline in modern computer vision systems.

7.8 SCALE-INVARIANT FEATURE TRANSFORMS

The Scale-Invariant Feature Transform (SIFT) is a widely used algorithm in computer vision for detecting and describing local features in images. Developed by David Lowe in 1999, SIFT is renowned for its robustness to changes in scale, rotation, illumination, and viewpoint, making it suitable for various tasks such as object recognition, image stitching, and 3D reconstruction.

Here's a detailed explanation of how SIFT works:

1. Scale-space Extrema Detection:

- SIFT operates on multiple scales of an image to detect features that are invariant to scale changes. This is achieved by constructing a scale-space representation of the image, which involves creating a series of blurred and downsampled versions of the original image.

- Gaussian kernels with different standard deviations (representing different scales) are convolved with the image to create a scale-space pyramid.

- At each scale level of the pyramid, potential keypoints are detected by finding local extrema in the scale-space images. These extrema represent regions of high contrast compared to their surrounding neighborhoods across multiple scales.

2. Keypoint Localization:

- Once potential keypoints are identified, SIFT performs precise localization to accurately determine their positions. This involves fitting a 3D quadratic function to the intensity values in the vicinity of each keypoint candidate.

- The localization process ensures that keypoints are accurately located at sub-pixel precision and eliminates unstable keypoints that are poorly localized.

3. Orientation Assignment:

- To achieve rotation invariance, each keypoint is assigned a dominant orientation based on local image gradient directions.
- A gradient orientation histogram is computed in a region around the keypoint, with orientations binned into discrete bins (e.g., 36 bins covering a full circle).
- The peak(s) in the histogram correspond to the dominant orientation(s) of the local image gradients, and the keypoint is assigned one or more dominant orientations based on these peaks.

4. Descriptor Generation:

- For each keypoint, a descriptor is generated to capture its local appearance information.
- A window around the keypoint is subdivided into smaller regions or bins.
- Within each bin, gradient orientations and magnitudes are computed, and histogram bins are populated based on these values.
- The resulting histograms across all bins are concatenated to form the keypoint descriptor, which encodes information about the local image gradients in the keypoint's neighborhood.
- The descriptor is designed to be highly distinctive and invariant to changes in illumination, viewpoint, and partial occlusion.

5. Keypoint Matching:

- Once descriptors are computed for keypoints in multiple images, keypoint matching is performed to establish correspondences between keypoints in different images.
- Typically, a nearest neighbor approach is used, where descriptors from one image are matched to the closest descriptors in another image based on a distance metric (e.g., Euclidean distance or cosine similarity).
- Additionally, techniques like ratio test or thresholding are often employed to filter out ambiguous matches and improve robustness.

SIFT has been widely used in various computer vision applications for more than two decades due to its reliability and effectiveness. While it can be computationally intensive, especially when dealing with large-scale datasets, SIFT remains a fundamental tool in the field of computer vision.

Strengths of SIFT:

- **Scale and Rotation Invariance**: SIFT's key advantage is its ability to find and match features regardless of image scale or rotation.

- **Robustness**: It's relatively robust to illumination changes and some geometric distortions.

Limitations of SIFT:

- **Computational Cost**: Compared to HOG, SIFT is computationally more expensive due to the scale-space representation and descriptor generation.
- **Sensitivity to Complex Deformations**: While robust to some distortions, SIFT might struggle with highly deformed objects.

Overall, SIFT is a cornerstone technique in computer vision for feature matching and object recognition. Its ability to handle scale and rotation variations makes it valuable in tasks like image stitching, 3D modeling, and robot navigation

7.9 HAAR-LIKE FEATURES

Haar-like features are simple rectangular filters used in object detection and recognition tasks, particularly in the Viola-Jones object detection framework. These features are named after Alfred Haar, a mathematician who first introduced them in the early 20th century. They are primarily used for detecting variations in pixel intensity in localized regions of an image.

Here's a detailed explanation of Haar-like features and their role in object detection:

1. Definition:

- Haar-like features are rectangular regions within an image where the pixel values are summed up in a specific pattern. These features are defined by their shape, size, and position within the image.
- Each Haar-like feature consists of a set of rectangular regions, typically arranged in a specific layout. These regions can be either white (representing areas of higher intensity) or black (representing areas of lower intensity).

2. Types of Haar-like Features:

- There are several types of Haar-like features, each capturing different patterns of intensity variations:
- **Edge Features:** These features capture the contrast between adjacent regions of different intensities, such as the transition from light to dark or vice versa.
- **Line Features:** Line features capture longer horizontal or vertical edges within an image.

- **Four-rectangle Features:** These features divide a rectangular region into four smaller rectangles and compute the difference between the sum of pixel intensities in the two diagonally opposite rectangles.
- **Three-rectangle Features:** Similar to four-rectangle features but with three rectangles, often used for capturing diagonal edges or corners.

3. Integral Image:

- To efficiently compute Haar-like features across different scales and positions, an integral image is used.
- An integral image is a data structure that allows rapid calculation of the sum of pixel intensities within any rectangular area of an image.
- Each pixel in the integral image contains the sum of all pixel intensities above and to the left of it in the original image.

4. Feature Evaluation:

- To use Haar-like features for object detection, a classifier evaluates each feature to determine whether it is relevant for detecting objects of interest.
- During training, the classifier learns which Haar-like features are discriminative for distinguishing between positive (object) and negative (non-object) examples.
- This learning process is typically done using machine learning algorithms such as AdaBoost, which selects a subset of the most informative features and combines them into a strong classifier.

