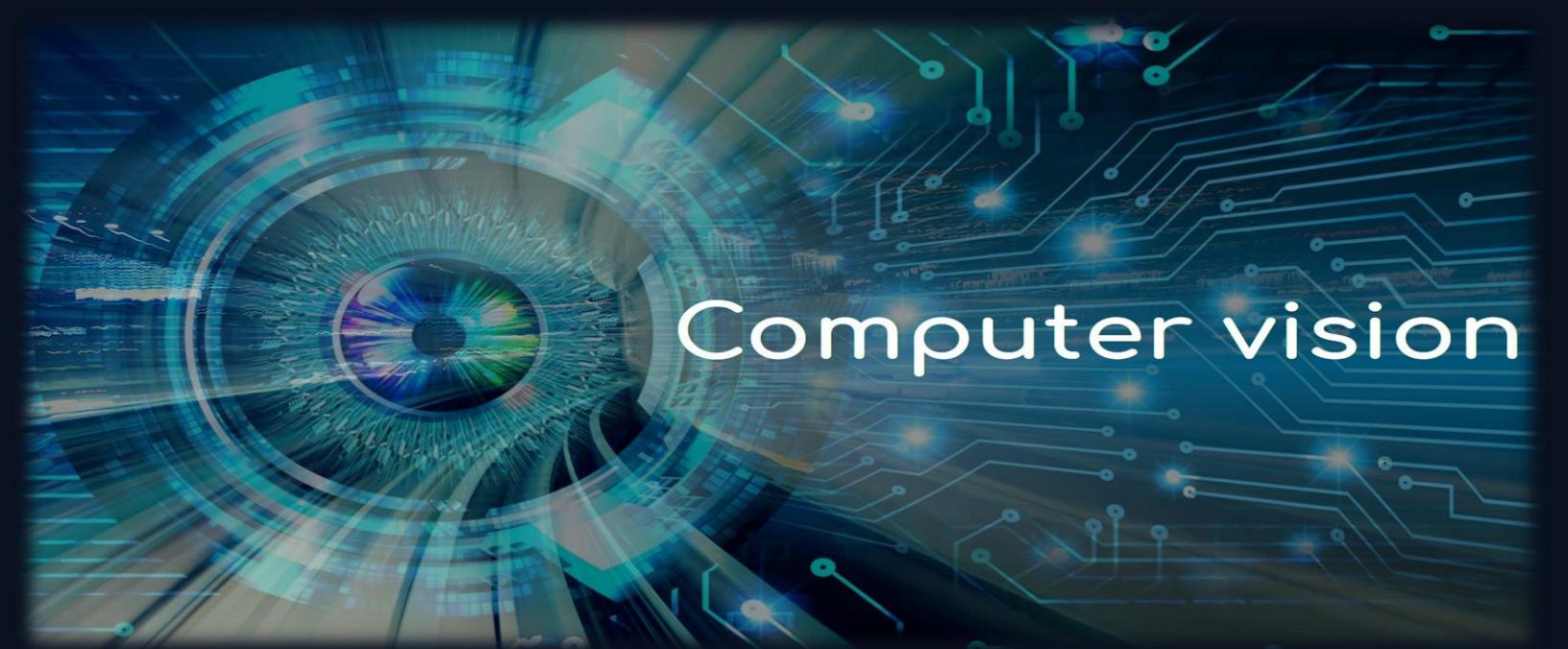


IMAGE PROCESSING

**Image Processing & Computer Vision LAB
with MATLAB**

Low level Vision



Computer vision

Eng. Elaf A.Saeed

Introduction to Lessons

Introduction

Computer vision is concerned with modeling and replicating human vision using computer software and hardware. Formally if we define computer vision then its definition would be that computer vision is a discipline that studies how to reconstruct, interrupt and understand a 3d scene from its 2d images in terms of the properties of the structure present in scene.

It needs knowledge from the following fields in order to understand and stimulate the operation of human vision system.

- Computer Science
- Electrical Engineering
- Mathematics
- Physiology
- Biology
- Cognitive Science

Computer Vision Hierarchy

Computer vision is divided into three basic categories that are as following:

Low-level vision: includes process image for feature extraction.

Intermediate-level vision: includes object recognition and 3D scene Interpretation

High-level vision: includes conceptual description of a scene like activity, intention and behavior.

In these lessons we take the low-level vision and A set of topics related to image processing will be explained.

Introduction to Lessons

Table of Contents

| | |
|---|-----|
| Introduction | 1 |
| Lesson#1 Introduction to Image Processing and Computer Vision..... | 3 |
| Lesson#2 Introduction to MATLAB Program | 65 |
| Lesson#3 Some important functions are used with image in MATLAB | 122 |
| Lesson#4 Read and Change any pixel in the image by using MATLAB | 152 |
| Lesson#5 Color Image Processing and color models in MATLAB (Model 1: RGB Color Model) | 158 |
| Lesson#6 Color Image Types (Indexed Images & RGB (TrueColor) Images) | 170 |
| Lesson#7 Math Operations with Images in MATLAB | 180 |
| Lesson#8 Algebra Operations with Images in MATLAB | 196 |
| Lesson#9 Concept of Sampling and Quantization with Images in MATLAB | 209 |
| Lesson#10 Zooming and Shrinking the Images in MATLAB | 224 |
| Lesson#11 Image Enhancement in the Spatial Model part1 – point Processing. | 235 |
| Lesson#12 Image Enhancement in the Spatial Model part2 – Histogram | 253 |
| Lesson#13 Special Domain – Convolution and Filters – Low pass Filter (LPF) | 275 |
| Lesson#14 Image Restoration part1 – Introduction and Noise Reduction | 294 |
| Lesson#15 High Pass Filter (HPF) – Edge Detection and Sharpening | 309 |
| Lesson#16 Frequency Domain – Filter in Frequency Domain | 341 |
| Lesson#17 Image Restoration – Deblurring Image by using Bilateral Filter – Inverse Filter – Winner Filter | 360 |
| Lesson#18 Image Segmentation – Threshold | 385 |
| Lesson#19 Morphology Process | 401 |
| Lesson#20 Image Pyramids | 414 |

Lesson#1 Image Processing & COMPUTER VISION

Introduction to Image Processing and Computer Vision

Eng. Elaf A.Saeed

Email: elafe1888@gmail.com

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

➤ Digital Image Processing Introduction

Introduction

Signal processing is a discipline in electrical engineering and in mathematics that deals with analysis and processing of analog and digital signals, and deals with storing, filtering, and other operations on signals. These signals include transmission signals, sound or voice signals, image signals, and other signals etc.

Out of all these signals, the field that deals with the type of signals for which the input is an image and the output is also an image is done in image processing. As its name suggests, it deals with the processing on images.

It can be further divided into analog image processing and digital image processing.

Analog image processing

Analog image processing is done on analog signals. It includes processing on two dimensional analog signals. In this type of processing, the images are manipulated by electrical means by varying the electrical signal. The common example include is the television image.

Digital image processing has dominated over analog image processing with the passage of time due to its wider range of applications.

Digital image processing

The digital image processing deals with developing a digital system that performs operations on a digital image.

What is an Image

An image is nothing more than a two-dimensional signal. It is defined by the mathematical function $f(x,y)$ where x and y are the two co-ordinates horizontally and vertically.

The value of $f(x,y)$ at any point gives the pixel value at that point of an image.

LESSON#1 IMAGE PROCESSING & COMPUTER VISION



Figure 1.1: Digital Image

The above figure 1.1 is an example of digital image that you are now viewing on your computer screen. But actually, this image is nothing but a two-dimensional array of numbers ranging between 0 and 255.

| | | |
|-----|-----|-----|
| 128 | 230 | 123 |
| 232 | 123 | 321 |
| 123 | 77 | 89 |
| 80 | 255 | 255 |

Each number represents the value of the function $f(x,y)$ at any point. In this case the value 128, 230 ,123 each represents an individual pixel value. The dimensions of the picture are actually the dimensions of this two dimensional array.

Relationship between a digital image and a signal

If the image is a two dimensional array, then what does it have to do with a signal?

In order to understand that, we need to first understand what is a signal?

Signal

In physical world, any quantity measurable through time over space or any higher dimension can be taken as a signal. A signal is a mathematical function, and it conveys some information.

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

A signal can be one dimensional or two dimensional or higher dimensional signal. One dimensional signal is a signal that is measured over time. The common example is a voice signal.

The two dimensional signals are those that are measured over some other physical quantities. The example of two dimensional signal is a digital image.

Relationship

Since anything that conveys information or broadcast a message in physical world between two observers is a signal. That includes speech or (human voice) or an image as a signal. Since when we speak, our voice is converted to a sound wave/signal and transformed with respect to the time to person we are speaking to. Not only this, but the way a digital camera works, as while acquiring an image from a digital camera involves transfer of a signal from one part of the system to the other.

How a digital image is formed?

Since capturing an image from a camera is a physical process. The sunlight is used as a source of energy. A sensor array is used for the acquisition of the image. So when the sunlight falls upon the object, then the amount of light reflected by that object is sensed by the sensors, and a continuous voltage signal is generated by the amount of sensed data. In order to create a digital image, we need to convert this data into a digital form. This involves sampling and quantization. (They are discussed later on). The result of sampling and quantization results in a two dimensional array or matrix of numbers which are nothing but a digital image.

Overlapping fields

Machine/Computer vision

Machine vision or computer vision deals with developing a system in which the input is an image and the output is some information. For example: Developing a

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

system that scans human face and opens any kind of lock. This system would look something like this in figure 1.2.

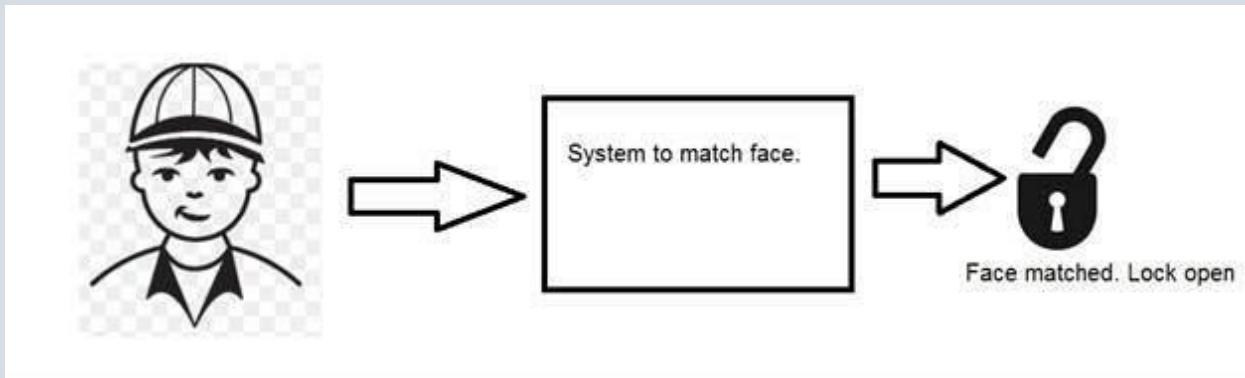


Figure 1.2: System Face Matched. Lock Open

Computer graphics

Computer graphics deals with the formation of images from object models, rather than the image is captured by some device. For example: Object rendering. Generating an image from an object model. Such a system would look something like this in figure 1.3.

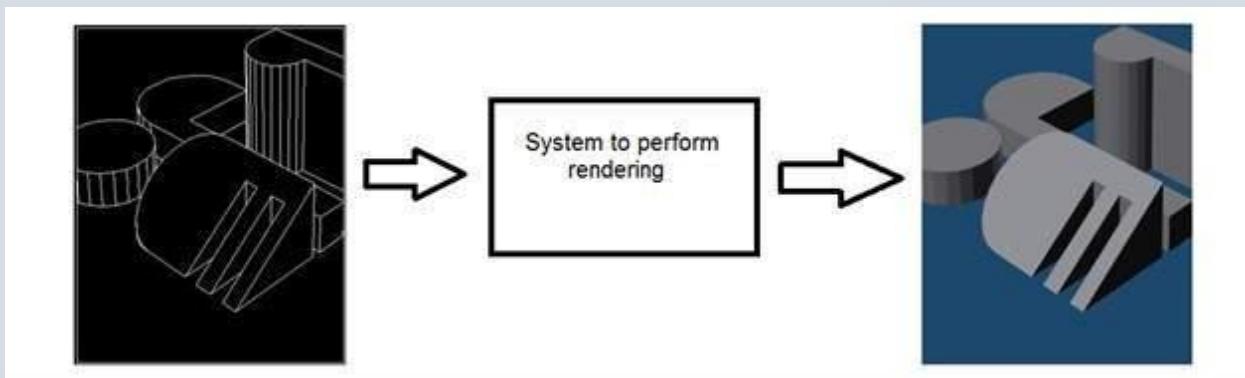


Figure 1.3: System Generating an Image from an object model

Artificial intelligence

Artificial intelligence is more or less the study of putting human intelligence into machines. Artificial intelligence has many applications in image processing. For example: developing computer aided diagnosis systems that help doctors in interpreting images of X-ray, MRI etc. and then highlighting conspicuous section to be examined by the doctor.

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

Signal processing

Signal processing is an umbrella and image processing lies under it. The amount of light reflected by an object in the physical world (3d world) is pass through the lens of the camera and it becomes a 2d signal and hence result in image formation. This image is then digitized using methods of signal processing and then this digital image is manipulated in digital image processing.

➤ Signals and Systems Introduction

Signals

In electrical engineering, the fundamental quantity of representing some information is called a signal. It does not matter what the information is i-e: Analog or digital information. In mathematics, a signal is a function that conveys some information. In fact, any quantity measurable through time over space or any higher dimension can be taken as a signal. A signal could be of any dimension and could be of any form.

Analog signals

A signal could be an analog quantity that means it is defined with respect to the time. It is a continuous signal. These signals are defined over continuous independent variables. They are difficult to analyze, as they carry a huge number of values. They are very much accurate due to a large sample of values. In order to store these signals, you require an infinite memory because it can achieve infinite values on a real line. Analog signals are denoted by sin waves.

For example:

Human voice

Human voice is an example of analog signals. When you speak, the voice that is produced travel through air in the form of pressure waves and thus belongs to a

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

mathematical function, having independent variables of space and time and a value corresponding to air pressure.

Another example is of sin wave which is shown in the figure 1.4 below.

$Y = \sin(x)$ where x is independent

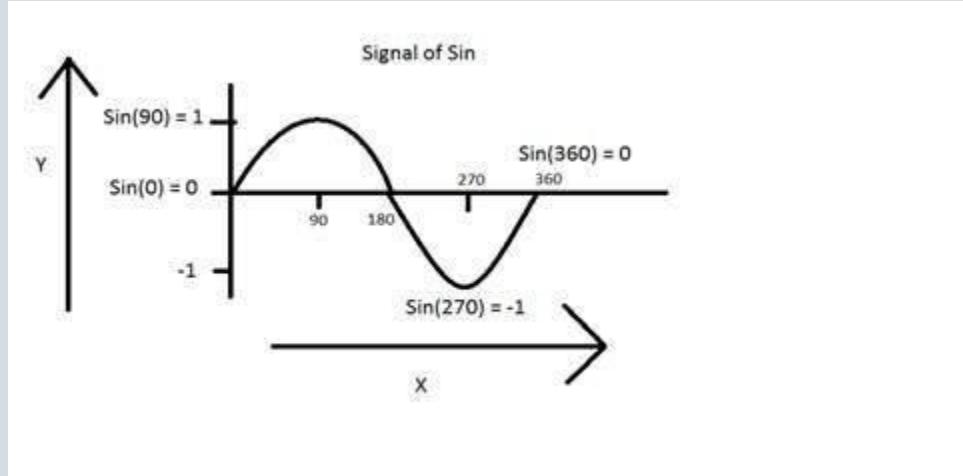


Figure 1.4: Sine Wave Analog Signal

Digital signals

As compared to analog signals, digital signals are very easy to analyze. They are discontinuous signals. They are the appropriation of analog signals.

The word digital stands for discrete values and hence it means that they use specific values to represent any information. In digital signal, only two values are used to represent something i-e: 1 and 0 (binary values). Digital signals are less accurate than analog signals because they are the discrete samples of an analog signal taken over some period of time. However digital signals are not subject to noise. So they last long and are easy to interpret. Digital signals are denoted by square waves.

For example:

Computer keyboard

Whenever a key is pressed from the keyboard, the appropriate electrical signal is sent to keyboard controller containing the ASCII value that particular key. For example, the electrical signal that is generated when keyboard key a is pressed, carry

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

information of digit 97 in the form of 0 and 1, which is the ASCII value of character a.

Difference between analog and digital signals

| Comparison element | Analog signal | Digital signal |
|----------------------------|---|---|
| Analysis | Difficult | Possible to analyze |
| Representation | Continuous | Discontinuous |
| Accuracy | More accurate | Less accurate |
| Storage | Infinite memory | Easily stored |
| Subject to Noise | Yes | No |
| Recording Technique | Original signal is preserved | Samples of the signal are taken and preserved |
| Examples | Human voice, Thermometer, Analog phones etc. | Computers, Digital Phones, Digital pens, etc. |

Systems

A system is defined by the type of input and output it deals with. Since we are dealing with signals, so in our case, our system would be a mathematical model, a piece of code/software, or a physical device, or a black box whose input is a signal and it performs some processing on that signal, and the output is a signal. The input is known as excitation and the output is known as response.



LESSON#1 IMAGE PROCESSING & COMPUTER VISION

Figure 1.5: Conversion System

In the above figure 1.5 a system has been shown whose input and output both are signals but the input is an analog signal. And the output is a digital signal. It means our system is actually a conversion system that converts analog signals to digital signals.

Let's have a look at the inside of this black box system

Conversion of analog to digital signals

Since there are lot of concepts related to this analog to digital conversion and vice-versa. We will only discuss those which are related to digital image processing. There are two main concepts that are involved in the conversion.

- **Sampling**
- **Quantization**

Sampling

Sampling as its name suggests can be defined as take samples. Take samples of a digital signal over x axis. Sampling is done on an independent variable. In case of this mathematical equation in figure 1.6:

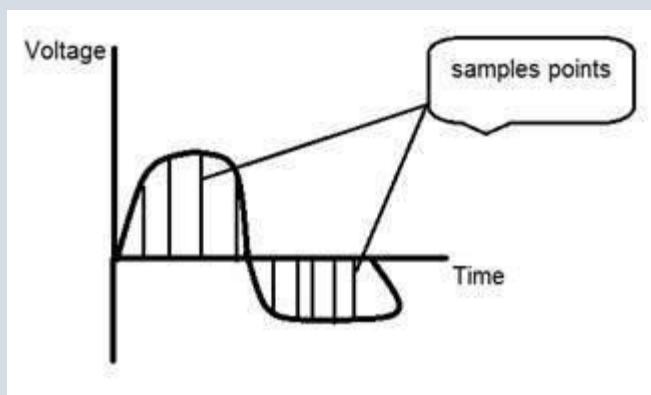


Figure 1.6: Sampling

Sampling is done on the x variable. We can also say that the conversion of x axis (infinite values) to digital is done under sampling.

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

Sampling is further divide into up sampling and down sampling. If the range of values on x-axis are less, then we will increase the sample of values. This is known as up sampling and its vice versa is known as down sampling.

Quantization

Quantization as its name suggest can be defined as dividing into quanta (partitions).

Quantization is done on dependent variable. It is opposite to sampling.

In case of this mathematical equation $y = \sin(x)$

Quantization is done on the Y variable. It is done on the y axis. The conversion of y axis infinite values to 1, 0, -1 (or any other level) is known as Quantization.

These are the two basics steps that are involved while converting an analog signal to a digital signal.

The quantization of a signal has been shown in the figure below 1.7.

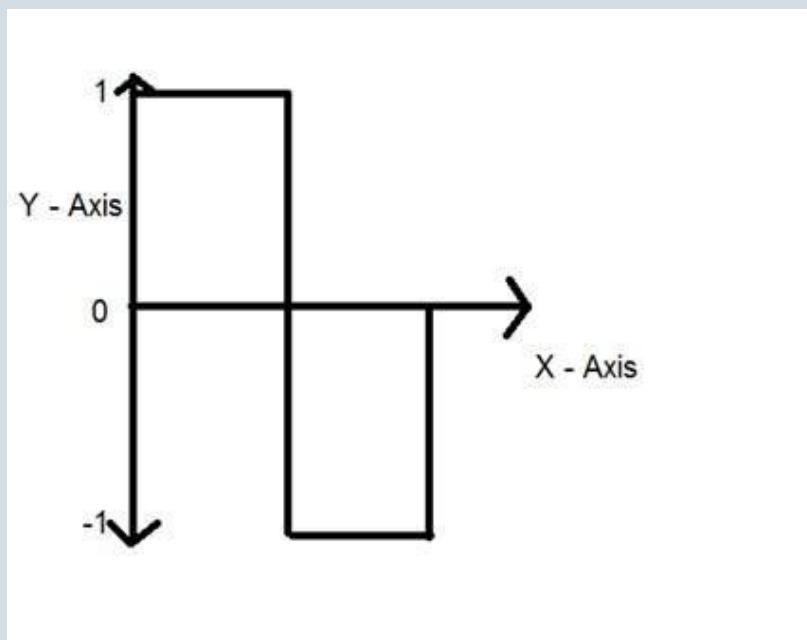


Figure 1.7: Quantization

Why do we need to convert an analog signal to digital signal?

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

The first and obvious reason is that digital image processing deals with digital images, that are digital signals. So whenever the image is captured, it is converted into digital format and then it is processed.

The second and important reason is, that in order to perform operations on an analog signal with a digital computer, you have to store that analog signal in the computer. And in order to store an analog signal, infinite memory is required to store it. And since that's not possible, so that's why we convert that signal into digital format and then store it in digital computer and then performs operations on it.

Continuous systems vs discrete systems

Continuous systems

The type of systems whose input and output both are continuous signals or analog signals are called continuous systems, as shown un figure 1.8.

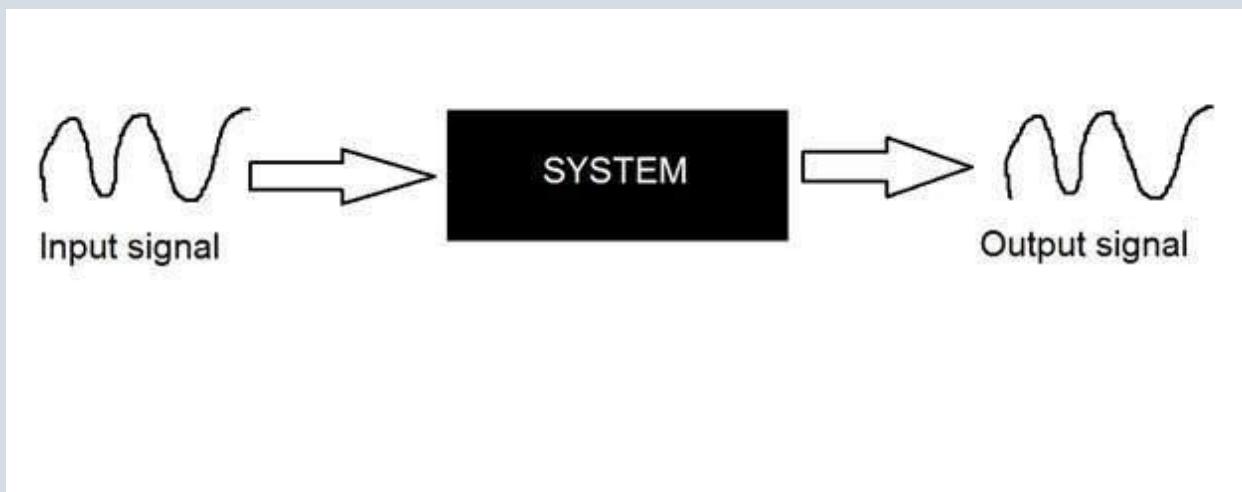


Figure 1.8: Continuous Systems

Discrete systems

The type of systems whose input and output both are discrete signals or digital signals are called digital systems, as shown un figure 1.9.

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

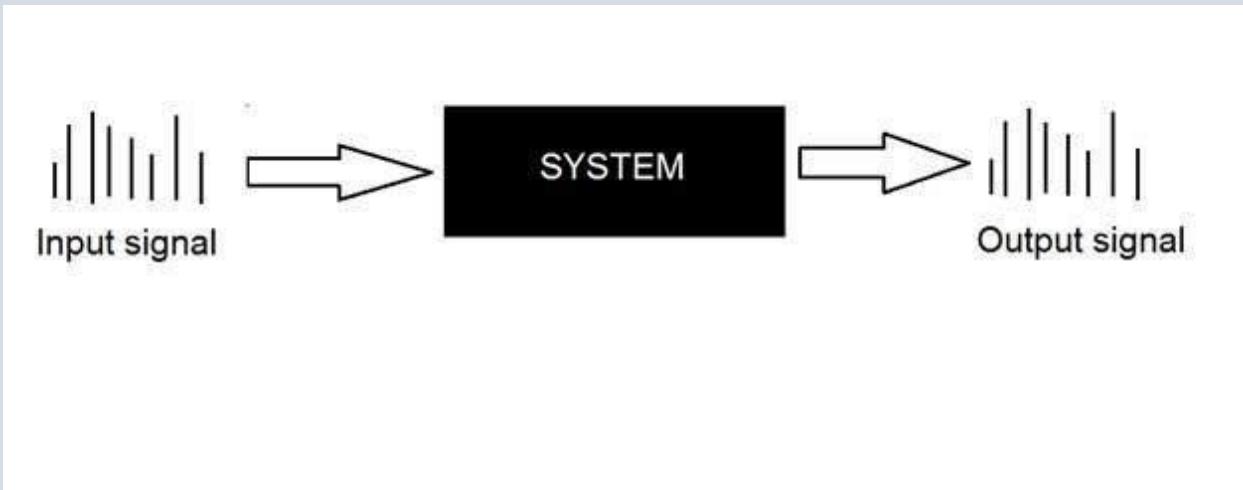


Figure 1.9: Digital Systems

➤ History of Photography

Origin of camera

The history of camera and photography is not exactly the same. The concepts of camera were introduced a lot before the concept of photography.

camera obscura

The history of the camera lies in ASIA. The principles of the camera were first introduced by a Chinese philosopher MOZI. It is known as camera obscura. The cameras evolved from this principle. The word camera obscura is evolved from two different words. Camera and Obscura. The meaning of the word camera is a room or some kind of vault and Obscura stands for dark. The concept which was introduced by the Chinese philosopher consist of a device, that project an image of its surrounding on the wall. However, it was not built by the Chinese. In figure 1.10 that shown the camera obscura.

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

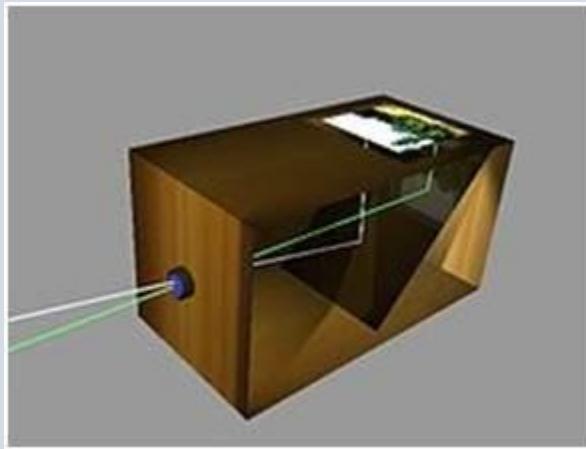


Figure 1.10: Camera Obscura

The creation of camera obscura

The concept of Chinese was bringing in reality by a Muslim scientist Abu Ali Al-Hassan Ibn al-Haitham commonly known as Ibn al-Haitham. He built the first camera obscura. His camera follows the principles of pinhole camera. He builds this device in somewhere around 1000.

Portable camera

In 1685, a first portable camera was built by Johann Zahn. Before the advent of this device, the camera consists of a size of room and were not portable. Although a device was made by an Irish scientist Robert Boyle and Robert Hooke that was a transportable camera, but still that device was very huge to carry it from one place to the other.

Origin of photography

Although the camera obscura was built in 1000 by a Muslim scientist. But its first actual use was described in the 13th century by an English philosopher Roger Bacon. Roger suggested the use of camera for the observation of solar eclipses.

Da Vinci

Although much improvement has been made before the 15th century, but the improvements and the findings done by **Leonardo di ser Piero da Vinci** was

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

remarkable. Da Vinci was a great artist, musician, anatomist, and a war engineer. He is credited for many inventions. His one of the most famous painting includes, the painting of Mona Lisa, as shown in figure 1.11.



Figure 1.11: painting of Mona Lisa

Da Vinci not only built a camera obscura following the principle of a pin hole camera but also uses it as drawing aid for his art work. In his work, which was described in Codex Atlanticus, many principles of camera obscura have been defined, as shown in figure 1.12.

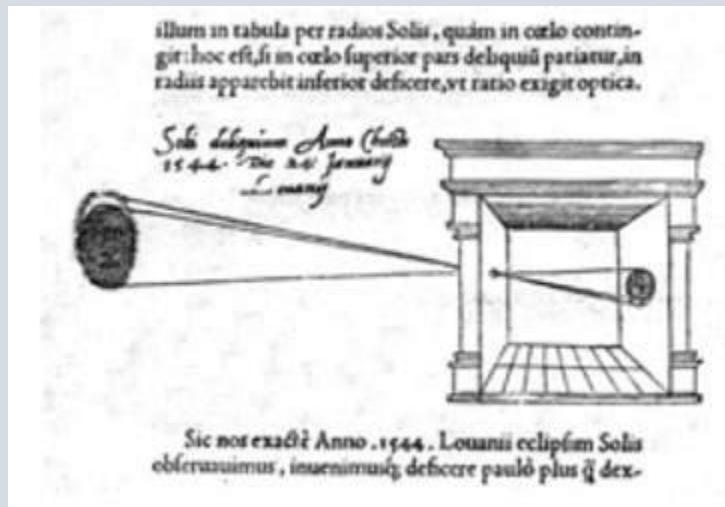


Figure 1.12: Principles of Camera Obscura

His camera follows the principle of a pin hole camera which can be described as

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

When images of illuminated objects penetrate through a small hole into a very dark room you will see [on the opposite wall] these objects in their proper form and color, reduced in size in a reversed position, owing to the intersection of rays.

First photograph

The first photograph was taken in 1814 by a French inventor Joseph Nicéphore Nièpce. He captures the first photograph of a view from the window at Le Gras, by coating the pewter plate with bitumen and after that exposing that plate to light, as shown in figure 1.13.



Figure 1.13: First photograph

First underwater photograph

The first underwater photograph was taken by an English mathematician William Thomson using a water tight box. This was done in 1856, as shown in figure 1.14.



Figure 1.14: First underwater photograph

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

The origin of film

The origin of film was introduced by an American inventor and a philanthropist known as George Eastman who is considered as the pioneer of photography. He founded the company called as Eastman Kodak, which is famous for developing films. The company starts manufacturing paper film in 1885. He first created the camera Kodak and then later Brownie. Brownie was a box camera and gain its popularity due to its feature of Snapshot, as shown in figure 1.15.



Figure 1.15: Box Camera

After the advent of the film, the camera industry once again got a boom and one invention lead to another.

Leica and Argus

Leica and argus are the two analog cameras developed in 1925 and in 1939 respectively. The camera Leica was built using a 35mm cine film, as shown in figure 1.16.

LESSON#1 IMAGE PROCESSING & COMPUTER VISION



Figure 1.16: Argus Camera

Argus was another camera analog camera that uses the 35mm format and was rather inexpensive as compared by Leica and became very popular.



Figure 1.17: Leica Camera

Analog CCTV cameras

In 1942 a German engineer Walter Bruch developed and installed the very first system of the analog CCTV cameras. He is also credited for the invention of color television in the 1960.

Photo Pac

The first disposable camera was introduced in 1949 by Photo Pac. The camera was only a one time use camera with a roll of film already included in it. The later versions of Photo Pac were water proof and even have the flash, as shown in figure 1.18.



Figure 1.18: Photo Pac

Digital Cameras

Mavica by Sony

Mavica (the magnetic video camera) was launched by Sony in 1981 was the first game changer in digital camera world. The images were recorded on floppy disks and images can be viewed later on any monitor screen.

It was not a pure digital camera, but an analog camera. But got its popularity due to its storing capacity of images on a floppy disk. It means that you can now store images for a long lasting period, and you can save a huge number of pictures on the floppy which are replaced by the new blank disc, when they got full. Mavica has the capacity of storing 25 images on a disk.

One more important thing that mavica introduced was its 0.3 mega pixel capacity of capturing photos, as shown in figure 1.19.

LESSON#1 IMAGE PROCESSING & COMPUTER VISION



Figure 1.19: Mavica by Sony

Digital Cameras

Fuji DS-1P camera by Fuji films 1988 was the first true digital camera

Nikon D1 was a 2.74 mega pixel camera and the first commercial digital SLR camera developed by Nikon, and was very much affordable by the professionals, that is shown in figure 1.20.



Figure 1.20: Nikon D1

Today digital cameras are included in the mobile phones with very high resolution and quality.

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

➤ Applications and Usage

Since digital image processing has very wide applications and almost all of the technical fields are impacted by DIP, we will just discuss some of the major applications of DIP. Digital Image processing is not just limited to adjust the spatial resolution of the everyday images captured by the camera. It is not just limited to increase the brightness of the photo, etc. Rather it is far more than that. Electromagnetic waves can be thought of as stream of particles, where each particle is moving with the speed of light. Each particle contains a bundle of energy. This bundle of energy is called a photon. The electromagnetic spectrum according to the energy of photon is shown in figure 1.21.

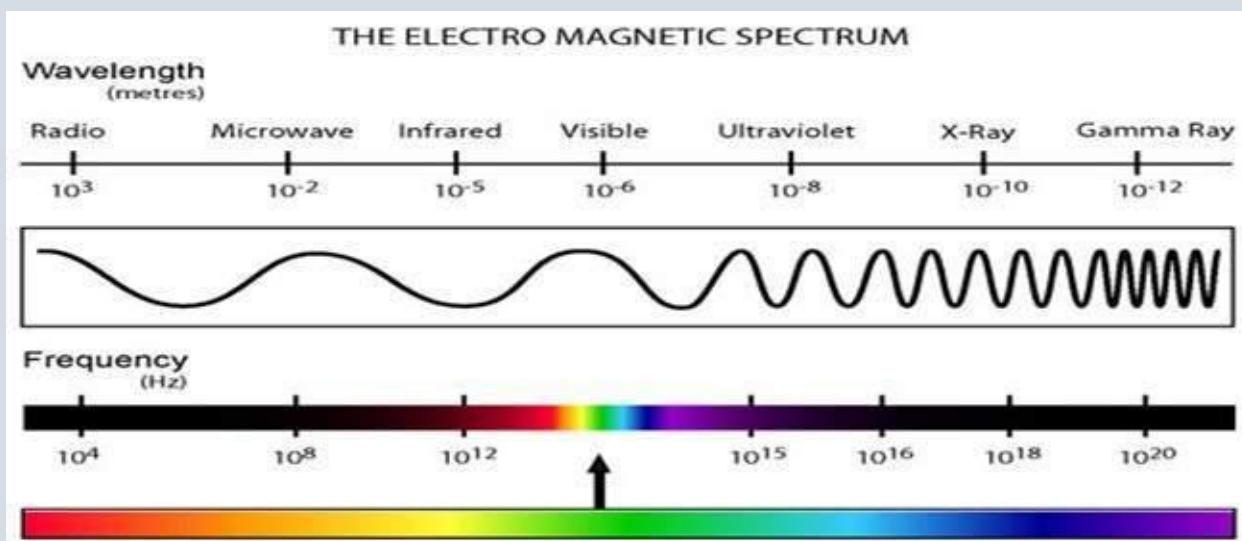


Figure 1.21: Electromagnetic Spectrum

In this electromagnetic spectrum, we are only able to see the visible spectrum. Visible spectrum mainly includes seven different colors that are commonly term as (VIBGOYR). VIBGOYR stands for violet, indigo, blue, green, orange, yellow and Red. But that does not nullify the existence of other stuff in the spectrum. Our human eye can only see the visible portion; in which we saw all the objects. But a camera can see the other things that a naked eye is unable to see. For example: x rays, gamma rays, etc. Hence the analysis of all that stuff too is done in digital image processing.

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

This discussion leads to another question which is

why do we need to analyze all that other stuff in EM spectrum too?

The answer to this question lies in the fact, because that other stuff such as X-ray has been widely used in the field of medical. The analysis of Gamma ray is necessary because it is used widely in nuclear medicine and astronomical observation. Same goes with the rest of the things in EM spectrum.

Applications of Digital Image Processing

Some of the major fields in which digital image processing is widely used are mentioned below

- Image sharpening and restoration
- Medical field
- Remote sensing
- Transmission and encoding
- Machine/Robot vision
- Color processing
- Pattern recognition
- Video processing
- Microscopic Imaging
- Others

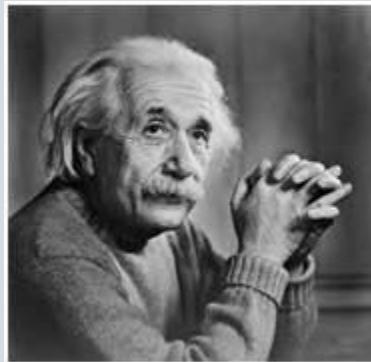
Image sharpening and restoration

Image sharpening and restoration refers here to process images that have been captured from the modern camera to make them a better image or to manipulate those images in way to achieve desired result. It refers to do what Photoshop usually does.

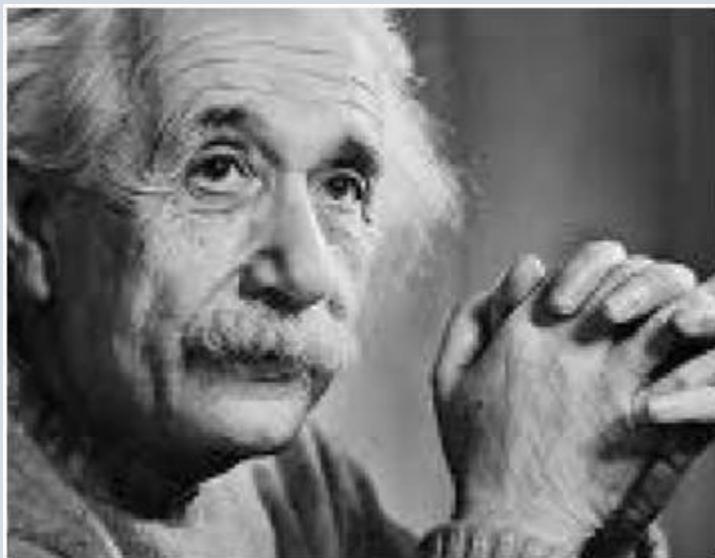
LESSON#1 IMAGE PROCESSING & COMPUTER VISION

This includes Zooming, blurring, sharpening, gray scale to color conversion, detecting edges and vice versa, Image retrieval and Image recognition. The common examples are:

The original image

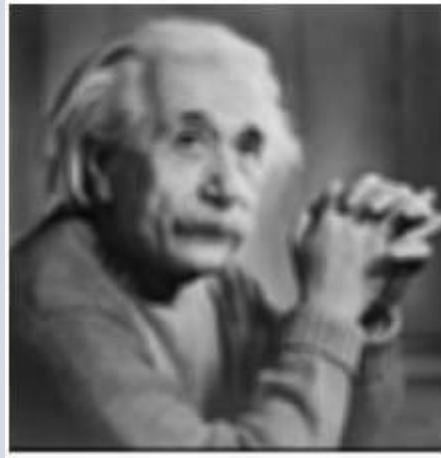


The zoomed image

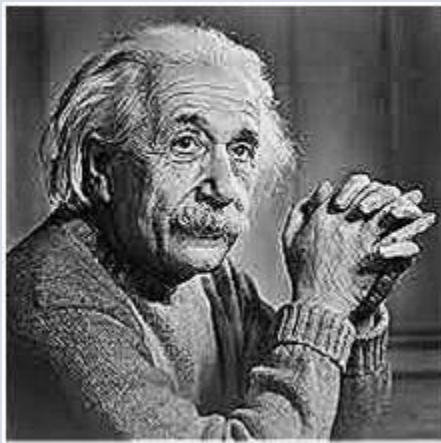


Blur image

LESSON#1 IMAGE PROCESSING & COMPUTER VISION



Sharp image



Edges



Medical field

The common applications of DIP in the field of medical is

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

- Gamma ray imaging
- PET scan
- X Ray Imaging
- Medical CT
- UV imaging

UV imaging

In the field of remote sensing, the area of the earth is scanned by a satellite or from a very high ground and then it is analyzed to obtain information about it. One particular application of digital image processing in the field of remote sensing is to detect infrastructure damages caused by an earthquake. As it takes longer time to grasp damage, even if serious damages are focused on. Since the area effected by the earthquake is sometimes so wide, that it not possible to examine it with human eye in order to estimate damages. Even if it is, then it is very hectic and time consuming procedure. So a solution to this is found in digital image processing. An image of the affected area is captured from the above ground and then it is analyzed to detect the various types of damage done by the earthquake, as it shown in figure 1.22.



Figure 1.22: UV Imaging

The key steps include in the analysis are

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

- The extraction of edges
- Analysis and enhancement of various types of edges

Transmission and encoding

The very first image that has been transmitted over the wire was from London to New York via a submarine cable. The picture that was sent is shown in figure 1.23 below.



Figure 1.23: First Transmission Picture

The picture that was sent took three hours to reach from one place to another.

Now just imagine, that today we are able to see live video feed, or live cctv footage from one continent to another with just a delay of seconds. It means that a lot of work has been done in this field too. This field does not only focus on transmission, but also on encoding. Many different formats have been developed for high or low bandwidth to encode photos and then stream it over the internet or etc.

Machine/Robot vision

Apart from the many challenges that a robot faces today, one of the biggest challenge still is to increase the vision of the robot. Make robot able to see things, identify them, identify the hurdles etc. Much work has been contributed by this field and a complete other field of computer vision has been introduced to work on it.

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

Hurdle detection

Hurdle detection is one of the common task that has been done through image processing, by identifying different type of objects in the image and then calculating the distance between robot and hurdles, as it shown in figure 1.24.



Figure 1.24: Hurdle Detection

Line follower robot

Most of the robots today work by following the line and thus are called line follower robots. This help a robot to move on its path and perform some tasks. This has also been achieved through image processing.



Figure 1.25: Line Follower Robot

Color processing

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

Color processing includes processing of colored images and different color spaces that are used. For example, RGB color model, YCbCr, HSV. It also involves studying transmission, storage, and encoding of these color images.

Pattern recognition

Pattern recognition involves study from image processing and from various other fields that includes machine learning (a branch of artificial intelligence). In pattern recognition, image processing is used for identifying the objects in an images and then machine learning is used to train the system for the change in pattern. Pattern recognition is used in computer aided diagnosis, recognition of handwriting, recognition of images etc.

Video processing

A video is nothing but just the very fast movement of pictures. The quality of the video depends on the number of frames/pictures per minute and the quality of each frame being used. Video processing involves noise reduction, detail enhancement, motion detection, frame rate conversion, aspect ratio conversion, color space conversion etc.

➤ Concept of Dimensions

We will look at this example in figure 1.26 in order to understand the concept of dimension.

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

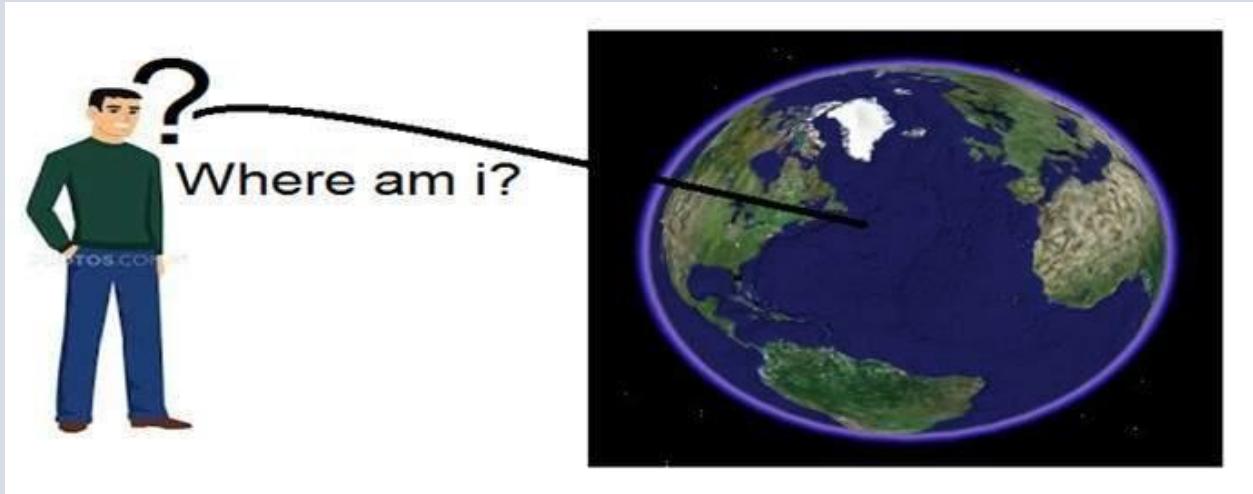


Figure 1.26: Example to understand the concept of dimension

Consider you have a friend who lives on moon, and he wants to send you a gift on your birthday present. He ask you about your residence on earth. The only problem is that the courier service on moon does not understand the alphabetical address, rather it only understands the numerical co-ordinates. So how do you send him your position on earth?

That's where comes the concept of dimensions. Dimensions define the minimum number of points required to point a position of any particular object within a space. So let's go back to our example again in which you have to send your position on earth to your friend on moon. You send him three pair of co-ordinates. The first one is called longitude, the second one is called latitude, and the third one is called altitude. These three co-ordinates define your position on the earth. The first two defines your location, and the third one defines your height above the sea level. So that means that only three co-ordinates are required to define your position on earth. That means you live in world which is 3 dimensional. And thus this not only answers the question about dimension, but also answers the reason, that why we live in a 3d world. Since we are studying this concept in reference to the digital image processing, so we are now going to relate this concept of dimension with an image.

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

Dimensions of image

So if we live in the 3d world, means a 3 dimensional world, then what are the dimensions of an image that we capture. An image is a two dimensional, that's why we also define an image as a 2 dimensional signal. An image has only height and width. An image does not have depth. Just have a look at this image in figure 1.27 below.

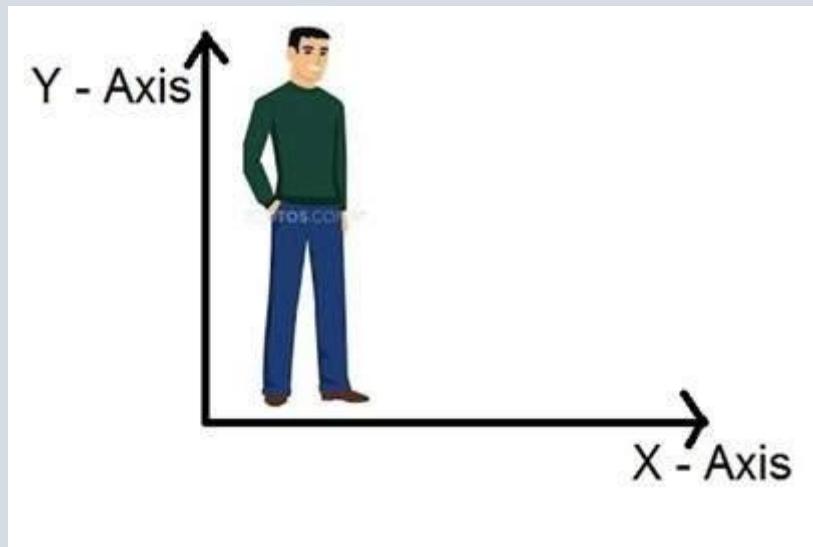


Figure 1.27: Image Look

If you would look at the above figure, it shows that it has only two axes which are the height and width axis. You cannot perceive depth from this image. That's why we say that an image is two dimensional signal. But our eye is able to perceive three dimensional objects.

How does television works?

If we look the image above, we will see that it is a two dimensional image. In order to convert it into three dimension, we need one other dimension. Let's take time as the third dimension, in that case we will move this two dimensional image over the third dimension time. The same concept that happens in television, that helps us perceive the depth of different objects on a screen. Does that mean that

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

what comes on the T.V or what we see in the television screen is 3d? Well we can yes.

The reason is that, in case of T.V we if we are playing a video. Then a video is nothing else but two dimensional pictures move over time dimension. As two dimensional objects are moving over the third dimension which is a time so we can say it is 3 dimensional.