5. Object Detection:

- In the Viola-Jones object detection framework, Haar-like features are used in a cascade of classifiers to efficiently detect objects in an image.
- The cascade consists of multiple stages, each containing a set of weak classifiers based on Haar-like features.
- At each stage, the classifier evaluates a subset of features, and if the region does not match the object being detected, it is quickly rejected without further processing.
- Regions that pass all stages of the cascade are considered positive detections and are further refined using additional techniques.

Advantages of Haar-like features:

- **Simplicity**: They are easy to compute and understand, making them efficient for real-time applications.

- **Effectiveness**: They can effectively capture basic geometric properties relevant for object detection, particularly for objects with well-defined shapes like faces.

Disadvantages of Haar-like features:

- **Limited Features**: They struggle to capture complex object variations in pose, shape, or illumination.
- **High False Positives**: The reliance on simple thresholds can lead to a high number of false positives, where non-object regions are mistakenly identified as the target object.

Haar-like features provide a computationally efficient way to capture basic patterns of intensity variation in images, making them well-suited for real-time object detection applications. While they are less flexible than more sophisticated feature descriptors like SIFT or HOG, they remain a foundational component in many object detection systems, particularly in scenarios where computational resources are limited.

7.10 SUMMARY

In this chapter, we embark on a journey into the heart of image feature extraction and description, exploring the diverse landscape of methodologies and algorithms that enable machines to discern and interpret visual information. From fundamental concepts to advanced techniques, each section sheds light on a specific aspect of this multifaceted domain, offering insights and practical knowledge to readers keen on mastering the art of image analysis.

At the boundary between raw pixel data and meaningful visual representations lies the crucial step of boundary processing. This chapter also discussed about the intricacies of extracting features from image boundaries, unraveling the underlying principles and methodologies that guide this process. Additionally, it explores the role of feature descriptors in capturing the salient characteristics of image boundaries, paving the way for robust and efficient feature representation.We also studiedthe principles of PCA and its applications in extracting informative features from high-dimensional data.

Corner detection lies at the heart of many computer vision applications, enabling the identification of salient keypoints in images. We explored the Harris Corner Detector, a seminal algorithm that revolutionized the field of feature detection. From its mathematical foundations to practical implementation strategies, readers are equipped with the knowledge to harness the power of corner detection in various visual tasks. We discussed how the Histogram of Oriented Gradients (HOG) emerges as a powerful technique for capturing local image structure and texture along with a comprehensive overview of SIFT, from its underlying principles to practical implementation strategies. Through intuitive explanations and illustrative examples, readers discover the versatility of Haar-like features and their applications across diverse domains.

By delving into the depths of image feature extraction and description, this chapter equips readers with the knowledge and tools to unravel the mysteries of visual data, paving the way for groundbreaking advancements in the field of computer vision. Whether you're a seasoned practitioner or an aspiring enthusiast, this chapter serves as a guiding beacon in your quest to master the art of image analysis.

7.11 LIST OF REFERENCES

Text Books:

1. Digital Image Processing by Rafael Gonzalez & Richard Woods, Pearson; 4th edition, 2018.
2. Think DSP: Digital Signal Processing in Python by Allen Downey, O'Reilly Media; 1st edition (August 16, 2016).

Reference Books:

1. Understanding Digital Image Processing, VipinTyagi, CRC Press, 2018.
2. Digital Signal and Image Processing by Tamal Bose, John Wiley 2010.
3. Hands-On Image Processing with Python by SandipanDey,Packt Publishing, 2018.
4. Fundamentals of Digital Images Processing by A K Jain, Pearson, 2010.

7.12 UNIT END EXERCISES

1. Explain the difference between feature detector versus descriptors.
2. Explain in detail aboutboundary processing and feature descriptor.
3. Write a note onprincipal components.
4. What do you mean by Principal Component Analysis (PCA)?
5. Explain thekey concepts associated with PCA.
6. Give the example of PCA.
7. What Are Principal Components?
8. How Principal Component Analysis (PCA) work ?
9. State theapplications of PCA.
10. Write a note on Harris Corner Detector.
11. Explain Blob detector.
12. Write a note on Histogram of Oriented Gradients.
13. What are Scale-invariant feature transforms ?
14. Write a note on Haar-like features.

❖❖❖❖

8

IMAGE SEGMENTATION

Unit Structure :

8.0 OBJECTIVES

- To get insights of image segmentation
- To understand different techniques and algorithms associated with segmentation

8.1 INTRODUCTION

Image segmentation is a fundamental task in computer vision that involves partitioning an image into multiple segments or regions based on certain characteristics such as color, intensity, texture, or semantic meaning. The goal of image segmentation is to simplify and/or change the representation of an image into something more meaningful and easier to analyze.

Objective: The primary objective of image segmentation is to simplify the representation of an image into more meaningful and easy-to-analyze parts. It aims to partition an image into distinct regions or objects based on certain features or criteria.

Types of Segmentation:

- **Semantic Segmentation :** This type of segmentation assigns a class label to each pixel in the image, effectively dividing the image into regions corresponding to different objects or regions of interest.
- **Instance Segmentation :** Instance segmentation not only assigns class labels to pixels but also distinguishes between individual object instances. Each instance of an object is segmented separately.
- **Boundary or Edge Detection :** This approach focuses on detecting edges or boundaries between different regions in an image. It doesn't segment the whole image but rather highlights the boundaries.
- **Clustering-Based Segmentation :** It groups pixels based on their similarity in terms of color, intensity, or other features. Common clustering algorithms like k-means or mean-shift are often used.
- **Region Growing :** This method starts with seed points and grows regions by adding neighboring pixels that meet certain similarity criteria.

Challenges:

- **Noise and Variability :** Images may contain noise, variations in illumination, or other artifacts, making it challenging to accurately segment objects.
- **Object Occlusion :** Objects in images may be partially occluded by other objects, complicating the segmentation process.
- **Complex Object Shapes :** Objects in images may have irregular shapes, making it difficult to accurately delineate their boundaries.

Applications:

- **Medical Imaging :** Image segmentation is used for identifying and delineating structures or abnormalities in medical images such as MRI scans or X-rays.
- **Autonomous Vehicles :** Segmentation helps in detecting and recognizing objects in the environment for tasks like lane detection, pedestrian detection, and obstacle avoidance.
- **Satellite Imaging :** It aids in land cover classification, urban planning, and environmental monitoring by segmenting satellite images into different land cover types.
- **Object Recognition and Tracking :** Segmentation is an essential component in object recognition and tracking systems for identifying and tracking objects in video streams.

In essence, image segmentation is a crucial preprocessing step in many computer vision tasks, enabling the extraction of meaningful information from images for further analysis and decision-making.

8.2 HOUGH TRANSFORM FOR DETECTING LINES AND CIRCLES

The Hough Transform is a powerful technique in computer vision and image processing used to detect geometric shapes such as lines and circles in an image. It was originally developed by Paul Hough in the 1960s for detecting lines in binary images and later extended by Richard Duda and Peter Hart in 1972 to detect arbitrary shapes like circles. The Hough Transform is particularly robust to noise and occlusion and can detect shapes even when they are incomplete or broken.

Here's a detailed explanation of the Hough Transform for detecting lines and circles:

Hough Transform for Line Detection:

1. Parameter Space Representation :

- In the Hough Transform for lines, each pixel in the image space corresponds to a parameter space representation. Instead of representing lines as slopes and intercepts (as in Cartesian space), lines are represented as points in a parameter space known as the Hough space.

- For a line defined by $y = mx + c$, where m is the slope and c is the y-intercept, the Hough space is represented by a 2D array where one axis represents mand the other represents c.

2. Voting :

- For each edge pixel in the binary image (typically obtained through edge detection techniques like Canny edge detector), we compute the possible lines that could pass through that pixel in parameter space.

- Each edge pixel votes for the possible lines it could belong to in the Hough space. This is done by incrementing the corresponding cells in the Hough space for each possible line.

3. Accumulator Thresholding:

- After all edge pixels have voted, we examine the Hough space to identify cells with high vote counts. These cells correspond to lines in the original image.

- By thresholding the accumulator array, we select only those cells with a high enough count, indicating significant line detections.

4. Line Extraction:

- After thresholding, lines are extracted from the parameter space representation by mapping back to the Cartesian space using the parameters represented by the selected cells.
- Each selected cell corresponds to a line in the original image.

Hough Transform for Circle Detection:

1. Parameter Space Representation:

- In the Hough Transform for circles, circles in the image are represented by their center coordinates (a, b) and their radius r. Thus, each pixel in the image space corresponds to a parameter space representation.
- The Hough space for circle detection is typically three-dimensional, with axes representing a,b, and r.

2. Voting:

- Similar to line detection, for each edge pixel in the binary image, we compute the possible circles that could pass through that pixel in parameter space.
- Each edge pixel votes for the possible circles it could belong to in the Hough space by incrementing the corresponding cells in the accumulator array.

3. Accumulator Thresholding:

- After all edge pixels have voted, we examine the accumulator array to identify cells with high vote counts, indicating potential circle detections.
- By thresholding the accumulator array, we select only those cells with a high enough count, indicating significant circle detections.

4. Circle Extraction:

- After thresholding, circles are extracted from the parameter space representation by mapping back to the Cartesian space using the parameters represented by the selected cells.
- Each selected cell corresponds to a circle in the original image.

Summary

- The Hough Transform provides a robust method for detecting lines and circles in images, even in the presence of noise and occlusion.

- It operates by transforming the spatial domain representation of the image into a parameter space where geometric shapes are represented explicitly.
- Voting and accumulator thresholding are key steps in both line and circle detection variants of the Hough Transform.
- Despite its computational complexity, the Hough Transform remains widely used in computer vision applications for its effectiveness in shape detection and robustness to various image conditions.

8.3 THRESHOLDING AND OTSU'S SEGMENTATION

Thresholding is a basic yet powerful image processing technique used to separate objects or regions of interest from the background in an image. It works by converting a grayscale or color image into a binary image, where pixels are classified as either foreground (object) or background based on their intensity values relative to a specified threshold value.

Thresholding:

1. Simple Thresholding:

- In simple thresholding, a fixed threshold value is applied to each pixel in the image. If the intensity of a pixel exceeds the threshold, it is classified as foreground; otherwise, it is classified as background.
- Mathematically, if I(x, y) represents the intensity of the pixel at coordinates (x, y), and T represents the threshold value, then the binary result B(x, y) is given by:

$B(x, y) = 1$ ……if $I(x, y) > T$

$= 0$ ……otherwise

2. Adaptive Thresholding:

- Adaptive thresholding adjusts the threshold value for each pixel based on the local neighborhood of that pixel. This is useful when the illumination varies across the image.
- Instead of using a single global threshold, adaptive methods compute thresholds for smaller regions of the image.

3. Otsu's Thresholding:

- Otsu's method is an automatic thresholding technique that selects the optimal threshold value by maximizing the between-class variance of the pixel intensities.
- It assumes that the image contains two classes of pixels: background and foreground.

- Otsu's method works by iterating through all possible threshold values and selecting the one that minimizes the intra-class variance (variance within each class) or maximizes the inter-class variance (variance between classes).
- The algorithm computes the histogram of pixel intensities and calculates the cumulative probabilities and means for each intensity level.
- Then, it iterates through all possible threshold values and calculates the between-class variance for each threshold.
- The threshold value that maximizes the between-class variance is selected as the optimal threshold.