Different dimensions of signals

1-dimension signal

The common example of a 1-dimension signal is a waveform. It can be mathematically represented as

$$F(x) = \text{waveform}$$

Where x is an independent variable. Since it is a one-dimension signal, so that's why there is only one variable x is used.

Pictorial representation of a one dimensional signal is given in figure 1.28 below:

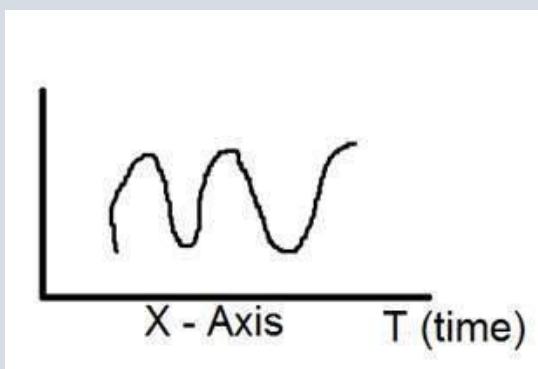


Figure 1.28: 1D Pictorial Signal

The above figure 1.28 shows a one dimensional signal.

Now this lead to another question, which is, even though it is a one dimensional signal, then why does it have two axes? The answer to this question is that even though it is a one dimensional signal, but we are drawing it in a two dimensional

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

space. Or we can say that the space in which we are representing this signal is two dimensional. That's why it looks like a two dimensional signal.

Perhaps you can understand the concept of one dimension better by looking at the figure 1.29 below.



Figure 1.29: Line

Now refer back to our initial discussion on dimension, Consider the above figure a real line with positive numbers from one point to the other. Now if we have to explain the location of any point on this line, we just need only one number, which means only one dimension.

2 dimensions' signal

The common example of a two dimensional signal is an image, which has already been discussed above. In figure 1.30 that shown the 2d image.

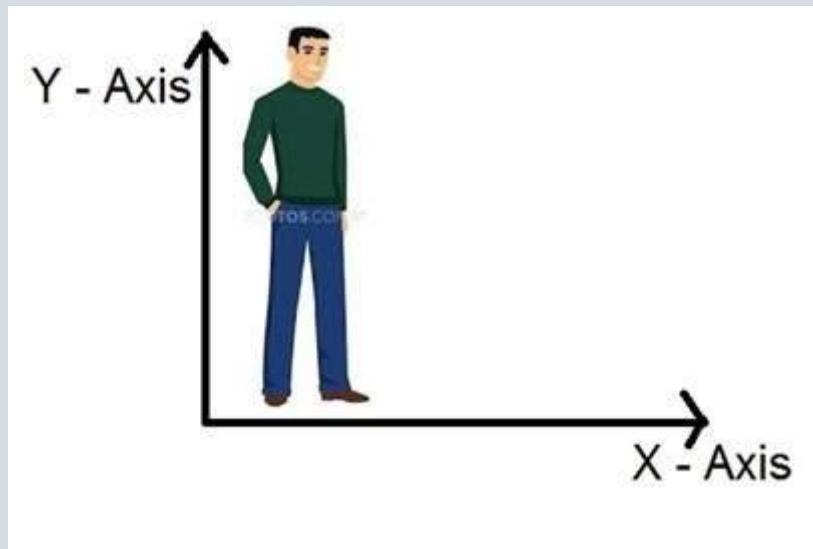


Figure 1.30: 2D Image

As we have already seen that an image is two dimensional signal, i-e: it has two dimensions. It can be mathematically represented as:

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

$$F(x, y) = \text{Image}$$

Where x and y are two variables. The concept of two dimension can also be explained in terms of mathematics as:

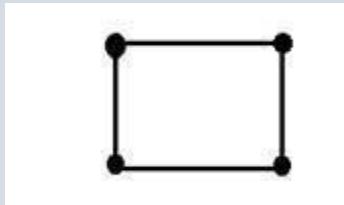


Figure 1.31: Four Corners of the Square

Now in the above figure 1.31, label the four corners of the square as A,B,C and D respectively. If we call, one-line segment in the figure AB and the other CD, then we can see that these two parallel segments join up and make a square. Each line segment corresponds to one dimension, so these two line segments correspond to 2 dimensions.

3-dimension signal

Three dimensional signal as it names refers to those signals which has three dimensions. The most common example has been discussed in the beginning which is of our world. We live in a three dimensional world. This example has been discussed very elaborately. Another example of a three dimensional signal is a cube or a volumetric data or the most common example would be animated or 3d cartoon character.

The mathematical representation of three dimensional signal is:

$$F(x,y,z) = \text{animated character.}$$

Another axis or dimension Z is involved in a three dimension, that gives the illusion of depth. In a Cartesian co-ordinate system, it can be viewed as shown in figure 1.32:

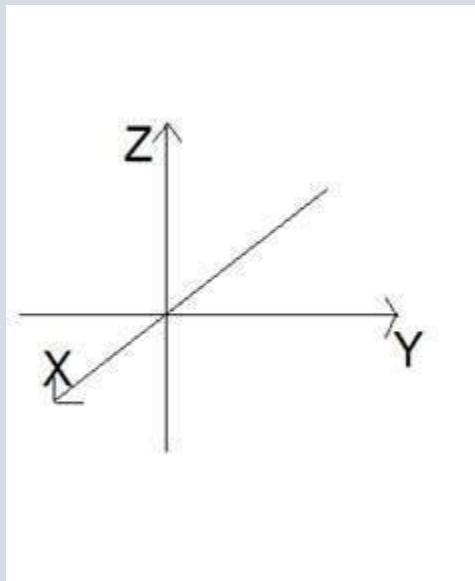


Figure 1.32: 3D Cartesian Co-ordinate System

4-dimension signal

In a four dimensional signal, four dimensions are involved. The first three are the same as of three dimensional signal which are: (X, Y, Z), and the fourth one which is added to them is T(time). Time is often referred to as temporal dimension which is a way to measure change. Mathematically a four d signal can be stated as:

$F(x,y,z,t)$ = animated movie.

The common example of a 4 dimensional signal can be an animated 3d movie. As each character is a 3d character and then they are moved with respect to the time, due to which we saw an illusion of a three dimensional movie more like a real world. So that means that in reality the animated movies are 4 dimensional i-e: movement of 3d characters over the fourth dimension time.

➤ Image Formation on Camera

How human eye works?

Before we discuss, the image formation on analog and digital cameras, we have to first discuss the image formation on human eye. Because the basic principle that is followed by the cameras has been taken from the way, the human eye works.

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

When light falls upon the particular object, it is reflected back after striking through the object. The rays of light when passed through the lens of eye, form a particular angle, and the image is formed on the retina which is the back side of the wall. The image that is formed is inverted. This image is then interpreted by the brain and that makes us able to understand things. Due to angle formation, we are able to perceive the height and depth of the object we are seeing.

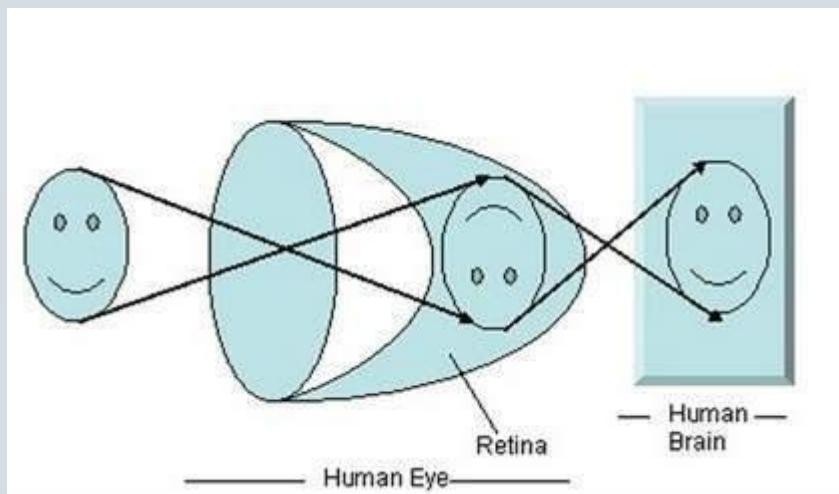


Figure 1.33: Human Eye Works

As you can see in the above figure 1.33, that when sun light falls on the object (in this case the object is a face), it is reflected back and different rays form different angle when they are passed through the lens and an invert image of the object has been formed on the back wall. The last portion of the figure denotes that the object has been interpreted by the brain and re-inverted.

Now let's take our discussion back to the image formation on analog and digital cameras.

Image formation on analog cameras

In figure 1.34 that shown the analog camera.

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

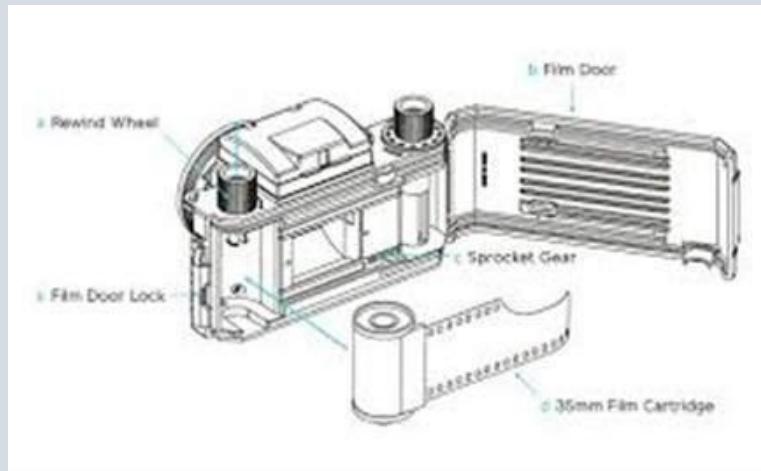


Figure 1.34: Analog Camera

In analog cameras, the image formation is due to the chemical reaction that takes place on the strip that is used for image formation.

A 35mm strip is used in analog camera. It is denoted in the figure 1.35 by 35mm film cartridge. This strip is coated with silver halide (a chemical substance).

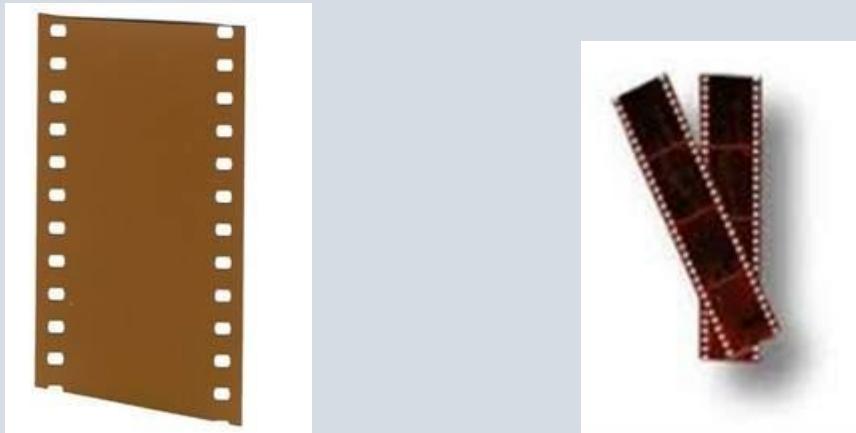


Figure 1.35: Stripe of Analog Camera

Light is nothing but just the small particles known as photon particles. So when these photon particles are passed through the camera, it reacts with the silver halide particles on the strip and it results in the silver which is the negative of the image. In order to understand it better, have a look at this equation.

Photons (light particles) + silver halide? silver? image negative.

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

This is just the basics, although image formation involves many other concepts regarding the passing of light inside, and the concepts of shutter and shutter speed and aperture and its opening but for now we will move on to the next part. Although most of these concepts have been discussed in our tutorial of shutter and aperture.

Image formation on digital cameras

In the digital cameras, the image formation is not due to the chemical reaction that take place, rather it is a bit more complex than this. In the digital camera, a CCD array of sensors is used for the image formation.

Image formation through CCD array

In figure 1.36 that shown the CCD array IC.

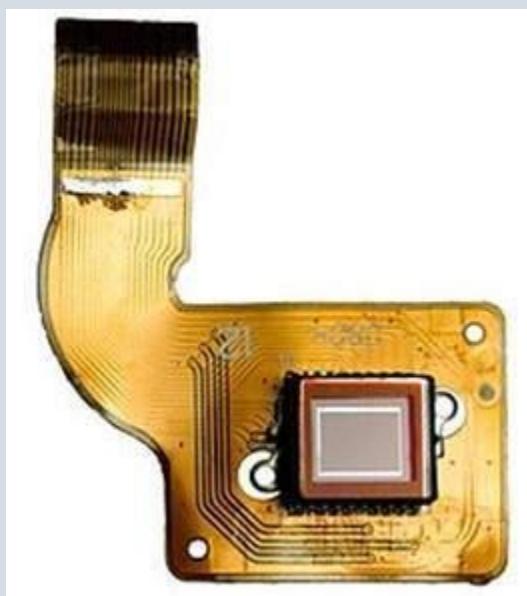


Figure 1.36: CCD array IC

CCD stands for charge-coupled device. It is an image sensor, and like other sensors it senses the values and converts them into an electric signal. In case of CCD it senses the image and convert it into electric signal etc. This CCD is actually in the shape of array or a rectangular grid. It is like a matrix with each cell in the matrix contains a sensor that senses the intensity of photon, as shown in figure 1.37.

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

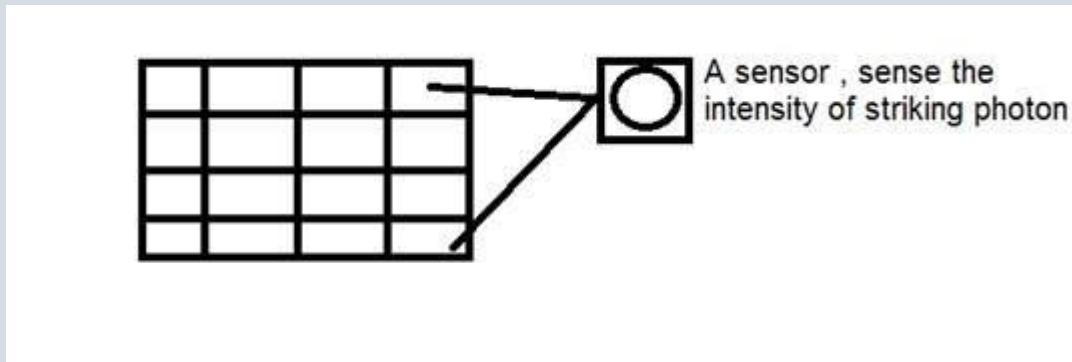


Figure 1.37: Array Sensors

Like analog cameras, in the case of digital too, when light falls on the object, the light reflects back after striking the object and allowed to enter inside the camera. Each sensor of the CCD array itself is an analog sensor. When photons of light strike on the chip, it is held as a small electrical charge in each photo sensor. The response of each sensor is directly equal to the amount of light or (photon) energy strike on the surface of the sensor. Since we have already define an image as a two dimensional signal and due to the two dimensional formation of the CCD array, a complete image can be achieved from this CCD array. It has limited number of sensors, and it means a limited detail can be captured by it. Also each sensor can have only one value against each photon particle that strike on it. So the number of photons striking(current) are counted and stored. In order to measure accurately these, external CMOS sensors are also attached with CCD array.

Introduction to pixel

The value of each sensor of the CCD array refers to each the value of the individual pixel. The **number of sensors = number of pixels**. It also means that each sensor could have only one and only one value.

Storing image

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

The charges stored by the CCD array are converted to voltage one pixel at a time. With the help of additional circuits, this voltage is converted into a digital information and then it is stored.

Each company that manufactures digital camera, make their own CCD sensors. That include, Sony, Mitsubishi, Nikon, Samsung, Toshiba, FujiFilm, Canon etc.

Apart from the other factors, the quality of the image captured also depends on the type and quality of the CCD array that has been used.

➤ Camera Mechanism

Aperture

Aperture is a small opening which allows the light to travel inside into camera. Here in figure 1.38 is the picture of aperture.

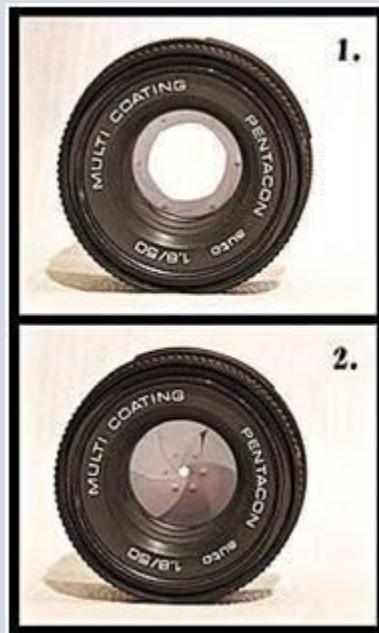


Figure 1.38: Aperture

You will see some small blades like stuff inside the aperture. These blades create an octagonal shape that can be opened closed. And thus it makes sense that, the more blades will open, the hole from which the light would have to pass would be bigger. The bigger the hole, the more light is allowed to enter.

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

Effect

The effect of the aperture directly corresponds to brightness and darkness of an image. If the aperture opening is wide, it would allow more light to pass into the camera. More light would result in more photons, which ultimately result in a brighter image.

The example of this is shown below

Consider these two photos

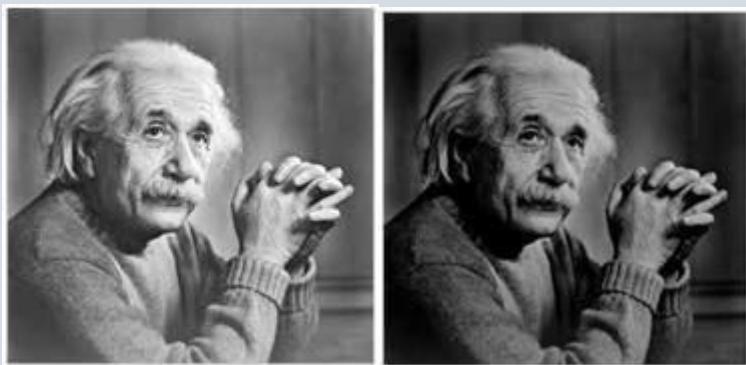


Figure 1.39: Effects of Aperture Open

The one on the right side looks brighter, it means that when it was captured by the camera, the aperture was wide open. As compare to the other picture on the left side, which is very dark as compare to the first one, that shows that when that image was captured, its aperture was not wide open.

Size

Now let's discuss the math's behind the aperture. The size of the aperture is denoted by a f value. And it is inversely proportional to the opening of aperture.

Here are the two equations, that best explain this concept.

Large aperture size = Small f value

Small aperture size = Greater f value

Pictorially it can be represented as shown in figure 1.40:

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

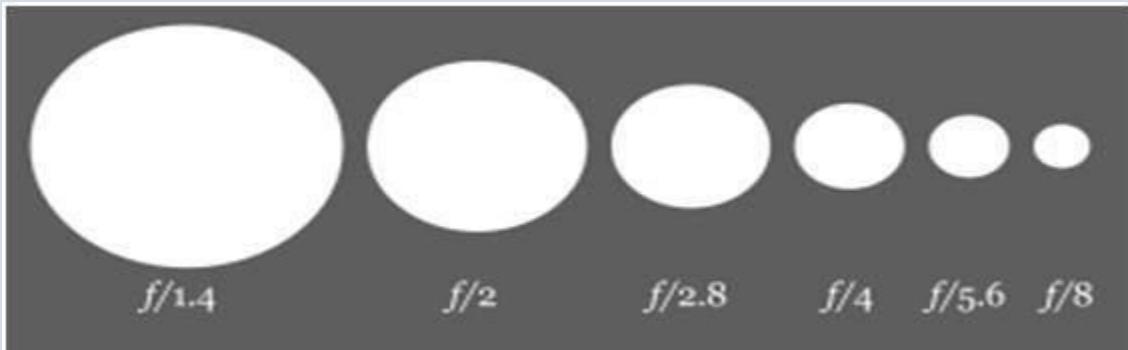


Figure 1.40: Pictorially of Aperture Size

Shutter

After the aperture, there comes the shutter. The light when allowed to pass from the aperture, falls directly on to the shutter. Shutter is actually a cover, a closed window, or can be thought of as a curtain. Remember when we talk about the CCD array sensor on which the image is formed. Well behind the shutter is the sensor. So shutter is the only thing that is between the image formation and the light, when it is passed from aperture.

As soon as the shutter is open, light falls on the image sensor, and the image is formed on the array.

Effect

If the shutter allows light to pass a bit longer, the image would be brighter.

Similarly, a darker picture is produced, when a shutter is allowed to move very quickly and hence, the light that is allowed to pass has very less photons, and the image that is formed on the CCD array sensor is very dark.

Shutter has further two main concepts:

- Shutter Speed
- Shutter time

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

Shutter speed

The shutter speed can be referred to as the number of times the shutter get open or close. Remember we are not talking about for how long the shutter get open or close.

Shutter time

The shutter time can be defined as

When the shutter is open, then the amount of wait time it takes till it is closed is called shutter time.

In this case we are not talking about how many times, the shutter got open or close, but we are talking about for how much time does it remain wide open.

For example:

We can better understand these two concepts in this way. That let's say that a shutter opens 15 times and then get closed, and for each time it opens for 1 second and then get closed. In this example, 15 is the shutter speed and 1 second is the shutter time.

Relationship

The relationship between shutter speed and shutter time is that they are both inversely proportional to each other.

This relationship can be defined in the equation below.

More shutter speed = less shutter time.

Less shutter speed = more shutter time.

Explanation:

The lesser the time required, the more is the speed. And the greater the time required, the less is the speed.

Applications

These two concepts together make a variety of applications. Some of them are given below.

Fast moving objects:

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

If you were to capture the image of a fast moving object, could be a car or anything.

The adjustment of shutter speed and its time would effect a lot.

So, in order to capture an image like this, we will make two amendments:

- Increase shutter speed
- Decrease shutter time

What happens is, that when we increase shutter speed, the more number of times, the shutter would open or close. It means different samples of light would allow to pass in. And when we decrease shutter time, it means we will immediately capture the scene, and close the shutter gate.

If you will do this, you get a crisp image of a fast moving object.

In order to understand it, we will look at this example. Suppose you want to capture the image of fast moving water fall.

You set your shutter speed to 1 second and you capture a photo. This is what you get



Then you set your shutter speed to a faster speed and you get.

LESSON#1 IMAGE PROCESSING & COMPUTER VISION



Then again you set your shutter speed to even more faster and you get.



You can see in the last picture, that we have increase our shutter speed to very fast, that means that a shutter gets opened or closed in 200th of 1 second and so we got a crisp image.

ISO

ISO factor is measured in numbers. It denotes the sensitivity of light to camera. If ISO number is lowered, it means our camera is less sensitive to light and if the ISO number is high, it means it is more sensitive.

Effect

The higher is the ISO, the more brighter the picture would be. IF ISO is set to 1600, the picture would be very brighter and vice versa.

Side effect

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

If the ISO increases, the noise in the image also increases. Today most of the camera manufacturing companies are working on removing the noise from the image when ISO is set to higher speed.

➤ Concept of Pixel

Pixel

Pixel is the smallest element of an image. Each pixel corresponds to any one value. In an 8-bit gray scale image, the value of the pixel between 0 and 255. The value of a pixel at any point correspond to the intensity of the light photons striking at that point. Each pixel stores a value proportional to the light intensity at that particular location.

PEL

A pixel is also known as PEL. You can have more understanding of the pixel from the pictures given below in figure 1.41.

In the above picture, there may be thousands of pixels, that together make up this image. We will zoom that image to the extent that we are able to see some pixels' division. It is shown in the image below in figure 1.41.



Figure 1.41: Pixels Division

Relationship with CCD array

We have seen that how an image is formed in the CCD array. So a pixel can also be defined as

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

The smallest division the CCD array is also known as pixel.

Each division of CCD array contains the value against the intensity of the photon striking to it. This value can also be called as a pixel.

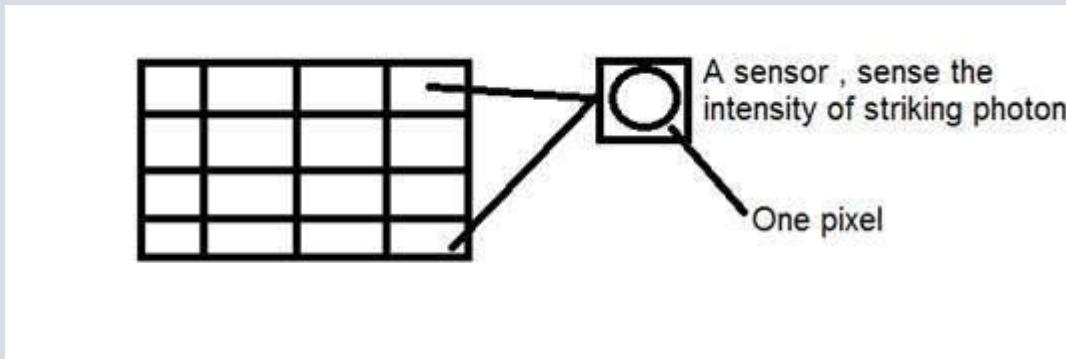


Figure 1.42: CCD Array

Calculation of total number of pixels

We have defined an image as a two dimensional signal or matrix. Then in that case the number of PEL would be equal to the number of rows multiply with number of columns.

This can be mathematically represented as below:

Total number of pixels = number of rows (X) number of columns

Or we can say that the number of (x, y) coordinate pairs make up the total number of pixels.

Gray level

The value of the pixel at any point denotes the intensity of image at that location, and that is also known as gray level.

We will see in more detail about the value of the pixels in the image storage and bits per pixel tutorial, but for now we will just look at the concept of only one-pixel value.

Pixel value. (0)

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

As it has already been defining that each pixel can have only one value and each value denotes the intensity of light at that point of the image. We will now look at a very unique value 0. The value 0 means absence of light. It means that 0 denotes dark, and it further means that whenever a pixel has a value of 0, it means at that point, black color would be formed.

Have a look at this image matrix

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

Now this image matrix has all filled up with 0. All the pixels have a value of 0. If we were to calculate the total number of pixels form this matrix, this is how we are going to do it.

Total no of pixels = total no. of rows X total no. of columns

$$= 3 \times 3$$

$$= 9.$$

It means that an image would be formed with 9 pixels, and that image would have a dimension of 3 rows and 3 columns and most importantly that image would be black. The resulting image that would be made would be something like this



Now why is this image all black. Because all the pixels in the image had a value of 0.

➤ Perspective Transformation

When human eyes see near things they look bigger as compare to those who are far away. This is called perspective in a general way. Whereas transformation is the

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

transfer of an object etc. from one state to another. So overall, the perspective transformation deals with the conversion of 3d world into 2d image. The same principle on which human vision works and the same principle on which the camera works. We will see in detail about why this happens, that those objects which are near to you look bigger, while those who are far away, look smaller even though they look bigger when you reach them.

We will start this discussion by the concept of frame of reference:

Frame of reference

Frame of reference is basically a set of values in relation to which we measure something, as shown in figure 1.43.

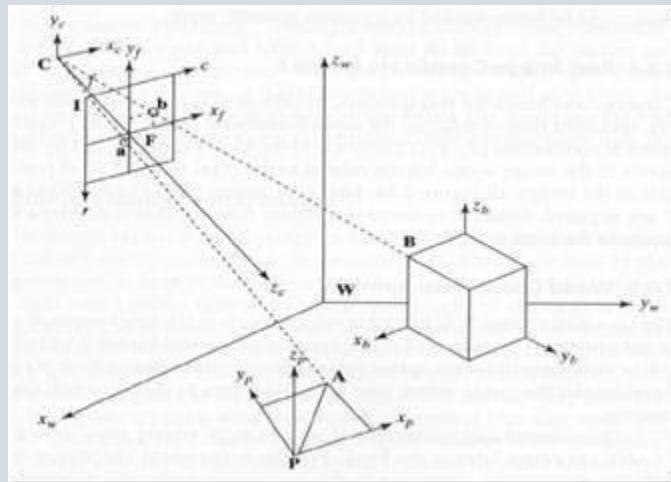


Figure 1.43: Frame of Reference

5 frames of reference

In order to analyze a 3d world/image/scene, 5 different frame of references are required.

- Object
- World
- Camera
- Image

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

- Pixel

Object coordinate frame

Object coordinate frame is used for modeling objects. For example, checking if a particular object is in a proper place with respect to the other object. It is a 3d coordinate system.

World coordinate frame

World coordinate frame is used for co-relating objects in a 3 dimensional world. It is a 3d coordinate system.

Camera coordinate frame

Camera co-ordinate frame is used to relate objects with respect of the camera. It is a 3d coordinate system.

Image coordinate frame

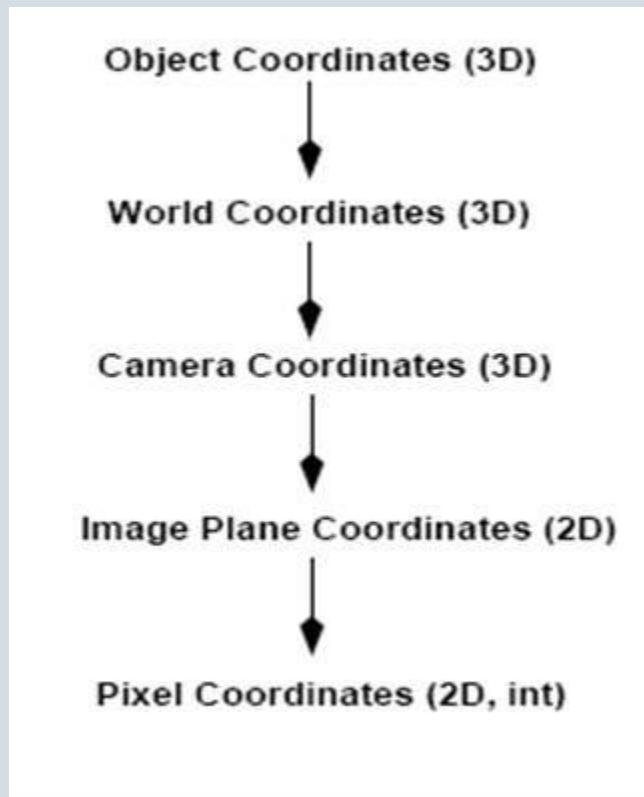
It is not a 3d coordinate system, rather it is a 2d system. It is used to describe how 3d points are mapped in a 2d image plane.

Pixel coordinate frame

It is also a 2d coordinate system. Each pixel has a value of pixel co-ordinates.

Transformation between these 5 frames

LESSON#1 IMAGE PROCESSING & COMPUTER VISION



That's how a 3d scene is transformed into 2d, with image of pixels.

Now we will explain this concept mathematically, as shown in figure 1.44.

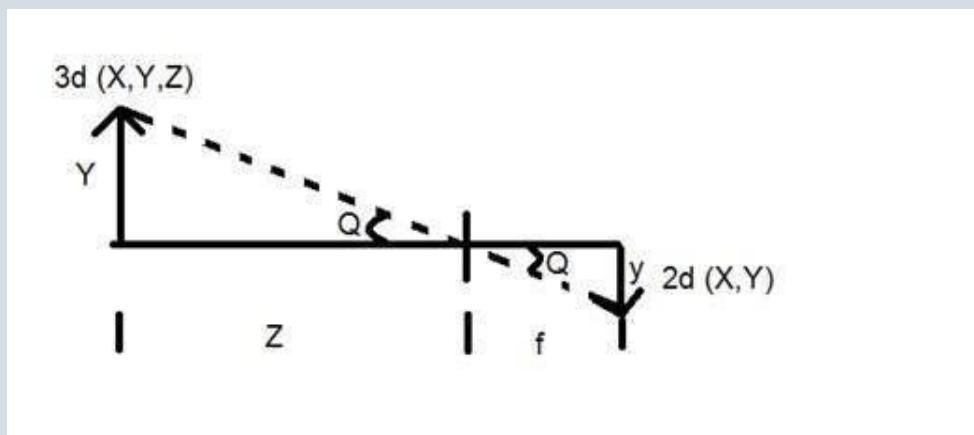


Figure 1.44: Transform 3d to 2d mathematically

Where

Y = 3d object

y = 2d Image

f = focal length of the camera

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

Z = distance between object and the camera

Now there are two different angles formed in this transform which are represented by Q.

The first angle is

$$\tan \theta = -\frac{y}{f}$$

Where minus denotes that image is inverted. The second angle that is formed is:

$$\tan \theta = \frac{Y}{Z}$$

Comparing these two equations we get

$$Y = -f \frac{Y}{Z}$$

From this equation, we can see that when the rays of light reflect back after striking from the object, passed from the camera, an invert image is formed.

We can better understand this, with this example.

For example

Calculating the size of image formed

Suppose an image has been taken of a person 5m tall, and standing at a distance of 50m from the camera, and we have to tell that what is the size of the image of the person, with a camera of focal length is 50mm.

Solution:

Since the focal length is in millimeter, so we have to convert everything in millimeter in order to calculate it.

So,

$Y = 5000 \text{ mm.}$

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

$f = 50 \text{ mm.}$

$Z = 50000 \text{ mm.}$

Putting the values in the formula, we get

$$Y = -f \frac{Y}{Z} = -50 \times 5000 / 50000$$

$= -5 \text{ mm.}$

Again, the minus sign indicates that the image is inverted.

➤ Concept of Bits Per Pixel

Bpp or bits per pixel denotes the number of bits per pixel. The number of different colors in an image depends on the depth of color or bits per pixel.

Bits in mathematics:

It's just like playing with binary bits.

How many numbers can be represented by one bit?

0

1

How many two bits' combinations can be made.

00

01

10

11

If we devise a formula for the calculation of total number of combinations that can be made from bit, it would be like this.

$$(2)^{\text{bpp}}$$

Where bpp denotes bits per pixel. Put 1 in the formula you get 2, put 2 in the formula, you get 4. It grows exponentially.

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

Number of different colors:

Now as we said it in the beginning, that the number of different colors depend on the number of bits per pixel.

The table for some of the bits and their color is given below.

| Bits per pixel | Number of colors |
|----------------|---|
| 1 bpp | 2 colors |
| 2 bpp | 4 colors |
| 3 bpp | 8 colors |
| 4 bpp | 16 colors |
| 5 bpp | 32 colors |
| 6 bpp | 64 colors |
| 7 bpp | 128 colors |
| 8 bpp | 256 colors |
| 10 bpp | 1024 colors |
| 16 bpp | 65536 colors |
| 24 bpp | 16777216 colors (16.7 million colors) |
| 32 bpp | 4294967296 colors (4294 million colors) |

This table shows different bits per pixel and the amount of color they contain.

Shades

You can easily notice the pattern of the exponential growth. The famous gray scale image is of 8 bpp , means it has 256 different colors in it or 256 shades.

Shades can be represented as:

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

$$\text{Shades} = \text{number of colors} = (2)^{\text{bpp}}$$

Color images are usually of the 24 bpp format, or 16 bpp.

We will see more about other color formats and image types in the tutorial of image types.

Color values:

We have previously seen in the tutorial of concept of pixel, that 0-pixel value denotes black color.

Black color:

Remember, 0-pixel value always denotes black color. But there is no fixed value that denotes white color.

White color:

The value that denotes white color can be calculated as:

$$\text{White color} = (2)^{\text{bpp}} - 1$$

In case of 1 bpp, 0 denotes black, and 1 denotes white.

In case 8 bpp, 0 denotes black, and 255 denotes white.

Gray color:

When you calculate the black and white color value, then you can calculate the pixel value of gray color.

Gray color is actually the midpoint of black and white. That said,

In case of 8bpp, the pixel value that denotes gray color is 127 or 128bpp (if you count from 1, not from 0).

Image storage requirements

After the discussion of bits per pixel, now we have everything that we need to calculate a size of an image.

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

Image size

The size of an image depends upon three things.

- Number of rows
- Number of columns
- Number of bits per pixel

The formula for calculating the size is given below.

Size of an image = rows * cols * bpp

It means that if you have an image, let's say this one in figure 1.45:

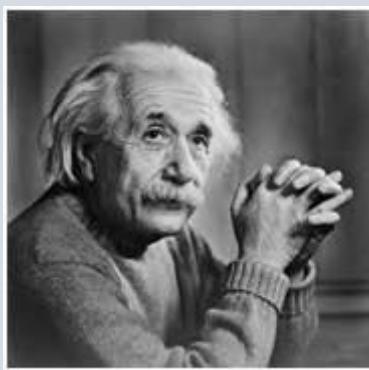


Figure 1.45: image

Assuming it has 1024 rows and it has 1024 columns. And since it is a gray scale image, it has 256 different shades of gray or it has bits per pixel. Then putting these values in the formula, we get

Size of an image = rows * cols * bpp

$$= 1024 * 1024 * 8$$

$$= 8388608 \text{ bits.}$$

But since it's not a standard answer that we recognize, so will convert it into our format.

Converting it into bytes = $8388608 / 8 = 1048576$ bytes.

Converting into kilo bytes = $1048576 / 1024 = 1024$ kb.

Converting into Megabytes = $1024 / 1024 = 1$ Mb.

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

That's how an image size is calculated and it is stored. Now in the formula, if you are given the size of image and the bits per pixel, you can also calculate the rows and columns of the image, provided the image is square (same rows and same column).

➤ Types of Images

There are many type of images, and we will look in detail about different types of images, and the color distribution in them.

The binary image

The binary image as it names states, contain only two pixel values 0 and 1.

Here 0 refers to black color and 1 refers to white color. It is also known as Monochrome.

Black and white image:

The resulting image that is formed hence consist of only black and white color and thus can also be called as Black and White image, as shown in figure 1.46.

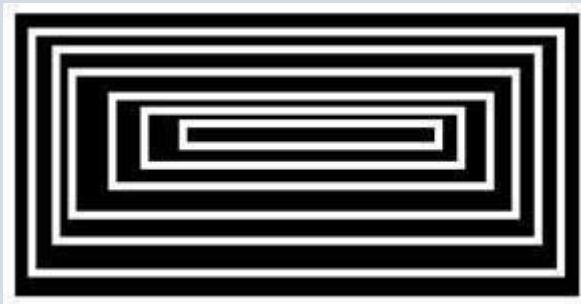


Figure 1.46: Black and White Image

No gray level

One of the interesting this about this binary image that there is no gray level in it.

Only two colors that are black and white are found in it.

Format

Binary images have a format of PBM (Portable bit map)

2, 3, 4,5, 6 bit color format

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

The images with a color format of 2, 3, 4, 5 and 6 bit are not widely used today.

They were used in old times for old TV displays, or monitor displays.

But each of these colors have more than two gray levels, and hence has gray color unlike the binary image.

In a 2 bit 4, in a 3 bit 8, in a 4 bit 16, in a 5 bit 32, in a 6 bit 64 different colors are present.

8 bit color format

8-bit color format is one of the most famous image format. It has 256 different shades of colors in it. It is commonly known as Grayscale image.

The range of the colors in 8 bit vary from 0-255. Where 0 stands for black, and 255 stands for white, and 127 stands for gray color.

This format was used initially by early models of the operating systems UNIX and the early color Macintoshes.

A grayscale image of Einstein is shown in figure 1.47 below:

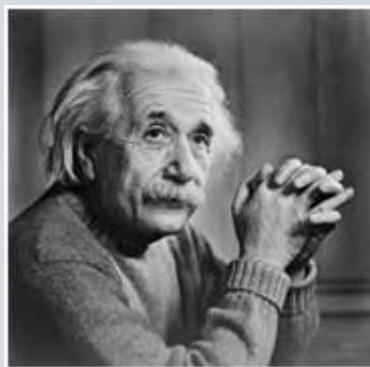


Figure 1.47: Grayscale image

Format

The format of these images are PGM (Portable Gray Map).

This format is not supported by default from windows. In order to see gray scale image, you need to have an image viewer or image processing toolbox such as Matlab.

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

Behind gray scale image:

As we have explained it several times in the previous tutorials, that an image is nothing but a two dimensional function, and can be represented by a two dimensional array or matrix. So in the case of the image of Einstein shown above, there would be two dimensional matrices in behind with values ranging between 0 and 255.

But that's not the case with the color images.

16 bit color format

It is a color image format. It has 65,536 different colors in it. It is also known as High color format.

It has been used by Microsoft in their systems that support more than 8-bit color format. Now in this 16 bit format and the next format we are going to discuss which is a 24 bit format are both color format.

The distribution of color in a color image is not as simple as it was in grayscale image.

A 16-bit format is actually divided into three further formats which are Red, Green and Blue. The famous (RGB) format.

It is pictorially represented in the image in figure 1.48 below.

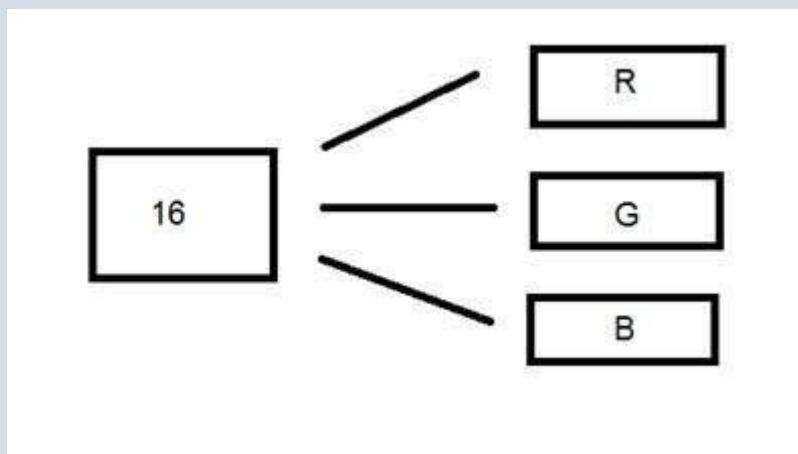


Figure 1.48: Grayscale Image

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

Now the question arises, that how would you distribute 16 into three. If you do it like this,

5 bits for R, 5 bits for G, 5 bits for B

Then there is one bit remains in the end.

So the distribution of 16 bit has been done like this.

5 bits for R, 6 bits for G, 5 bits for B.

The additional bit that was left behind is added into the green bit. Because green is the color which is most soothing to eyes in all of these three colors.

Note this is distribution is not followed by all the systems. Some have introduced an alpha channel in the 16 bit.

Another distribution of 16-bit format is like this:

4 bits for R, 4 bits for G, 4 bits for B, 4 bits for alpha channel.

Or some distribute it like this

5 bits for R, 5 bits for G, 5 bits for B, 1 bit for alpha channel.

24-bit color format

24-bit color format also known as true color format. Like 16-bit color format, in a 24-bit color format, the 24 bits are again distributed in three different formats of Red, Green and Blue, as shown in figure 1.49.

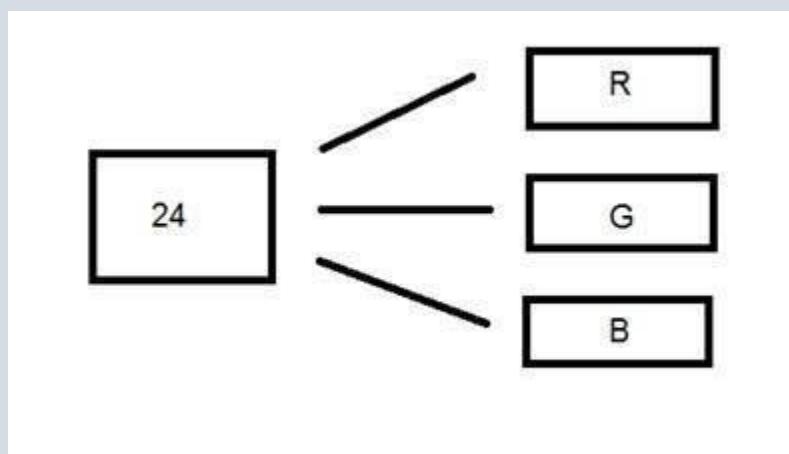


Figure 1.49: 24 Bits Image

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

Since 24 is equally divided on 8, so it has been distributed equally between three different color channels.

Their distribution is like this.

8 bits for R, 8 bits for G, 8 bits for B.

Behind a 24-bit image.

Unlike an 8-bit gray scale image, which has one matrix behind it, a 24-bit image has three different matrices of R, G, B. in figure 1.50 that shown the RGB image.



Figure 1.50: RGB Image

Format

It is the most common used format. Its format is PPM (Portable pixMap) which is supported by Linux operating system. The famous windows have its own format for it which is BMP (Bitmap).

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

Computer Vision and Computer Graphics

➤ Computer Vision

Computer vision is concerned with modeling and replicating human vision using computer software and hardware. Formally if we define computer vision then its definition would be that computer vision is a discipline that studies how to reconstruct, interrupt and understand a 3d scene from its 2d images in terms of the properties of the structure present in scene.

It needs knowledge from the following fields in order to understand and stimulate the operation of human vision system.

- Computer Science
- Electrical Engineering
- Mathematics
- Physiology
- Biology
- Cognitive Science

Computer Vision Hierarchy

- Computer vision is divided into three basic categories that are as following:
- Low-level vision: includes process image for feature extraction.
- Intermediate-level vision: includes object recognition and 3D scene Interpretation
- High-level vision: includes conceptual description of a scene like activity, intention and behavior.

Related Fields

Computer Vision overlaps significantly with the following fields:

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

Image Processing: it focuses on image manipulation.

Pattern Recognition: it studies various techniques to classify patterns.

Photogrammetry: it is concerned with obtaining accurate measurements from images.

Computer Vision Vs Image Processing

Image processing studies image to image transformation. The input and output of image processing are both images.

Computer vision is the construction of explicit, meaningful descriptions of physical objects from their image. The output of computer vision is a description or an interpretation of structures in 3D scene.

Example Applications

- Robotics
- Medicine
- Security
- Transportation
- Industrial Automation

Robotics Application

Localization-determine robot location automatically

Navigation

Obstacles avoidance

Assembly (peg-in-hole, welding, painting)

Manipulation (e.g. PUMA robot manipulator)

Human Robot Interaction (HRI): Intelligent robotics to interact with and serve people

Medicine Application

Classification and detection (e.g. lesion or cells classification and tumor detection)

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

2D/3D segmentation

3D human organ reconstruction (MRI or ultrasound)

Vision-guided robotics surgery

Industrial Automation Application

Industrial inspection (defect detection)

Assembly

Barcode and package label reading

Object sorting

Document understanding (e.g. OCR)

Security Application

Biometrics (iris, finger print, face recognition)

Surveillance-detecting certain suspicious activities or behaviors

Transportation Application

Autonomous vehicle

Safety, e.g., driver vigilance monitoring

Computer Graphics

Computer graphics are graphics created using computers and the representation of image data by a computer specifically with help from specialized graphic hardware and software. Formally we can say that Computer graphics is creation, manipulation and storage of geometric objects (modeling) and their images (Rendering).

The field of computer graphics developed with the emergence of computer graphics hardware. Today computer graphics is used in almost every field. Many powerful tools have been developed to visualize data. Computer graphics field became more popular when companies started using it in video games. Today it is a multibillion-dollar industry and main driving force behind the computer graphics development.

Some common applications areas are as following:

LESSON#1 IMAGE PROCESSING & COMPUTER VISION

- Computer Aided Design (CAD)
- Presentation Graphics
- 3d Animation
- Education and training
- Graphical User Interfaces

Computer Aided Design

Used in design of buildings, automobiles, aircraft and many other products

Use to make virtual reality system.

Presentation Graphics

Commonly used to summarize financial, statistical data

Use to generate slides

3d Animation

Used heavily in the movie industry by companies such as Pixar, DreamWorks

To add special effects in games and movies.

Education and training

Computer generated models of physical systems

Medical Visualization

3D MRI

Dental and bone scans

Stimulators for training of pilots etc.

Graphical User Interfaces

It is used to make graphical user interfaces objects like buttons, icons and other components

Lesson#2 Image Processing & COMPUTER VISION

Introduction to MATLAB Program

Eng. Elaf A.Saeed

Email: elafe1888@gmail.com

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

➤ Introduction to MATLAB

❖ Introduction

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation.

The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix.

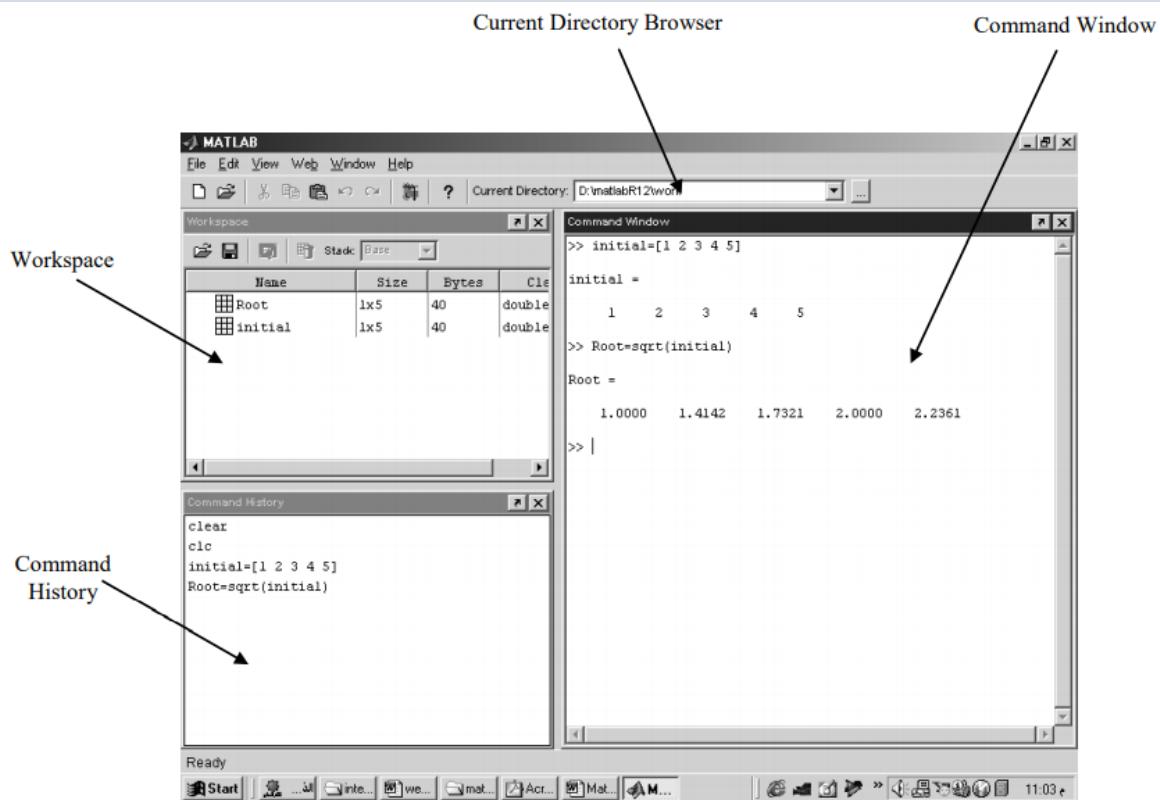
❖ Starting and Quitting MATLAB

- To start MATLAB, double-click the MATLAB shortcut icon  on your Windows desktop. You will know MATLAB is running when you see the special " >> " prompt in the MATLAB Command Window.
- To end your MATLAB session, select **Exit MATLAB** from the **File** menu in the desktop, or type **quit** (or **exit**) in the Command Window, or with easy way by click on close button  in control box.

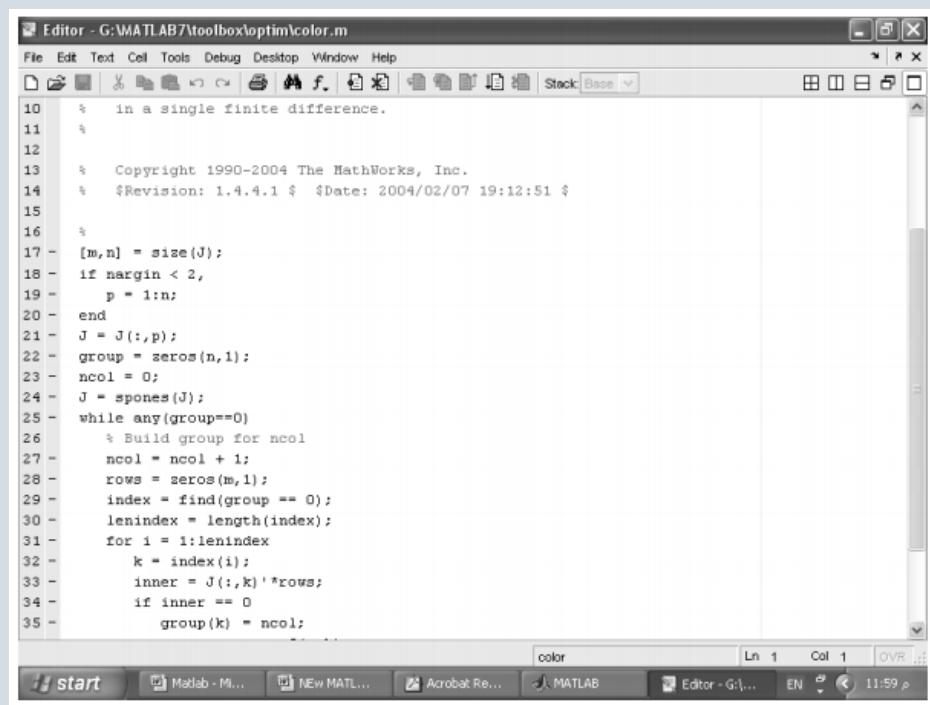
❖ Desktop Tools

- 1- **Command Window:** Use the Command Window to enter variables and run functions and M-files.
- 2- **Command History:** Statements you enter in the Command Window are logged in the Command History. In the Command History, you can view previously run statements, and copy and execute selected statements.
- 3- **Current Directory Browser:** MATLAB file operations use the current directory reference point. Any file you want to run must be in the current directory or on the search path.
- 4- **Workspace:** The MATLAB workspace consists of the set of variables (named arrays) built up during a MATLAB session and stored in memory.

LESSON#2 IMAGE PROCESSING & COMPUTER VISION



5- **Editor/Debugger Window:** Use the Editor/Debugger to create and debug M-files.



LESSON#2 IMAGE PROCESSING & COMPUTER VISION

❖ Basic Commands

- **clear Command:** Removes all variables from workspace.
- **clc Command:** Clears the Command window and homes the cursor.
- **help Command:** help <Topic> displays help about that Topic if it exist.
- **lookfor Command:** Provides help by searching through all the first lines of MATLAB help topics and returning those that contains a key word you specify.
- **edit Command:** enable you to edit (open) any M-file in Editor Window. This command doesn't open built-in function like, sqrt. See also type Command.
- **more command:** more on enables paging of the output in the MATLAB command window, and more off disables paging of the output in the MATLAB command window.

❖ Notes:

- A semicolon " ; " at the end of a MATLAB statement suppresses printing of results.

- If a statement does not fit on one line, use " . . . ", followed by Enter to indicate that the statement continues on the next line. For example:

```
>> S= sqrt (225)*30 /...
```

```
(20*sqrt (100)
```

- If we don't specify an output variable, MATLAB uses the variable ans (short for answer), to store the last results of a calculation.

- Use Up arrow and Down arrow to edit previous commands you entered in Command Window.

- Insert " % " before the statement that you want to use it as comment; the statement will appear in green color.

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

Now Try to do the following:

>> a=3

>> a=3; can you see the effect of semicolon " ; "

>> a+5 assign the sum of a and 5 to ans

>> b=a+5 assign the sum of a and 5 to b

>> clear a

>> a can you see the effect of clear command

>> clc clean the screen

>> b

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

➤ Working with Matrices

❖ Entering Matrix

The best way for you to get started with MATLAB is to learn how to handle matrices.

You only have to follow a few basic conventions:

- Separate the elements of a row with blanks or commas.
- Use a semicolon (;) to indicate the end of each row.
- Surround the entire list of elements with square brackets, [].

For Example

```
>> A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
```

MATLAB displays the matrix you just entered.

A =

| | | | |
|----|----|----|----|
| 16 | 3 | 2 | 13 |
| 5 | 10 | 11 | 8 |
| 9 | 6 | 7 | 12 |
| 4 | 15 | 14 | 1 |

Once you have entered the matrix, it is automatically remembered in the MATLAB workspace. You can refer to it simply as A. Also you can enter and change the values of matrix elements by using workspace window.

❖ Subscripts

The element in row i and column j of A is denoted by A(i,j). For example, A(4,2) is the number in the fourth row and second column. For the above matrix, A(4,2) is 15.

So to compute the sum of the elements in the fourth column of A, type

```
>> A(1,4) + A(2,4) + A(3,4) + A(4,4)
```

ans =

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

You can do the above summation, in simple way by using sum command.

If you try to use the value of an element outside of the matrix, it is an error.

```
>> t = A(4,5)
```

??? Index exceeds matrix dimensions.

On the other hand, if you store a value in an element outside of the matrix, the size increases to accommodate the newcomer. The initial values of other new elements are zeros.

```
>> X = A;
```

```
>> X(4,5) = 17
```

X =

| | | | | |
|----|----|----|----|----|
| 16 | 3 | 2 | 13 | 0 |
| 5 | 10 | 11 | 8 | 0 |
| 9 | 6 | 7 | 12 | 0 |
| 4 | 15 | 14 | 1 | 17 |

❖ Colon Operator

The colon " :" is one of the most important MATLAB operators. It occurs in several different forms. The expression

```
>> 1:10
```

is a row vector containing the integers from 1 to 10

1 2 3 4 5 6 7 8 9 10

To obtain nonunit spacing, specify an increment. For example,

```
>> 100: -7: 50
```

100 93 86 79 72 65 58 51

Subscript expressions involving colons refer to portions of a matrix.

```
>>A (1: k, j)
```

is the first k elements of the jth column of A.

The colon by itself refers to all the elements in a row or column of a matrix and the

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

keyword end refers to the last row or column. So

>> A (4,:) or >> A(4,1:end) give the same action

ans =

4 15 14 1

>> A (2, end)

ans =

8

❖ Basic Matrix Functions

| Command | Description |
|--|--|
| sum(x) >> x=[1 2 3 4 5 6]; >> sum(x) ans = 5 7 9 >> sum(x,2) ans= 6 15 >>sum(sum(x)) ans = 21 | The sum of the elements of x. For matrices, sum(x) is a row vector with the sum over each column. sum (x,dim) sums along the dimension dim. In order to find the sum of elements that are stored in matrix with n dimensions, you must use sum command n times in cascade form, this is also applicable for max, min, prod, mean, median commands. |

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

| | |
|---|---|
| mean(x) x=[1 2 3; 4 5 6]; >> mean(x) ans = 2.5 3.5 4.5 >> mean(x,2) ans = 2 5 >>mean(mean(x)) ans = 3.5000 | The average of the elements of x. For matrices, mean(x) is a row vector with the average over each column. mean (x,dim) averages along the dimension dim. |
| zeros(N) zeros(N,M) >> zeros(2,3) ans = 0 0 0 0 0 0 | Produce N by N matrix of zeros. Produce N by M matrix of zeros |
| ones(N) ones(N,M) >> ones(2,3) ans = 1 1 1 1 1 1 | Produce N by N matrix of ones. Produce N by M matrix of ones. |

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

| | |
|--|---|
| size(x) <pre>>> x=[1 2 3 4 5 6];</pre> ans = <pre>2 3</pre> | return the size (dimensions) of matrix x. |
| length(v) <pre>>> v=[1 2 3];</pre> ans = <pre>3</pre> | return the length (number of elements) of vector v. |
| numel(x) <pre>>> v=[55 63 34];</pre> ans = <pre>3</pre> <pre>>> x=[1 2 4 5 7 8];</pre> ans = <pre>6</pre> | returns the number of elements in array x. |
| single quote (') <pre>>> x=[1 2 3 4 5 6</pre> | Matrix transpose. It flips a matrix about its main diagonal and it turns a row vector into a column vector. |

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

```
7 8 9];  
>> x'  
ans =  
1 4 7  
2 5 8  
3 6 9  
>> v=[1 2 3];  
>> v'  
ans =  
1  
2  
3
```

max (x)
>> x=[1 2 3
4 5 6];
>> max (x)
ans =
4 5 6
>> max(max(x))
ans =
6

min (x)
>> x=[1 2 3
4 5 6];
>> min (x)
ans =

Find the largest element in a matrix or a vector.

Find the smallest element in a matrix or a vector.

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

| | | | | | | | | | | |
|---|---------|--|--------|---------|----------------------------------|---|---|---|---|---|
| <pre>1 2 3 >> min(min(x)) ans = 1</pre> | | | | | | | | | | |
| <pre>magic(N) >> magic(3) ans =</pre> <table style="margin-left: auto; margin-right: auto;"><tr><td>8</td><td>1</td><td>6</td></tr><tr><td>3</td><td>5</td><td>7</td></tr><tr><td>4</td><td>9</td><td>2</td></tr></table> | 8 | 1 | 6 | 3 | 5 | 7 | 4 | 9 | 2 | produce N Magic square. This command produces valid magic squares for all N>0 except N=2. |
| 8 | 1 | 6 | | | | | | | | |
| 3 | 5 | 7 | | | | | | | | |
| 4 | 9 | 2 | | | | | | | | |
| <pre>inv(x) >> x= [1 4; 5 8]; >> inv(x) ans =</pre> <table style="margin-left: auto; margin-right: auto;"><tr><td>-0.6667</td><td>0.3333</td></tr><tr><td>0.4167</td><td>-0.0833</td></tr></table> | -0.6667 | 0.3333 | 0.4167 | -0.0833 | produce the inverse of matrix x. | | | | | |
| -0.6667 | 0.3333 | | | | | | | | | |
| 0.4167 | -0.0833 | | | | | | | | | |
| <pre>diag(x) >> x= [1 2 3 4 5 6 7 8 9]; >> diag(x) ans =</pre> <table style="margin-left: auto; margin-right: auto;"><tr><td>1</td></tr></table> | 1 | Return the diagonal of matrix x. if x is a vector then this command produce a diagonal matrix with diagonal x. | | | | | | | | |
| 1 | | | | | | | | | | |

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

| | |
|---|--|
| <pre>5 9 >> v=[1 2 3]; >> diag(v) ans = 1 0 0 0 2 0 0 0 3</pre> | |
| prod(x) <pre>>> x= [1 2 3 4 5 6]; >> prod(x) ans = 4 10 18 >> prod(prod(x)) ans = 720</pre> | Product of the elements of x. For matrices, Prod(x) is a row vector with the product over each column. |
| median(x) <pre>x= [4 6 8 10 9 1 8 2 5]; >> median(x) ans = 8 6 5 >> median(x,2) ans =</pre> | The median value of the elements of x. For matrices, median (x) is a row vector with the median value for each column. median(x,dim) takes the median along the dimension dim of x. |

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

| | |
|---|---|
| <pre>6 9 5 >> median(median(x)) ans = 6</pre> | |
| <pre>sort(x,DIM,MODE) >> x = [3 7 5 0 4 2]; >> sort(x,1) ans = 0 4 2 3 7 5 >> sort(x,2) ans = 3 5 7 0 2 4 >> sort(x,2,'descend') ans = 7 5 3 4 2 0</pre> | <p>Sort in ascending or descending order.</p> <ul style="list-style-type: none">- For vectors, sort(x) sorts the elements of x in ascending order.For matrices, sort(x) sorts each column of x in ascending order. <p>DIM= 1 by default</p> <p>MODE= 'ascend' by default</p> |
| <pre>det(x) >> x= [5 1 8 4 7 3 2 5 6]; >> det(x)</pre> | Det is the determinant of the square matrix x. |

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

| | | | | | | | | | | |
|--|---|---|---|---|---|---|---|---|---|--|
| ans = 165 | | | | | | | | | | |
| tril(x) <code>>> x= [5 1 8 4 7 3 2 5 6];</code> >> tril(x) ans = <table><tr><td>5</td><td>0</td><td>0</td></tr><tr><td>4</td><td>7</td><td>0</td></tr><tr><td>2</td><td>5</td><td>6</td></tr></table> | 5 | 0 | 0 | 4 | 7 | 0 | 2 | 5 | 6 | Extract lower triangular part of matrix x. |
| 5 | 0 | 0 | | | | | | | | |
| 4 | 7 | 0 | | | | | | | | |
| 2 | 5 | 6 | | | | | | | | |
| triu(x) <code>>> x= [5 1 8 4 7 3 2 5 6];</code> >> triu(x) ans = <table><tr><td>5</td><td>1</td><td>8</td></tr><tr><td>0</td><td>7</td><td>3</td></tr><tr><td>0</td><td>0</td><td>6</td></tr></table> | 5 | 1 | 8 | 0 | 7 | 3 | 0 | 0 | 6 | Extract upper triangular part of matrix x. |
| 5 | 1 | 8 | | | | | | | | |
| 0 | 7 | 3 | | | | | | | | |
| 0 | 0 | 6 | | | | | | | | |

❖ Note

When we are taken away from the world of linear algebra, matrices become two-dimensional numeric arrays. Arithmetic operations on arrays are done element-by-element. This means that addition and subtraction are the same for arrays and

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

matrices, but that multiplicative operations are different. MATLAB uses a dot (.), or decimal point, as part of the notation for multiplicative array operations.

Example: Find the factorial of 5

```
>> x=2:5;
```

```
>> prod(x)
```

Example: if $x = [1,5,7,9,13,20,6,7,8]$, then

- replace the first five elements of vector x with its maximum value.
- reshape this vector into a 3 x 3 matrix.

solution

a)

```
>> x(1:5)=max(x)
```

b)

```
>> y(1,:)=x(1:3);
```

```
>> y(2,:)=x(4:6);
```

```
>> y(3,:)=x(7:9);
```

```
>> y
```

Example: Generate the following row vector $b = [1, 2, 3, 4, 5, \dots, 9, 10]$,

then transpose it to column vector.

solution

```
>> b=1:10
```

$b =$

1 2 3 4 5 6 7 8 9 10

```
>> b=b';
```

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

➤ Expressions

Like most other programming languages, MATLAB provides mathematical expressions, but unlike most programming languages, these expressions involve entire matrices. The building blocks of expressions are:

1- **Variable**

2- **Numbers**

3- **Operators**

4- **Functions**

1- Variable

MATLAB does not require any type declarations or dimension statements. When MATLAB encounters a new variable name, it automatically creates the variable and allocates the appropriate amount of storage. If the variable already exists, MATLAB changes its contents and, if necessary, allocates new storage. For example,

num_students = 25

creates a 1-by-1 matrix named **num_students** and stores the value 25 in its single element.

Variable names consist of a letter, followed by any number of letters, digits, or underscores. MATLAB uses only the first 31 characters of a variable name. MATLAB is case sensitive; it distinguishes between uppercase and lowercase letters. A and a are not the same variable.

2- Numbers

MATLAB uses conventional decimal notation, with an optional decimal point and leading plus or minus sign, for numbers. Scientific notation uses the letter (e) to specify a

power-of-ten scale factor. Imaginary numbers use either i or j as a suffix. Some examples of

legal numbers are

3 -99 0.0001 9.6397238 1.60210e-20
6.02252e23 1i 3+5j

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

3- Operators

| Operator | Description |
|----------|---------------------------------|
| + | Plus |
| - | Minus |
| * | Matrix multiply |
| .* | Array multiply |
| ^ | Matrix power |
| .^ | Array power |
| \ | Backslash or left matrix divide |
| / | Slash or right matrix divide |
| .\ | Left array divide |
| ./ | Right array divide |
| () | Specify evaluation order |

4- Functions

MATLAB provides a large number of standard elementary mathematical functions, including abs, sqrt, exp, and sin. Taking the square root or logarithm of a negative number is not an error; the appropriate complex result is produced automatically. MATLAB also provides many more advanced mathematical functions, including Bessel and gamma functions. Most of these functions accept complex arguments. For a list of the elementary mathematical functions, type

```
>> help elfun
```

For a list of more advanced mathematical and matrix functions, type

```
>> help specfun
```

```
>> help elmat
```

Some of the functions, like sqrt and sin, are built in. They are part of the MATLAB

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

core so they are very efficient, but the computational details are not readily accessible.

| Command | Description |
|-------------------------|---|
| <code>abs(x)</code> | Absolute value (magnitude of complex number). |
| <code>acos(x)</code> | Inverse cosine. |
| <code>angle(x)</code> | Phase angle (angle of complex number). |
| <code>asin(x)</code> | Inverse sine. |
| <code>atan(x)</code> | Inverse tangent. |
| <code>atan2(x,y)</code> | Four quadrant inverse tangent: $\tan^{-1}(x/y)$ |
| <code>ceil(x)</code> | Round towards plus infinity. |
| <code>conj(x)</code> | Complex conjugate. |
| <code>cos(x)</code> | Cosine of x, assumes radians. |
| <code>exp(x)</code> | Exponential: e^x . |
| <code>fix(x)</code> | Round towards zero. |
| <code>floor(x)</code> | Round towards minus infinity. |
| <code>imag(x)</code> | Complex imaginary part. |
| <code>log(x)</code> | Natural logarithm: $\ln(x)$. |
| <code>log10(x)</code> | Common (base 10) logarithm: $\log_{10}(x)$. |
| <code>log2(x)</code> | Base 2 logarithm: $\log_2(x)$. |
| <code>real(x)</code> | Complex real part. |
| <code>rem(x)</code> | Remainder after division. |
| <code>mod(x)</code> | Modulus after division. |

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

| | |
|-----------------------|--|
| <code>round(x)</code> | Round towards nearest integer. |
| <code>sign(x)</code> | Signum: return sign of argument. |
| <code>sin(x)</code> | Sine of x, assumes radians. |
| <code>sqrt(x)</code> | Square root. |
| <code>tan(x)</code> | Tangent of x, assumes radians. |
| <code>rand(n)</code> | returns an N-by-N matrix containing pseudorandom values drawn from the standard uniform distribution on the open interval(0,1). <code>rand(M,N)</code> returns an M-by-N matrix. <code>rand</code> returns a scalar. |
| <code>randn(n)</code> | returns an N-by-N matrix containing pseudorandom values drawn from the standard normal distribution on the open interval(0,1). <code>randn(M,N)</code> returns an M-by-N matrix. <code>randn</code> returns a scalar |

Several special functions provide values of useful constants.

| Command | Description |
|----------------------|--|
| <code>eps</code> | Floating point relative accuracy $\approx 2.2204e-016$. |
| <code>realmax</code> | Largest positive floating point number. |
| <code>realmin</code> | Smallest positive floating point number. |
| <code>pi</code> | 3.1415926535897.... |
| <code>i and j</code> | Imaginary unit $\sqrt{-1}$. |
| <code>inf</code> | Infinity, e.g. 1/0 |

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

NaN

Not A Number, e.g. 0/0

Examples of Expressions

1) $>> x = (1+\sqrt{5})/2$

$x =$

1.6180

$>> a = \text{abs}(3+4i)$

$a =$

5

$>> y=\sin(\pi/3) + \cos(\pi/4)-2 * \sqrt{3}$

$y =$

-1.8910

2) Solve the following system

$$x+y=1$$

$$x-y+z=0$$

$$x+y+z=2$$

Solution

$>> a=[1\ 1\ 0; 1\ -1\ 1; 1\ 1\ 1]; b=[1;0;2];$

$>> x=\text{inv}(a)*b$

or

$>> x=a\bslash b$

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

➤ Relational and Logical Operations

These operations and functions provide answers to True-False questions. One important use of this capability is to control the flow or order of execution of a series of MATLAB commands (usually in an M-file) based on the results of true/false questions. As inputs to all relational and logical expressions, MATLAB considers any nonzero number to be true, and zero to be False. The output of all relational and logical expressions produces one for True and zero for False, and the array is flagged as logical. That is, the result contains numerical values 1 and 0 that can be used in mathematical statement, but also allow logical array addressing.

❖ Relational Operations

| Operation | Description |
|-----------|-----------------------|
| $= =$ | Equal |
| $\sim =$ | Not equal |
| $<$ | Less than |
| $>$ | Greater than |
| $< =$ | Less than or equal |
| $> =$ | Greater than or equal |

Example:

```
>> a=1:9; b=9-a;  
>> t=a>4           %finds elements of (a) that are greater than 4.  
t =  
     0     0     0     0     1     1     1     1     1
```

Zeros appear where $a \leq 4$, and ones where $a > 4$.

```
>> t= (a==b)       %finds elements of (a) that are equal to those in (b).  
t =
```

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

0 0 0 0 0 0 0 0 0

❖ Logical Operation

| Operation | Description |
|-----------|----------------------|
| & | Logical AND and(a,b) |
| | Logical OR or(a,b) |
| ~ | Logical NOT |
| Xor (a,b) | Logical EXCLUSIVE OR |

Example:

```
>> a = [0 4 0 -3 -5 2];
>> b = ~a
b =
    1     0     1     0     0     0
>> c=a&b
c =
    0     0     0     0     0     0
```

Example: let $x = [2 \ -3 \ 5 ; 0 \ 11 \ 0]$, then

- find elements in x that are greater than 2
- find the number of nonzero elements in x

solution

a)

```
>> x>2
```

ans =

```
    0     0     1
    0     1     0
```

b)

```
>> t=~(~x);
>> sum(sum(t))
```

ans =

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

❖ Bitwise Operation

MATLAB also has a number of functions that perform bitwise logical operations.

If A, B unsigned integers then:

| Operation | Description |
|------------------------|--|
| bitand (A, B) | Bitwise AND. |
| bitor (A, B) | Bitwise OR. |
| bitset (A, BIT) | sets bit position BIT in A to 1. |
| bitget (A, BIT) | returns the value of the bit at position BIT in A. |
| xor (A, B) | Bitwise EXCLUSIVE OR. |

Example: if A=5, B=6 then:

>> bitget(A,3) → 5

ans = 1

Where A= 1 0 **1** 0 0 0 0 0

>> bitget(A ,(1:8))

ans =

1 0 1 0 0 0 0 0

>> bitand(A,B)

ans =

4

>> and(A,B)

ans =

1

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

❖ Logical Functions

MATLAB has a number of useful logical functions that operate on scalars, vectors, and matrices. Examples are given in the following list: -

| Function | Description |
|-------------------|---|
| any(x) | True if any element of a vector is a nonzero number or is logical 1 (TRUE). |
| all(x) | True if all elements of a vector are nonzero. |
| find(x) | Find indices of nonzero elements. |
| isnan(x) | True for Not-a-Number. |
| isinf(x) | True for infinite elements. |
| isempty(x) | True for empty array. |

Example: Let A= [4 9 7 0 5],

```
>> any(A)  
ans = 1  
>> all(A)  
ans = 0  
>> find(A)  
ans = 1 2 3 5
```

To remove zero elements from matrix

```
>> B=A(find(A));  
>> B  
B = 4 9 7 5
```

To find the location of maximum number of B

```
>> find(B==max(B)) ans = 2
```

➤ Plotting Function

MATLAB has extensive facilities for displaying vectors and matrices as graphs, as well as annotating and printing these graphs.

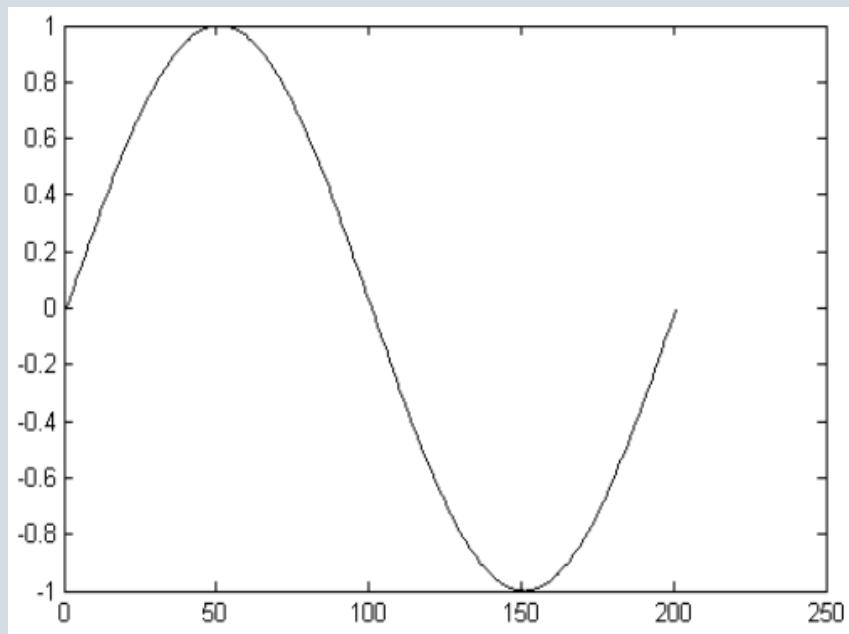
❖ Creating a Plot Using Plot Function

The plot function has different forms, depending on the input arguments. If y is a vector, $\text{plot}(y)$ produces a piecewise linear graph of the elements of y versus the index of the elements of y . If you specify two vectors as arguments, $\text{plot}(x,y)$ produces a graph of y versus x . For example, these statements use the colon operator to create a vector of x values ranging from zero to 2π , compute the sine of these values, and plot the result.

```
x = 0:pi/100:2*pi;
```

```
y = sin(x);
```

```
plot(y)
```

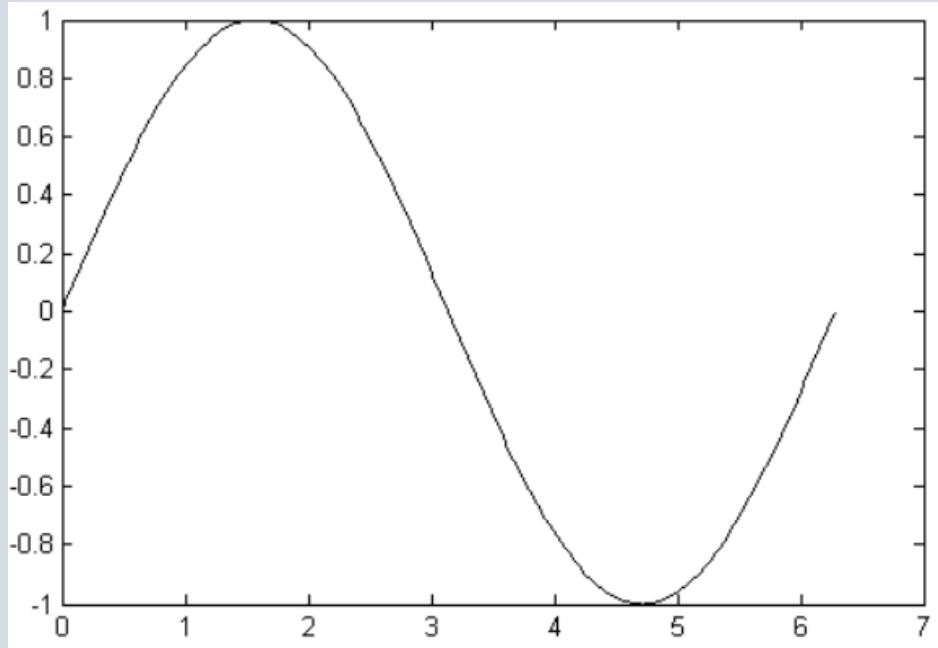


Now plot y variable by using:

```
plot(x,y)
```

can you see the difference at x-axis

LESSON#2 IMAGE PROCESSING & COMPUTER VISION



❖ Specifying Line Styles and Colors

Various line types, plot symbols and colors may be obtained with `plot(x,y,s)` where `s` is a character string made from one element from any or all the following 3 columns:

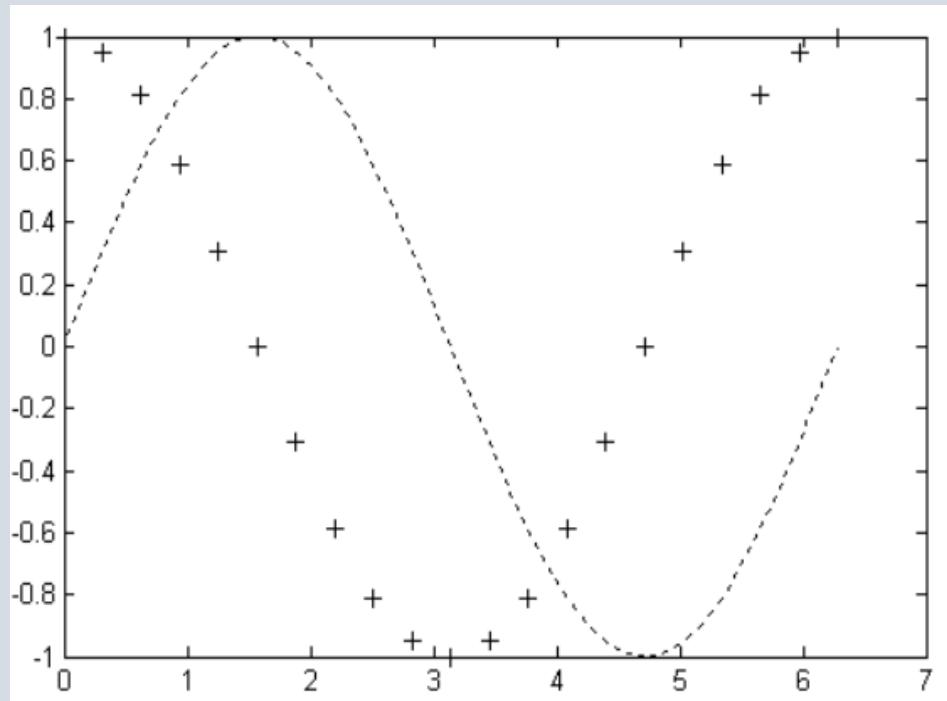
| Color | | Marker | | Line Style | |
|-------|---------|--------|--------------------|------------|----------|
| b | Blue | . | Point | - | solid |
| g | Green | o | Circle | : | dotted |
| r | Red | x | x-mark | -. | dash dot |
| c | Cyan | + | Plus | -- | dashed |
| m | Magenta | * | Star | (none) | no line |
| y | Yellow | s | Square | | |
| k | black | d | Diamond | | |
| | | v | Triangle (down) | | |

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

| | | | | | |
|--|--|---|--------------------|--|--|
| | | > | Triangle (left) | | |
| | | p | pentagram | | |
| | | h | hexagram | | |

Example:

```
x1 = 0:pi/100:2*pi;  
x2 = 0:pi/10:2*pi;  
plot(x1,sin(x1),'r:',x2,cos(x2),'r+')
```



❖ Adding Plots to an Existing Graph

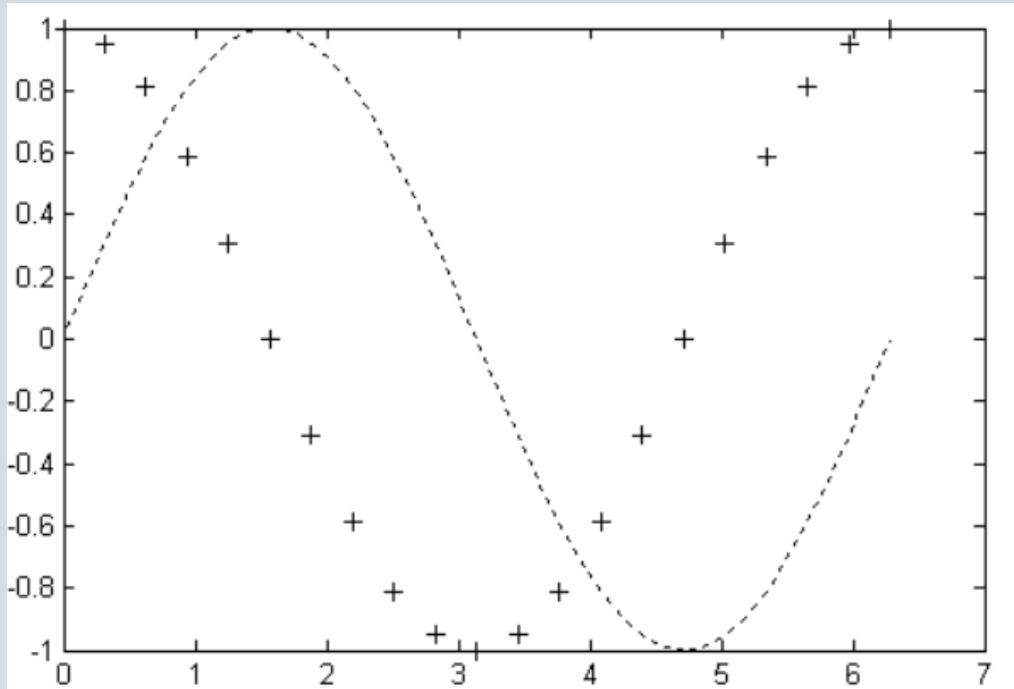
The **hold** command enables you to add plots to an existing graph. When you type **hold on** MATLAB does not replace the existing graph when you issue another plotting command; it adds the new data to the current graph, rescaling the axes if necessary.

Example:

```
x1 = 0:pi/100:2*pi;
```

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

```
x2 = 0:pi/10:2*pi;  
plot(x1,sin(x1),'r:')  
hold on  
plot(x2,cos(x2),'r+')
```



❖ Multiple Plots in One Figure

The subplot command enables you to display multiple plots in the same window or print them on the same piece of paper. Typing

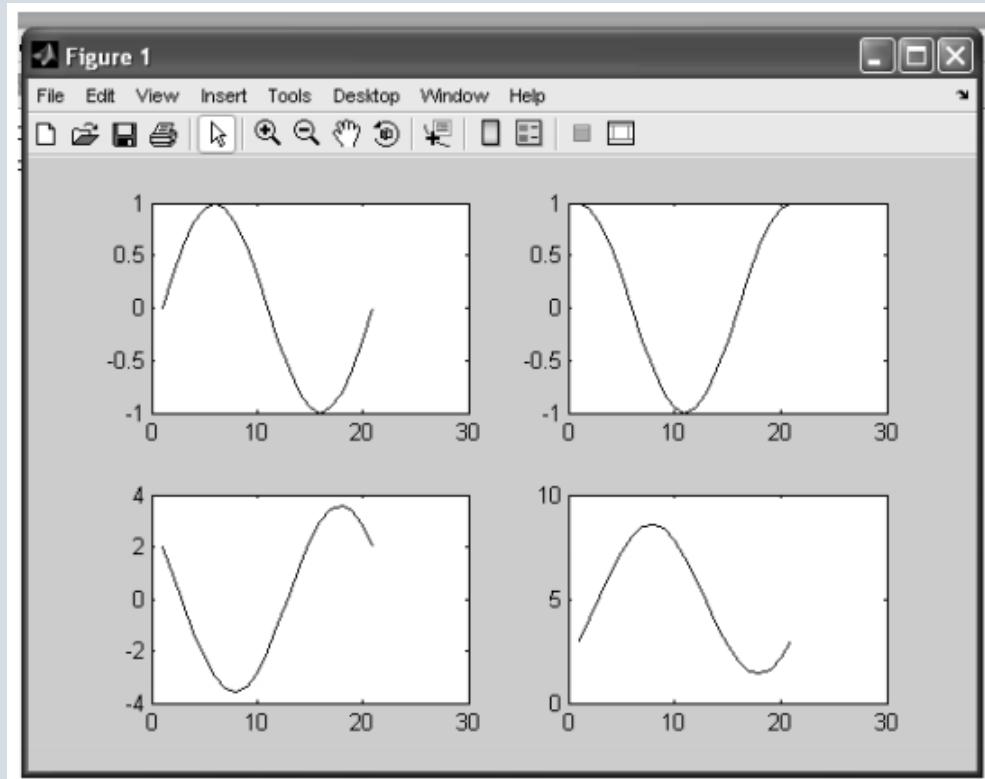
subplot (m, n, p)

partitions the figure window into an m-by-n matrix of small subplots and selects the pth subplot for the current plot. The plots are numbered along first the top row of the figure window, then the second row, and so on. For example, these statements plot data in four different sub regions of the figure window.

```
t = 0: pi/10:2*pi;  
x=sin(t); y=cos(t); z= 2*y-3*x; v=5-z;  
subplot (2,2,1); plot(x)
```

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

```
subplot (2,2,2); plot(y)  
subplot (2,2,3); plot(z)  
subplot (2,2,4); plot(v)
```



❖ Setting Axis Limits

By default, MATLAB finds the maxima and minima of the data to choose the axis limits to span this range. The axis command enables you to specify your own limits
axis([xmin xmax ymin ymax])

❖ Axis Labels and Titles

The xlabel, ylabel, and zlabel commands add x-, y-, and z-axis labels. The title command adds a title at the top of the figure and the text function inserts text anywhere in the figure.

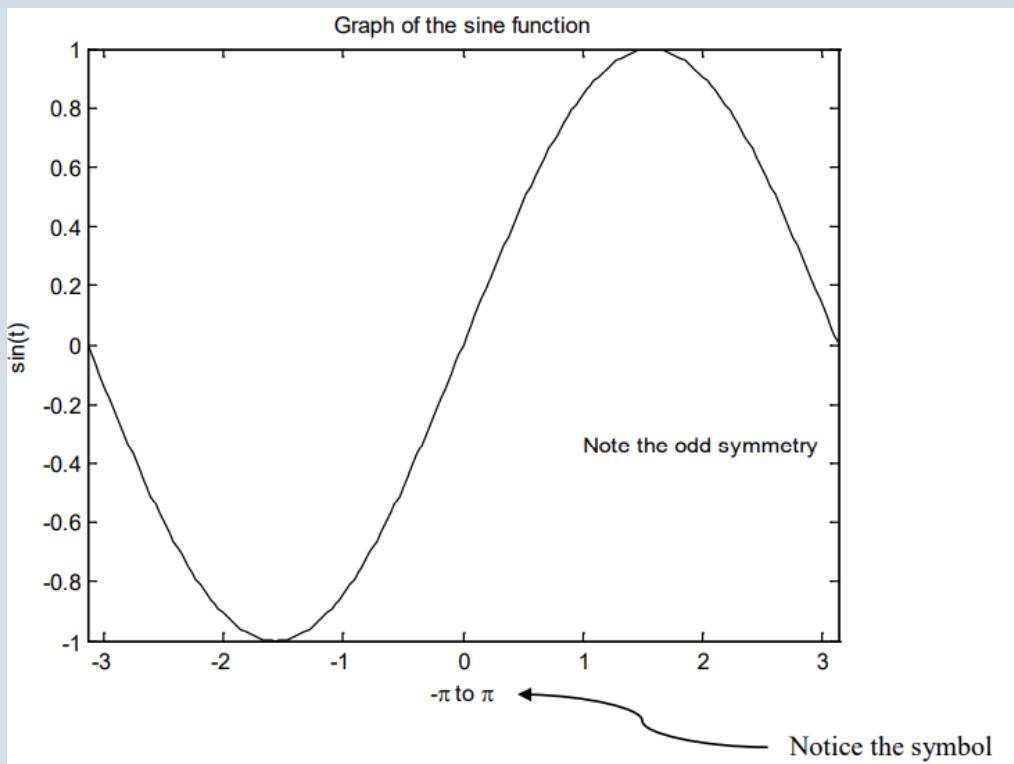
Example:

```
t = -pi:pi/100:pi;
```

```
y = sin(t);
```

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

```
plot(t,y)
axis([-pi pi -1 1])
xlabel ('-\pi to \pi')
ylabel ('sin(t)')
title ('Graph of the sine function')
text (1, -1/3,'Note the odd symmetry')
```



LESSON#2 IMAGE PROCESSING & COMPUTER VISION

➤ Complex and Statistical Functions

It is very easy to handle complex numbers in MATLAB. The special values i and j stand for $\sqrt{-1}$. Try `sqrt(-1)` to see how MATLAB represents complex numbers. The symbol i may be used to assign complex values, for example,

```
>> z = 2 + 3*i
```

represents the complex number $2 + 3i$ (real part 2, imaginary part 3).

You can also input a complex value like this:

```
>> z=2 + 3i
```

The imaginary part of a complex number may also be entered without an asterisk (*), 3i.

You can also input a complex value like this:

```
>> z=complex (2,3)
```

`z =`

$2.0000 + 3.0000i$

Example: Produce ten elements vector of random complex numbers and find the summation
of this vector

```
>> x=rand(1,10);
```

```
>> y=rand(1,10);
```

```
>> z=x+i*y
```

```
>> sum(z)
```

Or

```
>> z=complex(x,y)
```

```
>> sum(z)
```

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

All of the arithmetic operators (and most functions) work with complex numbers, such as $\text{sqrt}(2 + 3*i)$ and $\text{exp}(i*pi)$. Some functions are specific to complex numbers, like:

| Command | Description |
|---|--|
| <pre>>> A=3+4i A = 3.0000 + 4.0000i >> A = complex(3,4) A = 3.0000 + 4.0000i >> B = complex(-1,-3) B = -1.0000 - 3.0000i</pre> | Construct complex data from real and imaginary components. |
| <pre>>> abs(A) ans = 5 >> abs(B) ans = 3.1623</pre> | Absolute value and complex magnitude. |
| <pre>>> angle(A) ans = 0.9273 >> angle(A)*180/pi ans = 53.1301 >> angle(B)*180/pi</pre> | Phase angle. |

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

| | |
|---|-----------------------------------|
| ans = -108.4349 | |
| >> conj(A) ans = 3.0000 - 4.0000i | Complex conjugate. |
| >> conj(B) ans = -1.0000 + 3.0000i | |
| >> real(A) ans = 3 | Real part of complex number. |
| >> real(B) ans = -1 | |
| >> imag(A) ans = 4 | Imaginary part of complex number. |
| >> imag(B) ans = -3 | |

Example: Exchange the real and imaginary parts of the following matrix

$$A = 0.8147 + 0.1576i \quad 0.9058 + 0.9706i \quad 0.1270 + 0.9572i$$

$$\quad \quad \quad 0.9134 + 0.4854i \quad 0.6324 + 0.8003i \quad 0.0975 + 0.1419i$$

>> A= [0.8147 + 0.1576i, 0.9058 + 0.9706i, 0.1270 + 0.9572i

0.9134 + 0.4854i, 0.6324 + 0.8003i, 0.0975 + 0.1419i];

>> x=real(A);

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

```
>> y=imag(A);  
>> a=x;  
>> x=y;  
>> y=a;  
>> A=x+i*y  
A =  
0.1576 + 0.8147i 0.9706 + 0.9058i 0.9572 + 0.1270i  
0.4854 + 0.9134i 0.8003 + 0.6324i 0.1419 + 0.0975i
```

❖ Statistical Functions

| Function | Description |
|------------------|--|
| mean(x) | Average or mean value of array (x). |
| median(x) | Median value of array (x). |
| mode(x) | Most frequent values in array (x) When there are multiple values occurring equally frequently, mode returns the smallest of those values. For complex inputs, this is taken to be the first value in a sorted list of values. |
| std(x) | returns the standard deviation of array (x) $s = \text{std}(x) = \text{std}(x,0)$ and it is equal to $s = \left(\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2 \right)^{\frac{1}{2}}$ $\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$ $s = \text{std}(x,1)$, it is equal to |

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

| | |
|---------------|--|
| | >> x =[1 5 9 7 15 22]; >> s = std(x,0) s = 4.2426 7.0711 9.1924 >> s = std(x,1) s = 3.0000 5.0000 6.5000 |
| var(x) | Returns the variance of array (x), The variance is the square of the standard deviation (STD). s = var(x,0) when the summation normalized by N-1 s = var(x,1) when the summation normalized by N >> x =[1 5 9 7 15 22]; >> s = var(x,0) s = 18.0000 50.0000 84.5000 >> s = var(x,1) s = 9.0000 25.0000 42.2500 |

Example: If X = 3 3 1 4

0 0 1 1

0 1 2 4

Then

>> mode(x)

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

ans= [0 0 1 4]

and

>> mode(X,2)

ans= [3 ; 0 ; 0]

Example: If A = 1 2 4 4

3 4 6 6

5 6 8 8

5 6 8 8

>> median(A)

ans= [4 5 7 7]

>> median(A,2)

ans= [3 ; 5 ; 7 ; 7]

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

➤ Input / Output of Variables (Numbers and Strings)

❖ Characters and Text

Enter text into MATLAB using single quotes. For example,

```
>> S = 'Hello'
```

The result is not the same kind of numeric matrix or array we have been dealing with up to now. The string is actually a vector whose components are the numeric codes for the characters (the first 127 codes in ASCII). The length of S is the number of characters. It is a 1-by-5-character array. A quote within the string is indicated by two quotes.

Concatenation with square brackets joins text variables together into larger strings.

For

example,

```
>> h = ['MAT', 'LAB']
```

joins the strings horizontally and produces

h =

MATLAB

and the statement

```
>> v = ['MAT'; 'LAB']
```

joins the strings vertically and produces

v =

MAT

LAB

Note that both words in v have to have the same length. The resulting arrays are both character arrays; h is 1-by-6 and v is 2-by-3.