Applications:

- Thresholding is widely used in various image processing tasks such as object detection, image segmentation, and feature extraction.
- It is particularly useful for segmenting objects from the background in applications like document analysis, medical imaging, and quality control.

Otsu's Segmentation:

Steps:

- Compute the histogram of pixel intensities in the grayscale image.
- Normalize the histogram to obtain probabilities of each intensity level.
- Compute the cumulative probabilities and means for each intensity level.
- Iterate through all possible threshold values and calculate the between-class variance for each threshold.
- Select the threshold value that maximizes the between-class variance.
- Apply the selected threshold to the image to obtain the binary result.

the Otsu's method given by Nobuyuki Otsufor obtaining optimal thresholding. This is a variance-based method used to evaluate the least weighted variance between the foreground and background pixels. The essential factor is to measure the distribution of background and foreground pixels while iterating over all potential threshold values and then locating the threshold at which the dispersion is the smallest [C. Huang et.al., 37].

Algorithm

The algorithm repeatedly finds the threshold that reduces the variance belonging to the same class determined by the weighted sum of spread. Grayscale typically has hues between 0-255 (0-1 in case of float).

The following equation is utilized to calculate the variance at threshold t:

$$\sigma^2(t) = \omega_{bg}(t)\sigma^2_{bg}(t) + \omega_{fg}(t)\sigma^2_{fg}(t)$$

Where $\omega_{bg}(t)$ and $\omega_{fg}(t)$ represents the probability of pixels for value of t and σ2 represents the deviation of color values.

Let,

P_{all} :total pixel count,
$P_{BG}(t)$ and $P_{FG}(t)$: background and foreground pixels count at t,

So the updates are given by,

$$\omega_{bg}(t) = \frac{P_{BG}(t)}{P_{all}}$$

$$\omega_{fg}(t) = \frac{P_{FG}(t)}{P_{all}}$$

The variance is calculated using the below formula

$$\sigma^2(t) = \frac{\sum(x_i - \bar{x})^2}{N-1}$$

Where,

x_i and x bar: pixel value and its mean at i in the group (b_g or f_g)
N: number of pixels.

Advantages:

- Otsu's method automatically selects the threshold value without the need for manual tuning.
- It is robust to variations in illumination and background clutter.
- It works well for bimodal or nearly bimodal images where the foreground and background intensities are well-separated.

Limitations:

- Otsu's method assumes a bimodal distribution of pixel intensities, which may not hold true for all images.

- It may not perform well for images with complex intensity distributions or significant overlap between foreground and background intensities.

8.4 EDGE-BASED/REGION-BASED SEGMENTATION

Edge-based and region-based segmentation are two common approaches used in image processing to partition an image into meaningful regions or objects. Here's a detailed explanation of each approach with mathematical equations:

Edge-based Segmentation:

Edge-based segmentation relies on detecting significant changes in intensity or color, which often correspond to object boundaries or edges in the image.

1. Edge Detection:

- Edge detection is typically performed using techniques like the Sobel operator, Prewitt operator, or Canny edge detector. These operators compute the gradient magnitude of the image to highlight regions of rapid intensity change.

2. Gradient Magnitude:

- Let I(x, y) be the intensity of the pixel at coordinates(x, y). The gradient magnitude G(x, y) is computed as:

$G(x, y) = \text{sqrt}\{G_x(x, y)^2 + G_y(x, y)^2\}$

- Where $G_x(x, y)$and $G_y(x, y)$are the horizontal and vertical components of the gradient, respectively.

3. Thresholding:

- After edge detection, a thresholding operation is often applied to the gradient magnitude to obtain a binary edge map. Pixels with gradient magnitudes above a certain threshold are considered part of an edge.

4. Edge Linking and Refinement:

- Detected edges may not form continuous contours. Techniques such as edge linking (e.g., using the Hough transform) and edge refinement (e.g., applying morphological operations) can be employed to connect and smooth the detected edges.

Region-based Segmentation:

Region-based segmentation aims to partition an image into regions or objects based on certain criteria such as color similarity, texture, or intensity homogeneity.

1. Region Growing:

- Region growing algorithms start with seed points and iteratively grow regions by adding neighboring pixels that meet certain similarity criteria.
- One common criterion is intensity homogeneity, where pixels are added to a region if their intensity values are similar to those of the region's seed point.

2. Region Splitting and Merging:

- Region splitting divides an image into smaller regions based on certain criteria. For example, a region may be split if it contains significant intensity variations.
- Region merging combines adjacent regions that satisfy certain similarity criteria. This helps in merging regions that belong to the same object but were initially split due to noise or other factors.

Mathematical Equations for Region-based Segmentation:

1. Region Homogeneity Criterion:

- Let R_i represent a region in the image, and I(x, y)be the intensity of the pixel at coordinates (x, y) within the region. The region homogeneity criterion can be expressed as:

  $|I(x, y) - \mu_i| <= T$

- Where $\mu_i$is the mean intensity of region R_i, and T is a threshold representing the acceptable deviation from the mean.

2. Region Growing Algorithm:

- Given a seed point (x_s, y_s), the region growing algorithm iteratively adds neighboring pixels (x, y) to the region if they satisfy the homogeneity criterion.

3. Region Splitting and Merging Criteria:

- Region splitting may involve criteria such as local variance exceeding a threshold or the presence of significant edges within a region.
- Region merging criteria may include comparing the mean intensities of neighboring regions and merging regions with similar mean intensities.

Both edge-based and region-based segmentation have their strengths and weaknesses, and the choice between them depends on the specific characteristics of the image and the desired segmentation result. Edge-based methods are effective for detecting object boundaries, while region-based methods are useful for segmenting homogeneous regions. Often, a

combination of these techniques is employed for more robust segmentation results.