❖ Some String Function

| Function | Description |
|----------|-------------|
|----------|-------------|

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

| | |
|--|---|
| <pre>char (x) >> char(100) ans = d >> char([73 82 65 81]) ans = IRAQ</pre> | converts the array x that contains positive integers representing character codes into a MATLAB character array (the first 127 codes in ASCII). |
| <pre>double(s) >> double('z') ans = 122 >> double('ali') ans = 97 108 105</pre> | converts the character array to a numeric matrix containing floating point representations of the ASCII codes for each character. |
| <pre>strvcat(S1,S2,...) >> strvcat ('Hello', 'Hi', 'Bye') ans = Hello Hi Bye</pre> | joins S1,S2,... variables vertically together into larger string. |
| <pre>s = num2str(x) >> num2str(20) ans = 20 % as a string, not a number</pre> | converts character array representation of a matrix of numbers to a numeric matrix. |
| error (Msg) | displays the error message in the string |

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

| | |
|-----------------|---|
| | (Msg), and causes an error exit from the currently executing M-file to the keyboard. |
| lower(A) | Converts any uppercase characters in A to the corresponding lowercase character and leaves all other characters unchanged. |
| upper(x) | Converts any lower case characters in A to the corresponding upper case character and leaves all other characters unchanged |

Note

The printable characters in the basic ASCII character set are represented by the integers 32:127. (The integers less than 32 represent nonprintable control characters). Try

```
>> char (33:127)
```

To enter matrix or vector or single element:

```
>> x=input ('parameter= ')
```

```
parameter= 2
```

```
x =
```

```
2
```

```
>> x=input ('parameter= ')
```

```
parameter= [2 4 6]
```

```
x = 2 4 6
```

```
>> x=input ('parameter= ')
```

```
parameter= [1 2 3;4 5 6]
```

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

```
x = 1 2 3  
      4 5 6
```

To enter text:

```
>> x=input ('parameter= ')  
parameter= 'faaz'  
x =  
      faaz  
>> x=input ('parameter= ', 's')  
parameter= faaz  
x =      faaz
```

Notes the different
between two Statements

❖ Output of Variable

displays the array (x), without printing the array name. In all other ways it's the same as leaving the semicolon off an expression except that empty arrays don't display.

If (x) is a string, the text is displayed.

```
>> x=[1 2 3];  
>> x  
x =  
      1 2 3  
>> disp(x)  
      1 2 3
```

Example:

```
>> a=6;  
>> b=a;  
>> s='Ahmed has ';
```

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

```
>> w='Ali has ';  
>> t=' Dinars';  
>>disp([ s num2str(a) t]);  
>>disp([ w num2str(b) t]);
```

the execution result is:

Ahmed has 6 Dinars

Ali has 6 Dinars

An M-File is an external file that contains a sequence of MATLAB statements. By typing the filename in the Command Window, subsequent MATLAB input is obtained from the file. M-Files have a filename extension of ".m" and can be created or modified by using Editor/Debugger Window.

You need to save the program if you want to use it again later. To save the contents of the Editor, select File → Save from the Editor menu bar. Under Save file as, select a directory and enter a filename, which must have the extension .m, in the File name: box (e.g., faez.m). Click Save. The Editor window now has the title faez.m. If you make subsequent changes to faez.m an asterisk appears next to its name at the top of the Editor until you save the changes.

A MATLAB program saved from the Editor with the extension .m is called a script file, or simply a script. (MATLAB function files also have the extension .m. We therefore refer to both script and function files generally as M-files.).

The special significances of a script file are that: -

- 1- if you enter its name at the command-line prompt, MATLAB carries out each statement in it as if it were entered at the prompt.
- 2- Scripts M-file does not accept input arguments or return output arguments. They operate on data in the workspace.
- 3- The rules for script file names are the same as those for MATLAB variable

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

Names.

❖ Function Files

MATLAB enables you to create your own function M-files. A function M-file is similar to a script file in that it also has an .m extension. However, it differs from a script file in that it communicates with the MATLAB workspace only through specially designated input and output arguments.

* Functions operate on variables within their own workspace, separate from the workspace you access at the MATLAB command prompt.

General form of a function: A function M-file filename.m has the following general form:

```
function [ outarg1, outarg2,...] = filename (inarg1, inarg2,...)  
% comments to be displayed with help  
...  
outarg1 = ...;  
outarg2 = ...;
```

Note:

inarg1, inarg2,... are the input variables to the function filename.m

outarg1, outarg2,... are the output variables from the function filename.m

function The function file must start with the keyword function (in the function definition line).

Example:

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

sphecart.m ((function))

```
function [x,y,z] = sphecart(r,theta,rho)
%conversion from spherical to Cartesian
coordinates
x = r*cos(rho)*cos(theta);
y = r*cos(rho)*sin(theta);
z = r*sin(rho);
```

sphecart.m ((script file))

```
%conversion from spherical to Cartesian
coordinates

x = r*cos(rho)*cos(theta);
y = r*cos(rho)*sin(theta);
z = r*sin(rho);
%the values of r, rho, theta are be obtained from
workspace of command window
```

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

➤ Flow Control

Computer programming languages offer features that allow you to control the flow of command execution based on decision making structures. MATLAB has several flow control constructions:

- **if statement.**
- **switch and case statement.**
- **for statement.**
- **while statement.**
- **break statement.**

if statement

The if statement evaluates a logical expression and executes a group of statements when the expression is true. The optional elseif and else keywords provide for the execution

of alternate groups of statements. An end keyword, which matches the if, terminates the last group of statements.

The general form of if statement is:

```
if      expression 1
        group of statements 1
elseif   expression 2
        group of statements 2
else    expression 3
        group of statements 3
end
```

It is important to understand how relational operators and if statements work with

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

matrices. When you want to check for equality between two variables, you might use

```
if A == B
```

This is legal MATLAB code, and does what you expect when A and B are scalars. But when A and B are matrices, A == B does not test if they are equal, it tests Where they are

equal; the result is another matrix of 0's and 1's showing element-by-element equality. In fact, if A and B are not the same size, then A == B is an error. The proper way to check for equality between two matrices is to use the **isequal** function, if isequal (A,B)

Example:

```
A=input('A=');
B=input('B=');
if A > B
    'greater'
elseif A < B
    'less'
elseif A == B
    'equal'
else
    error ('Unexpected situation')
end
```

switch and case statement

The switch statement executes groups of statements based on the value of a variable or expression. The keywords case and otherwise delineate the groups.

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

Only the first matching case is executed. There must always be an end to match the switch. If the first case statement is true, the other case statements do not execute. The general form of switch statement is:

switch expression

case 0

statements 0

case 1

statements 1

otherwise

statements 3

end

Example

```
method = 'Bilinear';
switch lower(method)
case 'bilinear'
    disp('Method is bilinear')
case 'cubic'
    disp('Method is cubic')
case 'nearest'
    disp('Method is nearest')
otherwise
    disp('Unknown method.')
end
```

Execution result is:

Method is bilinear

for statement

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

The for loop repeats a group of statements a fixed, predetermined number of times.

The general form of for statement is:

for variable = initial value: step size: final value

statement

...

Statement

end

Example:

```
for i=1:5
```

```
for k=5:-1:1
```

```
m(i,k)=i*k;
```

```
end
```

```
end
```

```
>> m
```

```
m =
```

| | | | | |
|---|----|----|----|----|
| 1 | 2 | 3 | 4 | 5 |
| 2 | 4 | 6 | 8 | 10 |
| 3 | 6 | 9 | 12 | 15 |
| 4 | 8 | 12 | 16 | 20 |
| 5 | 10 | 15 | 20 | 25 |

A for loop cannot be terminated by reassigning the loop variable within the for loop:

```
for i=1:10
```

```
x(i)=sin (pi/i);
```

```
i=10; % this step do not effect on the for loop
```

```
end
```

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

x

i

Execution results are,

x=

| | | | | | |
|--------|--------|--------|--------|--------|--------|
| 0.0000 | 1.0000 | 0.8660 | 0.7071 | 0.5878 | 0.5000 |
| 0.4339 | 0.3827 | 0.3420 | 0.3090 | | |

i=

10

while statement

repeat statements an indefinite number of times under control of a logical condition.

A matching end delineates the statements.

The general form of while statement is:

while expression

statement

...

statement

end

Example: Here is a complete program, illustrating while, if, else, and end, that uses interval bisection method to find a zero of a polynomial.

a = 0; fa = -Inf;

b = 3; fb = Inf;

while b-a > eps*b

 x = (a+b)/2;

 fx = x^3-2*x-5;

 if sign(fx) == sign(fa)

 a = x; fa = fx;

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

```
else  
    b = x; fb = fx;  
end  
end
```

break

Terminate execution of while or for loop. In nested loops, break exits from the innermost loop only. If break is executed in an IF, SWITCH-CASE statement, it terminates the statement at that point.

Continue passes control to the next iteration of FOR or WHILE loop in which it appears, skipping any remaining statements in the body of the FOR or WHILE loop.

Example:

we can modify the previous example by using break command.

```
a = 0; fa = -Inf;  
b = 3; fb = Inf;  
while b-a > eps*b  
    x = (a+b)/2;  
    fx = x^3-2*x-5;  
    if fx == 0  
        break  
    elseif sign(fx) == sign(fa)  
        a = x; fa = fx;  
    else  
        b = x; fb = fx;  
    end  
end
```

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

Example: Without using the max command, find the maximum value of matrix (a)

where

```
a=[11 3 14;8 6 2;10 13 1]
```

Solution

```
a=[11 3 14;8 6 2;10 13 1]
```

```
temp=a(1);
```

```
[n,m]=size(a);
```

```
for i=1:n
```

```
    for j=1:m
```

```
        if a(i,j)>temp
```

```
            temp=a(i,j);
```

```
        end
```

```
    end
```

```
end
```

```
temp
```

the execution result is **14**

Example: Let $x = [2 \ 6; 1 \ 8]$, $y = [.8 \ -0.3; -0.1 \ 0.2]$, prove that y is not the inverse matrix of x .

Solution

```
z=inv(x);
```

```
if ~isequal(z,y)
```

```
    disp(' y is not the inverse matrix of x ')
```

```
end
```

the execution result is

y is not the inverse matrix of x

Exercises

Introduction to MATLAB

- 1- Use edit command to edit the **dct** function, then try to edit sin function. State the difference.
- 2- Use help command to get help about rand function.
- 3- Enter $a=3$; $b=5$; $c=7$, then clear the variable b only

Working with Matrices

- 4- If $x = [1 \ 4 \ ; \ 8 \ 3]$, find:
 - a) the inverse matrix of x.
 - b) the diagonal of x.
 - c) the sum of each column and the sum of whole matrix x.
 - d) the transpose of x.
- 5- If $x = [2 \ 8 \ 5 \ ; \ 9 \ 7 \ 1]$, $b = [2 \ 4 \ 5]$ find:
 - a) find the maximum and minimum of x.
 - b) find median value over each row of x.
 - c) add the vector b as a third row to x.
- 6- If $x = [2 \ 6 \ 12 \ ; \ 15 \ 6 \ 3 \ ; \ 10 \ 11 \ 1]$, then
 - a) replace the first row elements of matrix x with its average value.
 - b) reshape this matrix into row vector.
- 7- Generate a 4×4 Identity matrix.
- 8- Generate the following row vector $b = [5, 10, 15, 20, \dots, 95, 100]$, then find the number of elements in this vector.

Expressions

- 9- Write a MATLAB program to calculate the following expression and round the answers to the nearest integer.

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

- | | |
|-------------------------------------|----------------------------|
| a) $z = \sqrt{5x^2 + y^2}$ | where $x=2, y=4$ |
| b) $z = 4\cos(x) + j6\sin(x)$ | where $x=\pi/4$ |
| c) $z = 3\sin(x) + 4\cos(x) + 3e^y$ | where $x=\pi/3, y=2$ |
| d) $y = \sin(x)/x$ | where $0 \leq x \leq 2\pi$ |

10- Solve the following system

$$x + y - 2z = 3$$

$$2x + y = 7$$

$$x + y - z = 4$$

11- Use [round, fix, ceil, floor] commands to round the following numbers towards integer numbers:

| Before | After |
|---------------|--------------|
| 1.3 | 1 |
| 1.5 | 1 |
| 1.9 | 2 |
| 11.9 | 11 |
| -2.9 | -2 |
| -3.9 | -4 |
| 3.4 | 3 |

12- Write a Program to calculate the electromagnetic force between two electrons placed (in vacuum) at a distance ($r = 2*10^{-13}$ m) from each other. Charge of electron (Q) is $1.6*10^{-19}$ C.

Hint

$$\text{Electromagnetic Force} = K \frac{Q_1 Q_2}{r^2}$$

$$K = 9*10^9$$

13- Generate ten values from the uniform distribution on the interval [2, 3.5].

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

14- write a program to read three bits x, y, z, then compute:

- a) $v = (x \text{ and } y) \text{ or } z$
- b) $w = \text{not } (x \text{ or } y) \text{ and } z$
- c) $u = (x \text{ and not } (y)) \text{ or } (\text{not } (x) \text{ and } y)$

Relational and Logical Operations

13- Write a program for three bits parity generator using even-parity bit.

14- Write a program to convert a three bits binary number into its equivalent gray code.

15- if $q = [1\ 5\ 6\ 8\ 3\ 2\ 4\ 5\ 9\ 10\ 1]$, $x = [3\ 5\ 7\ 8\ 3\ 1\ 2\ 4\ 11\ 5\ 9]$,

then:

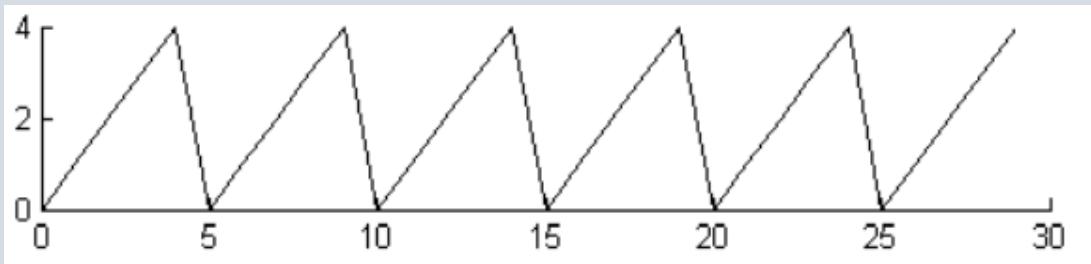
- a) find elements of (q) that are greater than 4.
- b) find elements of (q) that are equal to those in (x).
- c) find elements of (x) that are less than or equal to 7.

16- If $x = [10\ 3 ; 9\ 15]$, $y = [10\ 0 ; 9\ 3]$, $z = [-1\ 0 ; -3\ 2]$, what is the output of the following statements:

- a) $v = x > y$
- b) $w = z \geq y$
- c) $u = \sim z \& y$
- d) $t = x \& y < z$

Plotting Function

17- Plot sawtooth waveform as shown below

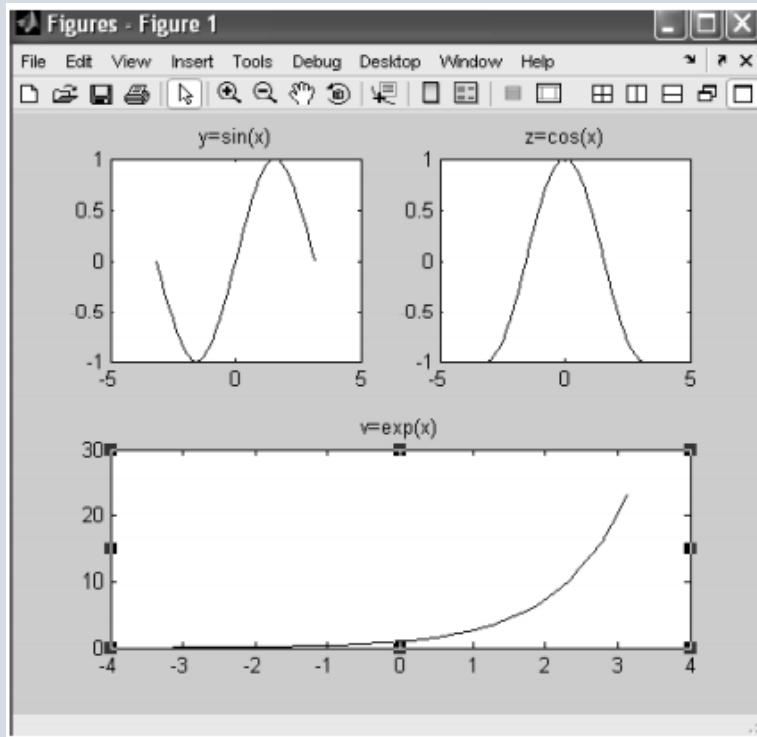


18- Plot Sinc function, where $\text{Sinc}(x) = \sin(x) / x$, and $-2\pi \leq x \leq 2\pi$

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

19- Plot $\sin(x)$ and $\cos(x)$ on the same figure, then on the same axis using different colors.

20- if $y=\sin(x)$, $z=\cos(x)$, $v=\exp(x)$, where $-\pi \leq x \leq \pi$
could you plot y , z , v as shown below!



Complex and Statistical Functions

21- Represent the following complex numbers in polar coordinate

$$Z = 2 + 5j$$

$$Y = -3 - 3j$$

$$D = -2 + 6j$$

22- Find the conjugate of the numbers above

23- Represent the following numbers in rectangular coordinate.

$$W = 5 \angle 30^\circ$$

$$A = 2.5 \angle -20^\circ$$

$$Q = 3e^{1.5} \angle -73^\circ$$

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

24- Compute the standard deviation by using the following equations then compare the result with that one obtained by std command.

$$s = \left(\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2 \right)^{\frac{1}{2}}$$

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

25- Write a program to compute the most frequent numbers in vectors (x), and (y) if

x=a*b

y=a*c

a=[1 3]

b=[2 3 5 ; 4 7 8]

b=[2 3 3 ; 4 7 7]

Input / Output of Variables

26- If x=[1 5 9; 2 7 4], then

a) display the last two elements by using disp command.

b) display the sum of each row as show below

The sum of 1st row =

The sum of 2nd row =

27- Write a program in M-File to read 3 x 3 Matrix, then display the diagonal of matrix as

shown below:

The Diagonal of This Matrix = []

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

28- Write a program to read a string, then replace each character in the string with its following character in ASCII code*.

29- The Table shown below lists the degrees of three students, Write a program in M-file to read these degrees and calculate the average degree for each student.

| Name | Mathematics | Electric Circuits | Communication |
|--------|-------------|-------------------|---------------|
| Ahmed | 80 | 80 | 80 |
| Waleed | 75 | 80 | 70 |
| Hasan | 80 | 90 | 85 |

Then display results as shown below

| Name | Degree |
|--------|--------|
| <hr/> | |
| Ahmed | 80 |
| Waleed | 75 |
| Hasan | 85 |

30- Write a group of statements that carry out the same action of upper and lower functions.

Flow Control

31- The value of s could be calculated from the equation below:

$$s = \begin{cases} \sqrt{y^2 - 4xz} & \text{if } y \geq 4xz \\ \text{inf} & \text{if } y < 4xz \end{cases}$$

write a MATLAB program in M-File to do the following steps:-

- input the value of x, y, z
- caluclate s
- print the output as shown below

x = . . .

LESSON#2 IMAGE PROCESSING & COMPUTER VISION

y = . . .

z = . . .

s = . . .

32- Write a program to find the current I in the circuit shown below

- a) By using conditional statements.
- b) Without using any conditional statements.

Lesson#3 Image Processing & COMPUTER VISION

**Some important functions are used with image in
MATLAB**

Eng. Elaf A.Saeed
Email: elafe1888@gmail.com

LAB Content

Read the Image in MATLAB

Display the Image in MATLAB

Write the Image in MATLAB

Convert RGB Image to Gray Image in MATLAB

Convert Gray Image to B&W Image in MATLAB

Crop Image in MATLAB

Resize the Image in MATLAB

Rotate the Image in MATLAB

The functions to display multiple Images in MATLAB

Table of Functions

| Function Name | Description |
|------------------------------|--|
| imread | Read image from graphics file. reads the image from the file specified by filename, inferring the format of the file from its contents. If filename is a multi-image file, then imread reads the first image in the file. |
| imgetfile | Display Open Image dialog box. |
| imshow | - Display image. |
| imshow(I) | - displays the grayscale image I. |
| imshow(I,[low high]) | - displays the grayscale image I, specifying the display range for I in [low high]. |
| imshow(RGB) | - displays the truecolor image RGB. |
| imshow(BW) | - displays the binary image BW. imshow displays pixels with the value 0 (zero) as black and pixels with the value 1 as white. |
| imshow(X,map) | - displays the indexed image X with the colormap map. A color map matrix may have any number of rows, but it must have exactly 3 columns. Each row is interpreted as a color, with the first element specifying the intensity of red light, the second green, and the third blue. Color intensity can be specified on the interval 0.0 to 1.0. |
| Imshowpair | - Compare differences between images |
| obj = imshowpair(A,B) | - creates a composite RGB image showing A and B overlaid in different color bands. To choose another type of visualization of the two images, use the method argument. If A and B are different sizes, imshowpair pads |

LESSON#3 IMAGE PROCESSING & COMPUTER VISION

| | |
|---|--|
| | the smaller dimensions with zeros on the bottom and right edges so that the two images are the same size. By default, imshowpair scales the intensity values of A and B independently from each other. imshowpair returns obj, an image object. |
| rgb2gray | Convert RGB image or colormap to grayscale. |
| imbinarize | Binarize 2-D grayscale image or 3-D volume by thresholding. |
| im2bw | Convert image to binary image, based on threshold. |
| imcrop | Crop image. |
| imrotate | Resize image. |
| figure figure figure(Name,Value) | - Create figure window. - Creates a new figure window using default property values. The resulting figure is the current figure. modifies properties of the figure using one or more name-value pair arguments. |
| subplot subplot(m,n,p) | - Create axes in tiled positions. - divides the current figure into an m-by-n grid and creates axes in the position specified by p. MATLAB® numbers subplot positions by row. The first subplot is the first column of the first row, the second subplot is the second column of the first row, and so on. If axes exist in |

LESSON#3 IMAGE PROCESSING & COMPUTER VISION

the specified position, then this command makes the axes the current axes.

LESSON#3 IMAGE PROCESSING & COMPUTER VISION

Theory

Image processing is a method to perform some operations on an image, in order to get an enhanced image or to extract some useful information from it. It is a type of signal processing in which input is an image and output may be image or characteristics/features associated with that image. Nowadays, image processing is among rapidly growing technologies. It forms core research area within engineering and computer science disciplines too.

Image processing basically includes the following three steps:

- Importing the image via image acquisition tools.
- Analyzing and manipulating the image.
- Output in which result can be altered image or report that is based on image analysis.

There are two types of methods used for image processing namely, analogue and digital image processing. Analogue image processing can be used for the hard copies like printouts and photographs. Image analysts use various fundamentals of interpretation while using these visual techniques. Digital image processing techniques help in manipulation of the digital images by using computers. The three general phases that all types of data have to undergo while using digital technique are pre-processing, enhancement, and display, information extraction.

The goal of this lab

It is the knowledge of the basic functions that are used in image processing in MATLAB.

LESSON#3 IMAGE PROCESSING & COMPUTER VISION

• Read the Image in MATLAB

1- **imread** → Read image from graphics file.

reads the image from the file specified by filename, inferring the format of the file from its contents. If filename is a multi-image file, then imread reads the first image in the file.

Syntax:

```
A = imread(filename)
```

Examples

1- save the image in the current folder.

2- Write in the command window the following:

```
>> x= imread('logo.jpg')
```

The Result

This will display the matrix of the image value.

3- Save the image in the in another path. For example, in the local disk D.

4- Write in the command window the following:

```
>> x= imread('D:\logo.jpg')
```

The Result

This will display the matrix of the image value.

5- Save the image in the on the desktop.

6- Write in the command window the following:

```
>> x= imread('C:\Users\Lenovo\Desktop\logo.jpg')
```

LESSON#3 IMAGE PROCESSING & COMPUTER VISION

The Result

This will display the matrix of the image value.

7- Save the image in the new folder.

8- Write in the command window the path of the image

2- **imgetfile** → Display Open Image dialog box.

Open the current folder to select the image.

Example

1- save the image in the current folder.

2- Write in the command window the following:

```
>> imgetfile
```

The Result

This will open the current folder to select the image.

- **Display the Image in MATLAB**

There are many functions to display the image in MATLAB. The functions are:

1- **imshow** → Display image.

a. **imshow(I)** displays the grayscale image I.

Example

1- save the image in the current folder.

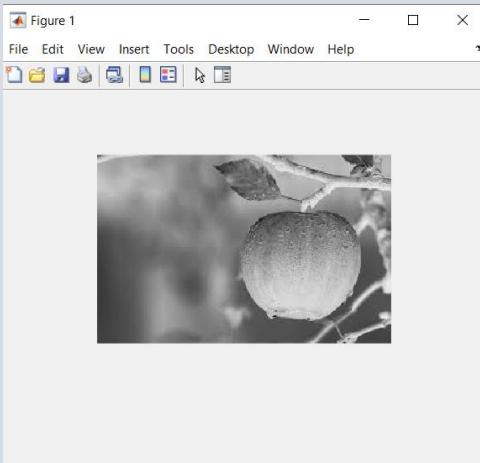
2- Write in the command window the following:

```
>> y= imshow('greeks.jpg')
```

LESSON#3 IMAGE PROCESSING & COMPUTER VISION

The Result

This will open the current folder to select the image.



- b. **imshow(I,[low high])** displays the grayscale image I, specifying the display range for I in [low high]. The value low (and any value less than low) displays as black; the value high (and any value greater than high) displays as white. Values in between are displayed as intermediate shades of gray, using the default number of gray levels. If you use an empty matrix ([]) for [low high], imshow uses [min(I(:)) max(I(:))]; that is, the minimum value in I is displayed as black, and the maximum value is displayed as white.

Note: The image must be a **png** extension because all other extensions are not affected.

Example

1. save the image in the current folder.
2. Write in the command window the following:

```
>> c=imread('bird1.png');
```

LESSON#3 IMAGE PROCESSING & COMPUTER VISION

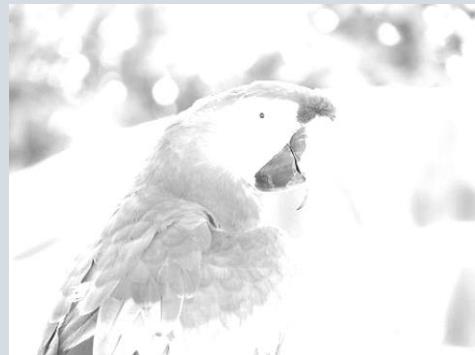
```
>> s= imshow(c,[20 150])
```

The Result

In figure bellow that shows the output image.



Before



After

- c. **imshow(RGB)** displays the truecolor image RGB.

Example

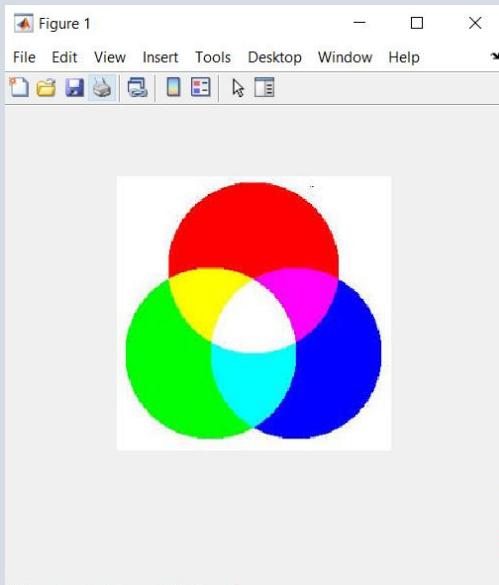
1. save the image in the current folder.
2. Write in the command window the following:

```
>> imshow('RGB.jpg')
```

The Result

This will open the RGB image.

LESSON#3 IMAGE PROCESSING & COMPUTER VISION



- d. **imshow(BW)** displays the binary image BW. imshow displays pixels with the value 0 (zero) as black and pixels with the value 1 as white.

Example

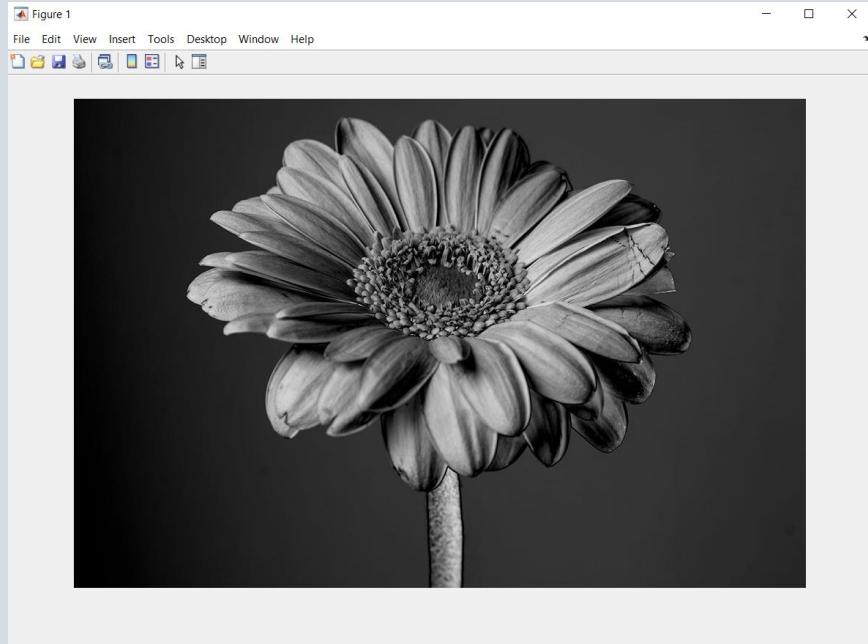
- 1- save the image in the current folder.
- 2- Write in the command window the following:

```
>> imshow('BW.jpg')
```

The Result

This will open the Black and White image.

LESSON#3 IMAGE PROCESSING & COMPUTER VISION



- e. **imshow(X,map)** displays the indexed image X with the colormap map. A color map matrix may have any number of rows, but it must have exactly 3 columns. Each row is interpreted as a color, with the first element specifying the intensity of red light, the second green, and the third blue. Color intensity can be specified on the interval 0.0 to 1.0.
 - f. **imshowpair** → Compare differences between images.
obj = imshowpair(A,B) creates a composite RGB image showing A and B overlaid in different color bands. To choose another type of visualization of the two images, use the method argument. If A and B are different sizes, imshowpair pads the smaller dimensions with zeros on the bottom and right edges so that the two images are the same size. By default, imshowpair scales the intensity values of A and B independently from each other. imshowpair returns obj, an image object.
- Write the Image in MATLAB

Imwrite → Write image to graphics file.

LESSON#3 IMAGE PROCESSING & COMPUTER VISION

imwrite(A,filename) writes image data A to the file specified by filename, inferring the file format from the extension. imwrite creates the new file in your current folder. The bit depth of the output image depends on the data type of A and the file format. For most formats:

- If A is of data type uint8, then imwrite outputs 8-bit values.
- If A is of data type uint16 and the output file format supports 16-bit data (JPEG, PNG, and TIFF), then imwrite outputs 16-bit values. If the output file format does not support 16-bit data, then imwrite returns an error.
- If A is a grayscale or RGB color image of data type double or single, then imwrite assumes that the dynamic range is [0,1] and automatically scales the data by 255 before writing it to the file as 8-bit values. If the data in A is single, convert A to double before writing to a GIF or TIFF file.
- If A is of data type logical, then imwrite assumes that the data is a binary image and writes it to the file with a bit depth of 1, if the format allows it. BMP, PNG, or TIFF formats accept binary images as input arrays.
- If A contains indexed image data, you should additionally specify the map input argument.

Example

1- save the image in the current folder.

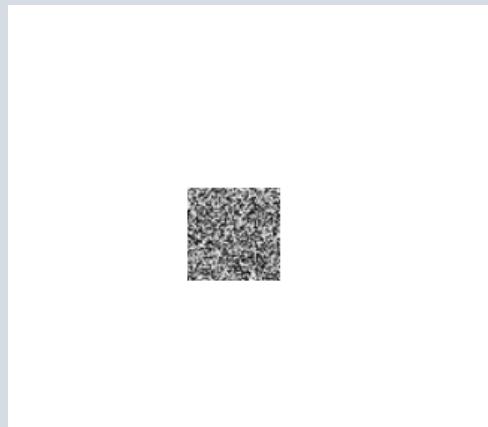
2- Write in the command window the following:

```
>> A = rand(50); %great the random matrix 50x50  
>> imwrite(A,'myGray.png') %write the image and store it in the  
current folder
```

The Result

This will store the gray image in the current folder.

LESSON#3 IMAGE PROCESSING & COMPUTER VISION



The output image

- Convert RGB Image to Gray Image in MATLAB

rgb2gray → Convert RGB image or colormap to grayscale.

Example

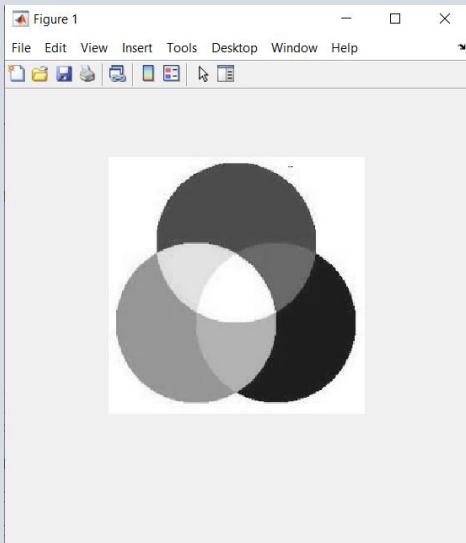
- 3- save the image in the current folder.
- 4- Write in the command window the following:

```
>> h= imread('RGB.jpg');  
>> x=rgb2gray(h);  
>> imshow(x)
```

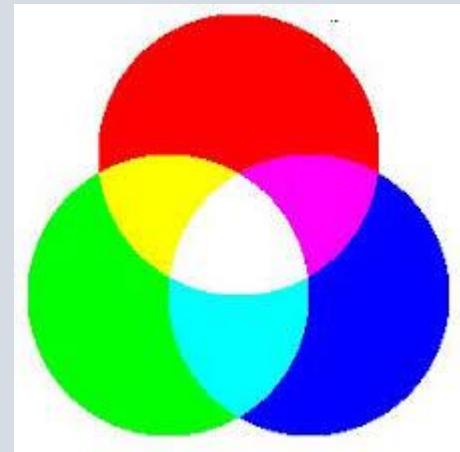
The Result

This will open the gray image.

LESSON#3 IMAGE PROCESSING & COMPUTER VISION



After convert



original image

- Convert Gray Image to B&W Image in MATLAB

- 1- **imbinarize** → Binarize 2-D grayscale image or 3-D volume by thresholding.

NOTE:

imbinarize function display the matrix values of the image. So expected I to be one of these types: uint8, uint16, uint32, int8, int16, int32, single, double.

Error: `>> x=imbinarize('birds.png');`

Example

- 1- save the image in the current folder.
- 2- Write in the command window the following:

`>> x=imread('birds.png');` %read the image values

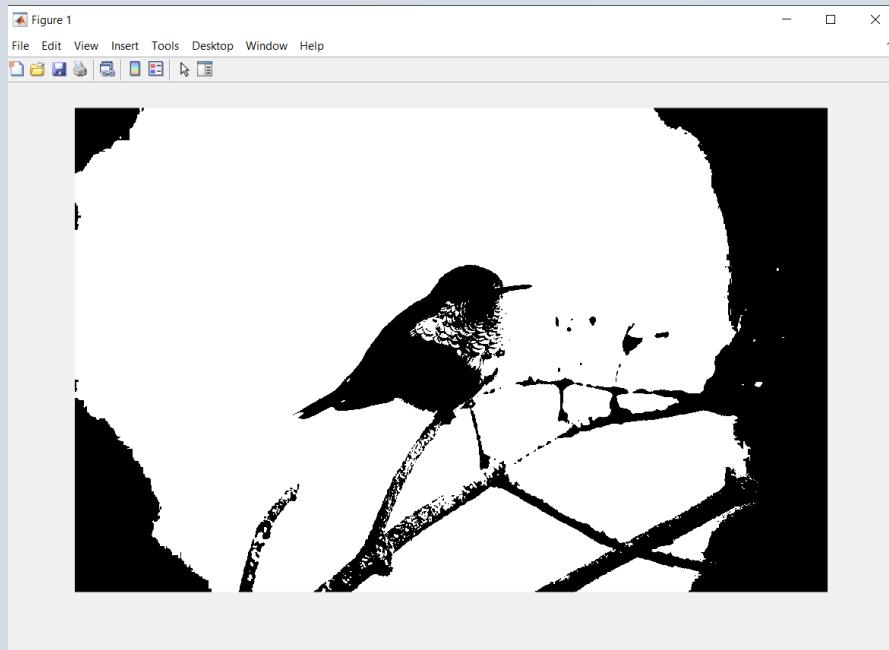
`>> y=imbinarize(x);` %convert the image to B&W

LESSON#3 IMAGE PROCESSING & COMPUTER VISION

```
>> imshow(y) %show the image
```

The Result

This will open the W&B image.



Black and white image

LESSON#3 IMAGE PROCESSING & COMPUTER VISION



Original image

2- **im2bw** → Convert image to binary image, based on threshold

BW = im2bw(I,level) converts the grayscale image I to binary image BW, by replacing all pixels in the input image with luminance greater than level with the value 1 (white) and replacing all other pixels with the value 0 (black).

level — Luminance threshold

0.5 (default) | number in the range [0, 1]

Luminance threshold, specified as a number in the range [0, 1].

Data Types: single | double | int16 | uint8 | uint16

Example

1- save the image in the current folder.

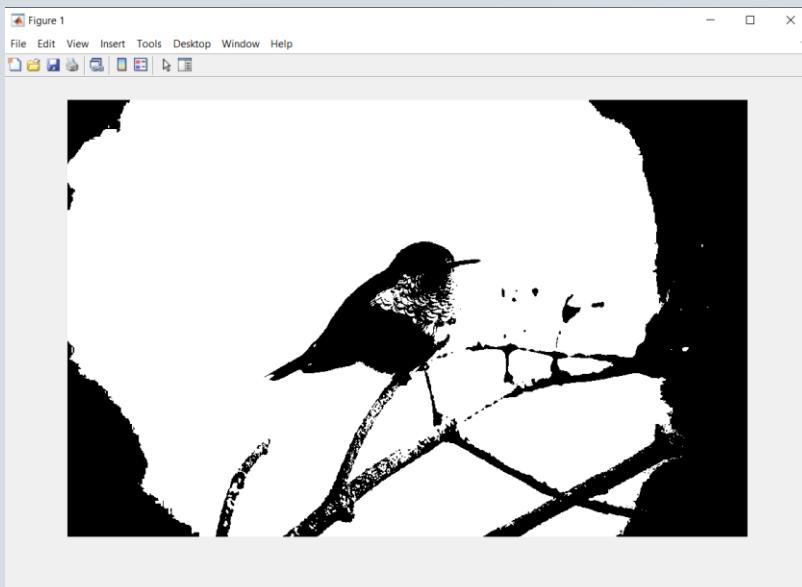
LESSON#3 IMAGE PROCESSING & COMPUTER VISION

2- Write in the command window the following:

```
>> x=imread('birds.png'); %read the image values  
>> y=im2bw(x); %convert the image to B&W  
>> imshow(y) %show the image
```

The Result

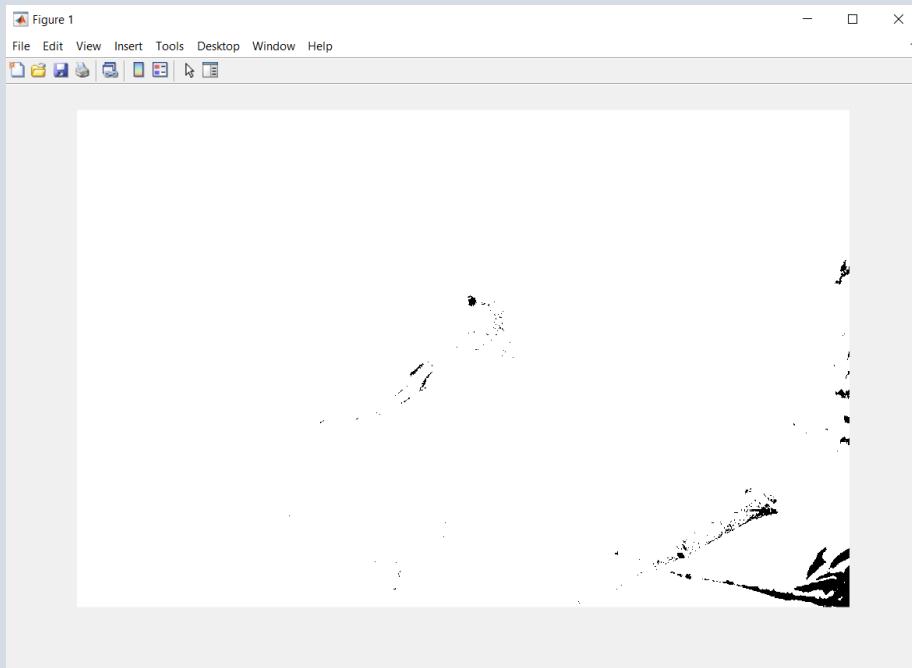
This will open the W&B image.



If level=0.1

```
>> y=im2bw(x,0.1);  
>> imshow(y)
```

LESSON#3 IMAGE PROCESSING & COMPUTER VISION



Threshold level=0.1

- Crop Image in MATLAB

imcrop → Crop image

J = imcrop(I) displays the image I in a figure window and creates an interactive Crop Image tool associated with the image. I can be a grayscale image, a truecolor image, or a logical array.

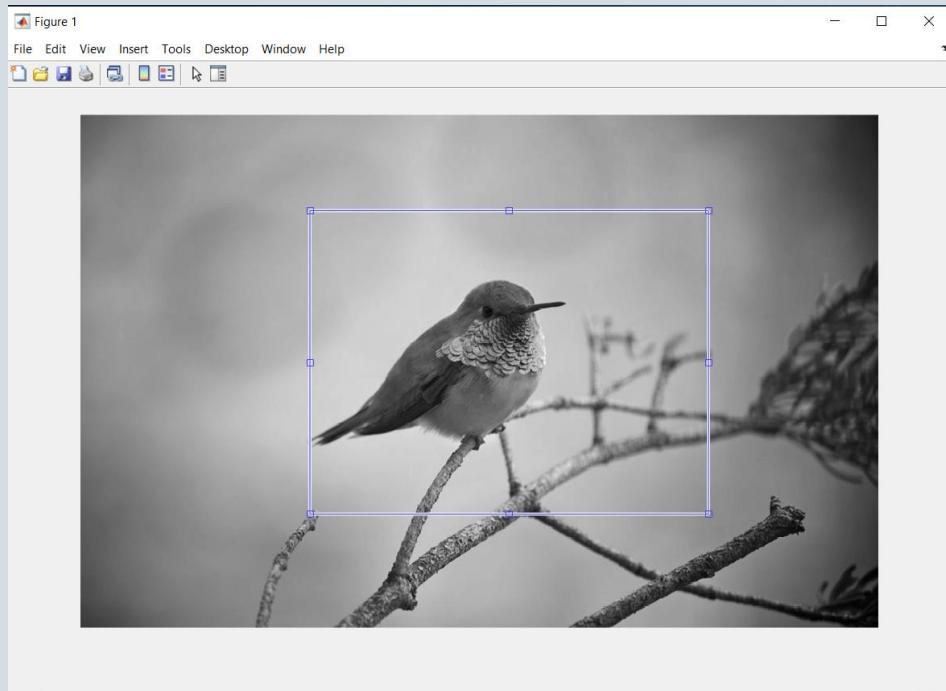
Example

- 1- save the image in the current folder.
- 2- Write in the command window the following:

```
>> c=imread('birds.png'); %read the image values
```

```
>> z=imcrop(c) %crop the image
```

LESSON#3 IMAGE PROCESSING & COMPUTER VISION

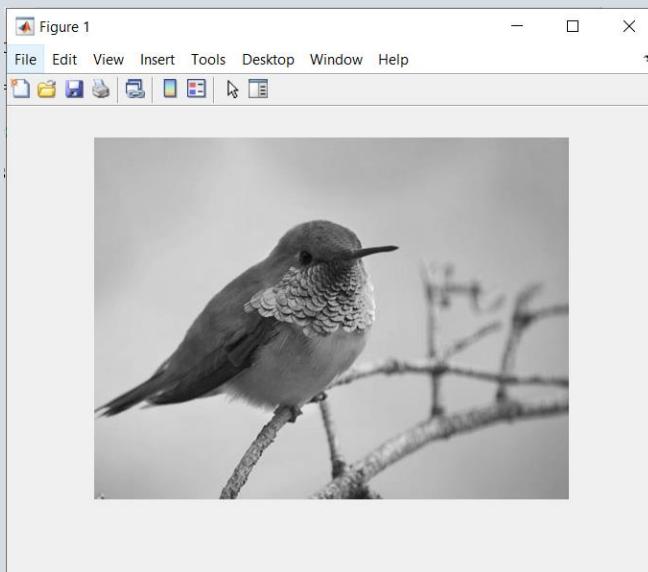


R.C on the image → select crop image → then write in the command window:

```
>> x=imshow(z) %show the image
```

The Result

This will open the W&B image.



LESSON#3 IMAGE PROCESSING & COMPUTER VISION

Crop image



Original image

- Resize the Image in MATLAB

imresize → Resize image

B = imresize(A,scale) returns image B that is scale times the size of A. The input image A can be a grayscale, RGB, or binary image. If A has more than two dimensions, imresize only resizes the first two dimensions. If scale is in the range [0, 1], B is smaller than A. If scale is greater than 1, B is larger than A. By default, imresize uses bicubic interpolation.

Example

- 1- save the image in the current folder.
- 2- Open the new script.
- 3- Write the following code:

```
% resize the image  
clc  
clear
```

LESSON#3 IMAGE PROCESSING & COMPUTER VISION

```
%input the image
image1=imread('birds.png');
%display the size of original image
size(image1)

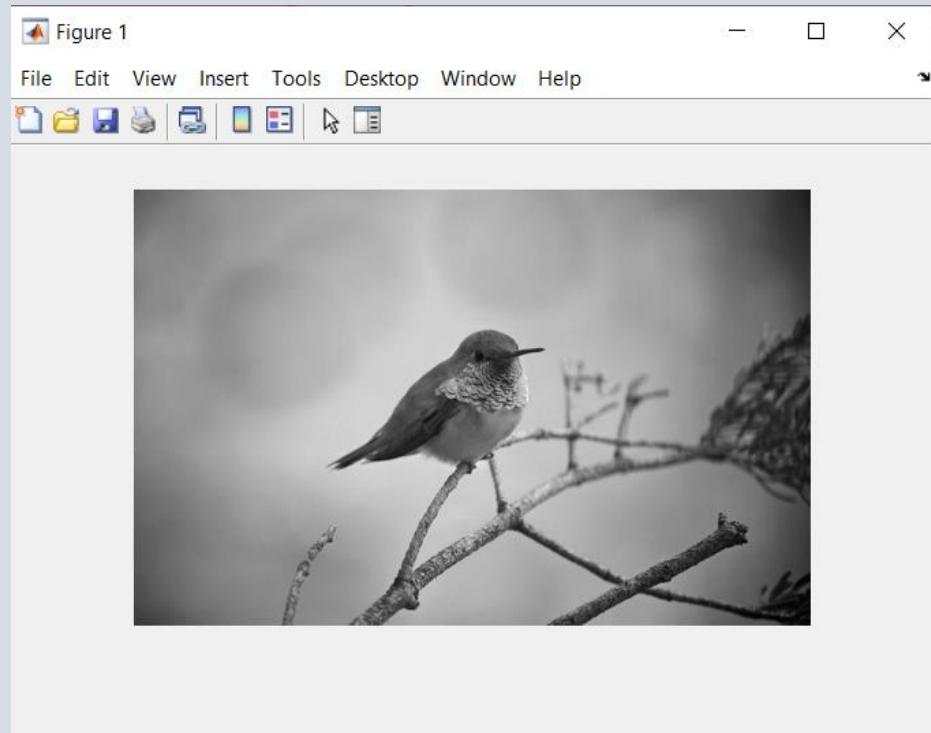
%resize the image with scale 1/2
image2 = imresize(image1, 0.5);
%display the resize image
disp(size(image2))

%resize the image with scale 2
image3 = imresize(image1, 2);
%display the resize image
disp(size(image3))

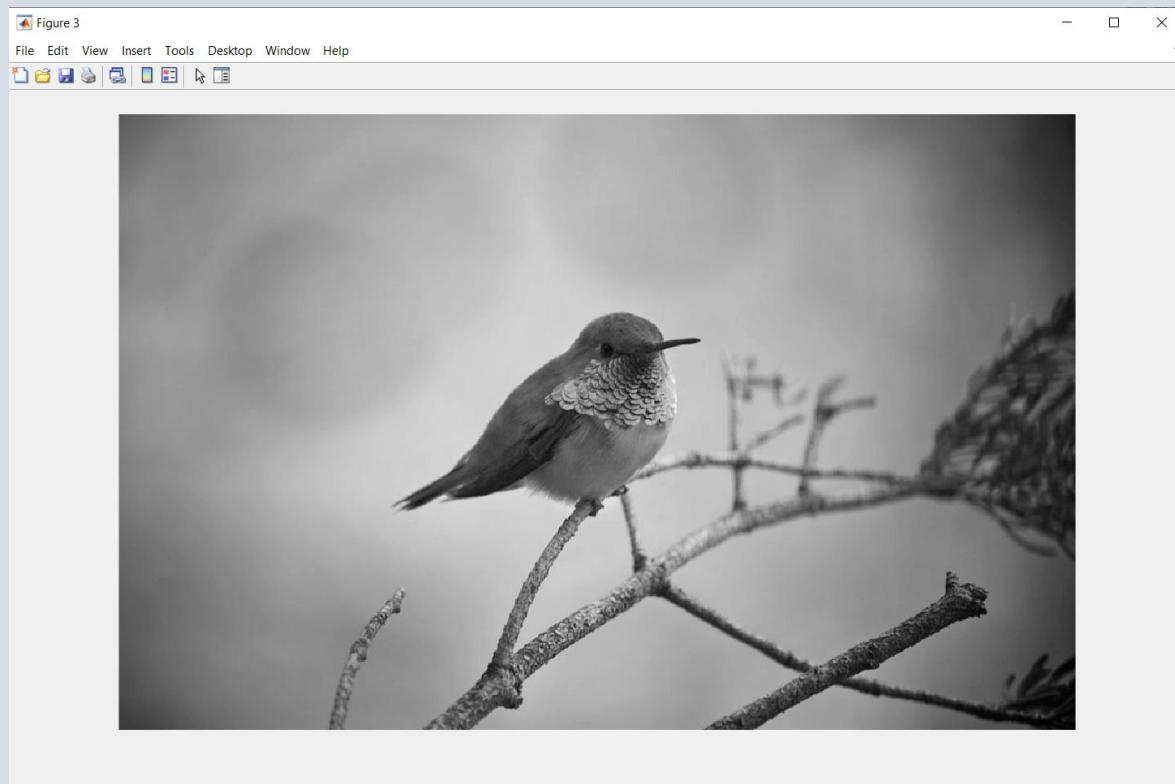
%display image
imshow(image2)
figure
imshow(image3)
```

The Result

LESSON#3 IMAGE PROCESSING & COMPUTER VISION



Size: 319 496



Size: 1274 1982

LESSON#3 IMAGE PROCESSING & COMPUTER VISION

- Rotate the Image in MATLAB

imrotate → Rotate image

`J = imrotate(I,angle)` rotates image I by angle degrees in a counterclockwise direction around its center point. To rotate the image clockwise, specify a negative value for angle. `imrotate` makes the output image J large enough to contain the entire rotated image. `imrotate` uses nearest neighbor interpolation, setting the values of pixels in J that are outside the rotated image to 0 (zero).

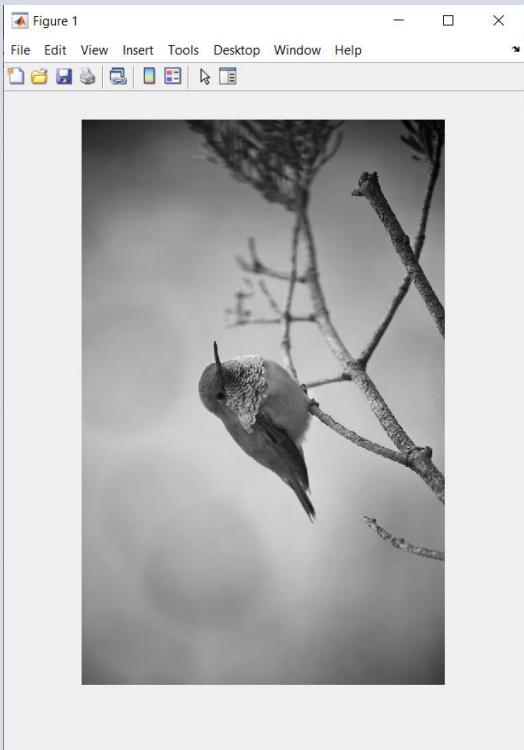
Example

- 1- save the image in the current folder.
- 2- Write in the command window the following:

```
>> c=imread('birds.png');  
  
>> rot= imrotate(c,90);  
  
>> r= imshow(rot)
```

The Result

LESSON#3 IMAGE PROCESSING & COMPUTER VISION



Rotate image



original image

- The functions to display multiple Images in MATLAB

1- **figure** → Create figure window.

figure creates a new figure window using default property values. The resulting figure is the current figure.

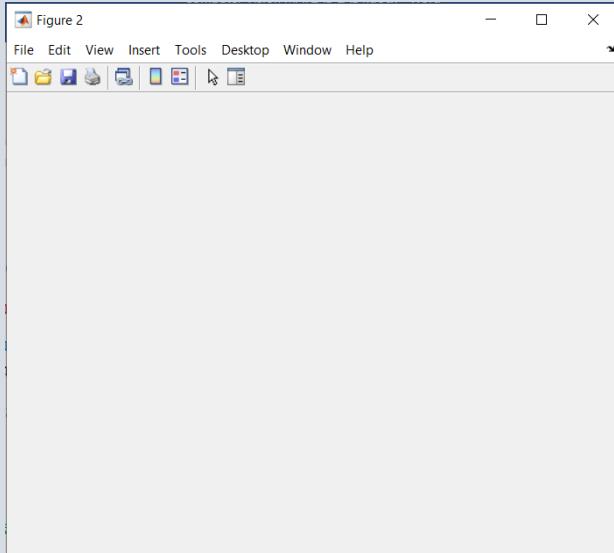
Example

Write in the command window the following:

```
>> figure
```

The Result

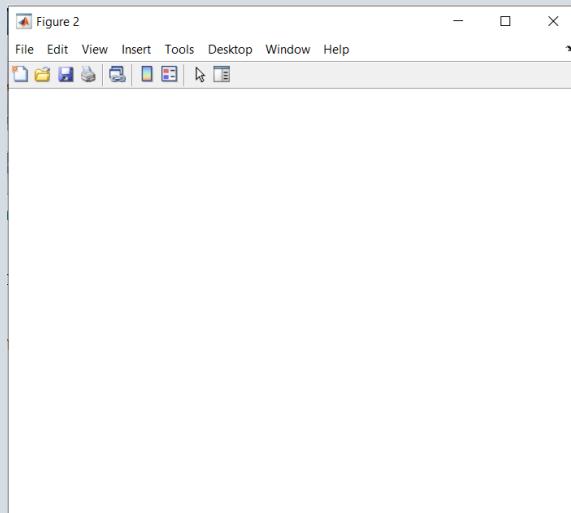
LESSON#3 IMAGE PROCESSING & COMPUTER VISION



figure(Name,Value) modifies properties of the figure using one or more name-value pair arguments.

To specify the background color:

For example, **figure('Color','white')** sets the background color to white.



To specify the figure Title:

LESSON#3 IMAGE PROCESSING & COMPUTER VISION

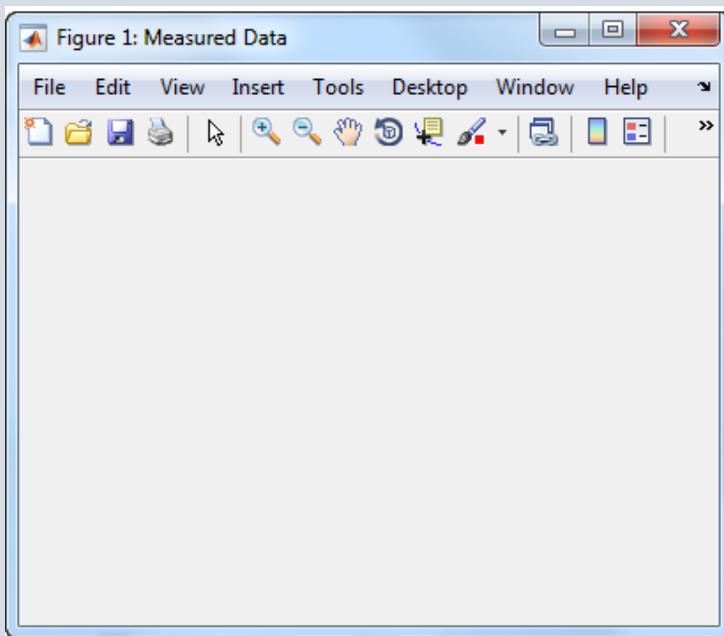
Example1

Create a figure, and specify the Name property. By default, the resulting title includes the figure number.

- Write in the command window the following:

```
>> figure('Name','Measured Data');
```

The Result



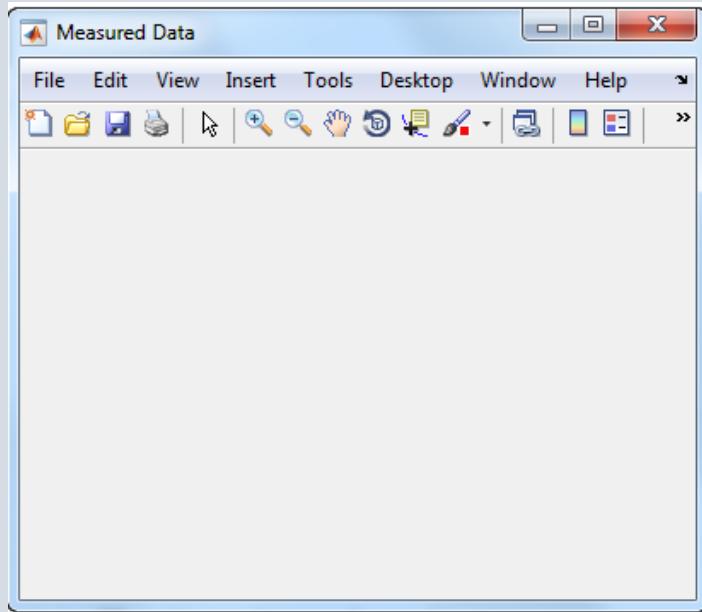
Example2

Specify the Name property again, but this time, set the NumberTitle property to 'off'. The resulting title does not include the figure number.

- Write in the command window the following:

```
>> figure('Name','Measured Data','NumberTitle','off');
```

LESSON#3 IMAGE PROCESSING & COMPUTER VISION



To work with multiple figures:

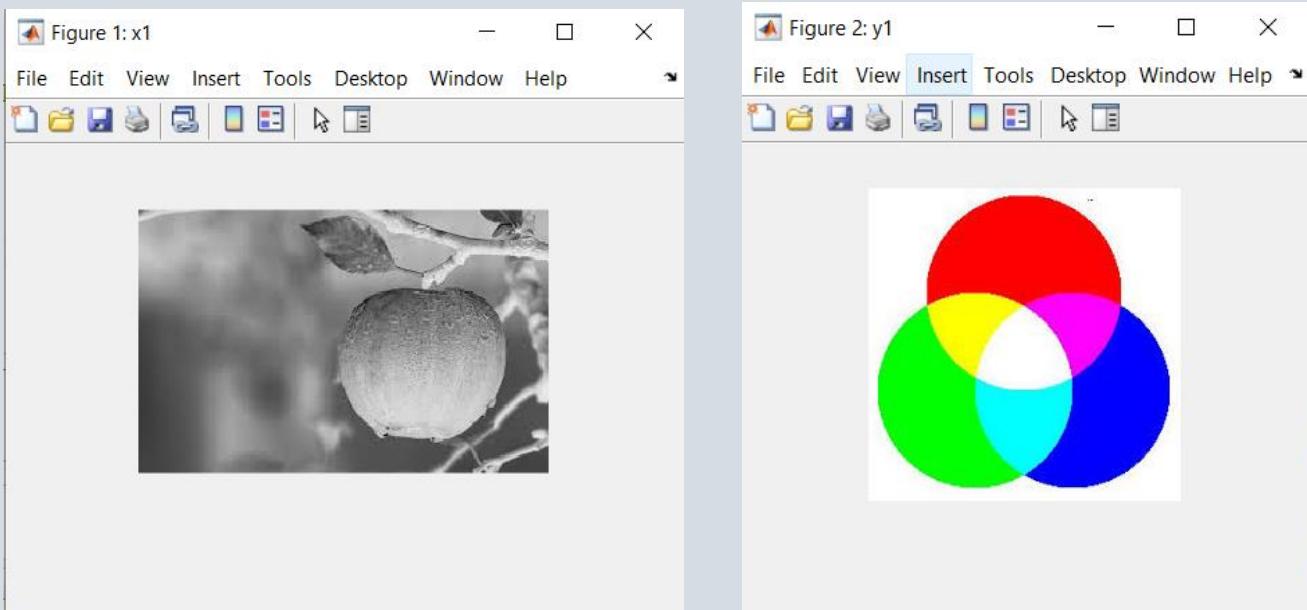
Example

- 1- save the image in the current folder.
- 2- Open the new script.
- 3- Write the following code:

```
clear  
clc  
  
x= imread('greeks.jpg');  
y= imread ('RGB.jpg');  
  
figure('Name','x1')  
x1= imshow(x)  
  
figure('Name','y1')  
y1= imshow (y)
```

LESSON#3 IMAGE PROCESSING & COMPUTER VISION

The Result



2- **subplot** → Create axes in tiled positions.

subplot(m,n,p) divides the current figure into an m-by-n grid and creates axes in the position specified by p. MATLAB® numbers subplot positions by row. The first subplot is the first column of the first row, the second subplot is the second column of the first row, and so on. If axes exist in the specified position, then this command makes the axes the current axes.

Example

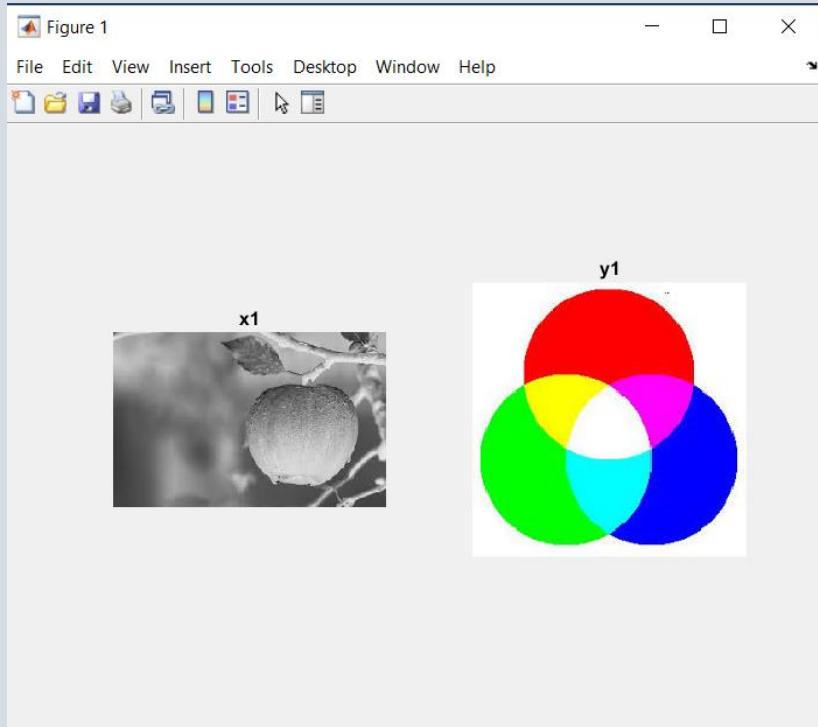
- 1- save the image in the current folder.
- 2- Open the new script.
- 3- Write the following code:

```
clear  
clc
```

LESSON#3 IMAGE PROCESSING & COMPUTER VISION

```
x= imread('greeks.jpg');  
y= imread ('RGB.jpg');  
subplot(1,2,1)  
x1= imshow(x)  
title('x1')  
subplot(1,2,2)  
y1= imshow (y)  
title('y1')
```

The Result



LESSON#3 IMAGE PROCESSING & COMPUTER VISION

Note

You can edit multiple specification to the figure directly from the figure window.

From tool bar you can edit the properties. such as, x-axis, y-axis, title, zoom in, zoom out, ...etc.

Report

1- Write the m-file to display the 3- images, Each image separately.

in these figures display the following:

- a- In the first figure Display the RGB image.
- b- In the second figure, change the color image to gray.
- c- In the third figure, change the gray image to a black and white image.

2- Write the m-file to display the 3- images in the same figure. The

figure contains the following:

- a- The original image.
- b- Displays the original image in a larger size by scale 2.
- c- Display the original image in a 90-degree rotation

Lesson#4 Image Processing & COMPUTER VISION

**Read and Change any pixel in the image by using
MATLAB**

Eng. Elaf A.Saeed
Email: elafe1888@gmail.com

LAB Content

Read any pixel in the image.

Change any pixel in the image

Table of Functions

| Function Name | Description |
|---------------|---|
| imread | Read image from graphics file. reads the image from the file specified by filename, inferring the format of the file from its contents. If filename is a multi-image file, then imread reads the first image in the file. |

Theory

The goal of this lab

Learn to read a pixel and how to change its value, learn to copy a specific part of an image, and other basic operations in MATLAB.

Procedure

1. Read any pixel in the image.

- **imread** → Read image from graphics file.
reads the image from the file specified by filename, inferring the format of the file from its contents. If filename is a multi-image file, then imread reads the first image in the file.

Syntax:

`A = imread(filename)`

A. Read Gray image Pixel

- 1- save the image in the current folder.
- 2- Write in the command window the following:

```
>> x=imread('logo.jpg')
```

```
>>x(100,100)
```

The Result

This will display the matrix of the image value.

B. Read RGB image Pixel

- 1- save the image in the current folder.
- 2- Write in the command window the following:

LESSON#4 IMAGE PROCESSING & COMPUTER VISION

```
>> x= imread('gift.jpg')

%to read red color in pixel (100,100)

>> x(50,50,1)

ans =

252

%to read green color in pixel (100,100)

>> x(50,50,2)

ans =

255

%to read blue color in pixel (100,100)

>> x(50,50,3)

ans =

253
```

2. Change any pixel in the image

A- Gray image

```
%to change the value in pixel (50,50)- gray image

>> imread('rose.jpg');

>> x(50,50)

ans =

252

>> x(50,50)=100;
```

LESSON#4 IMAGE PROCESSING & COMPUTER VISION

```
>> x(50,50)
```

ans =

100

B- RGB image

- **Red color**

```
>> x=imread('gift.jpg');
```

```
>> x(50,50,1)
```

ans =

252

```
>> x(50,50,1)=100;
```

```
>> x(50,50,1)
```

ans =

100

- **Green Color**

```
>> x(50,50,2)
```

ans =

255

```
>> x(50,50,2)=100;
```

```
>> x(50,50,2)
```

ans =

100

- **Blue Color**

```
>> x(50,50,3)
```

LESSON#4 IMAGE PROCESSING & COMPUTER VISION

ans =

253

>> x(50,50,3)=100;

>> x(50,50,3)

ans =

100

Report

- 1- Write the m-file to display the gray image, and change the value of pixel in (r=20 and c=30) to 200.
- 2- Write the m-file to display the color image and change the value of pixel in (r=40 and c=60) to 60, do this for the red color, green and blue color.

Lesson#5 Image Processing & COMPUTER VISION

**Color Image Processing and color models in MATLAB
(Model 1: RGB Color Model)**

Eng. Elaf A.Saeed
Email: elafe1888@gmail.com

LAB Content

Extracting the Color Spaces in MATLAB

CMY Color in MATLAB

Table of Functions

| Function Name | Description |
|--|---|
| size | - Display image. |
| sz = size(A) | <ul style="list-style-type: none"> - returns a row vector whose elements contain the length of the corresponding dimension of A. For example, if A is a 3-by-4 matrix, then size(A) returns the vector [3 4]. The length of sz is ndims(A). If A is a table or timetable, then size(A) returns a two-element row vector consisting of the number of rows and the number of table variables. |
| szdim = size(A,dim) [m,n] = size(A) | <ul style="list-style-type: none"> - returns the length of dimension dim. - returns the number of rows and columns when A is a matrix. |
| [sz1,...,szN] = size(A) | - returns the length of each dimension of A separately. |

Theory

Color Image Processing

The use of color is important in image processing because:

- Color is a powerful descriptor that simplifies object identification and extraction.
- Humans can discern thousands of color shades and intensities, compared to about only two dozen shades of gray.

Color image processing is divided into two major areas:

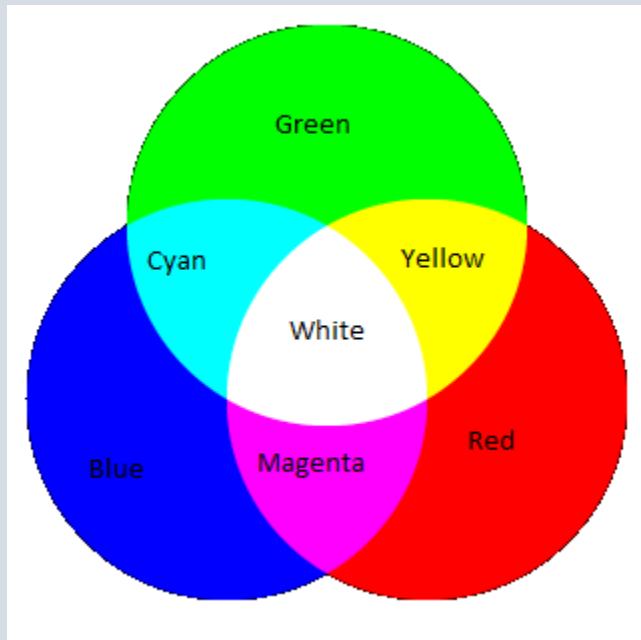
- Full-color processing: images are acquired with a full-color sensor, such as a color TV camera or color scanner.
- Pseudo color processing: The problem is one of assigning a color to a particular monochrome intensity or range of intensities.

Color Fundamentals

Colors are seen as variable combinations of the primary colors of light: red (R), green (G), and blue (B). The primary colors can be mixed to produce the secondary colors: magenta (red+blue), cyan (green+blue), and yellow (red+green). Mixing the three primaries, or a secondary with its opposite primary color, produces white light.

In figure 3.1 that shown the Primary and secondary colors of light.

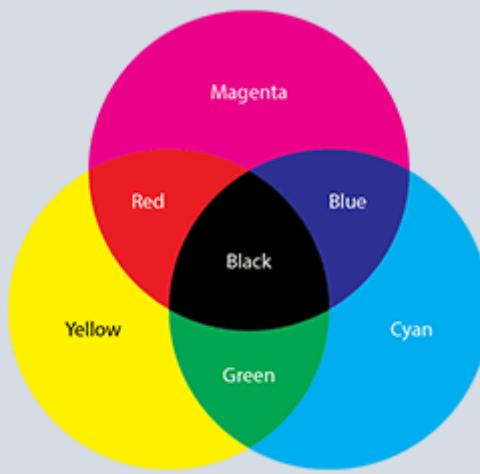
LESSON#5 IMAGE PROCESSING & COMPUTER VISION



Primary and secondary colors of light

RGB colors are used for color TV, monitors, and video cameras.

However, the primary colors of pigments are cyan(C), magenta(M), and yellow(Y), and the secondary colors are red, green, and blue. A proper combination of the three pigment primaries, or a secondary with its opposite primary, produces black.



Primary and secondary colors of pigments

❖ **CMY colors are used for color printing.**

Color characteristics

LESSON#5 IMAGE PROCESSING & COMPUTER VISION

The characteristics used to distinguish one color from another are:

- **Brightness**: means the amount of intensity (i.e. color level).
- **Hue**: represents dominant color as perceived by an observer.
- **Saturation**: refers to the amount of white light mixed with a hue.

Color Models

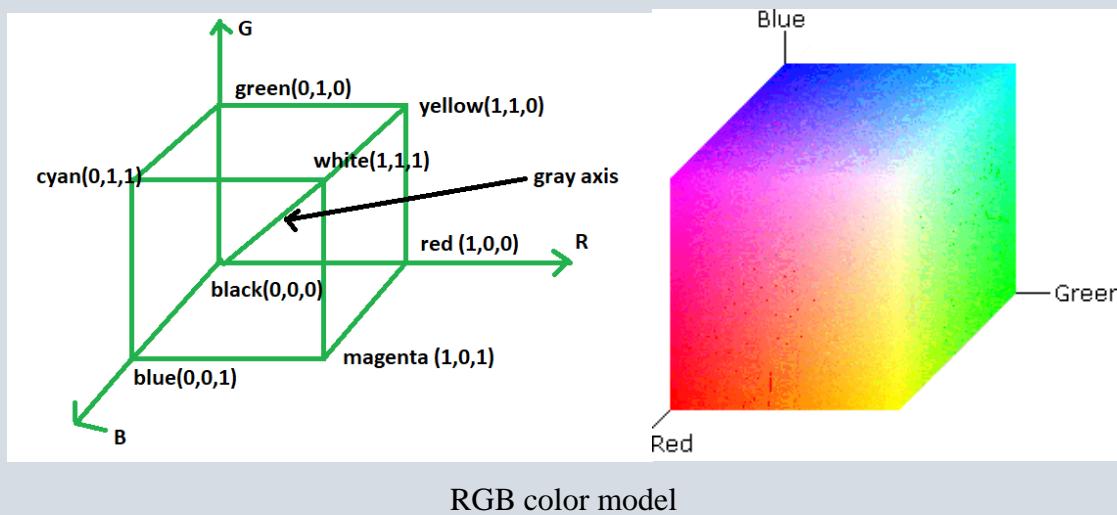
The purpose of a color model is to facilitate the specification of colors in some standard way. A color model is a specification of a coordinate system and a subspace within that system where each color is represented by a single point. Color models most commonly used in image processing are:

- **RGB** model for color monitors and video cameras
- **CMY** and **CMYK** (cyan, magenta, yellow, black) models for color printing
- **HSI** (hue, saturation, intensity) model

In this lab we discuss the first model is RGB color model.

The RGB color model

In this model, each color appears in its primary colors red, green, and blue. This model is based on a Cartesian coordinate system. The color subspace is the cube shown in the figure 3.3 below. The different colors in this model are points on or inside the cube, and are defined by vectors extending from the origin.



LESSON#5 IMAGE PROCESSING & COMPUTER VISION

All color values R, G, and B have been normalized in the range [0, 1]. However, we can represent each of R, G, and B from 0 to 255. Each RGB color image consists of three component images, one for each primary color as shown in the figure 3.4 below. These three images are combined on the screen to produce a color image.

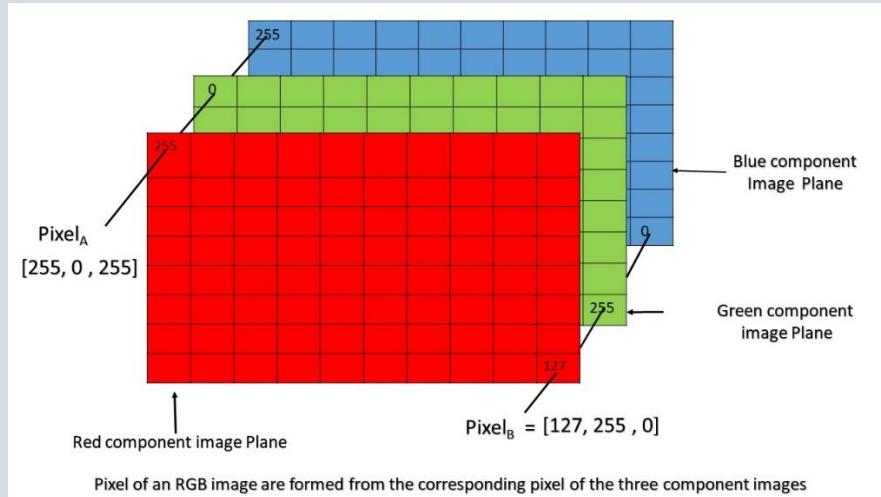


Figure 3.4: RGB color image

The total number of bits used to represent each pixel in RGB image is called pixel depth. For example, in an RGB image if each of the red, green, and blue images is an 8-bit image, the pixel depth of the RGB image is 24-bits. The figure below shows the component images of an RGB image.



A full-color image and its RGB component images

LESSON#5 IMAGE PROCESSING & COMPUTER VISION

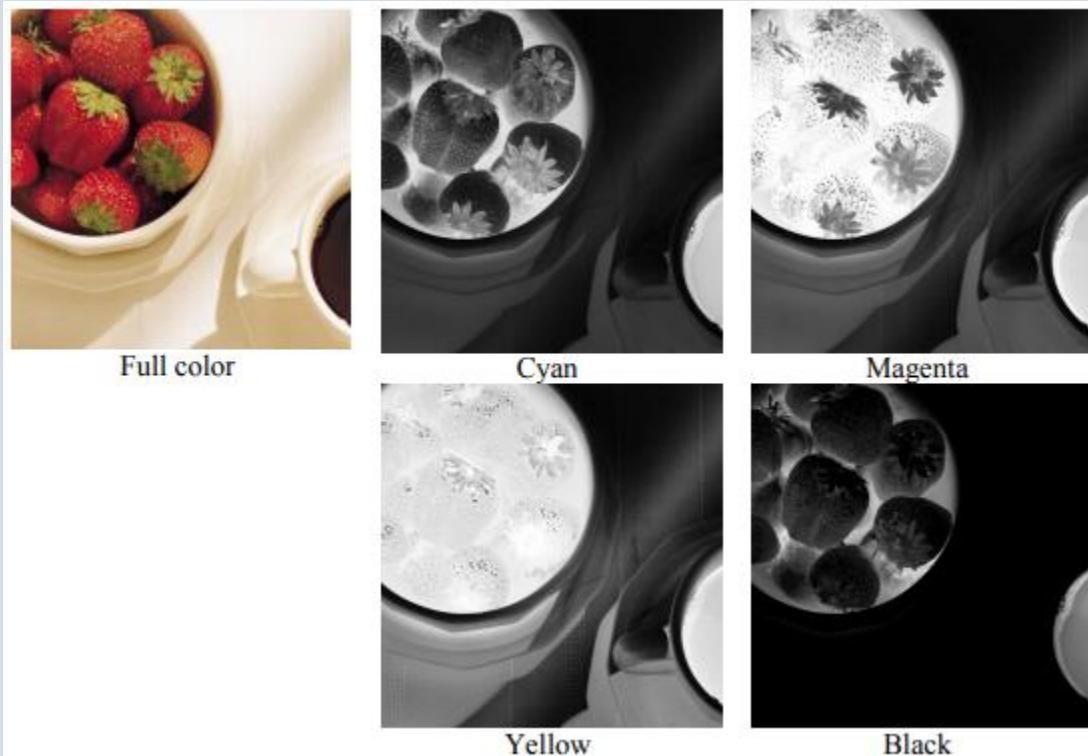
The CMY and CMYK color model

Cyan, magenta, and yellow are the primary colors of pigments. Most printing devices such as color printers and copiers require CMY data input or perform an RGB to CMY conversion internally. This conversion is performed using the equation

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

where, all color values have been normalized to the range [0, 1]. In printing, combining equal amounts of cyan, magenta, and yellow produce muddy-looking black. In order to produce true black, a fourth color, black, is added, giving rise to the CMYK color model.

The figure below shows the CMYK component images of an RGB image.



A full-color image and its CMYK component images

LESSON#5 IMAGE PROCESSING & COMPUTER VISION

The HIS color model

The RGB and CMY color models are not suited for describing colors in terms of human interpretation. When we view a color object, we describe it by its hue, saturation, and brightness(intensity). Hence the HSI color model has been presented. The HSI model decouples the intensity component from the color-carrying information (hue and saturation) in a color image. As a result, this model is an ideal tool for developing color image processing algorithms.

The goal of this lab

It is the knowledge how to separate the RGB, CMY and CMYK colors in MATLAB.

- Extracting the Color Spaces in MATLAB
 - ❖ **Size** → Array size.
- **sz = size(A)** returns a row vector whose elements contain the length of the corresponding dimension of A. For example, if A is a 3-by-4 matrix, then **size(A)** returns the vector [3 4]. The length of sz is **ndims(A)**.
If A is a table or timetable, then **size(A)** returns a two-element row vector consisting of the number of rows and the number of table variables.
- **szdim = size(A,dim)** returns the length of dimension dim.
- **[m,n] = size(A)** returns the number of rows and columns when A is a matrix.
- **[sz1,...,szN] = size(A)** returns the length of each dimension of A separately.

Procedure

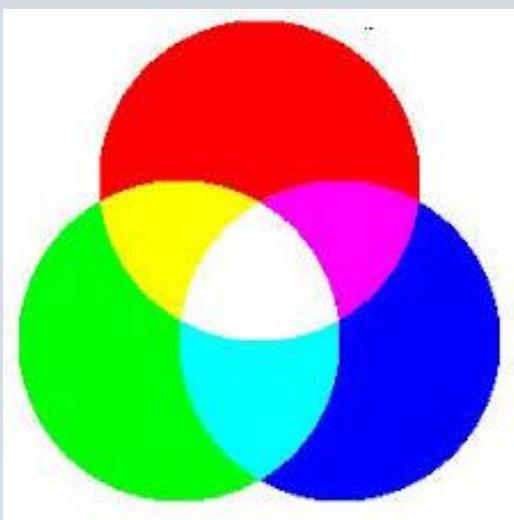
- 1- save the image in the current folder.
- 2- Open new script file and write the following:

```
clc
clear
%input the image
image1=imread('RGB.jpg');
```

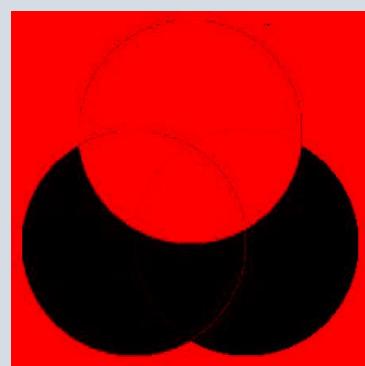
LESSON#5 IMAGE PROCESSING & COMPUTER VISION

```
% take the size of the image
[r c d]=size (image1)
%make the matrix of zeros
z=zeros(r,c);
%extract the red color
tempr=image1;
tempr(:,:,2)=z;
tempr(:,:,3)=z;
figure
imshow(tempr)
%extract the green color
tempg=image1;
tempg(:,:,1)=z;
tempg(:,:,3)=z;
figure
imshow(tempg)
%extract the blue color
tempb=image1;
tempb(:,:,1)=z;
tempb(:,:,2)=z;
figure
imshow(tempb)
```

The Result

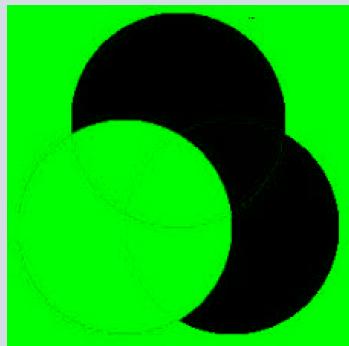


Original Image

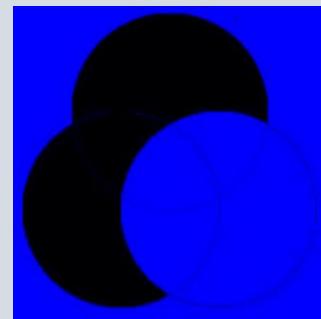


Extract Red

LESSON#5 IMAGE PROCESSING & COMPUTER VISION



Extract Green



Extract Blue

Extract Color Images

- CMY Color in MATLAB

Procedure

3- save the image in the current folder.

4- Open new script file and write the following:

```
clc
clear all
close all
i= imgetfile;
im= imread (i);
% color models

figure (1);
subplot (2,2,1),imshow(im), title('original image');

% Read Image
imr= im;
imr(:,:,:,2:3)=0; % Green & Blue =0
subplot (2,2,2),imshow(imr), title('Red Image');
% Green Image
img= im;
```

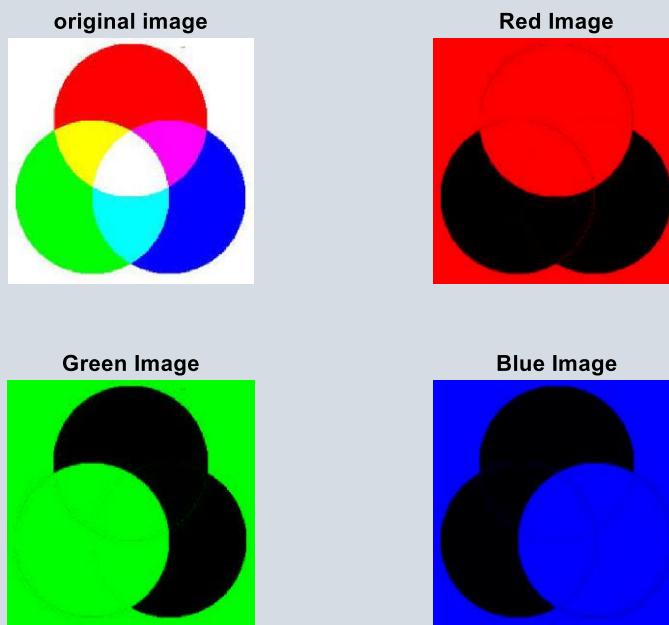
LESSON#5 IMAGE PROCESSING & COMPUTER VISION

```
img(:,:,1:2:3)=0; % Red & Blue =0
subplot (2,2,3),imshow(img), title('Green Image');
% Blue Image
imb= im;
imb(:,:,1:2)=0; % Red & Green =0
subplot (2,2,4),imshow(imb), title('Blue Image');

%CMY Model

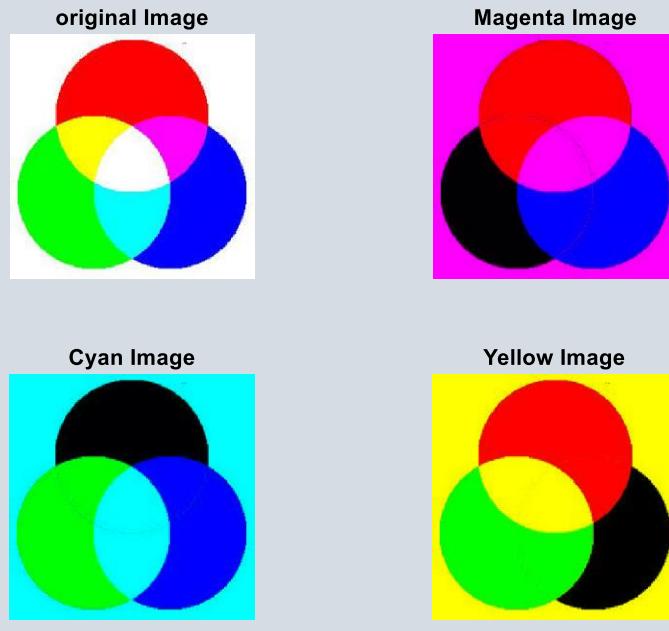
figure (2);
subplot(2,2,1), imshow(im),title('original Image');
% Magenta = Red + Blue
Mag= imr+imb;
subplot(2,2,2),imshow(Mag), title('Magenta Image');
%Cyan= Green + Blue
cyn= img + imb;
subplot(2,2,3), imshow(cyn), title('Cyan Image');
%Yellow= imr + img
Yellow= imr + img;
subplot(2,2,4), imshow(Yellow), title('Yellow Image');
```

The Result



RGB Model

LESSON#5 IMAGE PROCESSING & COMPUTER VISION



CMY Model

Report

1. Repeated the procedure but display the images in the same figure.
2. Explain the types of size function in MATLAB with examples.
3. Write the program to generate the CMYK colors in any image.
4. What is the different between RGB and CMY? With examples.

Lesson#6 Image Processing & COMPUTER VISION

**Color Image Types (Indexed Images & RGB (TrueColor)
Images)**

Eng. Elaf A.Saeed
Email: elafe1888@gmail.com

LAB Content

Index image

RGB (TrueColor) Images

Table of Functions

| Function Name | Description |
|--|---|
| Image <code>image(C)</code> | <ul style="list-style-type: none"> - Display image from array. - Each element of C specifies the color for 1 pixel of the image. The resulting image is an m-by-n grid of pixels where m is the number of rows and n is the number of columns in C. The row and column indices of the elements determine the centers of the corresponding pixels. |
| colormap <code>colormap(target,map)</code> | <ul style="list-style-type: none"> - View and set current colormap. - sets the colormap for the figure, axes, or chart specified by target, instead of for the current figure. |
| imtool <code>imtool(X,cmap)</code> | <ul style="list-style-type: none"> - Open Image Viewer app - displays the indexed image X with colormap cmap in the Image Viewer. |

Theory

Image Types

Indexed Images

An indexed image consists of a data matrix, X, and a colormap matrix, map. map is an m-by-3 array of class double containing floating-point values in the range [0, 1].

Each row of map specifies the red, green, and blue components of a single color.

An indexed image uses “direct mapping” of pixel values to colormap values. The color of each image pixel is determined by using the corresponding value of X as an index into map. Values of X therefore must be integers. The value 1 points to the first row in map, the value 2 points to the second row, and so on. Display an indexed image with the statements:

image(X); colormap(map)

A **colormap** is often stored with an indexed image and is automatically loaded with the image when you use the imread function. The next figure 6.1 illustrates the structure of an indexed image. The pixels in the image are represented by integers, which are pointers (indices) to color values stored in the colormap.

LESSON#6 IMAGE PROCESSING & COMPUTER VISION

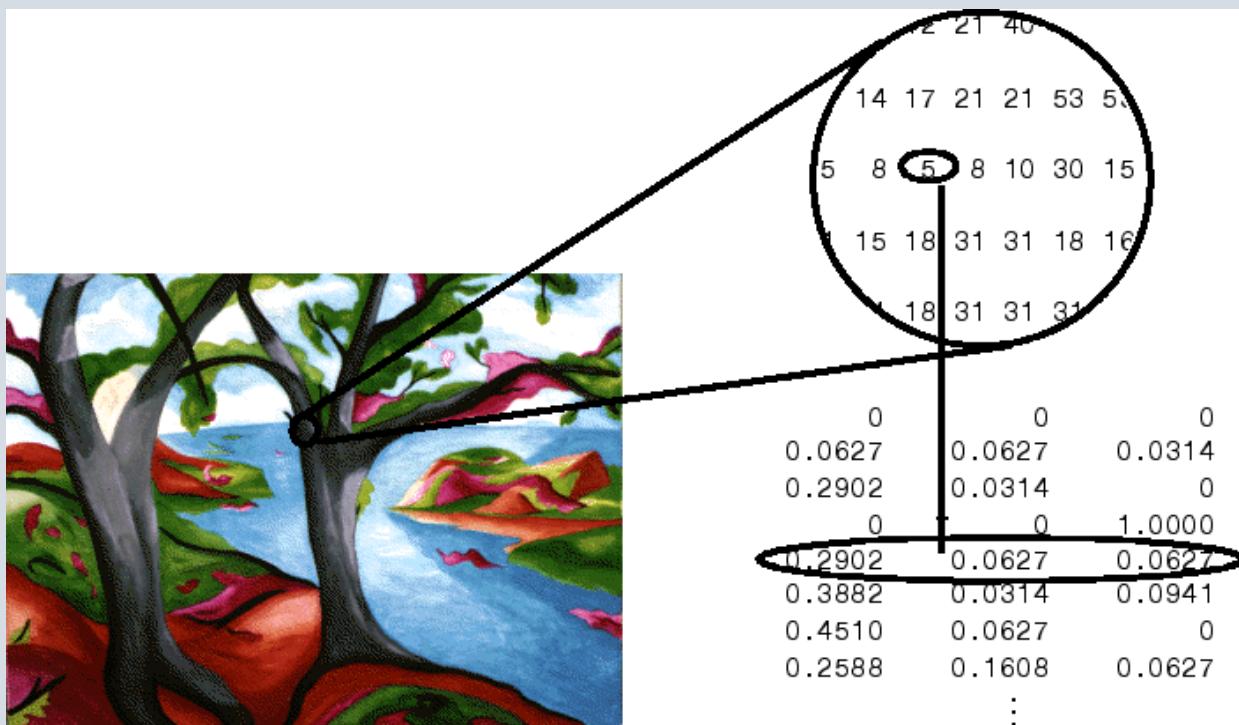


Figure 6.1: structure of an indexed image

The relationship between the values in the image matrix and the colormap depends on the class of the image matrix. If the image matrix is of class double, the value 1 points to the first row in the colormap, the value 2 points to the second row, and so on. If the image matrix is of class uint8 or uint16, there is an offset—the value 0 points to the first row in the colormap, the value 1 points to the second row, and so on. The offset is also used in graphics file formats to maximize the number of colors that can be supported. In the preceding image, the image matrix is of class double. Because there is no offset, the value 5 points to the fifth row of the colormap.

To read indexed image:

```
>> [x map]=imread('RGB.jpg');  
>> imtool(x,map)
```

RGB (Truecolor) Images

An RGB image, sometimes referred to as a truecolor image, is stored as an m-by-n-by-3 data array that defines red, green, and blue color components for each

LESSON#6 IMAGE PROCESSING & COMPUTER VISION

individual pixel. RGB images do not use a palette. The color of each pixel is determined by the combination of the red, green, and blue intensities stored in each color plane at the pixel's location. Graphics file formats store RGB images as 24-bit images, where the red, green, and blue components are 8 bits each. This yields a potential of 16 million colors. The precision with which a real-life image can be replicated has led to the nickname “truecolor image.”

An RGB MATLAB® array can be of class double, uint8, or uint16. In an RGB array of class double, each color component is a value between 0 and 1. A pixel whose color components are (0,0,0) is displayed as black, and a pixel whose color components are (1,1,1) is displayed as white. The three-color components for each pixel are stored along the third dimension of the data array. For example, the red, green, and blue color components of the pixel (10,5) are stored in RGB(10,5,1), RGB(10,5,2), and RGB(10,5,3), respectively.

To display the truecolor image RGB, use the image function:

image(RGB)

The next figure 6.2 shows an RGB image of class double.

LESSON#6 IMAGE PROCESSING & COMPUTER VISION

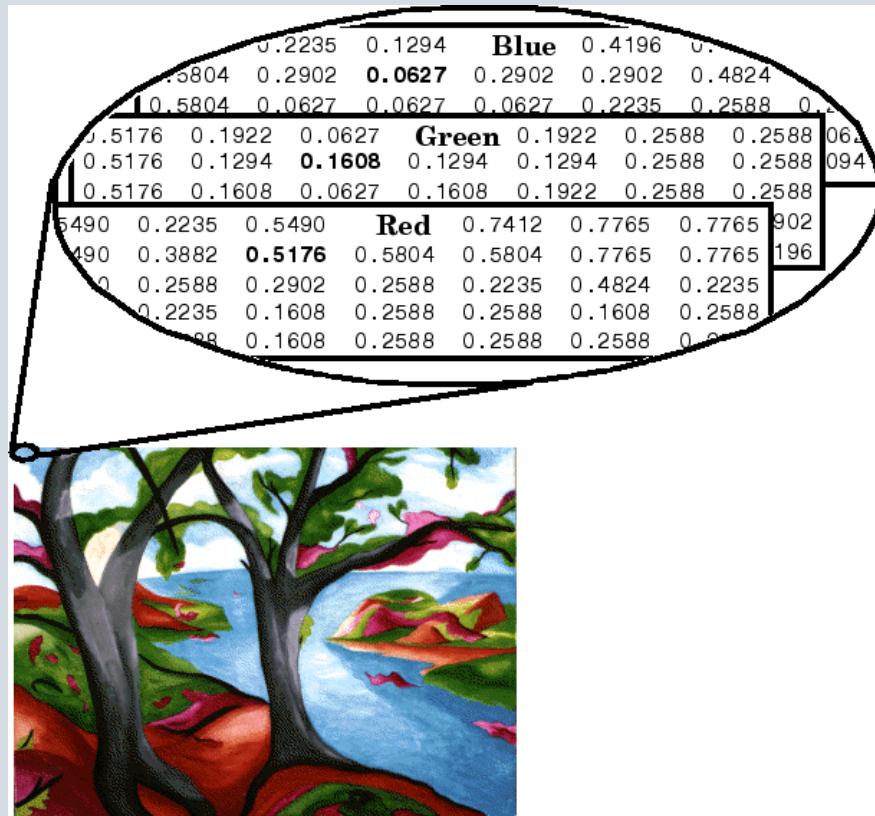


Figure 6.2: RGB image of class double

To determine the color of the pixel at (2,3), look at the RGB triplet stored in (2,3,1:3). Suppose (2,3,1) contains the value 0.5176, (2,3,2) contains 0.1608, and (2,3,3) contains 0.0627. The color for the pixel at (2,3) is **0.5176 0.1608 0.0627**

The goal of this lab

It is the knowledge how to separate the RGB, CMY and CMYK colors in MATLAB.