8.5 REGION GROWING

Region growing is a fundamental method for image segmentation, particularly in cases where objects of interest exhibit uniform characteristics such as intensity or color. This technique starts from seed points and iteratively adds neighboring pixels to form regions that meet certain homogeneity criteria. Here's a detailed explanation of region growing with mathematical equations:

Region Growing Algorithm:

1. Initialization:

- Select seed points within the image. These seed points can be chosen manually or automatically based on certain criteria.
- Let S represent the set of seed points.

2. Homogeneity Criterion:

- Define a homogeneity criterion that determines whether a pixel should be added to the region.
- Let I(x, y) represent the intensity (or color) of the pixel at coordinates (x, y).
- The homogeneity criterion typically involves comparing the intensity (or color) of neighboring pixels to the seed point or the current region.
- One common criterion is to check if the intensity of a pixel is within a certain threshold of the mean intensity of the region.

3. Region Growing Process:

- Start with the seed points in S.
- For each seed point (x_s, y_s) in S, initialize a region R containing only the seed point.
- Iterate through the neighboring pixels of each pixel in R (e.g., 4 or 8 connectivity).
- If a neighboring pixel (x, y)satisfies the homogeneity criterion, add it to the region R and mark it as visited.
- Continue this process until no more pixels can be added to the region.
- Repeat the above steps for each seed point in S.

Mathematical Equations:

1. Homogeneity Criterion:

- Let $\mu_R$represent the mean intensity (or color) of region R.
- The homogeneity criterion can be expressed as:

  $|I(x, y) - \mu_R| \leq T$

- Where T is a threshold representing the acceptable deviation from the mean.

2. Region Growing Process:

- Start with the seed points:

  S = {(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)}

- For each seed point (x_s, y_s) in S, initialize the region:

  R ={(x_s, y_s)

- Iterate through the neighboring pixels (x, y)of each pixel in R:
- If (x, y) satisfies the homogeneity criterion, add it to R:
- $R = R \cup \{(x,y)\}$

Advantages:

- Region growing is simple to implement and computationally efficient.
- It can effectively segment regions with homogeneous characteristics.
- It can handle irregular shapes and objects with varying sizes.

Limitations:

- The choice of seed points can significantly affect the segmentation result.
- Region growing may fail if the homogeneity criterion is not well-defined or if the image contains noise or artifacts.
- It may be sensitive to variations in illumination or shading.

In summary, region growing is a versatile technique for image segmentation, widely used in various applications such as medical image analysis, object recognition, and satellite image processing. By iteratively expanding regions based on predefined criteria, region growing can effectively partition images into meaningful regions or objects.

8.6 REGION SPLITTING AND MERGING

Region splitting and merging is another important technique for image segmentation, particularly useful when dealing with complex images containing objects with varying characteristics or cluttered backgrounds. This method involves dividing regions into smaller segments based on certain criteria (splitting) and then merging adjacent segments that satisfy specific similarity conditions (merging). Here's a detailed explanation of region splitting and merging along with mathematical equations:

Region Splitting:

1. Homogeneity Criterion for Splitting:

- Define a criterion to determine when a region should be split into smaller segments.
- Common criteria include variations in intensity, texture, or color within a region.
- For example, a region may be split if the local variance of pixel intensities exceeds a certain threshold.

2. Splitting Process:

- Start with the initial regions obtained through an initial segmentation method or seed points.
- For each region that satisfies the splitting criterion, divide it into smaller segments.

Mathematical Equations for Region Splitting:

1. Homogeneity Criterion for Splitting:

- Let R represent a region in the image, andI(x, y) be the intensity (or color) of the pixel at coordinates (x, y) within region R.
- Define a splitting criterion based on the local variance of pixel intensities:

If $\sigma R > T_{split}$, split region R

Where $\sigma R$is the standard deviation of pixel intensities within regionR, and $T_{split}$is a threshold representing the maximum allowable variance.

Region Merging:

1.Homogeneity Criterion for Merging:

- Define a criterion to determine when adjacent regions should be merged into a single region.

- Common criteria include comparing the mean intensity (or color) of neighboring regions and checking for color or intensity similarity.
- For example, two regions may be merged if their mean intensities are similar or if their color histograms exhibit significant overlap.

2. Merging Process:

- Start with the initial regions obtained through segmentation or splitting.
- For each pair of adjacent regions that satisfy the merging criterion, merge them into a single region.

Mathematical Equations for Region Merging:

1. Homogeneity Criterion for Merging:

- Let R_1 and R_2represent two adjacent regions in the image, with mean intensities μ_1 and

  μ_2, respectively.
- Define a merging criterion based on the difference in mean intensities:

If

$|\mu 1-\mu 2|<$Tmerge, merge regions R1 and R2

Where Tmergeis a threshold representing the maximum allowable difference in mean intensities.

Advantages:

- Region splitting and merging can handle complex images with varying characteristics.
- It allows for the segmentation of objects with irregular shapes and varying sizes.
- The method can be adapted to different types of images and segmentation tasks.

Limitations:

- The performance of region splitting and merging heavily depends on the choice of criteria and thresholds.
- The method can be computationally expensive, especially for large images or complex segmentation tasks.
- It may produce over-segmentation or under-segmentation if the criteria are not appropriately tuned.

8.7 WATERSHED ALGORITHM

The Watershed algorithm is a powerful method for image segmentation, particularly in scenarios where objects of interest are touching or overlapping. It views the grayscale image as a topographic surface, where pixel intensities represent elevations, and the goal is to partition the surface into catchment basins (regions) corresponding to distinct objects. Here's a detailed explanation of the Watershed algorithm along with mathematical equations and a diagram:

Watershed Algorithm:

1. Gradient Computation:

- Compute the gradient magnitude of the grayscale image to highlight regions of rapid intensity change, which often correspond to object boundaries.
- The gradient magnitude image represents the topographic surface where higher values correspond to steeper regions.