Procedure

- **Index image**

1. save the image in the current folder.
2. Write in the command window the following:

LESSON#6 IMAGE PROCESSING & COMPUTER VISION

```
>> [x map]=imread('RGB.jpg'); %read the image  
>> imtool(x,map) % display the image
```

The Result

In figure 6.3 that shows the image when we can read the index value of each.

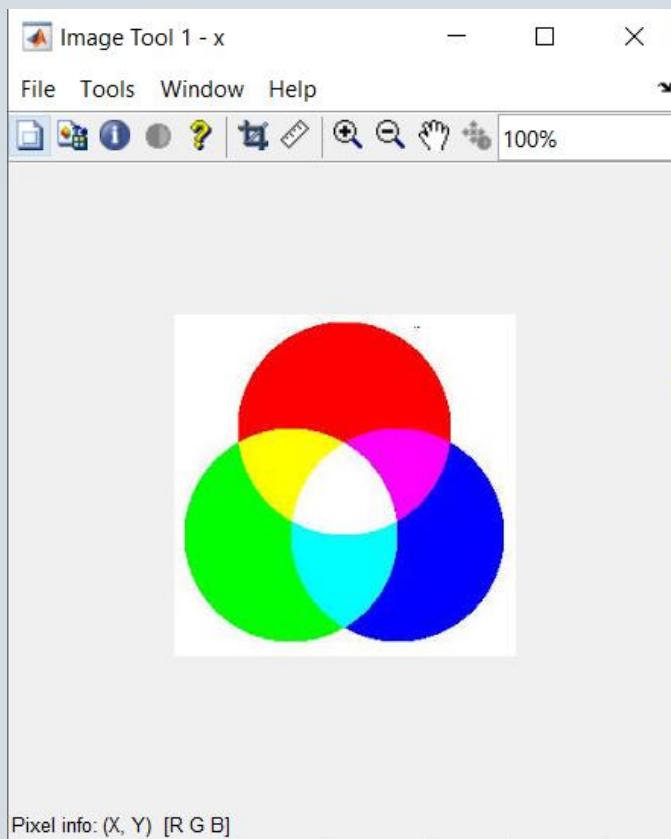


Figure 6.3: Index Image

- RGB (Truecolor) Images
3. save the image in the current folder.
 4. Open new script file and write the following:

```
% Truecolor Image  
clc  
clear all  
close all
```

LESSON#6 IMAGE PROCESSING & COMPUTER VISION

```
i=imread('RGB1.jpg');
figure
imshow(i)
R=i(2,3,1) %Red color value
G=i(2,3,2) %Green color value
B=i(2,3,3) %Blue color value
% Display all Color methode1
RGB1=i(2,3,1:3)
% Display all Color methode2
RGB2=i(2,3,:)
%impixel function to display all color values
impixel(i,100,100)
```

The Result

R =

uint8

203

G =

uint8

223

B =

uint8

212

1×1×3 uint8 array

RGB1(:,:,1) =

203

RGB1(:,:,2) =

LESSON#6 IMAGE PROCESSING & COMPUTER VISION

223

RGB1(:, :, 3) =

212

1×1×3 uint8 array

RGB2(:, :, 1) =

203

RGB2(:, :, 2) =

223

RGB2(:, :, 3) =

212

ans =

169 61 0

In figure below that shows the original image.



LESSON#6 IMAGE PROCESSING & COMPUTER VISION

Report

1. Explain what is the grayscale image with examples in MATLAB.
2. Explain each of the 2, 3, 4, 5, 6-bit color format.

Lesson#7 Image Processing & COMPUTER VISION

Math Operations with Images in MATLAB

Eng. Elaf A.Saeed
Email: elafe1888@gmail.com

LAB Content

Add Operation

Subtract Operation

Multiply Operation

Divide Operation

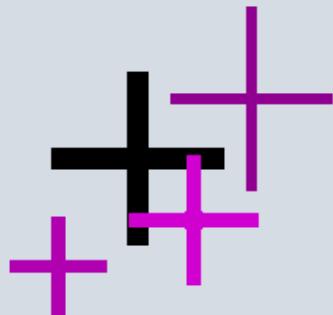
Table of Functions

| Function Name | Description |
|--|---|
| Imadd <code>imadd(X,Y)</code> | - Add two images or add constant to image. adds each element in array X with the corresponding element in array Y and returns the sum in the corresponding element of the output array Z. |
| imsubtract <code>imsubtract(X,Y)</code> | subtracts each element in array Y from the corresponding element in array X and returns the difference in the corresponding element of the output array Z. |
| Immultiply <code>Z= immultiply(X,Y)</code> | Multiply two images or multiply image by constant. multiplies each element in array X by the corresponding element in array Y and returns the product in the corresponding element of the output array Z. |
| Imdivide <code>Z = imdivide(X,Y)</code> | Divide one image into another or divide image by constant. <code>Z = imdivide(X,Y)</code> divides each element in the array X by the corresponding element in array Y and returns the result in the corresponding element of the output array Z. |

Theory

Image Arithmetic

1- Pixel Addition



Addition - pointwise addition: image + image (or constant)

In its most straightforward implementation, this operator takes as **input** two identically sized images and produces as output a third image of the **same size** as the first two, in which each pixel value is the sum of the values of the corresponding pixel from each of the two input images. More sophisticated versions allow more than two images to be combined with a single operation.

A common variant of the operator simply allows a specified constant to be added to every pixel.

How It Works

The addition of two images is performed straightforwardly in a single pass. The output pixel values are given by:

$$Q(i, j) = P_1(i, j) + P_2(i, j)$$

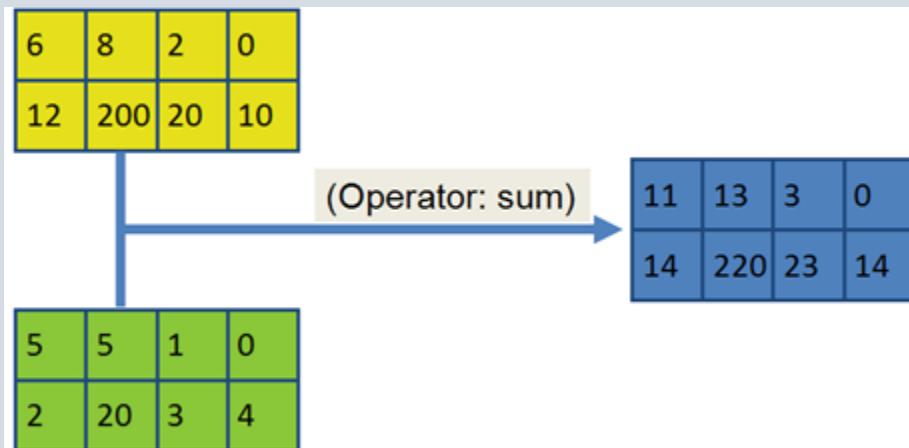
Or if it is simply desired to add a **constant** value C to a single image then:

$$Q(i, j) = P_1(i, j) + C$$

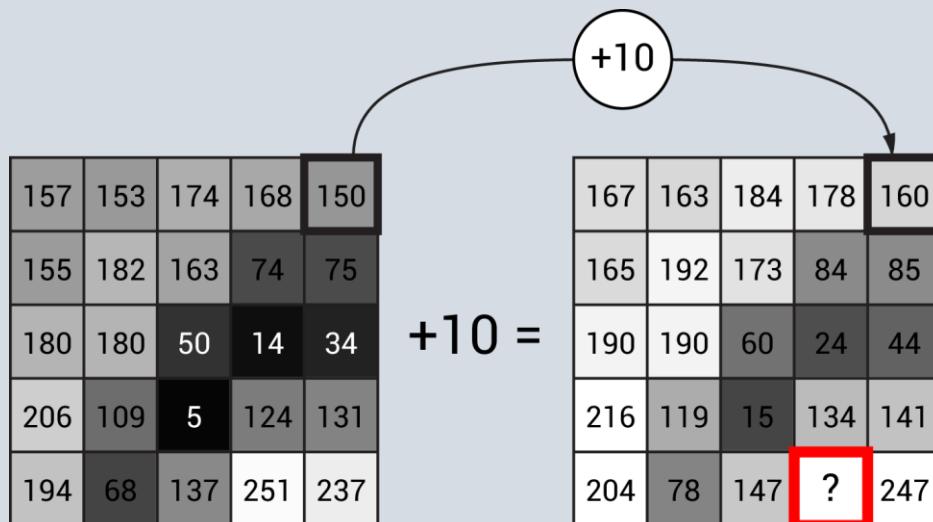
LESSON#7 IMAGE PROCESSING & COMPUTER VISION

If the pixel values in the input images are actually vectors rather than scalar values (e.g. for color images) then the individual components (e.g. red, blue and green components) are simply added separately to produce the output value.

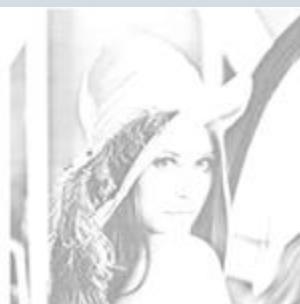
Add two images:



Add scalar to image:



$$+ 128 =$$



LESSON#7 IMAGE PROCESSING & COMPUTER VISION



2- Pixel Subtraction



Subtraction - pointwise subtraction: image - image (or constant)

The pixel subtraction operator takes two images as input and produces as output a third image whose pixel values are simply those of the first image minus the corresponding pixel values from the second image. It is also often possible to just use a single image as input and subtract a constant value from all the pixels. Some versions of the operator will just output the absolute difference between pixel values, rather than the straightforward signed output.

How It Works

The subtraction of two images is performed straightforwardly in a single pass. The output pixel values are given by:

$$Q(i, j) = P_1(i, j) - P_2(i, j)$$

Or if the operator computes absolute differences between the two input images then:

$$Q = |P_1(i, j) - P_2(i, j)|$$

Or if it is simply desired to subtract a constant value C from a single image then:

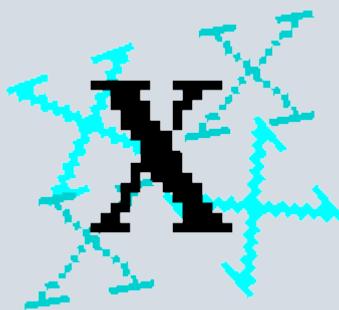
LESSON#7 IMAGE PROCESSING & COMPUTER VISION

$$Q = P_1(i, j) - C$$

If the pixel values in the input images are actually vectors rather than scalar values (e.g. for color images) then the individual components (e.g. red, blue and green components) are simply subtracted separately to produce the output value.



3- Pixel Multiplication and Scaling



Multiplication - pointwise multiplication: images * image (or constant)

Like other image arithmetic operators, multiplication comes in two main forms. The first form takes two input images and produces an output image in which the pixel

LESSON#7 IMAGE PROCESSING & COMPUTER VISION

values are just those of the first image, multiplied by the values of the corresponding values in the second image. The second form takes a single input image and produces output in which each pixel value is multiplied by a specified constant. This latter form is probably the more widely used and is generally called scaling.

How It Works

The multiplication of two images is performed in the obvious way in a single pass using the formula:

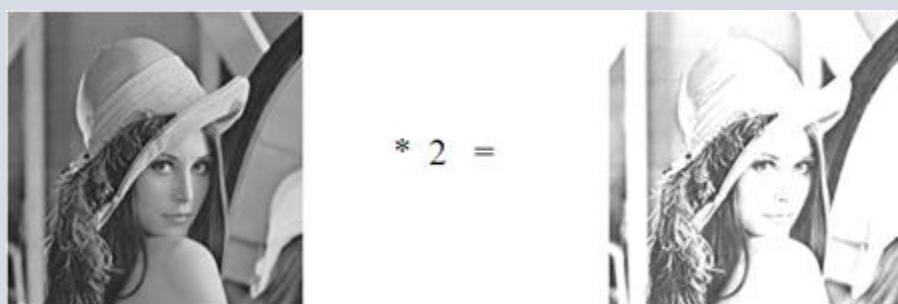
$$Q(i, j) = P_1(i, j) \times P_2(i, j)$$

Scaling by a constant is performed using:

$$Q(i, j) = P_1(i, j) \times C$$

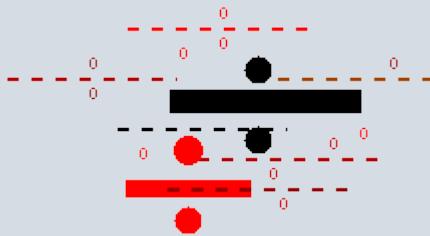
Note that the constant is often a floating-point number, and may be less than one, which will reduce the image intensities. It may even be negative if the image format supports that.

If the pixel values are actually vectors rather than scalar values (e.g. for color images) then the individual components (e.g. ref{rgb}{red, blue and green components}) are simply multiplied separately to produce the output value.



4- Pixel Division

LESSON#7 IMAGE PROCESSING & COMPUTER VISION



Division - pointwise division: images / image (or constant)

The image division operator normally takes two images as input and produces a third whose pixel values are just the pixel values of the first image divided by the corresponding pixel values of the second image. Many implementations can also be used with just a single input image, in which case every pixel value in that image is divided by a specified constant.

How It Works

The division of two images is performed in the obvious way in a single pass using the formula:

$$Q(i, j) = P_1(i, j) \div P_2(i, j)$$

Division by a constant is performed using:

$$Q(i, j) = P_1(i, j) \div C$$

If the pixel values are actually vectors rather than scalar values (e.g. for color images) than the individual components (e.g. red, blue and green components) are simply divided separately to produce the output value.

- Multiplication and division are used to adjust the brightness of an image.

The goal of this lab

It is the knowledge how to use arithmetic operation with image in MATLAB.

Procedure

1- Add Operation

LESSON#7 IMAGE PROCESSING & COMPUTER VISION

1. save the image in the current folder.
2. Open new script and write the following:

A-Read two grayscale uint8 images into the workspace.

```
I = imread('rice.png');  
J = imread('cameraman.tif');  
%Add the images. Specify the output as type uint16 to  
avoid truncating the result.  
K = imadd(I,J,'uint16');  
%Display the result.  
imshow(K,[])
```

The Result

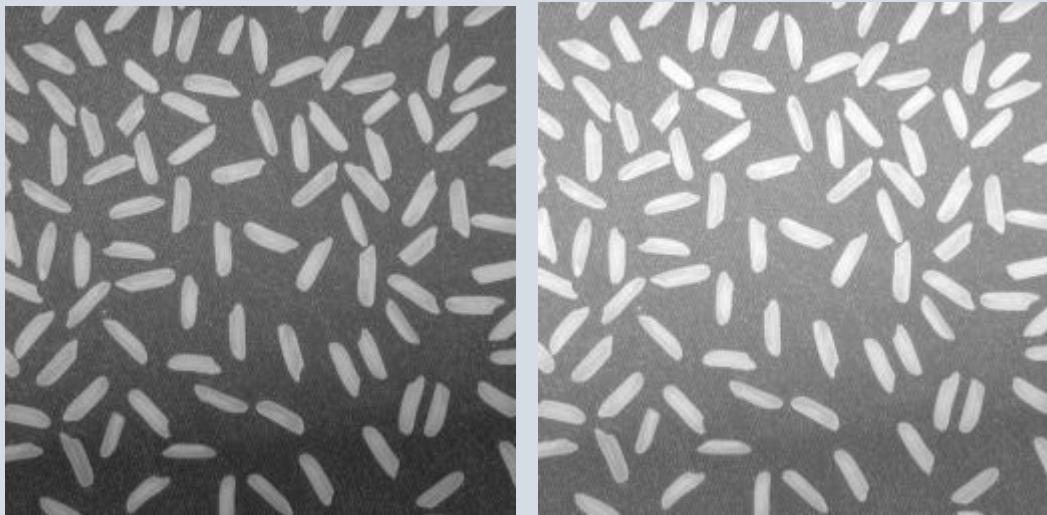


B- Add a Constant to an Image.

```
I = imread('rice.png');  
%Add a constant to the image.  
J = imadd(I,50);  
%Display the original image and the result.  
imshow(I)  
figure  
imshow(J)
```

The Result

LESSON#7 IMAGE PROCESSING & COMPUTER VISION



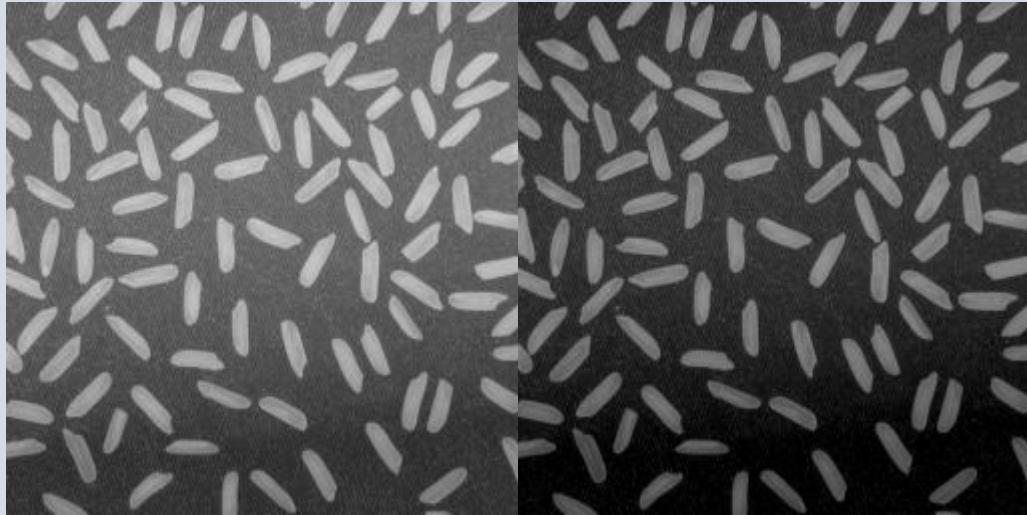
2- Subtract Operation

1. save the image in the current folder.
2. Open new script and write the following:

A-Subtract a Constant from an Image

```
I = imread('rice.png');  
%Subtract a constant value from the image.  
J = imsubtract(I,50);  
%Display the original image and the result.  
imshow(I)  
figure  
imshow(J)
```

LESSON#7 IMAGE PROCESSING & COMPUTER VISION



B- Subtract Image Background

```
I = imread('rice.png');
%Estimate the background.

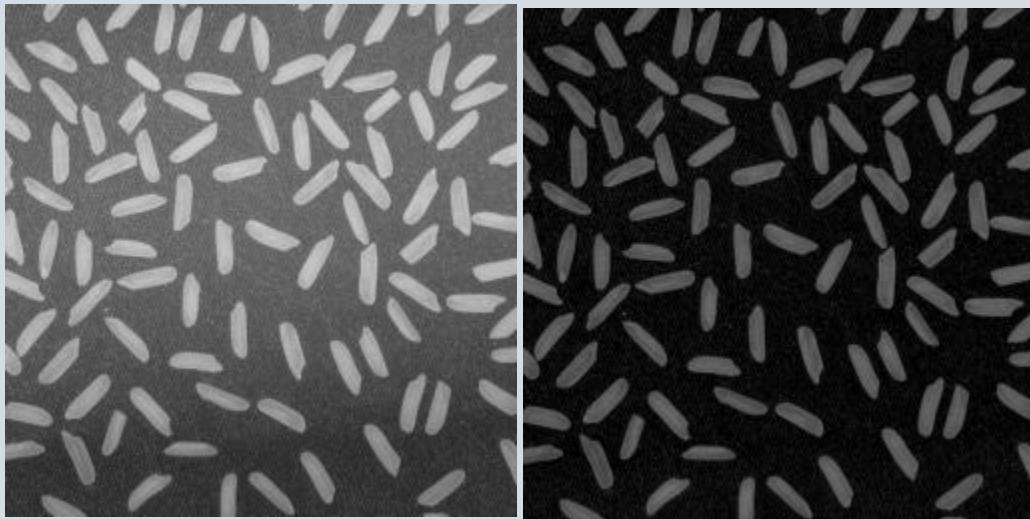
background = imopen(I,strel('disk',15));
%Subtract the background from the image.

J = imsubtract(I,background);
%Display the original image and the processed image.

imshow(I)
figure
imshow(J)
```

The Result

LESSON#7 IMAGE PROCESSING & COMPUTER VISION



3- Multiply Operation

1. save the image in the current folder.
2. Open new script and write the following:

A-Multiply an Image by Itself

```
%Read a grayscale image into the workspace, then  
convert the image to uint8.  
I = imread('moon.tif');  
I16 = uint16(I);  
%Multiply the image by itself. Note that immultiply  
converts the class of the image from uint8 to uint16  
before performing the multiplication to avoid  
truncating the results.  
J = immultiply(I16,I16);  
%Show the original image and the processed image.  
imshow(I)  
figure  
imshow(J)
```

The Result

LESSON#7 IMAGE PROCESSING & COMPUTER VISION



A- Scale an Image by a Constant Factor

```
%Read an image into the workspace.  
I = imread('moon.tif');  
%Scale each value of the image by a constant factor  
%of 0.5.  
J = immultiply(I,0.5);  
%Display the original image and the processed image.  
imshow(I)  
figure  
imshow(J)
```

The Result



4-Divide Operation

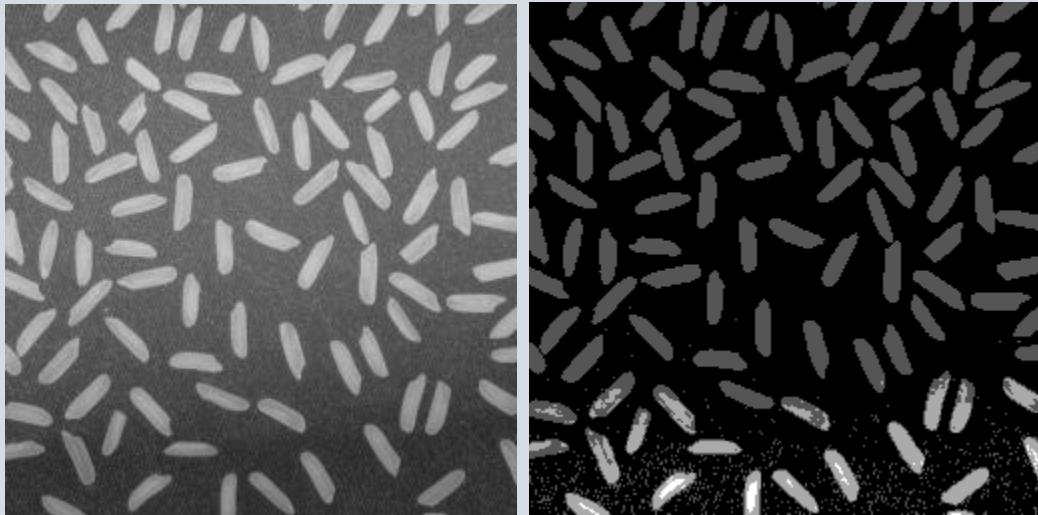
1. save the image in the current folder.
2. Open new script and write the following:

A-Divide Image Background

```
%Read a grayscale image into the workspace.  
I = imread('rice.png');  
%Estimate the background.  
background = imopen(I,strel('disk',15));  
%Divide out the background from the image.  
J = imdivide(I,background);  
%Display the original image and the processed image.  
imshow(I)  
figure  
imshow(J, [ ])
```

The Result

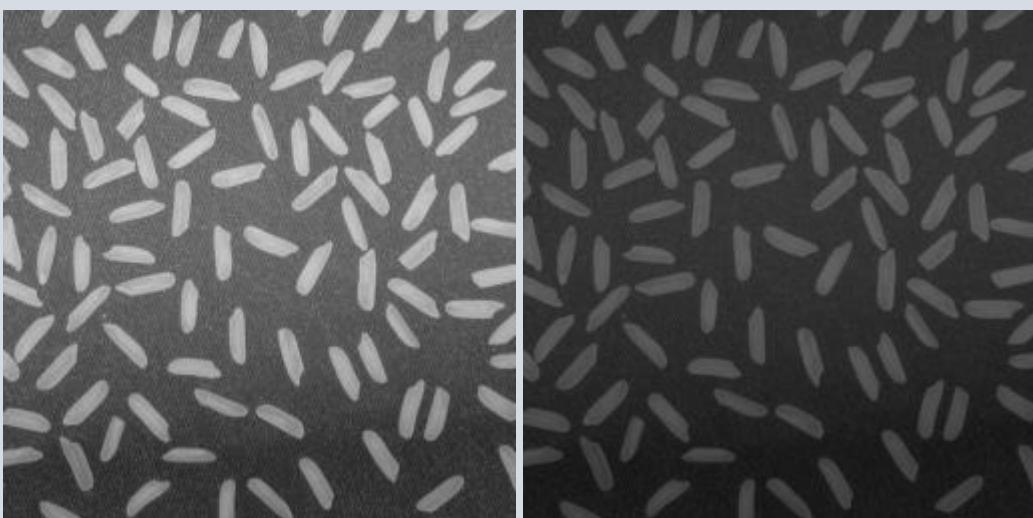
LESSON#7 IMAGE PROCESSING & COMPUTER VISION



B-Divide an Image by a Constant Factor

```
%Read an image into the workspace.  
I = imread('rice.png');  
%Divide each value of the image by a constant factor  
of 2.  
J = imdivide(I,2);  
%Display the original image and the processed image.  
imshow(I)  
figure  
imshow(J)
```

The Result



Report

1. Write the script code in MATLAB to do the following:
 - a) Read the two-color images.
 - b) Add the two images.
 - c) Add the images with scale.
 - d) Subtract the two images.
 - e) Subtract the images with scale.
 - f) multiply the two images.
 - g) multiply the images with scale.
 - h) divide the two images.
 - i) divide the images with scale.

Lesson#8 Image Processing & COMPUTER VISION

Algebra Operations with Images in MATLAB

Eng. Elaf A.Saeed
Email: elafe1888@gmail.com

LAB Content

AND, OR, XOR and NOT Gate.

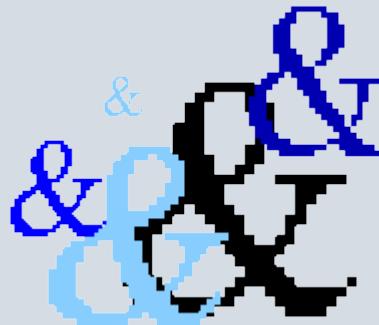
Table of Functions

| Function Name | Description |
|--|---|
| bitand $C = \text{bitand}(A, B)$ | Bit-wise AND. $C = \text{bitand}(A, B)$ returns the bit-wise AND of A and B. |
| bitor $C = \text{bitor}(A, B)$ | Bit-wise OR. $C = \text{bitor}(A, B)$ returns the bit-wise OR of A and B. |
| bitcmp $\text{cmp} = \text{bitcmp}(A)$ | Bit-wise complement. $\text{cmp} = \text{bitcmp}(A)$ returns the bit-wise complement of A. |
| bitxor $C = \text{bitxor}(A, B)$ | Bit-wise XOR. $C = \text{bitxor}(A, B)$ returns the bit-wise XOR of A and B. |

Theory

Image Arithmetic

A-Logical AND/NAND



Logical AND/NAND - pointwise logical ANDing/NANDing of two binary images

AND and NAND are examples of logical operators having the truth-tables shown in Figure

| A | B | Q |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

AND

| A | B | Q |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

NAND

How It Works

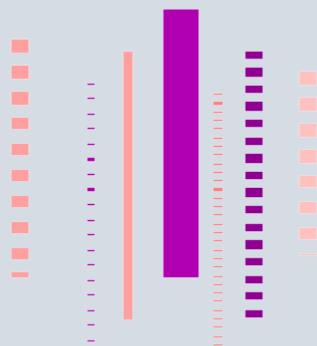
The operation is performed straightforwardly in a single pass. It is important that all the input pixel values being operated on have the same number of bits in them or unexpected things may happen. Where the pixel values in the input images are not simple 1-bit numbers, the AND operation is normally (but not always) carried out individually on each corresponding bit in the pixel values, in bitwise fashion.

LESSON#8 IMAGE PROCESSING & COMPUTER VISION



B- Logical OR/NOR

Logical OR/NOR - pointwise logical ORing/NORing of two binary images



OR and NOR are examples of logical operators having the truth-tables shown in Figure.

| A | B | Q |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

OR

| A | B | Q |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

NOR

As can be seen, the output values of NOR are simply the inverses of the corresponding output values of OR.

LESSON#8 IMAGE PROCESSING & COMPUTER VISION



C-Logical XOR/XNOR

Logical XOR/XNOR - pointwise logical XORing/XNORing of two binary images.



XOR and XNOR are examples of logical operators having the truth-tables shown in Figure.

| A | B | Q |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

XOR

| A | B | Q |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

XNOR

The XOR function is only true if just one (and only one) of the input values is true, and false otherwise. **XOR** stands for **exclusive OR**. As can be seen, the output values of XNOR are simply the inverse of the corresponding output values of **XOR**.

LESSON#8 IMAGE PROCESSING & COMPUTER VISION

The XOR (and similarly the XNOR) operator typically takes two binary or graylevel images as input, and outputs a third image whose pixel values are just those of the first image, XORed with the corresponding pixels from the second. A variation of this operator takes a single input image and XORs each pixel with a specified constant value in order to produce the output.

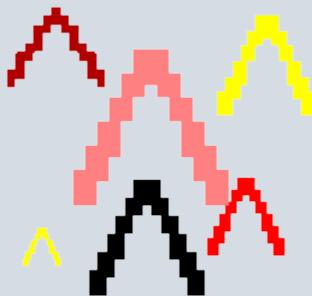
How It Works

The operation is performed straightforwardly in a single pass. It is important that all the input pixel values being operated on have the same number of bits in them, or unexpected things may happen. Where the pixel values in the input images are not simple 1-bit numbers, the XOR operation is normally (but not always) carried out individually on each corresponding bit in the pixel values, in bitwise fashion.



D-Invert/Logical NOT

Logical XOR/XNOR - pointwise logical XORing/XNORing of two binary images.



Logical NOT or invert is an operator which takes a binary or graylevel image as input and produces its photographic negative, i.e. dark areas in the input image become light and light areas become dark.

LESSON#8 IMAGE PROCESSING & COMPUTER VISION

How It Works

To produce the photographic negative of a binary image we can employ the logical NOT operator. Its truth-table is shown in Figure.

| A | Q |
|---|---|
| 0 | 1 |
| 1 | 0 |

NOT

Each pixel in the input image having a logical 1 (often referred to as foreground) has a logical 0 (associated with the background in the output image and *vice versa*). Hence, applying logical NOT to a binary image changes its polarity.

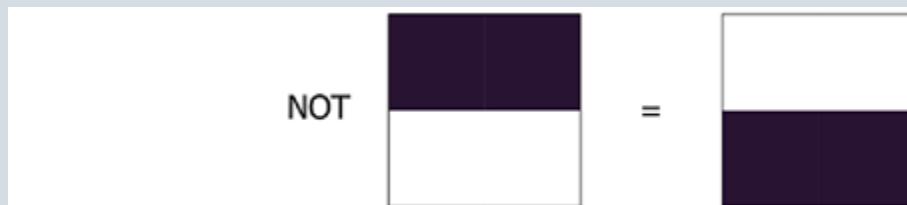
The logical NOT can also be used for a graylevel image being stored in byte pixel format by applying it in a bitwise fashion. The resulting value for each pixel is the input value subtracted from 255:

$$Q(i, j) = 255 - P(i, j)$$

Some applications of *invert* also support *integer* or *float* pixel format. In this case, we can't use the logical NOT operator, therefore the pixel values of the inverted image are simply given by

$$Q(i, j) = -P(i, j)$$

If this output image is normalized for an 8-bit display, we again obtain the photographic negative of the original input image.



The goal of this lab

It is the knowledge how to use **Algebra operation** with image in MATLAB.

Procedure

- AND, OR, XOR and NOT Gate

1. save the image in the current folder.
2. Open new script and write the following:

```
%%
%Reading Required Images & required Preprocessing
clc
clear all
close all
x=imread('circle.png');
imshow(x);
title('Plus Structure');
figure;
y=imread('circle3.png');
[a b c]=size(x);
y=imresize(y, [a,b]);
imshow(y);
title('Octagon Structure');
%%
%And Operation
figure;
z=bitand(x,y);
imshow(z);
title('Output of And operation');
%%
%Or Operation
figure;
z=bitor(x,y);
imshow(z);
title('Output of Or operation');
%%
%Not Operation
```

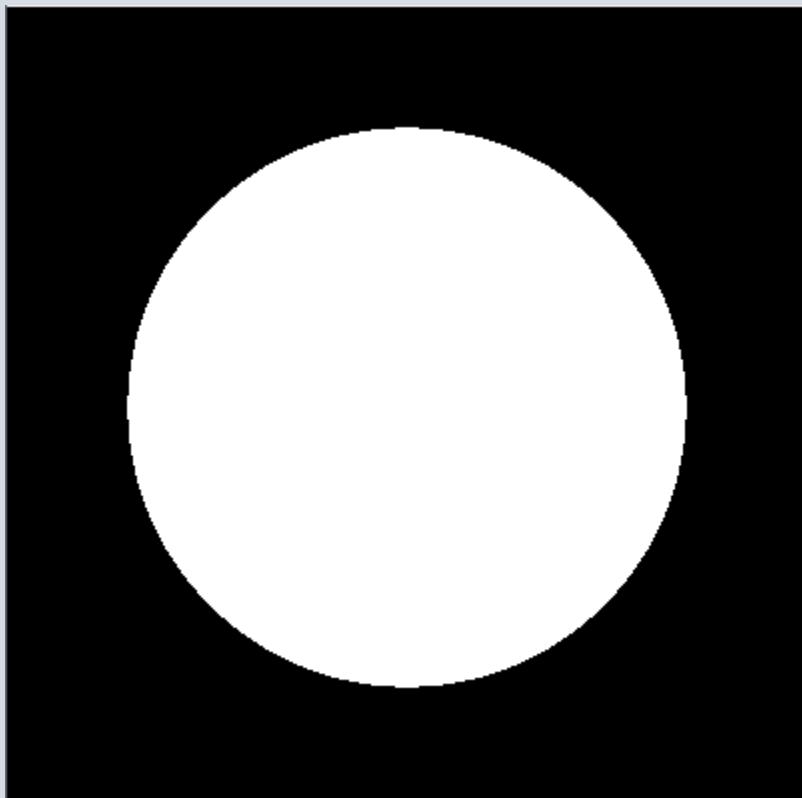
LESSON#8 IMAGE PROCESSING & COMPUTER VISION

```
z=bitcmp(x);  
imshow(z);  
title('Output of Not operation');  
%%  
%XOR Operation  
figure;  
z=bitxor(x,y);  
imshow(z);  
title('Output of XOR operation');  
A-
```

The Result

LESSON#8 IMAGE PROCESSING & COMPUTER VISION

circle Structure



circle3 Structure

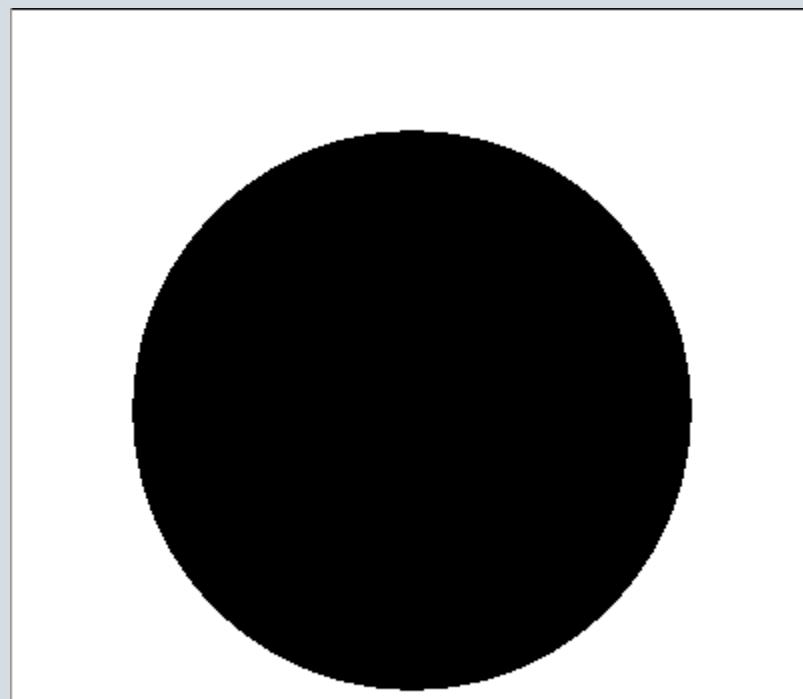


LESSON#8 IMAGE PROCESSING & COMPUTER VISION

Output of And operation



Output of Not operation



LESSON#8 IMAGE PROCESSING & COMPUTER VISION

Output of XOR operation



Report

1. Write the script code in MATLAB to do the following:

A-



LESSON#8 IMAGE PROCESSING & COMPUTER VISION

B-



Lesson#9 Image Processing & COMPUTER VISION

**Concept of Sampling and Quantization with Images in
MATLAB**

Eng. Elaf A.Saeed
Email: elafe1888@gmail.com

LAB Content

Sampling image

Quantization Images

Table of Functions

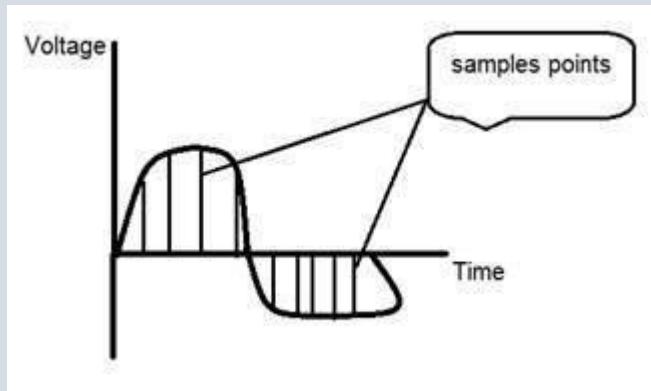
| Function Name | Description |
|--|---|
| griddedInterpolant F = griddedInterpolant(V) | <ul style="list-style-type: none"> - Gridded data interpolation. - uses the default grid to create the interpolant. When you use this syntax, griddedInterpolant defines the grid as a set of points whose spacing is 1 and range is [1, size(V,i)] in the ith dimension. Use this syntax when you want to conserve memory and are not concerned about the absolute distances between points. |
| uint8 | 8-bit unsigned integer arrays. |
| round Y = round(X) | <ul style="list-style-type: none"> - Round to nearest decimal or integer. - rounds each element of X to the nearest integer. In the case of a tie, where an element has a fractional part of exactly 0.5, the round function rounds away from zero to the integer with larger magnitude. |

Theory

Concept of Quantization

Digitizing a signal

As we have seen in the previous tutorials, that digitizing an analog signal into a digital, requires two basic steps. Sampling and quantization. Sampling is done on x axis. It is the conversion of x axis (infinite values) to digital values. The below figure shows sampling of a signal.



structure of an indexed image

Sampling with relation to digital images

The concept of sampling is directly related to zooming. The more samples you take, the more pixels, you get. Oversampling can also be called as zooming.

This has been discussed under sampling and zooming tutorial.

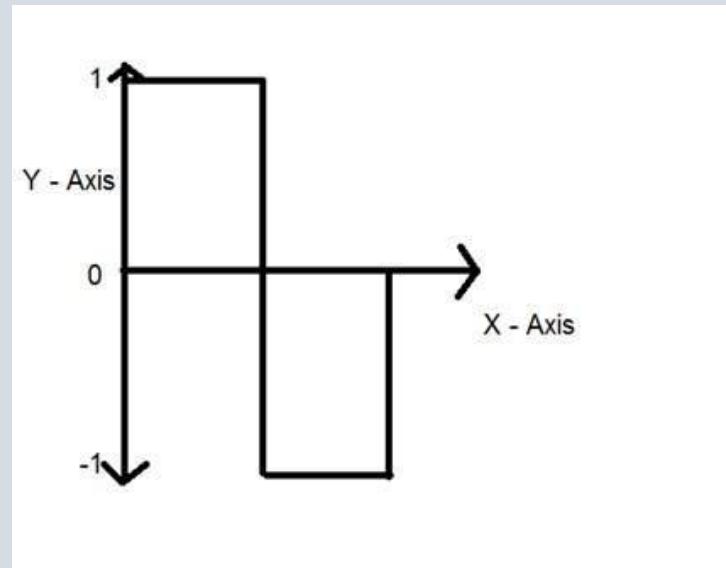
But the story of digitizing a signal does not end at sampling too, there is another step involved which is known as Quantization.

What is quantization

Quantization is opposite to sampling. It is done on y axis. When you are quantizing an image, you are actually dividing a signal into quanta(partitions).

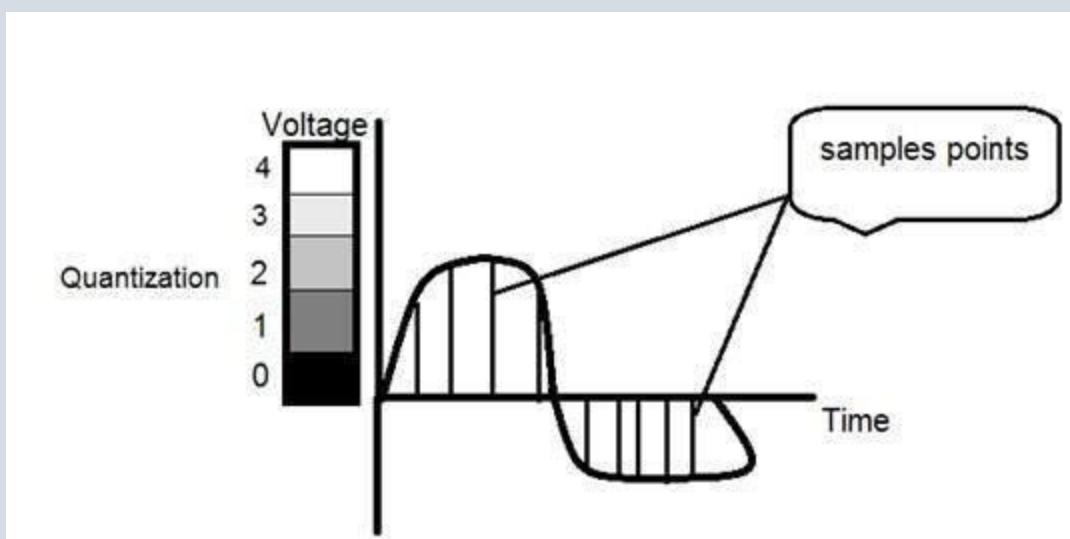
LESSON#9 IMAGE PROCESSING & COMPUTER VISION

On the x axis of the signal, are the co-ordinate values, and on the y axis, we have amplitudes. So digitizing the amplitudes is known as Quantization. In figure Here, that shows how it is done



Quantization

You can see in this image, that the signal has been quantified into three different levels. That means that when we sample an image, we actually gather a lot of values, and in quantization, we set levels to these values. This can be more clear in the figure below.



Quantization levels

LESSON#9 IMAGE PROCESSING & COMPUTER VISION

In the figure shown in sampling, although the samples have been taken, but they were still spanning vertically to a continuous range of gray level values. In the figure shown above, these vertically ranging values have been quantized into 5 different levels or partitions. Ranging from 0 black to 4 white. This level could vary according to the type of image you want.

The relation of quantization with gray levels has been further discussed below.

Relation of Quantization with gray level resolution:

The quantized figure 7.3 shown above has 5 different levels of gray. It means that the image formed from this signal, would only have 5 different colors. It would be a black and white image more or less with some colors of gray. Now if you were to make the quality of the image better, there is one thing you can do here. Which is, to increase the levels, or gray level resolution up. If you increase this level to 256, it means you have a **gray scale image**. Which is far better than simple black and white image.

Now 256, or 5 or whatever level you choose is called **gray level**. The gray level resolution which is,

$$L = 2^k$$

The gray level can be defined in two ways. Which were these two.

- Gray level = number of bits per pixel (BPP). (k in the equation)
- Gray level = number of levels per pixel.

In this case we have gray level is equal to 256. If we have to calculate the number of bits, we would simply put the values in the equation. In case of 256 levels, we have 256 different shades of gray and 8 bits per pixel, hence the image would be a gray scale image.

Reducing the gray level

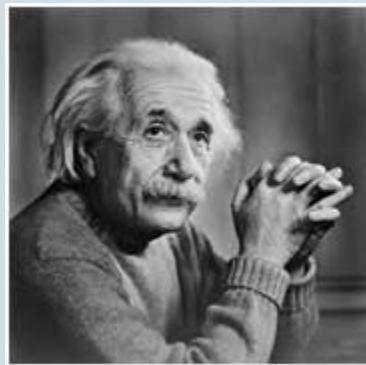
LESSON#9 IMAGE PROCESSING & COMPUTER VISION

Now we will reduce the gray levels of the image to see the effect on the image.

For example

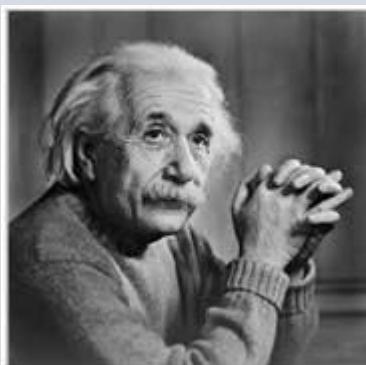
Let's say you have an image of 8bpp, that has 256 different levels. It is a grayscale image and the image looks something like this.

256 Gray Levels



Now we will start reducing the gray levels. We will first reduce the gray levels from 256 to 128.

128 Gray Levels

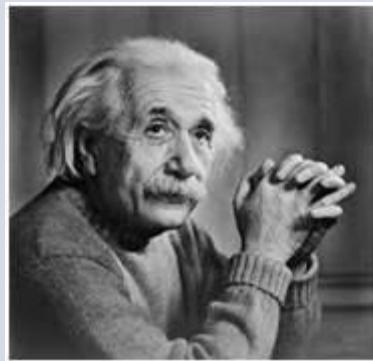


There is not much effect on an image after decrease the gray levels to its half.

Let's decrease some more.

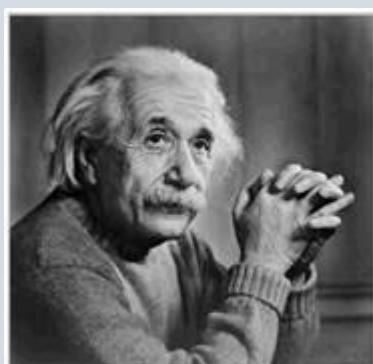
64 Gray Levels

LESSON#9 IMAGE PROCESSING & COMPUTER VISION



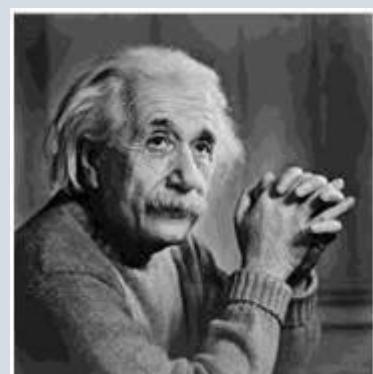
Still not much effect, then let's reduce the levels more.

32 Gray Levels



Surprised to see, that there is still some little effect. May be its due to reason, that it is the picture of Einstein, but lets reduce the levels more.

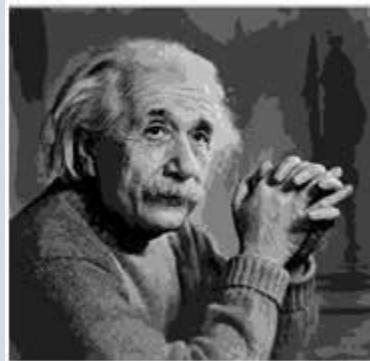
16 Gray Levels



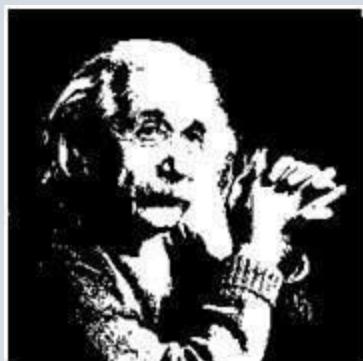
Boom here, we go, the image finally reveals, that it is effected by the levels.

8 Gray Levels

LESSON#9 IMAGE PROCESSING & COMPUTER VISION



4 Gray Levels



That's the last level we can achieve, because if reduce it further, it would be simply a black image, which cannot be interpreted.

Contouring

There is an interesting observation here, that as we reduce the number of gray levels, there is a special type of effect start appearing in the image, which can be seen clear in 16 gray level picture. This effect is known as Contouring.

Procedure

- Sampling image

1. save the image in the current folder.
2. Open new script and write the following:

```
% Sampling Image
```

```
clc
```

```
clear all
```

LESSON#9 IMAGE PROCESSING & COMPUTER VISION

```
close all
i=imread('birds.png');
figure
imshow(i)
% 512 to 256
%get the properties of the image and double I return
precision of Image
F = griddedInterpolant(double(i))
%Dimention of image in terms of x,y and z
[sx,sy,sz]=size(i)
% Reduce Sampling rate from 512 to 256 in x dimention
xq_256=(1:512/256:sx);
% Reduce Sampling rate from 512 to 256 in y dimention
yq_256= (1:512/256:sy);
%convert x,y into 8bit unsigned integer image
vq_256=uint8(F({xq_256,yq_256}));
figure
imshow(vq_256)

% 512 to 128
xq_128=(1:512/128:sx);
yq_128= (1:512/128:sy);
vq_128=uint8(F({xq_128,yq_128}));
figure
imshow(vq_128)

% 512 to 64
xq_64=(1:512/64:sx);
```

LESSON#9 IMAGE PROCESSING & COMPUTER VISION

```
yq_64= (1:512/64:sy);  
vq_64=uint8(F({xq_64, yq_64}));  
figure  
imshow(vq_64)
```

The Result

In figure 6.2 that shows the image when we can read the index value of each.



Original

LESSON#9 IMAGE PROCESSING & COMPUTER VISION



256 → 512



512 → 128



512 → 64

Figure 6.2: Index Image

LESSON#9 IMAGE PROCESSING & COMPUTER VISION

- Quantization Images

1. save the image in the current folder.
2. Open new script file and write the following:

```
% Quantization Image
clc
clear all
close all
im=imread('greeks.jpg'); % read image
imshow(im) % display image
b=[2 3 4 6 7 8];
for i=1:length(b)
    d= 2^b(i);
    z=round(im/d);
    figure
    imshow(z*d) %Z*d for more resolution
end
```

The Result



Original

LESSON#9 IMAGE PROCESSING & COMPUTER VISION



2 level

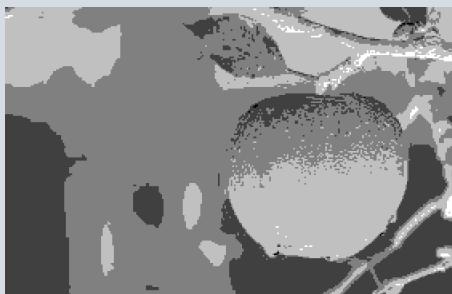


3 level

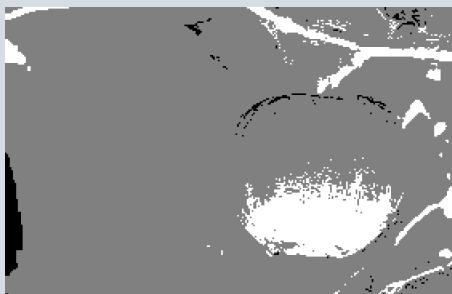


4 level

LESSON#9 IMAGE PROCESSING & COMPUTER VISION



6 level



7 level



8 level

LESSON#9 IMAGE PROCESSING & COMPUTER VISION

Report

1. Write a matlab function: $[im_q] = \text{quant}(im, N)$ that performs uniform quantization of the image im to N gray levels ($0 \leq N \leq 255$).
2. Explain what is the problem of “false contours” that occurs in image quantization? Suggest a method to reduce this artifact.

Lesson#10 Image Processing & COMPUTER VISION

Zooming and Shrinking the Images in MATLAB

Eng. Elaf A.Saeed
Email: elafe1888@gmail.com

LAB Content

Zoom-in & Zoom-out without built-in MATLAB Function

Zoom-in & Zoom-out with built-in MATLAB Function

Table of Functions

| Function | Description |
|---|---|
| Name | |
| Imresize B = imresize(A,scale) | <p>Resize image.</p> <p>B = imresize(A,scale) returns image B that is scale times the size of A. The input image A can be a grayscale, RGB, or binary image. If A has more than two dimensions, imresize only resizes the first two dimensions. If scale is in the range [0, 1], B is smaller than A. If scale is greater than 1, B is larger than A. By default, imresize uses bicubic interpolation.</p> |

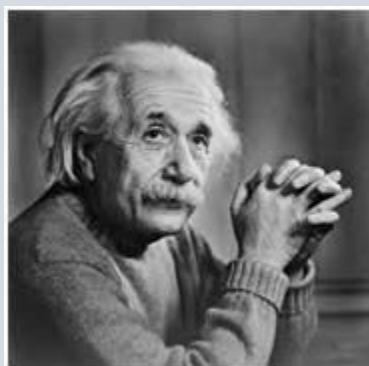
Theory

Concept of Zooming

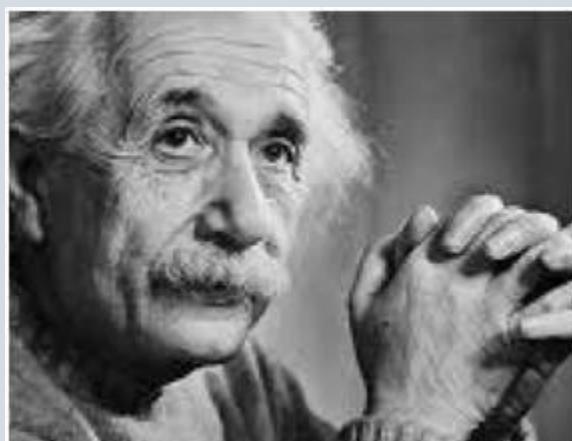
Zooming

Zooming simply means enlarging a picture in a sense that the details in the image became more visible and clearer. Zooming an image has many wide applications ranging from zooming through a camera lens, to zoom an image on internet etc.

For example



is zoomed into



You can zoom something at two different steps.

The first step includes zooming before taking a particular image. This is known as preprocessing zoom. This zoom involves hardware and mechanical movement.

The second step is to zoom once an image has been captured. It is done through many different algorithms in which we manipulate pixels to zoom in the required portion.

LESSON#10 IMAGE PROCESSING & COMPUTER VISION

Optical Zoom vs digital Zoom

These two types of zoom are supported by the cameras.

Optical Zoom:

The optical zoom is achieved using the movement of the lens of your camera. An optical zoom is actually a true zoom. The result of the optical zoom is far better than that of digital zoom. In optical zoom, an image is magnified by the lens in such a way that the objects in the image appear to be closer to the camera. In optical zoom the lens is physically extend to zoom or magnify an object.

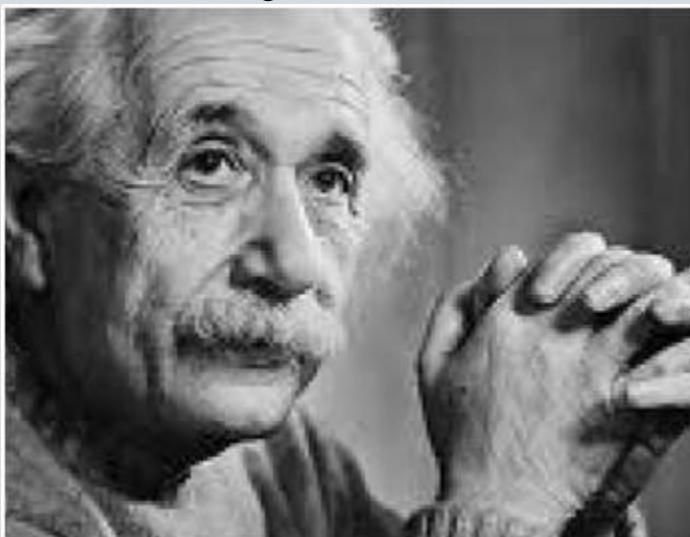
Digital Zoom:

Digital zoom is basically image processing within a camera. During a digital zoom, the center of the image is magnified and the edges of the picture got crop out. Due to magnified center, it looks like that the object is closer to you.

During a digital zoom, the pixels get expand, due to which the quality of the image is compromised.

The same effect of digital zoom can be seen after the image is taken through your computer by using an image processing toolbox / software, such as Photoshop.

The following picture is the result of digital zoom done through one of the following methods given below in the zooming methods.



Now since we are learning digital image processing, we will not focus on how an image can be zoomed optically using lens or other stuff. Rather we will focus on the methods, that enable to zoom a digital image.

Zooming methods:

Although there are many methods that do this job, but we are going to discuss the most common of them here.

LESSON#10 IMAGE PROCESSING & COMPUTER VISION

They are listed below.

- Pixel **replication** or (Nearest neighbor interpolation).
- Zero order hold method (or linear **interpolation**).
- Zero **Interpolation**.

Each of the methods have their own **advantages and disadvantages**. We will start by discussing pixel replication.

Replication

❖ Pixel replication or (Nearest neighbor interpolation)

Introduction:

It is also known as Nearest neighbor interpolation. As its name suggest, in this method, we just replicate the neighboring pixels. As we have already discussed in the tutorial of Sampling, that zooming is nothing but increase amount of sample or pixels. This algorithm works on the same principle.

Working:

In this method we create new pixels form the already given pixels. Each pixel is replicated in this method n times row wise and column wise and you got a zoomed image. It's as simple as that.

For example:

if you have an image of 2 rows and 2 columns and you want to zoom it twice or 2 times using pixel replication, here how it can be done.

| | |
|---|---|
| 1 | 2 |
| 3 | 4 |

For a better understanding, the image has been taken in the form of matrix with the pixel values of the image.

The above image has two rows and two columns, we will first zoom it row wise.

Row wise zooming:

When we zoom it row wise, we will just simple copy the rows pixels to its adjacent new cell.

Here how it would be done.

LESSON#10 IMAGE PROCESSING & COMPUTER VISION

| | | | |
|---|---|---|---|
| 1 | 1 | 2 | 2 |
| 3 | 3 | 4 | 4 |

As you can see that in the above matrix, each pixel is replicated twice in the rows.

Column size zooming:

The next step is to replicate each of the pixel column wise, that we will simply copy the column pixel to its adjacent new column or simply below it.

Here how it would be done.

| | | | |
|---|---|---|---|
| 1 | 1 | 2 | 2 |
| 1 | 1 | 2 | 2 |
| 3 | 3 | 4 | 4 |
| 3 | 3 | 4 | 4 |

New image size:

As it can be seen from the above example, that an original image of 2 rows and 2 columns has been converted into 4 rows and 4 columns after zooming. That means the new image has a dimension of

(Original image rows * zooming factor, Original Image cols * zooming factor)

Advantage and disadvantage:

One of the **advantages** of this zooming technique is, it is very simple. You just have to copy the pixels and nothing else.

The **disadvantage** of this technique is that image got zoomed but the output is very blurry. And as the zooming factor increased, the image got more and more blurred. That would eventually result in fully blurred image.

Interpolation

A- Zero order hold (or linear Interpolation)

Introduction

Zero order hold method is another method of zooming. It is also known as zoom twice. Because it can only zoom twice. We will see in the below example that why it does that.

Working

In zero order hold method, we pick two adjacent elements from the rows respectively and then we add them and divide the result by two, and place their

LESSON#10 IMAGE PROCESSING & COMPUTER VISION

result in between those two elements. We first do this row wise and then we do this column wise.

For example

Lets take an image of the dimensions of 2 rows and 2 columns and zoom it twice using zero order hold.

| | |
|---|---|
| 1 | 2 |
| 3 | 4 |

First, we will zoom it row wise and then column wise.

Row wise zooming

| | | |
|---|---|---|
| 1 | 1 | 2 |
| 3 | 3 | 4 |

As we take the first two numbers: $(2 + 1) = 3$ and then we divide it by 2, we get 1.5 which is approximated to 1. The same method is applied in the row 2.

Column wise zooming

| | | |
|---|---|---|
| 1 | 1 | 2 |
| 2 | 2 | 3 |
| 3 | 3 | 4 |

We take two adjacent column pixel values which are 1 and 3. We add them and got 4. 4 is then divided by 2 and we get 2 which is placed in between them. The same method is applied in all the columns.

New image size

As you can see that the dimensions of the new image are 3×3 where the original image dimensions are 2×2 . So it means that the dimensions of the new image are based on the following formula

(2(number of rows) minus 1) X (2(number of columns) minus 1)

Advantages and disadvantage.

One of the advantage of this zooming technique , that it does not create as blurry picture as compare to the nearest neighbor interpolation method. But it also has a disadvantage that it can only run on the power of 2. It can be demonstrated here.

Reason behind twice zooming:

- Consider the above image of 2 rows and 2 columns. If we have to zoom it 6 times, using zero order hold method, we cannot do it. As the formula shows us this.
- It could only zoom in the power of 2 2,4,8,16,32 and so on.

LESSON#10 IMAGE PROCESSING & COMPUTER VISION

- Even if you try to zoom it, you cannot. Because at first when you will zoom it two times, and the result would be same as shown in the column wise zooming with dimensions equal to 3x3. Then you will zoom it again and you will get dimensions equal to 5 x 5. Now if you will do it again, you will get dimensions equal to 9 x 9.

Whereas according to the formula of yours the answer should be 11x11. As $(6(2) \text{ minus } 1) \times (6(2) \text{ minus } 1)$ gives 11 x 11.

B-Zero Interpolation

Example

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 8 | 6 | 7 |
| 0 | 1 | 2 | 3 |

The original image is 4x4 size.

After zero interpolation:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 2 | 0 | 3 | 0 | 4 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 6 | 0 | 7 | 0 | 8 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 0 | 8 | 0 | 6 | 0 | 7 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 2 | 0 | 3 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

We show the result is 8x8 size of image.

Procedure

- Zoom-in & Zoom-out without built-in MATLAB Function
 - save the image in the current folder.
 - Open new script and write the following:

```
clc
```

LESSON#10 IMAGE PROCESSING & COMPUTER VISION

```
clear all
close all

a= imread('cameraman.tif');
[m,n] = size(a);

%zoom in
for i=1:2*m
    for j=1:2*n
        p= i-floor(i/2);
        q= j-floor(j/2);
        b(i,j)=a(p,q);
    end
end
figure;
imshow(a)
title('original image');
figure; imshow(b)
title('zoom in image');

%%
%zoom out (shrinking)
for i= 1:m/2
    for j= 1:n/2
        p= i*2;
        q= j*2;
        c(i,j)=a(p,q);
    end
end
figure; imshow(c);
title ('zoomed out or shrinked image')
```

The Result

LESSON#10 IMAGE PROCESSING & COMPUTER VISION

original image



zoom in image



LESSON#10 IMAGE PROCESSING & COMPUTER VISION

zoomed out or shrinked image



- Zoom-in & Zoom-out with built-in MATLAB Function

Or we can use the function (**imresize**) to change the image size.

```
clc  
clear all  
close all  
  
a= imread('cameraman.tif');  
b= imresize(a,1/2);  
figure, imshow(b)
```

The Result



Report

1. write the MATLAB program to resize the image by 3 times greater than the original image and then shrink the original image to quarter.

Lesson#11 Image Processing & COMPUTER VISION

**Image Enhancement in the Spatial Model part1 – point
Processing.**

Eng. Elaf A.Saeed
Email: elafe1888@gmail.com

LAB Content

Negative Transformation

Log Transformation

Power Transformation

Table of Functions

| Function Name | Description |
|---|--|
| Imshowpair <code>obj = imshowpair(A,B)</code> <code>obj = imshowpair(A,RA,B,RB)</code> <code>obj = imshowpair(__,method)</code> | <ul style="list-style-type: none"> - Compare differences between images. -creates a composite RGB image showing A and B overlaid in different color bands. To choose another type of visualization of the two images, use the method argument. If A and B are different sizes, imshowpair pads the smaller dimensions with zeros on the bottom and right edges so that the two images are the same size. By default, imshowpair scales the intensity values of A and B independently from each other. imshowpair returns obj, an image object. - displays the differences between images A and B, using the spatial referencing information provided in RA and RB. RA and RB are spatial referencing objects. - uses the visualization method specified by method. |
| <code>obj = imshowpair(__,Name,Value)</code> | <ul style="list-style-type: none"> -specifies additional options with one or more Name,Value pair arguments, using any of the previous syntaxes. |
| im2double <code>I2 = im2double(I)</code> | <ul style="list-style-type: none"> - Convert image to double precision. - converts the image I to double precision. I can be a grayscale intensity image, a truecolor image, or a |

LESSON#11 IMAGE PROCESSING & COMPUTER VISION

I2 = im2double(I,'indexed')

binary image. im2double rescales the output from integer data types to the range [0, 1].
- converts the indexed image I to double precision.
im2double adds an offset of 1 to the output from integer data types.

Theory

Transformation

Transformation is a function. A function that maps one set to another set after performing some operations.

Digital Image Processing system

We have already seen in the introductory tutorials that in digital image processing, we will develop a system that whose input would be an image and output would be an image too. And the system would perform some processing on the input image and gives its output as a processed image. It is shown in figure 5.1 below.

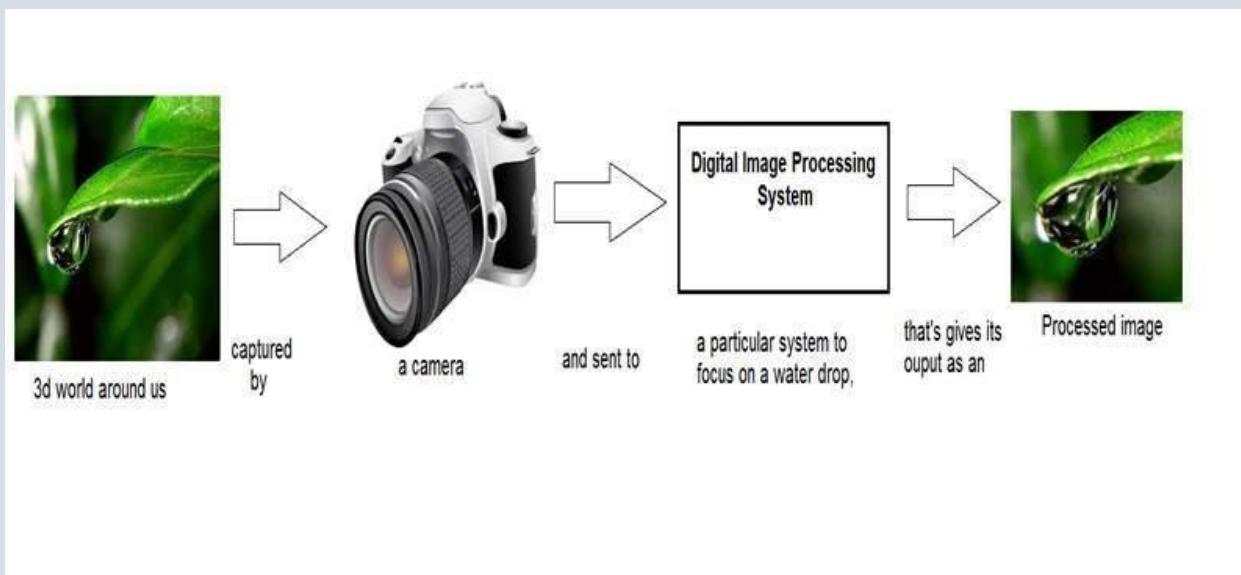


Figure 5.1: Image Transformation

Now function applied inside this digital system that process an image and convert it into output can be called as **transformation function**.

As it shows transformation or relation, that how an image1 is converted to image2.

Image transformation.

Consider this equation: $G(x,y) = T\{ f(x,y) \}$

In this equation,

F(x,y) = input image on which transformation function has to be applied.

LESSON#11 IMAGE PROCESSING & COMPUTER VISION

G(x,y) = the output image or processed image.

T is the transformation function.

This relation between input image and the processed output image can also be represented as.

$$s = T(r)$$

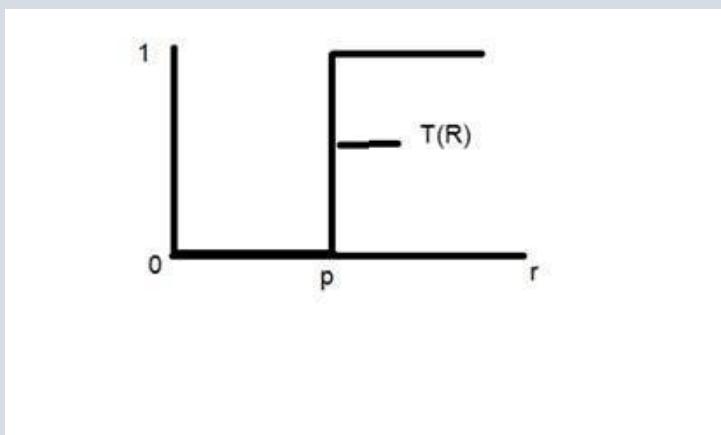
where r is actually the pixel value or gray level intensity of $f(x,y)$ at any point. And s is the pixel value or gray level intensity of $g(x,y)$ at any point.

The basic gray level transformation has been discussed in our tutorial of basic gray level transformations.

Now we are going to discuss some of the very basic transformation functions.

Examples

Consider this transformation function.



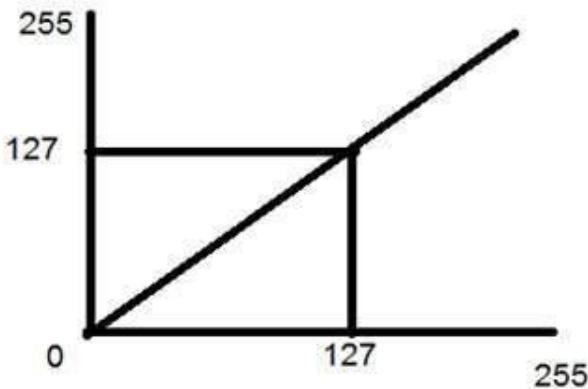
Let's take the point r to be 256, and the point p to be 127. Consider this image to be a one bpp image. That means we have only two levels of intensities that are 0 and 1. So in this case the transformation shown by the graph can be explained as. All the pixel intensity values that are below 127 (point p) are 0, means black. And all the pixel intensity values that are greater than 127, are 1, that means white. But at the exact point of 127, there is a sudden change in transmission, so we cannot tell that at that exact point, the value would be 0 or 1.

LESSON#11 IMAGE PROCESSING & COMPUTER VISION

Mathematically this transformation function can be denoted as:

$$g(x,y) = \begin{cases} 0 & f(x,y) < 127 \\ 1 & f(x,y) > 127 \end{cases}$$

Consider another transformation like this



Now if you will look at this particular graph, you will see a straight transition line between input image and output image.

It shows that for each pixel or intensity value of input image, there is a same intensity value of output image. That means the output image is exact replica of the input image.

It can be mathematically represented as:

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

the input and output image would be in this case are shown below.

LESSON#11 IMAGE PROCESSING & COMPUTER VISION



Image enhancement

Enhancing an image provides better contrast and a more detailed image as compare to non-enhanced image. Image enhancement has very applications. It is used to enhance medical images, images captured in remote sensing, images from satellite etc.

The transformation function has been given below

$$s = T(r)$$

where r is the pixels of the input image and s is the pixels of the output image. T is a transformation function that maps each value of r to each value of s. Image enhancement can be done through gray level transformations which are discussed below.

Gray level transformation

There are three basic gray level transformation.

- **Linear (Negative Transformation)**
- **Logarithmic**
- **Power – law**

The overall graph of these transitions has been shown in figure 5.2 below.

LESSON#11 IMAGE PROCESSING & COMPUTER VISION

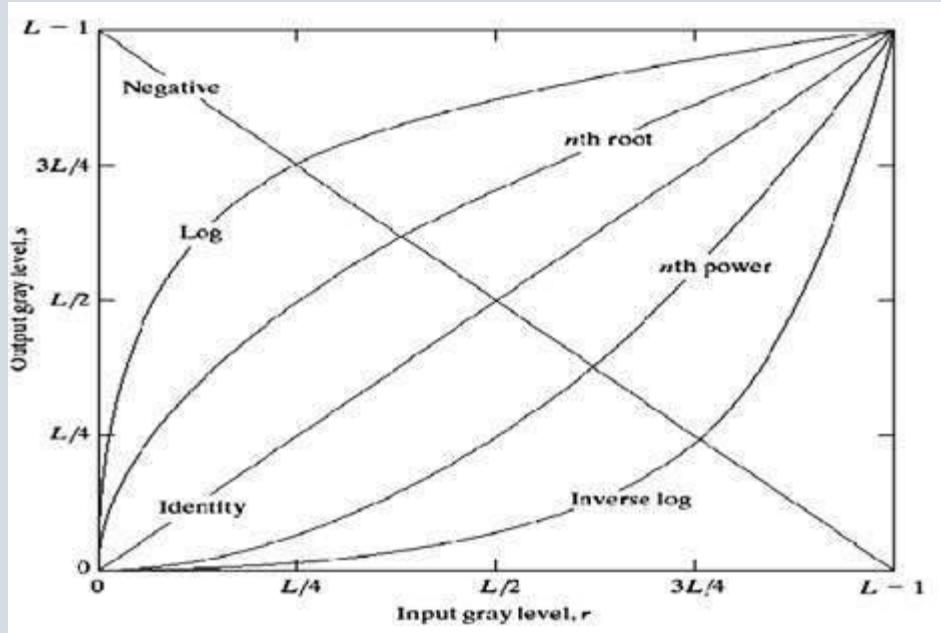


Figure 5.2: Gray Level Transformation

- **Linear transformation**

First we will look at the linear transformation. Linear transformation includes **simple identity** and **negative transformation**. Identity transformation has been discussed in our tutorial of image transformation, but a brief description of this transformation has been given here.

Identity transition is shown by a straight line. In this transition, each value of the input image is directly mapped to each other value of output image. That results in the same input image and output image. And hence is called identity transformation. It has been shown in figure 5.3 below:

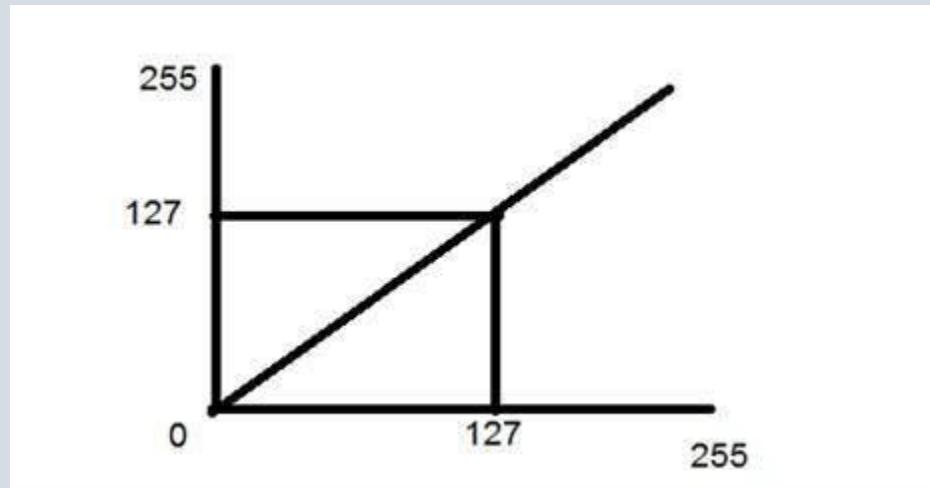


Figure 5.3: Linear Transformation

Negative transformation

The second linear transformation is negative transformation, which is invert of identity transformation. In negative transformation, each value of the input image is subtracted from the L-1 and mapped onto the output image.

The result is somewhat like this shown in **figure 5.4**.



Figure 5.4: Negative Transformation

In this case the following transition has been done.

$$s = (L - 1) - r$$

since the input image of Einstein is an 8 bpp image, so the number of levels in this image are 256. Putting 256 in the equation, we get this

$$s = 255 - r$$

LESSON#11 IMAGE PROCESSING & COMPUTER VISION

So each value is subtracted by 255 and the result image has been shown above. So what happens is that, the lighter pixels become dark and the darker picture becomes light. And it results in image negative.

It has been shown in the graph in **figure 5.5** below.

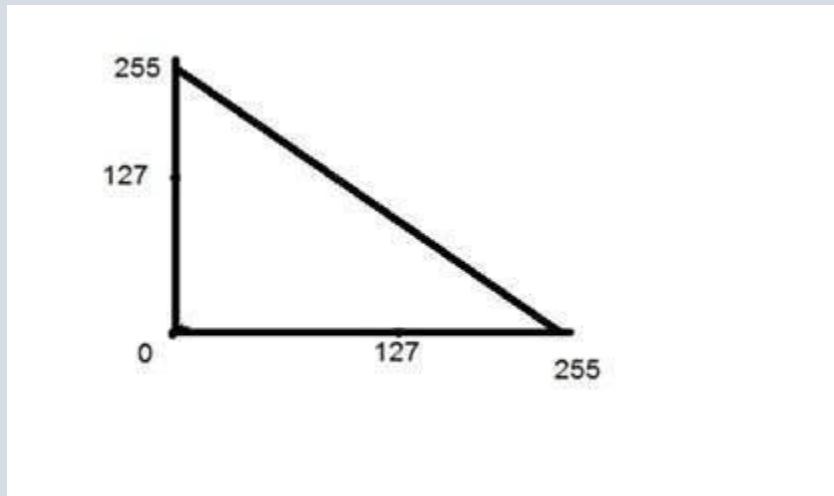


Figure 5.5: Negative Transformation

- **Logarithmic transformations**

The log transformations can be defined by this formula

$$s = c \log(r + 1).$$

Where s and r are the pixel values of the output and the input image and c is a constant. 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. So 1 is added, to make the minimum value at least 1.

During log transformation, the dark pixels in an image are expanded as compare to the higher pixel values. The higher pixel values are kind of compressed in log transformation. This result in following image enhancement.

The value of c in the log transform adjust the kind of enhancement you are looking for, that is shows in figure 5.5.

Input Image

output Image

LESSON#11 IMAGE PROCESSING & COMPUTER VISION

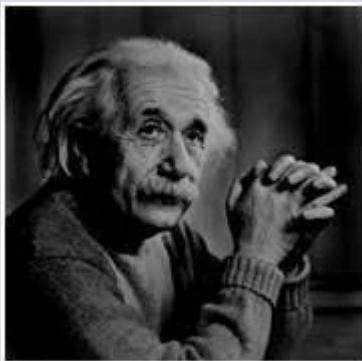


Figure 5.5: Log Transformation

The inverse log transform is opposite to log transform.

- **Power – Law transformations**

There are further two transformations in power law transformations, that include **nth** power and **nth** root transformation. These transformations can be given by the expression: $s=cr^\gamma$

This symbol γ is called **gamma**, due to which this transformation is also known as **gamma transformation**.

Variation in the value of γ varies the enhancement of the images. Different display devices / monitors have their **own gamma** correction, that's why they display their image at different intensity.

This type of transformation is used for enhancing images for different type of display devices. The gamma of different display devices is different. For example, Gamma of CRT lies in between of 1.8 to 2.5, that means the image displayed on CRT is dark.

Correcting gamma

$$s=cr^\gamma$$

$$s=cr^{(1/2.5)}$$

The same image but with different gamma values has been shown in figure 5.6.

For example,

LESSON#11 IMAGE PROCESSING & COMPUTER VISION

Gamma = 10



Gamma = 8



Gamma = 6



Figure 5.6: Power Transformation

The goal of this lab

It is the knowledge how to gray level transformation in MATLAB.

The function that is used in this lab:

- **Imshowpair** → Compare differences between images

obj = imshowpair(A,B) creates a composite RGB image showing A and B overlaid in different color bands. To choose another type of visualization of the two images, use the method argument. If A and B are different sizes, imshowpair pads the smaller dimensions with zeros on the bottom and right edges so that the two images are the same size. By default, imshowpair scales the intensity values of A and B independently from each other. imshowpair returns obj, an image object.

obj = imshowpair(A,RA,B,RB) displays the differences between images A and B, using the spatial referencing information provided in RA and RB. RA and RB are spatial referencing objects.

obj = imshowpair(___,method) uses the visualization method specified by method.

obj = imshowpair(___,Name,Value) specifies additional options with one or more Name,Value pair arguments, using any of the previous syntaxes.

LESSON#11 IMAGE PROCESSING & COMPUTER VISION

- **im2double** → Convert image to double precision.

I2 = im2double(I) converts the image I to double precision. I can be a grayscale intensity image, a **truecolor** image, or a binary image. **im2double** rescales the output from integer data types to the range [0, 1].

I2 = im2double(I,'indexed') converts the indexed image I to double precision. **im2double** adds an offset of 1 to the output from integer data types.

Procedure

- Negative Transformation

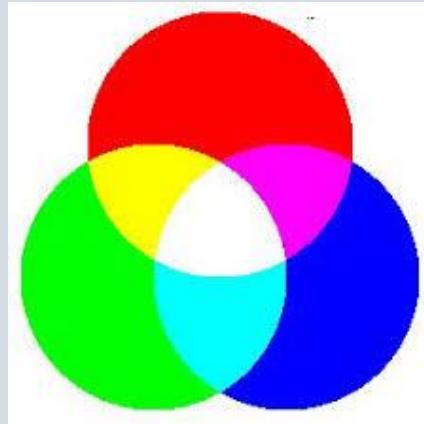
- 1- save the image in the current folder.
- 2- Open new script file and write the following:

```
clc
clear all
close all
m= imgetfile; %to select image from current folder
x= imread(m); %to read image
y= rgb2gray(x); %convert RGB to gray image
%take the maximum pixel in the image
v= max (y(:));
neg_im =v-y;
% show the pair of image in the same figure
imshowpair(y, neg_im, 'montage');
```

The Result

In figure 5.6 that shows the negative image.

LESSON#11 IMAGE PROCESSING & COMPUTER VISION



Original Image

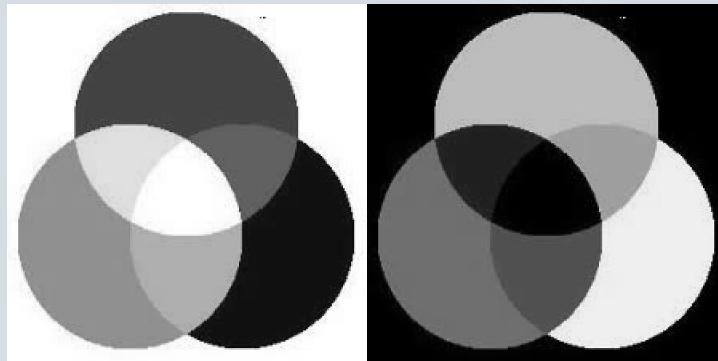


Figure 5.6: Negative Transformation Images

• Log Transformation

1. save the image in the current folder.
2. Open new script file and write the following:

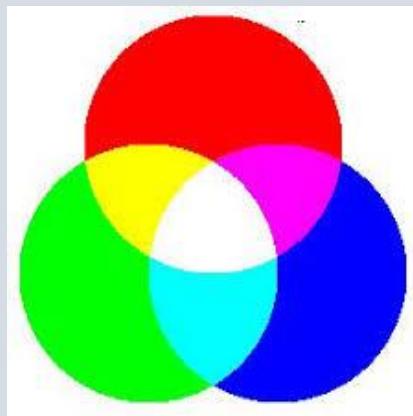
```
% Log Transformation  
clc  
clear all
```

LESSON#11 IMAGE PROCESSING & COMPUTER VISION

```
close all  
m= imgetfile; %to select image from current folder  
x= imread(m); %to read image  
y= rgb2gray(x); %convert RGB to gray image  
figure  
subplot(1,2,1),imshow (y),title('original Image');  
  
t= double(y)  
s= 0.1 * log(t+1);  
subplot(1,2,2),imshow(s), title('Log Image');
```

The Result

In figure 5.7 that shows the log transformation image.



Original Image

LESSON#11 IMAGE PROCESSING & COMPUTER VISION

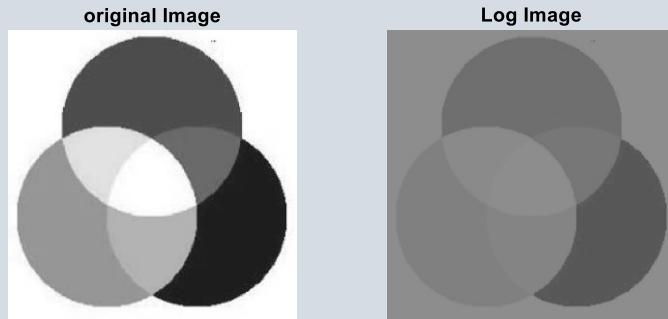


Figure 5.7: Log Transformation Images

• Power Transformation

3. save the image in the current folder.
4. Open new script file and write the following:

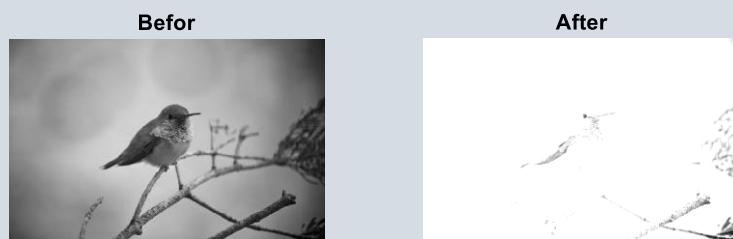
```
% Power Transformation  
clc  
clear all  
close all  
im= imgetfile; %to select image from current folder  
a= imread(im); %to read image  
ad= im2double(a); %Convert image to double precision  
x=ad; y=ad;  
[r,c]=size(ad)  
factor=6;  
for i=1:r
```

LESSON#11 IMAGE PROCESSING & COMPUTER VISION

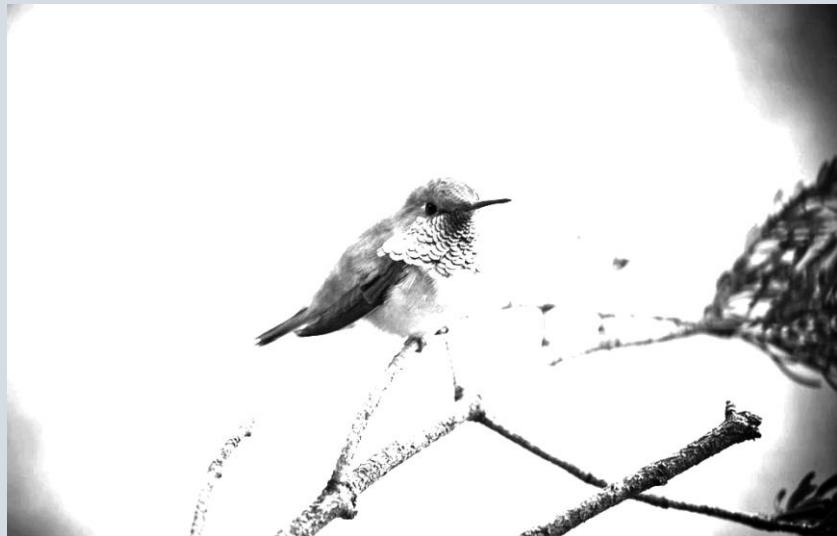
```
for j=1:c  
    x(i,j)=factor *log(1+ad(i,j)); %Log trans.  
    y(i,j)=factor *ad(i,j)^2; %power trans.  
end  
end  
%Log trans.  
subplot(1,2,1),imshow (ad),title('Befor');  
subplot(1,2,2),imshow (x),title('After');  
%power trans.  
figure,imshow(y);
```

\ The Result

In figure 5.8 that shows the log and power transformation image.



Log Transformation



Power Transformation

Figure 5.7: Log & power Transformation Images

Report

- Make the negative transformation by using the for loop.
- Explain What is the different between log transformation and power transformation.

Lesson#12 Image Processing & COMPUTER VISION

**Image Enhancement in the Spatial Model part2 –
Histogram**

Eng. Elaf A.Saeed
Email: elafe1888@gmail.com

LAB Content

Grayscale Image Histogram

Histogram Equalization

Histogram Stretch

Histogram Sliding

Table of Functions

| Function Name | Description |
|---|--|
| imhist <code>[counts,binLocations] = imhist(I)</code> | <p>Histogram of image data.</p> <p><code>[counts,binLocations] = imhist(I)</code> calculates the histogram for the grayscale image I. The imhist function returns the histogram counts in counts and the bin locations in binLocations. The number of bins in the histogram is determined by the image type.</p> |
| histeq <code>J = histeq(I)</code> | <p>Enhance contrast using histogram equalization.</p> <p><code>J = histeq(I)</code> transforms the grayscale image I so that the histogram of the output grayscale image J has 64 bins and is approximately flat.</p> |
| imadjust <code>J = imadjust(I)</code> | <p>Adjust image intensity values or color map.</p> <p><code>J = imadjust(I)</code> maps the intensity values in grayscale image I to new values in J. By default, imadjust saturates the bottom 1% and the top 1% of all pixel values. This operation increases the contrast of the output image J.</p> |

Theory

Image Statistic

Calculating Image Statistics

The statistical properties of an image provide useful information, such as the total, mean, standard deviation, and variance of the pixel values. The statistical is Knowing some statistical matters for pictures.

Image Statistic Types

In Figure 8.1 that shown the Image Statistic Types.

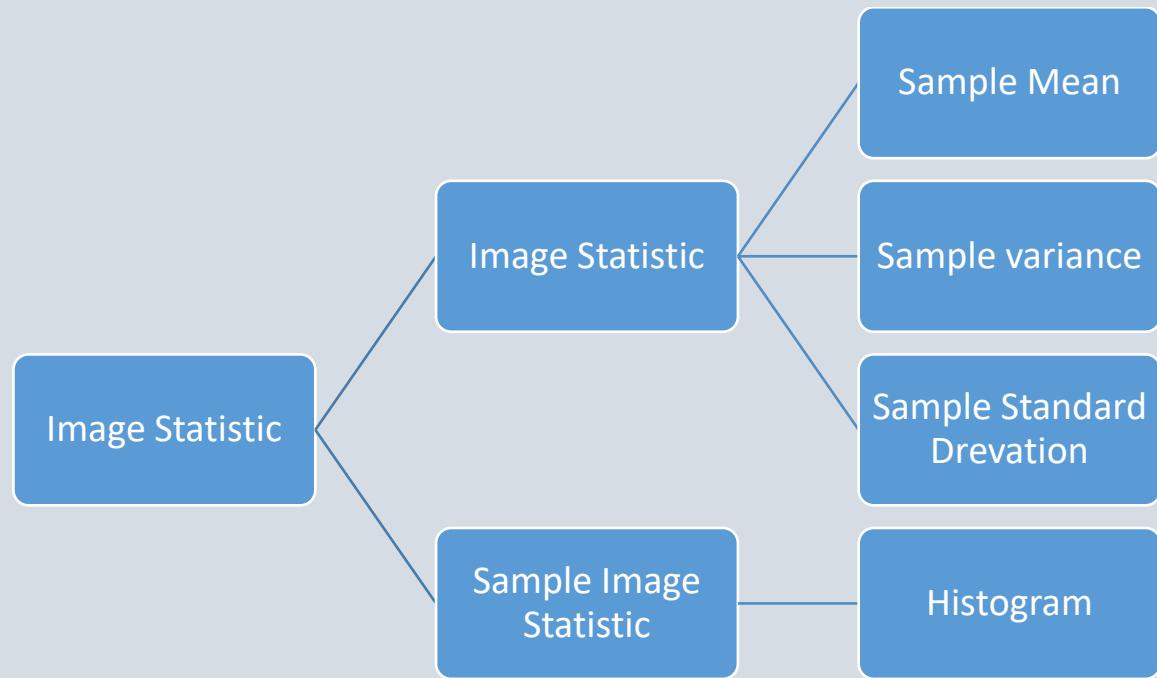


Figure 8.1: Image Statistic Types

Sample Image Statistic

Histograms

A histogram is a graph. A graph that shows frequency of anything. Usually histogram have bars that represent frequency of occurring of data in the whole data set. A Histogram has two axes the x axis and the y axis. The x axis contains event

LESSON#12 IMAGE PROCESSING & COMPUTER VISION

whose frequency you have to count. The y axis contains frequency. The different heights of bar show different frequency of occurrence of data. Usually a histogram looks like this shows in figure 8.2.

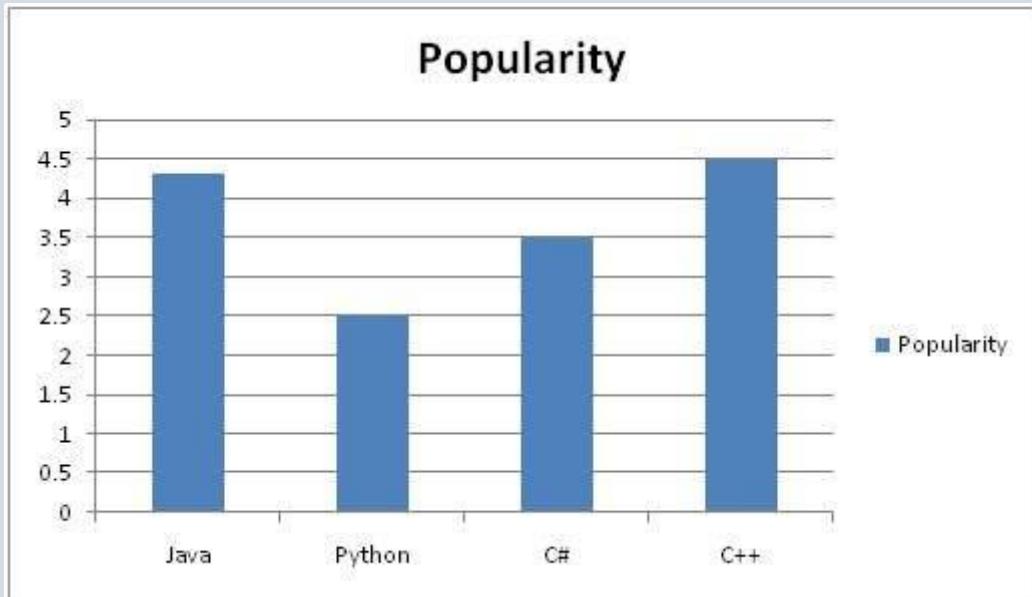


Figure 8.2: Histogram

Now we will see an example of this histogram is build

Example

Consider a class of programming students and you are teaching python to them. At the end of the semester, you got this result that is shown in table. But it is very messy and does not show your overall result of class. So you have to make a histogram of your result, showing the overall frequency of occurrence of grades in your class. Here how you are going to do it.

Result sheet

LESSON#12 IMAGE PROCESSING & COMPUTER VISION

| Name | Grade |
|--------|-------|
| John | A |
| Jack | D |
| Carter | B |
| Tommy | A |
| Lisa | C+ |
| Derek | A- |
| Tom | B+ |

Histogram of result sheet

Now what you are going to do is, that you have to find what comes on the x and the y axis. There is one thing to be sure, that y axis contains the frequency, so what comes on the x axis. X axis contains the event whose frequency has to be calculated. In this case x axis contains grades, as shown in figure 8.3.

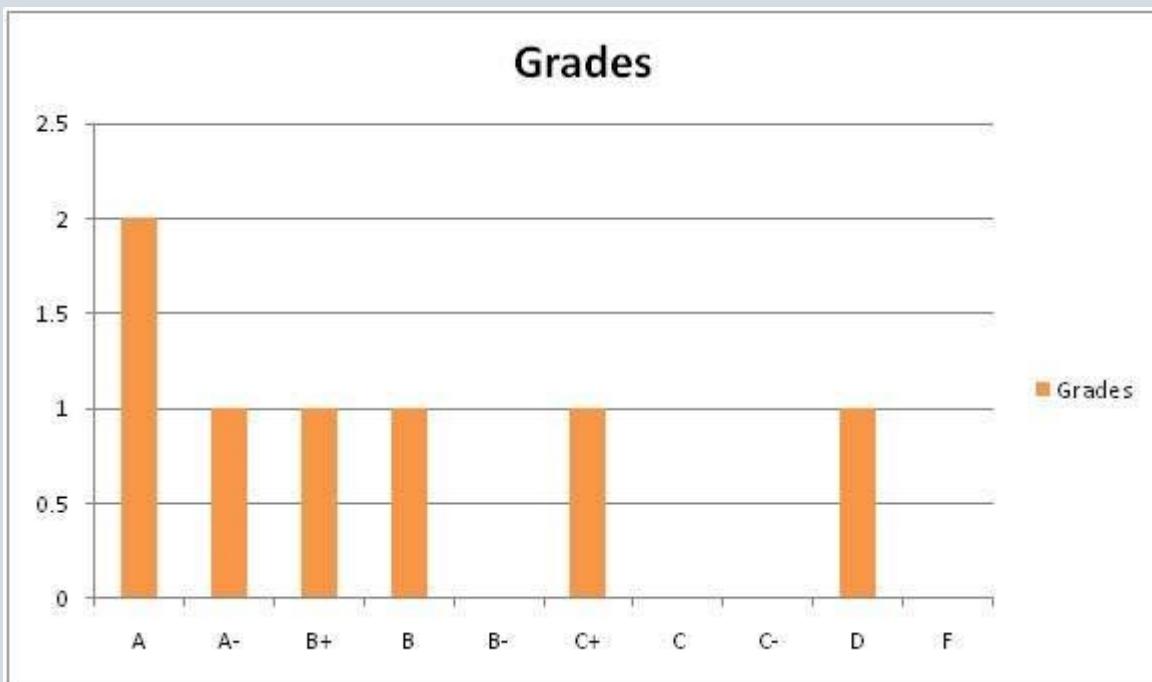


Figure 8.3: Histogram Grades

Now we will how do we use a histogram in an image.