2. Marker Selection:

- Identify markers or seeds within the image that correspond to potential objects or regions of interest. These markers can be manually selected or obtained through other segmentation methods.
- Markers can be placed at local minima in the gradient magnitude image or based on user input.

3. Flood Fill Simulation:

- Simulate a flooding process on the topographic surface starting from the markers.
- The flooding process is analogous to filling basins with water, where each basin corresponds to a catchment area or region.

4. Boundary Identification:

- As the flooding process progresses, boundaries between adjacent basins are formed where the water from different basins meets.
- These boundaries represent the segmentation result, delineating the boundaries between different objects or regions.

Mathematical Equations:

1. Gradient Magnitude Calculation:

Let I(x, y) represent the intensity of the pixel at coordinates (x, y), and G(x, y) denote the gradient magnitude:

$G(x, y) = |\nabla I(x,y)|$

- Where $|\nabla I(x,y)|$represents the gradient vector.

2. Marker Selection:

Markers can be manually defined or obtained automatically. Let M represent the set of markers:

$M = \{m_1, m_2, ..., m_n\}$

Each marker m_i can be represented as a point in the image domain (x_i, y_i) along with an associated label.

3. Flood Fill Simulation:

Simulate the flooding process by propagating water from the markers throughout the image.

At each iteration, water flows from each marker to neighboring pixels with lower elevations (lower intensity values).

Pixels are labeled with the marker labels as the flooding process progresses.

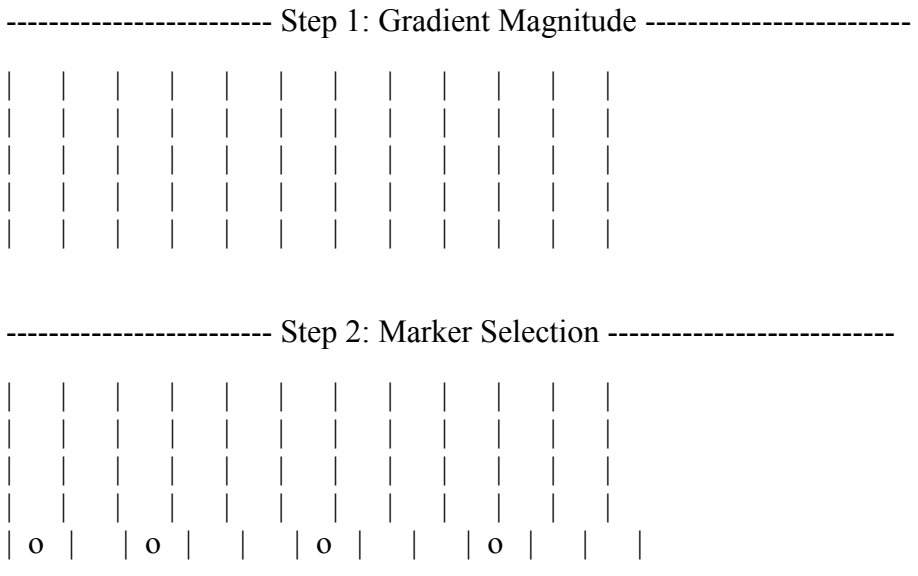
The process continues until all pixels are labeled.

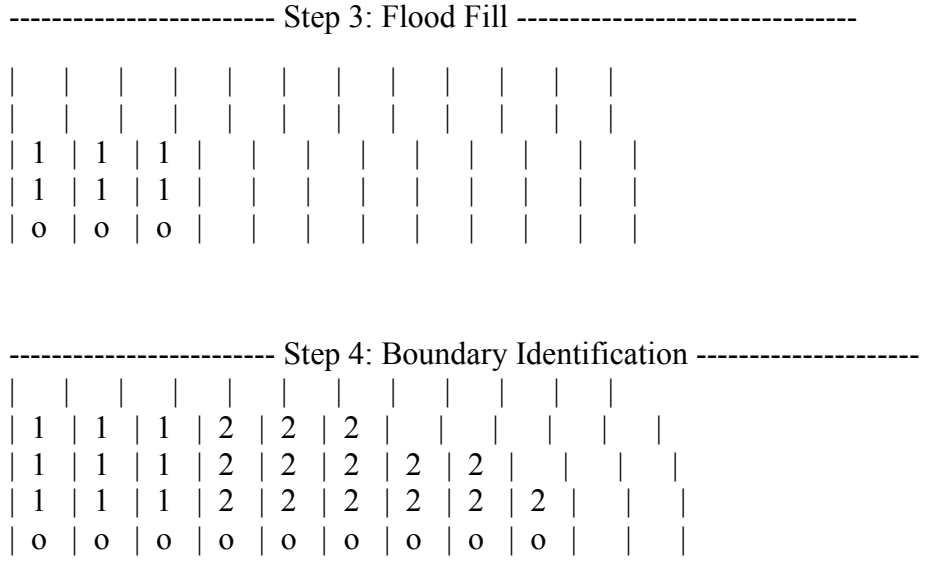
4. Boundary Identification:

Boundaries between adjacent regions are formed where water from different markers meets during the flooding process.

These boundaries can be represented as watershed lines or contours.

Diagram:





In the diagram:

- Step 1 illustrates the computation of the gradient magnitude.
- Step 2 shows the selection of markers.
- Step 3 depicts the flood fill simulation, where the labels propagate from the markers.
- Step 4 demonstrates the identification of boundaries between regions.

Advantages:

- The Watershed algorithm can segment complex images with overlapping objects.
- It does not require a priori knowledge of the number of objects in the image.
- It can handle objects with irregular shapes and sizes.

Limitations:

- The algorithm may produce over-segmentation, especially in areas with texture or noise.
- It can be sensitive to the choice of markers and parameters.
- Post-processing steps may be required to refine the segmentation result.

In summary, the Watershed algorithm is a powerful tool for image segmentation, particularly useful in scenarios where objects are touching or overlapping. By simulating a flooding process on the gradient magnitude image, the algorithm can partition the image into distinct regions corresponding to different objects or structures.

8.8 ACTIVE CONTOURS

Active Contours, also known as snakes, are curve-evolving algorithms used for image segmentation and object tracking. These algorithms aim to deform a curve iteratively until it aligns with the boundary of an object of interest in an image. Here's a detailed explanation of Active Contours along with mathematical equations:

Active Contours Algorithm:

1. Initialization:

- Initialize a curve (or contour) within the image domain, typically close to the boundary of the object to be segmented.
- The curve can be represented parametrically or implicitly, depending on the specific implementation.

2. Energy Minimization:

- Define an energy functional that quantifies the properties of the curve and its relationship with the image.
- The energy functional typically consists of two components:
- Internal Energy: Encodes the smoothness and regularity of the curve, preventing it from deforming excessively.
- External Energy: Measures the compatibility between the curve and the image, attracting the curve towards object boundaries or edges.

3. Curve Evolution:

- Deform the curve iteratively to minimize the energy functional.
- The curve evolution process involves updating the positions of the curve's control points (or parameters) based on the gradient descent or other optimization techniques.

4. Convergence:

- Iterate until the curve converges to the desired object boundary or until a termination criterion is met (e.g., maximum number of iterations, negligible change in energy).

Mathematical Equations:

1. Energy Functional:

- The energy functional E is typically defined as the sum of internal and external energies:

E=Einternal+α·Eexternal

Where αis a weighting parameter controlling the influence of the external energy.

2. Internal Energy:

- The internal energy Einternalmeasures the smoothness and regularity of the curve:

$$E_{\text{internal}} = \int_0^1 (\kappa(s) - \kappa_0(s))^2 \, ds$$

- Where κ(s)represents the curvature of the curve at arc length s, and κ0 (s)is a reference curvature.

3. External Energy:

- The external energy Eexternalmeasures the compatibility between the curve and the image:

$$E_{\text{external}} = -\int_0^1 W(s) \cdot |\nabla I(\mathbf{x}(s))|^2 \, ds$$

- Where W(s) is a weighting function that emphasizes image gradients near the curve, and ∇I(**x**(s))represents the image gradient at the curve's position.

4. Curve Evolution:

- The curve evolves according to the gradient descent of the energy functional:

$$\frac{\partial \mathbf{x}}{\partial t} = -\frac{\partial E}{\partial \mathbf{x}}$$

- Wherex represents the curve's control points (or parameters), and t is the evolution time.

Advantages:

- Active Contours can accurately segment objects with irregular shapes and complex boundaries.
- They are robust to noise and cluttered backgrounds.
- The method allows for interactive refinement and user guidance.

Limitations:

- Active Contours may struggle with concave or fragmented objects.
- They can be sensitive to the choice of parameters and initialization.

- The algorithm may require significant computational resources, especially for large images or complex objects.

In summary, Active Contours are versatile techniques for image segmentation, widely used in medical imaging, remote sensing, and computer vision applications. By minimizing an energy functional that balances curve smoothness and image compatibility, Active Contours can accuratelydelineate object boundaries and provide precise segmentation results.

8.9 MORPHOLOGICAL SNAKES

Morphological snakes, also known as morphological active contours or morphological geodesic active contours, are a variation of the active contour model that combines geometric deformations with morphological operations. These snakes are particularly useful for segmenting objects with irregular shapes and varying contrast levels in images. Here's a detailed explanation of morphological snakes along with mathematical equations.

Morphological Snakes Algorithm:

1. Initialization:

- Initialize a curve (or contour) within the image domain, typically close to the boundary of the object to be segmented.
- The curve can be represented parametrically or implicitly, similar to traditional active contours.

2. Energy Minimization:

- Define an energy functional that quantifies the properties of the curve and its relationship with the image, incorporating both geometric and morphological terms.
- The energy functional consists of three components:
- Internal Energy: Encodes the smoothness and regularity of the curve.
- External Energy: Measures the compatibility between the curve and the image, attracting the curve towards object boundaries or edges.
- Morphological Energy: Incorporates morphological operations to enhance the curve's response to image features.

3. Curve Evolution:

- Deform the curve iteratively to minimize the energy functional, similar to traditional active contours.
- However, morphological snakes employ morphological operations such as dilation and erosion to adjust the curve's shape and position.

4. Convergence:

- Iterate until the curve converges to the desired object boundary or until a termination criterion is met.

Mathematical Equations:

1. Energy Functional:

- The energy functional E is defined as the sum of internal, external, and morphological energies:

$E = E_{internal} + \alpha E_{external} + \beta E_{morphological}$

Where $\alpha$and $\beta$are weighting parameters controlling the influence of the external and morphological energies, respectively.

2. Internal Energy:

- The internal energy $E_{internal}$measures the smoothness and regularity of the curve, similar to traditional active contours.

3. External Energy:

- The external energy $E_{external}$measures the compatibility between the curve and the image, similar to traditional active contours.

4. Morphological Energy:

- The morphological energy $E_{morphological}$incorporates morphological operations to enhance the curve's response to image features.
- This energy term can be defined based on morphological operations such as dilation, erosion, opening, or closing applied to the image and the curve.

Advantages:

- Morphological snakes combine the advantages of traditional active contours with the flexibility of morphological operations.
- They can handle objects with irregular shapes and varying contrast levels.
- The method is robust to noise and cluttered backgrounds.

Limitations:

- Morphological snakes may struggle with concave or fragmented objects.
- They can be sensitive to the choice of parameters and initialization.
- The algorithm may require significant computational resources, especially for large images or complex objects.

Thus, morphological snakes offer a powerful approach to image segmentation, particularly suitable for applications requiring accurate delineation of object boundaries in challenging imaging conditions. By integrating morphological operations with active contour models, morphological snakes can achieve precise segmentation results in various domains such as medical imaging, remote sensing, and computer vision.

8.10 GRABCUT ALGORITHMS

The GrabCut algorithm is an iterative method for foreground object segmentation in images, developed by Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. It combines graph cuts with Gaussian mixture modeling to efficiently segment objects based on user input. Here's a detailed explanation of the GrabCut algorithm along with mathematical equations and a corresponding diagram:

GrabCut Algorithm:

1. Initialization:

- The algorithm begins with an initial bounding box or user-defined scribbles indicating the foreground and background regions.
- Pixels within the bounding box are classified as either foreground or background based on their color similarity to the scribbles.

2. Gaussian Mixture Modeling:

- Represent the pixel colors within the bounding box as a mixture of Gaussian distributions, one for foreground and one for background.
- Learn the parameters (mean and covariance) of these Gaussian distributions using the Expectation-Maximization (EM) algorithm.

3. Graph Construction:

- Construct a graph where each pixel in the image is a node, and the edges between nodes represent the pairwise relationships between pixels.
- Assign weights to edges based on color similarity and spatial proximity.

4. Graph Cut Optimization:

- Use graph cuts to iteratively optimize the segmentation by minimizing the energy function, which consists of data and smoothness terms.
- The data term encourages pixels to be assigned to the foreground or background based on their color likelihood under the Gaussian mixture models.

- The smoothness term penalizes abrupt changes in the segmentation, promoting smooth object boundaries.

5. Foreground and Background Refinement:

- Update the foreground and background models based on the new segmentation result.
- Refine the Gaussian mixture models and reassign pixels to foreground or background based on the updated models.

6. Convergence:

- Iterate the optimization process until convergence, typically based on a predefined number of iterations or when the segmentation result stabilizes.

Mathematical Equations:

1. **Gaussian Mixture Model (GMM):**
   $P(x) = \sum_{k=1}^{K} \pi_k \cdot \mathcal{N}(x|\mu_k, \Sigma_k)$
   Where:
   - x is the pixel intensity,
   - π_k is the weight of component k,
   - μ_k is the mean, and
   - Σ_k is the covariance matrix of component k.
2. **Data Term:**
   $D(x) = -\log(P(x|\text{foreground})) - \log(P(x|\text{background}))$
3. **Smoothness Term:**
   $S(x, y) = \lambda \cdot \exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right)$
   Where:
   - x and y are neighboring pixels,
   - λ controls the strength of the term, and
   - σ controls the spatial proximity.
4. **Energy Function:**
   $E(S) = \sum_{x \in V} D(x) + \sum_{(x,y) \in E} S(x, y)$

Advantages:

- GrabCut is an efficient and interactive method for foreground object segmentation.
- It does not require extensive user input and can adapt to complex object shapes and backgrounds.

- The algorithm provides accurate segmentation results, particularly for images with well-defined object boundaries.

Limitations:

- GrabCut may struggle with images containing multiple objects with similar colors or textures.
- It can be sensitive to the initialization and may require manual adjustments for challenging cases.
- The algorithm may not perform well on images with low contrast or ambiguous object boundaries.

In summary, the GrabCut algorithm offers a versatile and effective approach to foreground object segmentation in images, combining Gaussian mixture modeling with graph cuts to achieve accurate and efficient results. With its interactive nature and robust performance, GrabCut has become a popular choice for various computer vision and image editing applications.

8.11 SUMMARY

We have seen thatmain goal of image segmentation is to simplify the representation of an image into more meaningful and easy-to-analyze parts. It aims to partition an image into distinct regions or objects based on certain features or criteria.We have also observed thatthe Hough Transform provides a robust method for detecting lines and circles in images, even in the presence of noise and occlusion.It operates by transforming the spatial domain representation of the image into a parameter space where geometric shapes are represented explicitly.

Thresholding is a fundamental image processing technique used for image segmentation, and Otsu's method provides an automatic way to select an optimal threshold value based on the image's intensity distribution

We also explored region splitting and merging which is a flexible technique for image segmentation, suitable for a wide range of applications such as medical imaging, remote sensing, and scene analysis. By iteratively dividing and merging regions based on predefined criteria, this method can effectively partition images into meaningful segments or objects.

8.12 LIST OF REFERENCES

Text Books:

1. Digital Image Processing by Rafael Gonzalez & Richard Woods, Pearson; 4th edition, 2018.
2. Think DSP: Digital Signal Processing in Python by Allen Downey, O'Reilly Media; 1st edition (August 16, 2016).

Reference Books:

1. Understanding Digital Image Processing, VipinTyagi, CRC Press, 2018.
2. Digital Signal and Image Processing by Tamal Bose, John Wiley 2010.
3. Hands-On Image Processing with Python by SandipanDey,Packt Publishing, 2018.
4. Fundamentals of Digital Images Processing by A K Jain, Pearson, 2010.

8.13 UNIT END EXERCISES

1. Explain in detail about Hough Transform for detecting lines and circles.
2. Write a note on Thresholding and Otsu's segmentation.
3. What is Edge-based/region-based segmentation?
4. Explain the concept of region growing.
5. Discuss in detail about region splitting and merging.
6. Explain watershed algorithm.
7. Write a note on Active Contours along with its mathematical formulations.
8. Write a note on Morphological snakesalong with its mathematical formulations.
9. Write a note on GrabCut algorithmsalong with its mathematical formulations.