Histogram of an image

LESSON#12 IMAGE PROCESSING & COMPUTER VISION

Histogram of an image, like other histograms also shows frequency. But an image histogram, shows frequency of pixels' intensity values. In an image histogram, the x axis shows the gray level intensities and the y axis shows the frequency of these intensities.

For example

In figure 8.4 that shows the grayscale image.

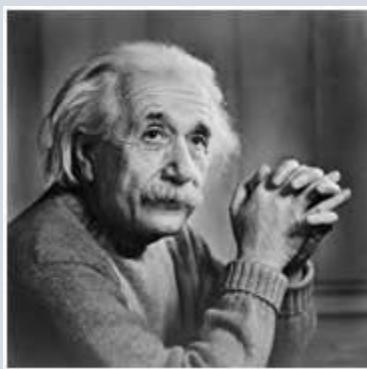


Figure 8.4: The grayscale Image

The histogram of the above picture of the Einstein would be something like this shows in figure 8.5.

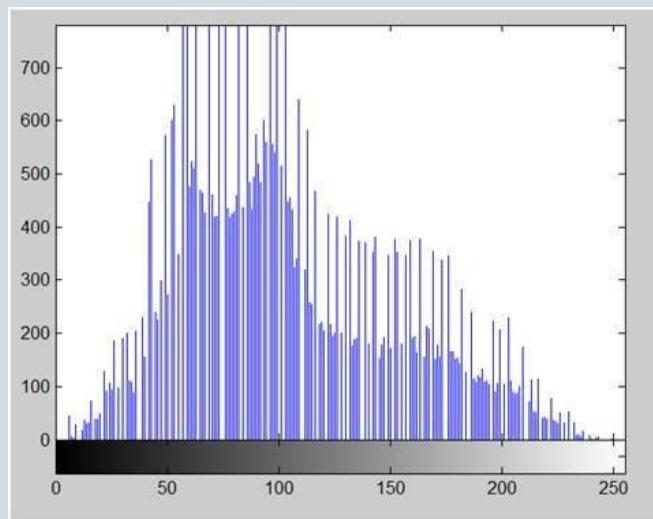


Figure 8.5: The Histogram of Image

The x axis of the histogram shows the range of pixel values. Since it's an 8 bpp image, that means it has 256 levels of gray or shades of gray in it. That's why the

LESSON#12 IMAGE PROCESSING & COMPUTER VISION

range of x axis starts from 0 and end at 255 with a gap of 50. Whereas on the y axis, is the count of these intensities.

As you can see from the graph, that most of the bars that have high frequency lies in the first half portion which is the darker portion. That means that the image we have got is darker. And this can be proved from the image too.

Applications of Histograms

Histograms has many uses in image processing. The first use as it has also been discussed above is the analysis of the image. We can predict about an image by just looking at its histogram. It's like looking an x ray of a bone of a body.

The second use of histogram is for brightness purposes. The histograms have wide application in image brightness. Not only in brightness, but histograms are also used in adjusting contrast of an image. Another important use of histogram is to equalize an image. And last but not the least, histogram has wide use in thresholding. This is mostly used in computer vision.

Histogram sliding

In histogram sliding, we just simply shift a complete histogram rightwards or leftwards. Due to shifting or sliding of histogram towards right or left, a clear change can be seen in the image. In this lab we are going to use histogram sliding for manipulating brightness.

Brightness

Brightness is a relative term. Brightness can be defined as intensity of light emit by a particular light source.

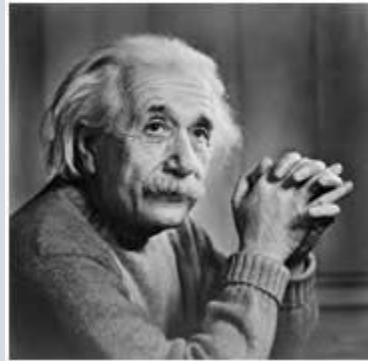
Contrast

Contrast can be defined as the difference between maximum and minimum pixel intensity in an image.

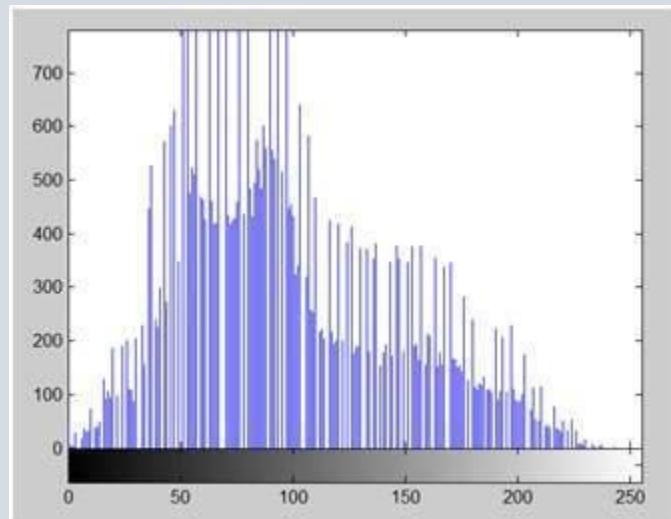
LESSON#12 IMAGE PROCESSING & COMPUTER VISION

A. Sliding Histograms

Increasing brightness using histogram sliding.

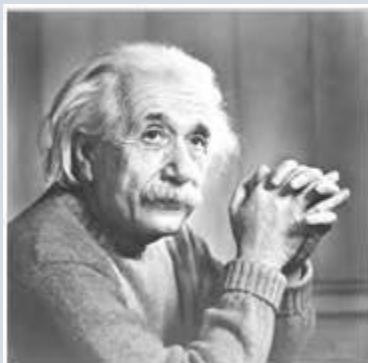


Histogram of this image has been shown below.



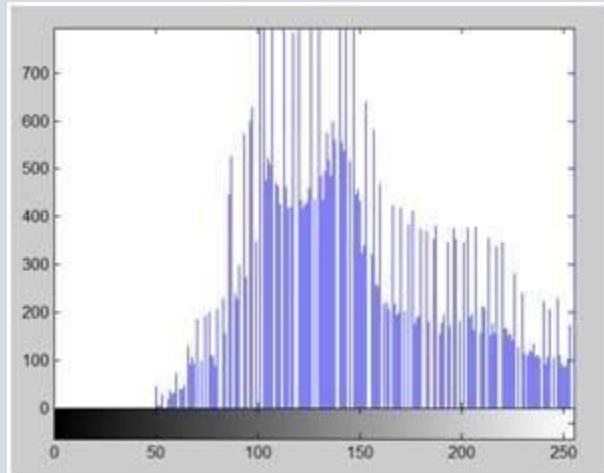
Here what we got after adding 50 to each pixel intensity.

The image has been shown below.

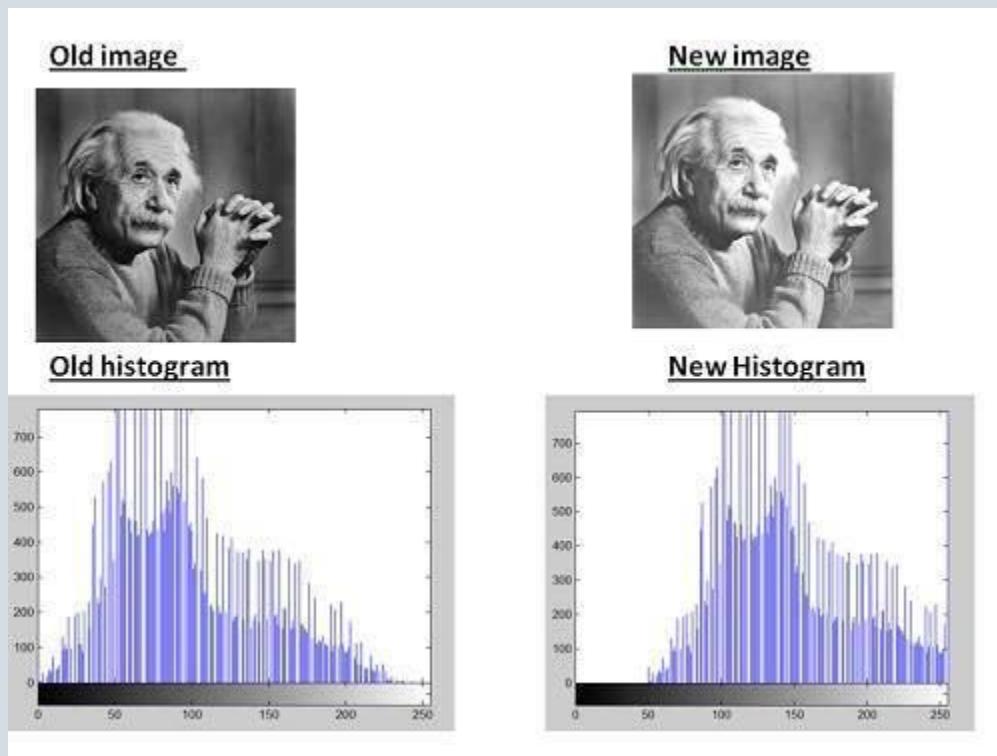


And its histogram has been shown below.

LESSON#12 IMAGE PROCESSING & COMPUTER VISION



Lets compare these two images and their histograms to see that what change have to got.



Conclusion

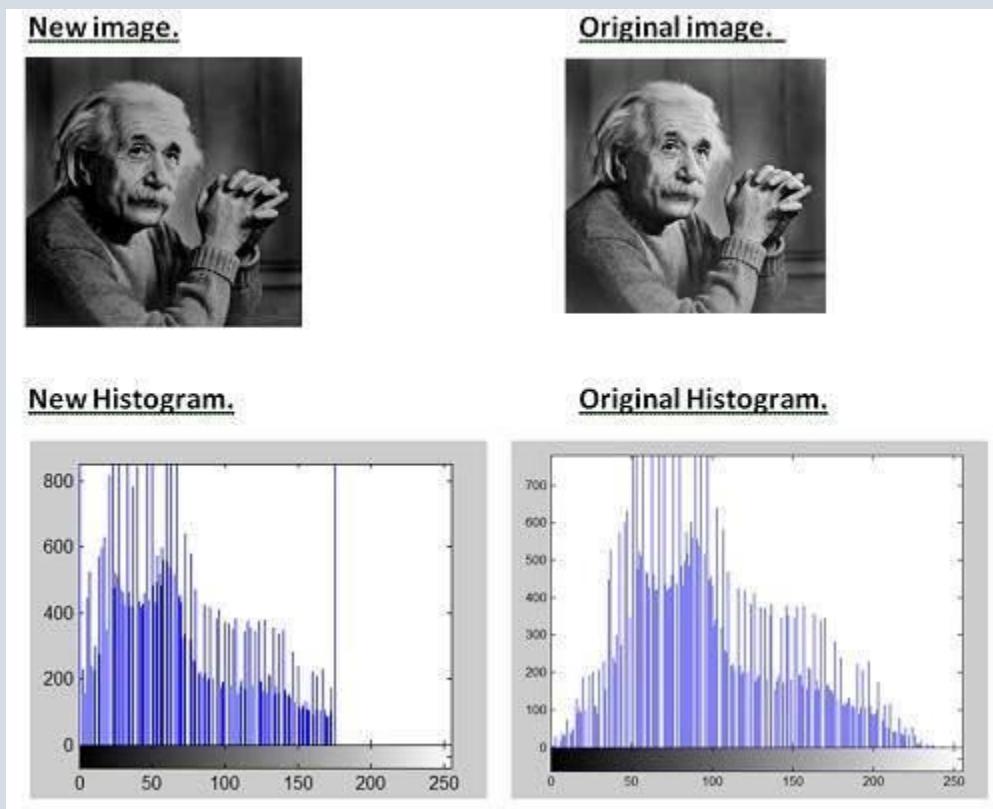
As we can clearly see from the new histogram that all the pixels values has been shifted towards right and its effect can be seen in the new image.

Decreasing brightness using histogram sliding

LESSON#12 IMAGE PROCESSING & COMPUTER VISION

Now if we were to decrease brightness of this new image to such an extent that the old image look brighter, we got to subtract some value from all the matrix of the new image. The value which we are going to subtract is 80. Because we already add 50 to the original image and we got a new brighter image, now if we want to make it darker, we have to subtract at least more than 50 from it.

And this what we got after subtracting 80 from the new image.



Conclusion

It is clear from the histogram of the new image, that all the pixel values have been shifted towards right and thus, it can be validated from the image that new image is darker and now the original image look brighter as compare to this new image.

B. Histogram stretching

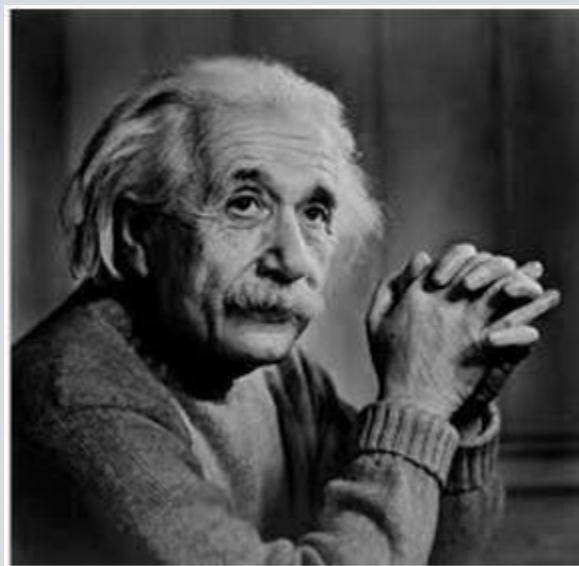
There are two methods of enhancing contrast. The first one is called Histogram stretching that increase contrast. The second one is called Histogram equalization that enhance contrast.

Contrast

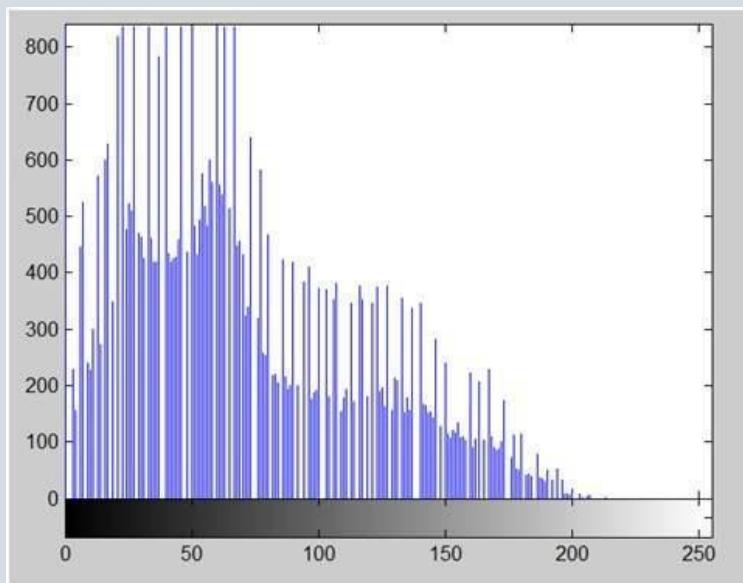
LESSON#12 IMAGE PROCESSING & COMPUTER VISION

Contrast is the difference between maximum and minimum pixel intensity.

Consider this image.



The histogram of this image is shown below.



Now we calculate contrast from this image.

Contrast = 225.

Now we will increase the contrast of the image.

Increasing the contrast of the image

LESSON#12 IMAGE PROCESSING & COMPUTER VISION

The formula for stretching the histogram of the image to increase the contrast is

$$g(x,y) = \frac{f(x,y) - f_{\min}}{f_{\max} - f_{\min}} * 2^{bpp}$$

The formula requires finding the minimum and maximum pixel intensity multiply by levels of gray. In our case the image is 8bpp, so levels of gray are 256.

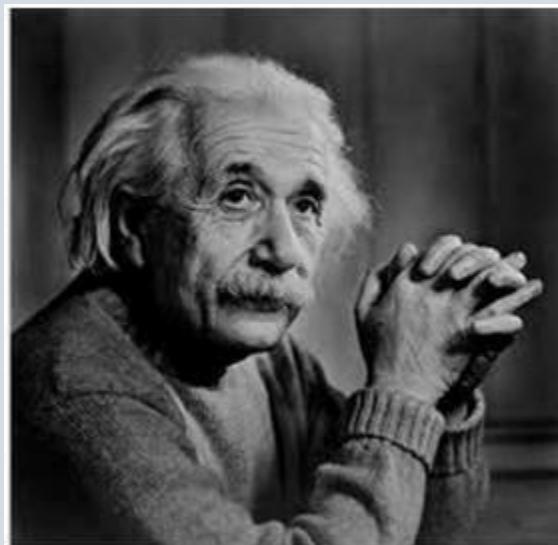
The minimum value is 0 and the maximum value is 225. So the formula in our case is

$$g(x,y) = \frac{f(x,y) - 0}{225 - 0} * 255$$

where **f (x, y)** denotes the value of each pixel intensity. For each $f(x,y)$ in an image , we will calculate this formula.

After doing this, we will be able to enhance our contrast.

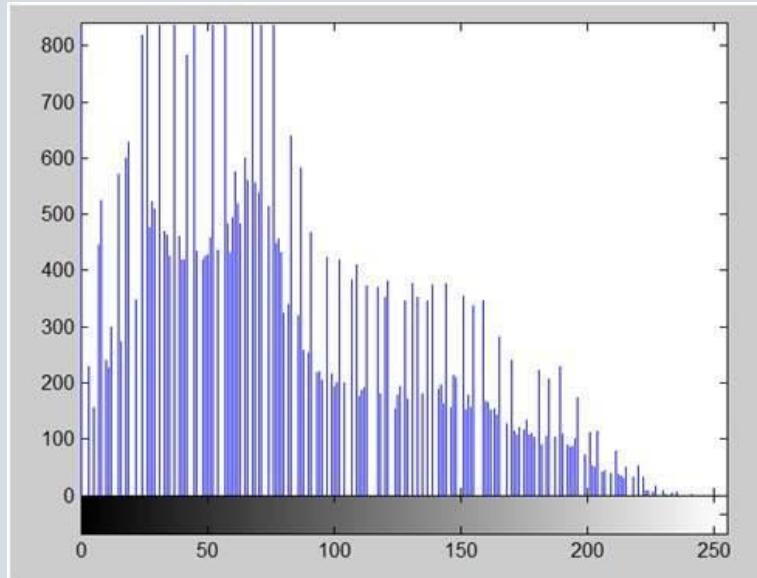
The following image appear after applying histogram stretching.



The stretched histogram of this image has been shown below.

Note the shape and symmetry of histogram. The histogram is now stretched or in other means expand. Have a look at it.

LESSON#12 IMAGE PROCESSING & COMPUTER VISION



In this case the contrast of the image can be calculated as

Contrast = 240

Hence, we can say that the contrast of the image is increased.

Note: this method of increasing contrast does not work always, but it fails on some cases.

Failing of histogram stretching

As we have discussed, that the algorithm fails on some cases. Those cases include images with when there is pixel intensity 0 and 255 are present in the image

Because when pixel intensities 0 and 255 are present in an image, then in that case they become the minimum and maximum pixel intensity which ruins the formula like this.

Original Formula

$$g(x,y) = \frac{f(x,y) - f_{min}}{f_{max} - f_{min}} * 2^{bpp}$$

Putting fail case values in the formula:

LESSON#12 IMAGE PROCESSING & COMPUTER VISION

$$g(x,y) = \frac{f(x,y) - 0}{255 - 0} * 255$$

Simplify that expression gives

$$g(x,y) = \frac{f(x,y)}{255} * 255$$
$$g(x,y) = f(x,y)$$

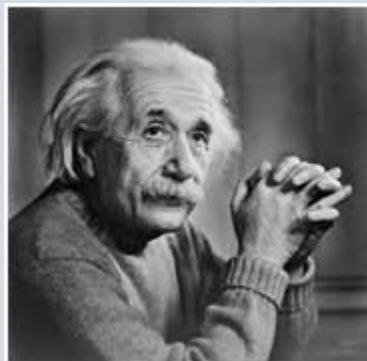
That means the output image is equal to the processed image. That means there is no effect of histogram stretching has been done at this image.

C. Histogram Equalization

Histogram equalization is used to enhance contrast. It is not necessary that contrast will always be increase in this. There may be some cases were histogram equalization can be worse. In that cases the contrast is decreased.

Let's start histogram equalization by taking this image below as a simple image.

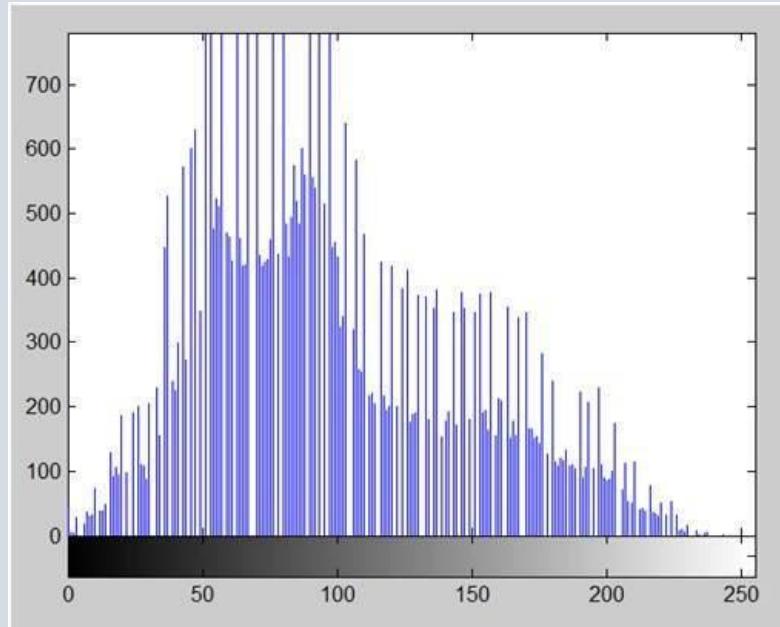
Image



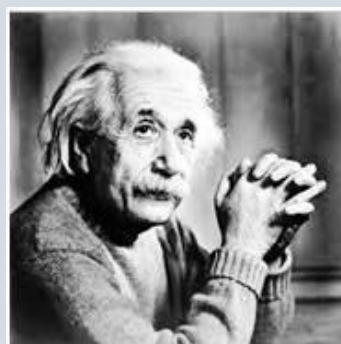
Histogram of this image

The histogram of this image has been shown below.

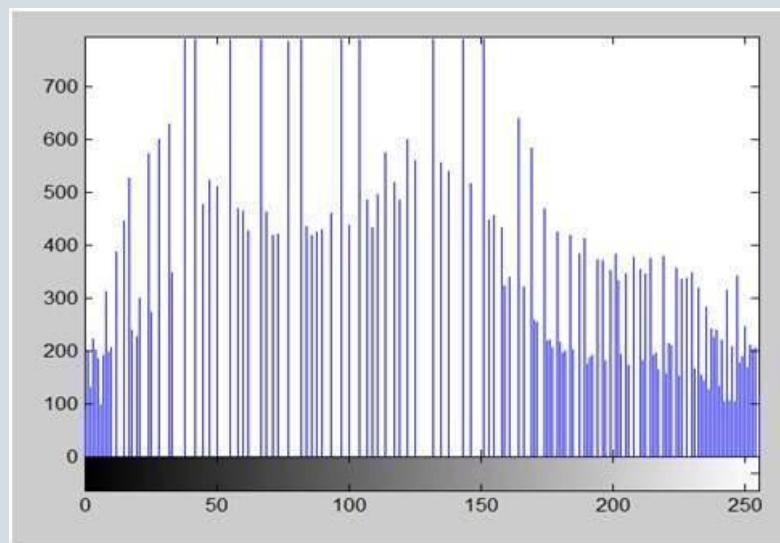
LESSON#12 IMAGE PROCESSING & COMPUTER VISION



Histogram Equalization Image

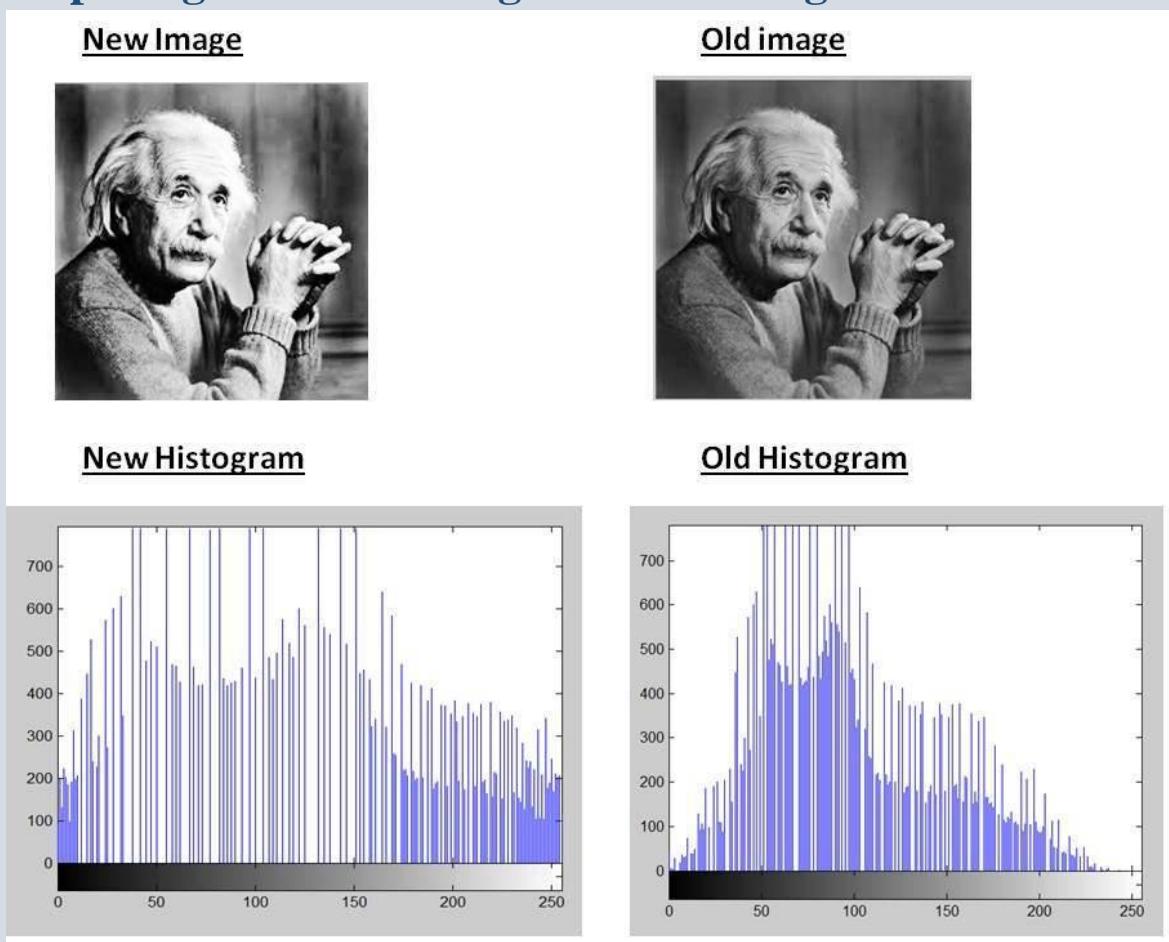


Histogram Equalization histogram



LESSON#12 IMAGE PROCESSING & COMPUTER VISION

Comparing both the histograms and images



Conclusion

As you can clearly see from the images that the new image contrast has been enhanced and its histogram has also been equalized. There is also one important thing to be note here that during **histogram equalization** the overall shape of the **histogram changes**, whereas in **histogram stretching** the overall shape of histogram remains same.

The goal of this lab

It is the knowledge how to use histogram with images in MATLAB.

Procedure

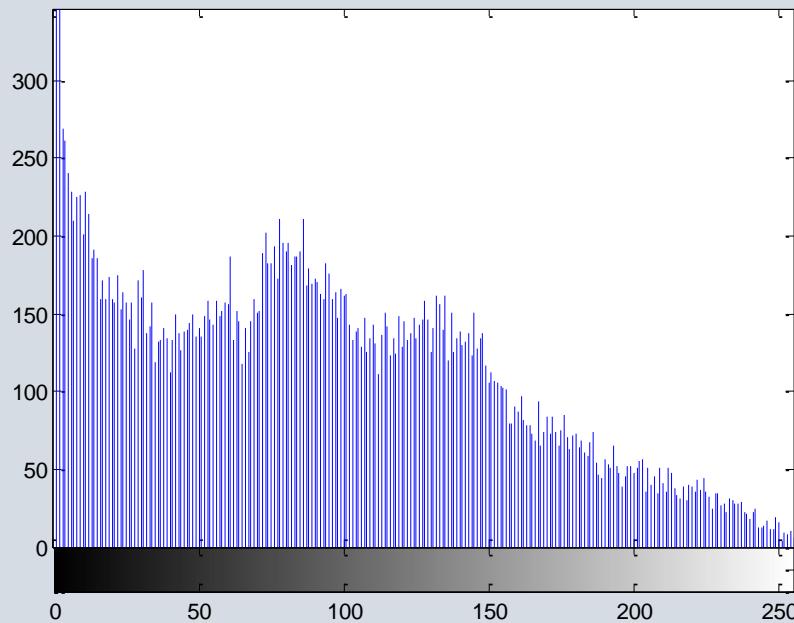
- Grayscale Image Histogram

LESSON#12 IMAGE PROCESSING & COMPUTER VISION

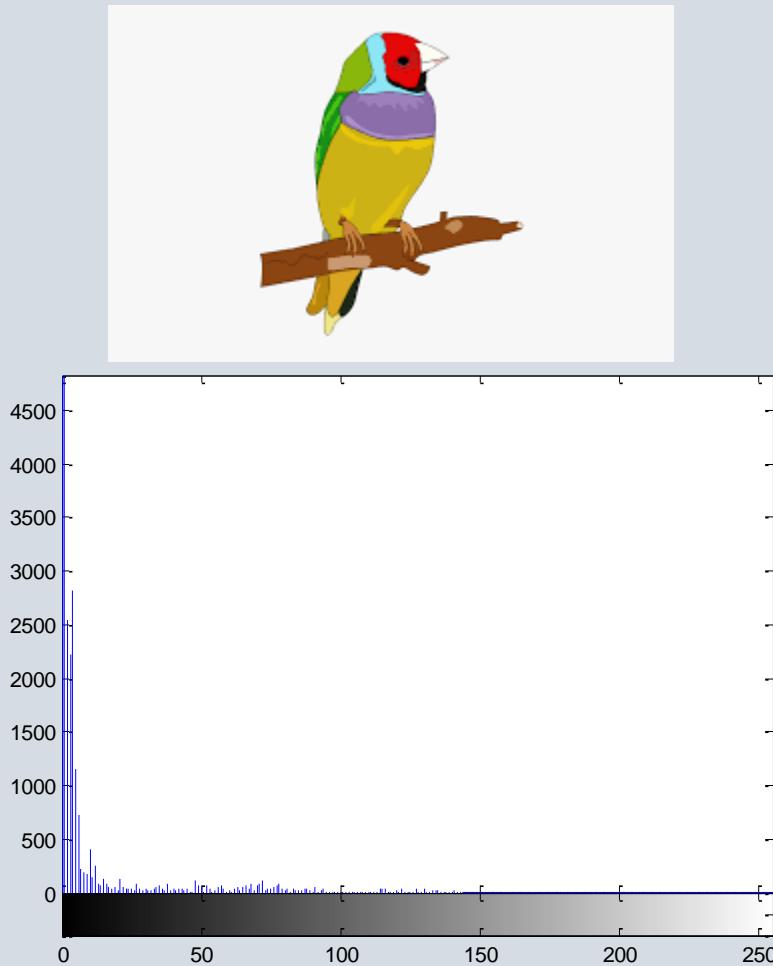
- 1- save the image in the current folder.
- 2- Open new script file and write the following:

```
%Image Histogram
clc
clear
close all
% GrayScale Histogram
A = imread('bird1.png');
figure
imshow(A)
figure
imhist(A);%Histogram of image data
B = imread('bird2.png');
figure
imshow(B)
figure
imhist(B)%Histogram of image data
```

The Result



LESSON#12 IMAGE PROCESSING & COMPUTER VISION



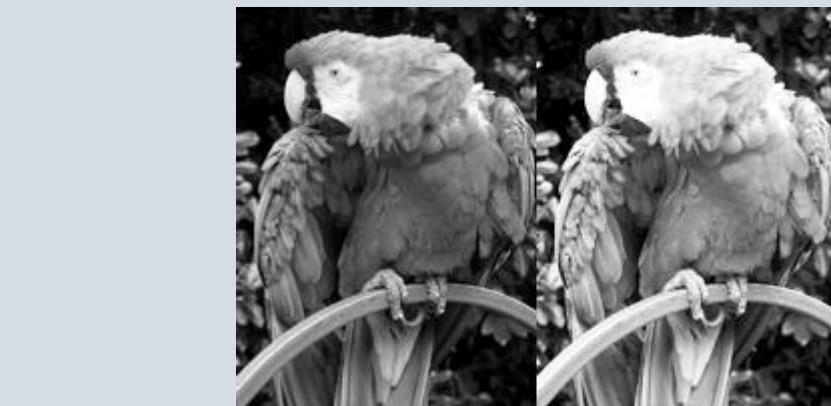
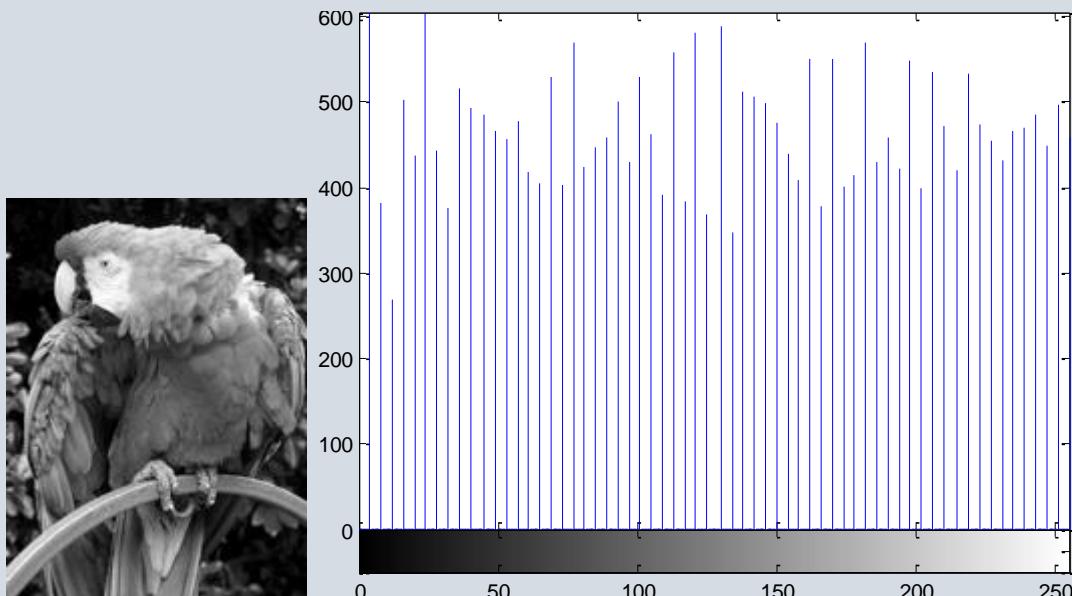
• Histogram Equalization

1. save the image in the current folder.
2. Open new script file and write the following:

```
% Histogram Equalization
A = imread('bird1.png');
figure
imshow(A)
hisequa= histeq(A); %Enhance contrast
%using histogram equalization
figure
imhist(hisequa)
figure
imshowpair(A,hisequa, 'montage')
```

LESSON#12 IMAGE PROCESSING & COMPUTER VISION

The Result

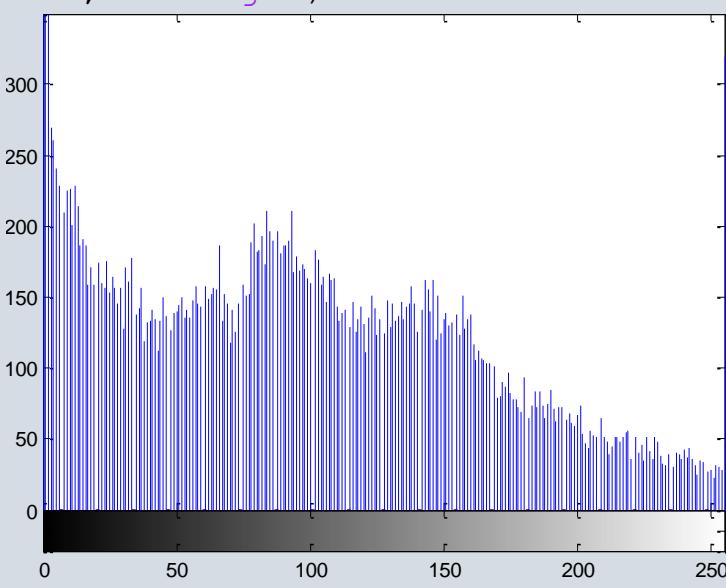


- Histogram Stretch

```
% Histogram Stretch
A = imread('bird1.png');
figure
imshow(A)
figure
his_strech=imadjust(A);%Adjust image
%intensity values or colormap
imhist(his_strech)
figure
```

LESSON#12 IMAGE PROCESSING & COMPUTER VISION

```
imshowpair(A,his_strech, 'montage')
```



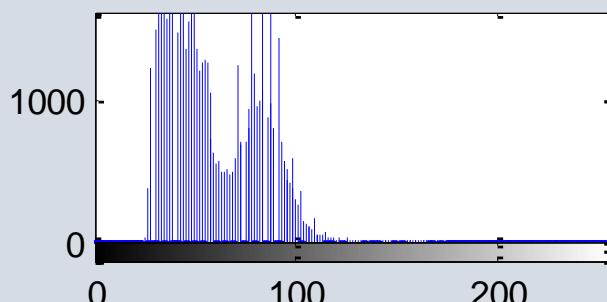
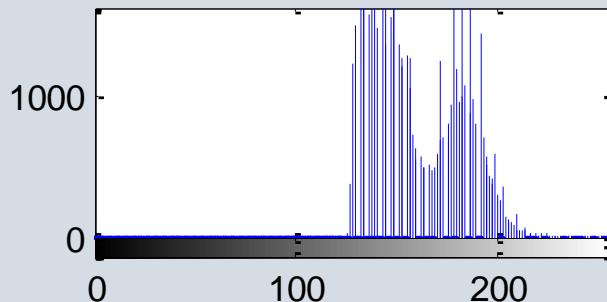
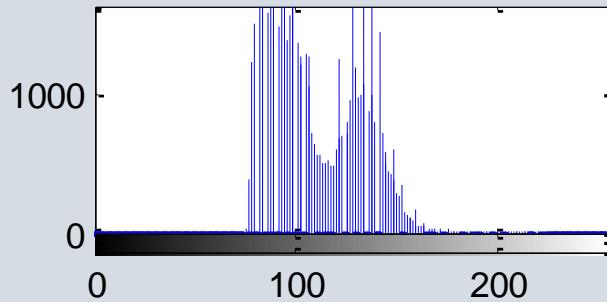
- Histogram Sliding

```
clc
clear all
close all
warning off
I = imread('pout.tif');
x=I;
subplot(3,2,1);
imshow(I);
subplot(3,2,2);
imhist(I);
%shift to right
```

LESSON#12 IMAGE PROCESSING & COMPUTER VISION

```
subplot(3,2,3);
I=I+50;
imshow(I);
subplot(3,2,4);
imhist(I);
% shift to left
I=x;
I=I-50;
subplot(3,2,5);
imshow(I);
subplot(3,2,6);
imhist(I);
```

The Result



Report

- 1- Open the new script in MATLAB program to make histogram for the color image for each band (color) separately.

Lesson#13 Image Processing & COMPUTER VISION

**Special Domain – Convolution and Filters – Low pass
Filter (LPF)**

Eng. Elaf A.Saeed
Email: elafe1888@gmail.com

LAB Content

Mean (average) Filter without Function.

Mean (average) Filter with Function.

Table of Functions

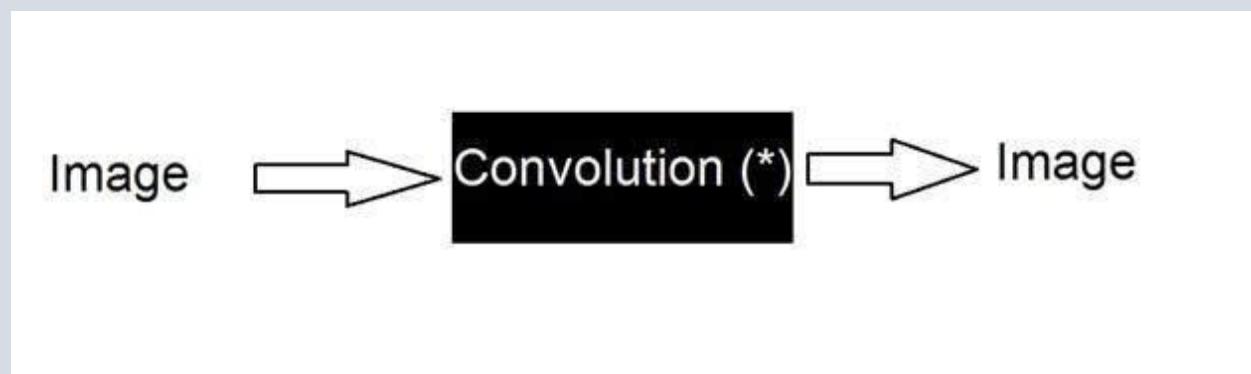
| Function Name | Description |
|-----------------|---|
| Imfilter | N-D filtering of multidimensional images. B = imfilter(A,h) filters the multidimensional array A with the multidimensional filter h and returns the result in B |

Theory

Image Convolution

Here we are going to discuss another method of dealing with images. This other method is known as convolution. Usually the black box(system) used for image processing is an LTI system or linear time invariant system. By linear we mean that such a system where output is always linear, neither log nor exponent or any other. And by time invariant we mean that a system which remains same during time.

So now we are going to use this third method. It can be represented as.



It can be mathematically represented as two ways

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

It can be explained as the “mask convolved with an image”.

Or

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

It can be explained as “image convolved with mask”.

There are two ways to represent this because the convolution operator(*) is **commutative**. The $h(x,y)$ is the mask or filter.

LESSON#13 IMAGE PROCESSING & COMPUTER VISION

What is mask?

Mask is also a signal. It can be represented by a two-dimensional matrix. The mask is usually of the order of 1x1, 3x3, 5x5, 7x7. A mask should always be in odd number, because otherwise you cannot find the mid of the mask. Why do we need to find the mid of the mask? The answer lies below, in topic of, how to perform convolution?

How to perform convolution?

In order to perform convolution on an image, following steps should be taken.

- Flip the mask (horizontally and vertically) only once
- Slide the mask onto the image.
- Multiply the corresponding elements and then add them
- Repeat this procedure until all values of the image has been calculated.

Example of convolution

Let's perform some convolution. Step 1 is to flip the mask.

Mask

Let's take our mask to be this.

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

Flipping the mask horizontally

| | | |
|---|---|---|
| 3 | 2 | 1 |
|---|---|---|

LESSON#13 IMAGE PROCESSING & COMPUTER VISION

6

5

4

9

8

7

Flipping the mask vertically

9

8

7

6

5

4

3

2

1

Image

Let's consider an image to be like this

2

4

6

8

10

12

14

16

18

Convolution

Convolving mask over image. It is done in this way. Place the center of the mask at each element of an image. Multiply the corresponding elements and then add them, and paste the result onto the element of the image on which you place the center of mask.

LESSON#13 IMAGE PROCESSING & COMPUTER VISION

| | | | |
|----|---|----|----|
| 9 | 8 | 7 | |
| 6 | 2 | 5 | 4 |
| 3 | 8 | 2 | 10 |
| 14 | | 16 | 18 |

The box in red color is the mask, and the values in the orange are the values of the mask. The black color box and values belong to the image. Now for the first pixel of the image, the value will be calculated as

$$\begin{aligned}\text{First pixel} &= (5*2) + (4*4) + (2*8) + (1*10) \\ &= 10 + 16 + 16 + 10 \\ &= 52\end{aligned}$$

Place 52 in the original image at the first index and repeat this procedure for each pixel of the image.

Why Convolution

Convolution can achieve something, that the **histogram and transform** function methods of manipulating images can't achieve. Those include the **blurring, sharpening, edge detection, noise reduction** etc.

Concept of Mask

A mask is a filter. Concept of masking is also known as spatial filtering. Masking is also known as filtering. In this concept we just deal with the filtering operation that is performed directly on the image.

A sample mask has been shown below

| | | |
|----|---|---|
| -1 | 0 | 1 |
| -1 | 0 | 1 |

-1

0

1

What is filtering

The process of filtering is also known as convolving a mask with an image. As this process is same of convolution so filter masks are also known as convolution masks.

How it is done

The general process of filtering and applying masks is consists of moving the filter mask from point to point in an image. At each point (x,y) of the original image, the response of a filter is calculated by a pre-defined relationship. All the filters values are pre-defined and are a standard.

Types of filters

Generally, there are two types of filters. One is called as linear filters or smoothing filters and others are called as frequency domain filters.

Why filters are used?

Filters are applied on image for multiple purposes. The two most common uses are as following:

- Filters are used for **Blurring** and **noise reduction** → LPF
- Filters are used for **edge detection** and **sharpness** → HPF

Blurring and noise reduction

Filters are most commonly used for blurring and for noise reduction. Blurring is used in preprocessing steps, such as removal of small details from an image prior to large object extraction.

Masks for blurring

The common masks for blurring are.

- Box filter
- Weighted average filter

In the process of blurring we reduce the edge content in an image and try to make the transitions between different pixel intensities as smooth as possible.

Noise reduction is also possible with the help of blurring.

LESSON#13 IMAGE PROCESSING & COMPUTER VISION

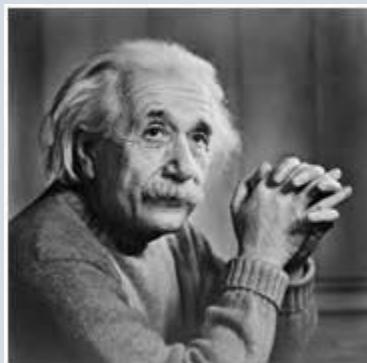
Edge Detection and sharpness

Masks or filters can also be used for edge detection in an image and to increase sharpness of an image.

What are edges

We can also say that sudden changes of discontinuities in an image are called as edges. Significant transitions in an image are called as edges. A picture with edges is shown below.

Original picture

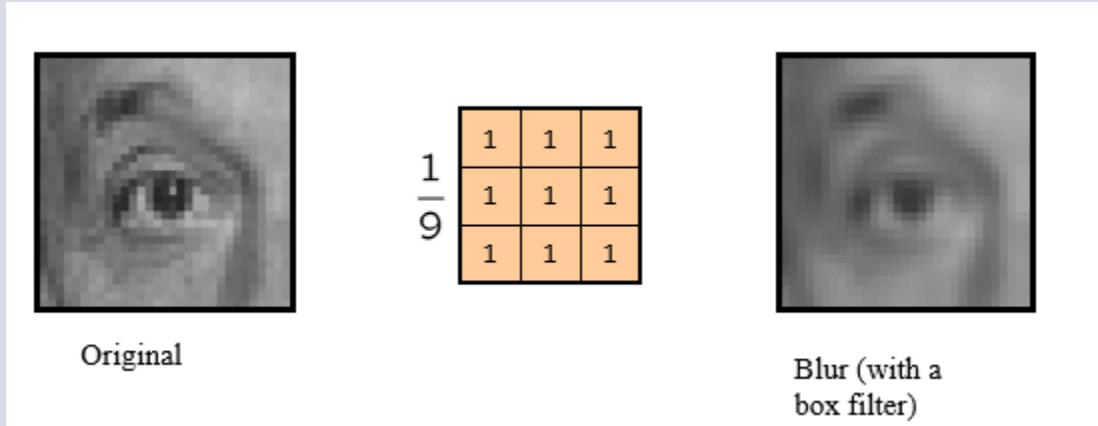


Same picture with edges

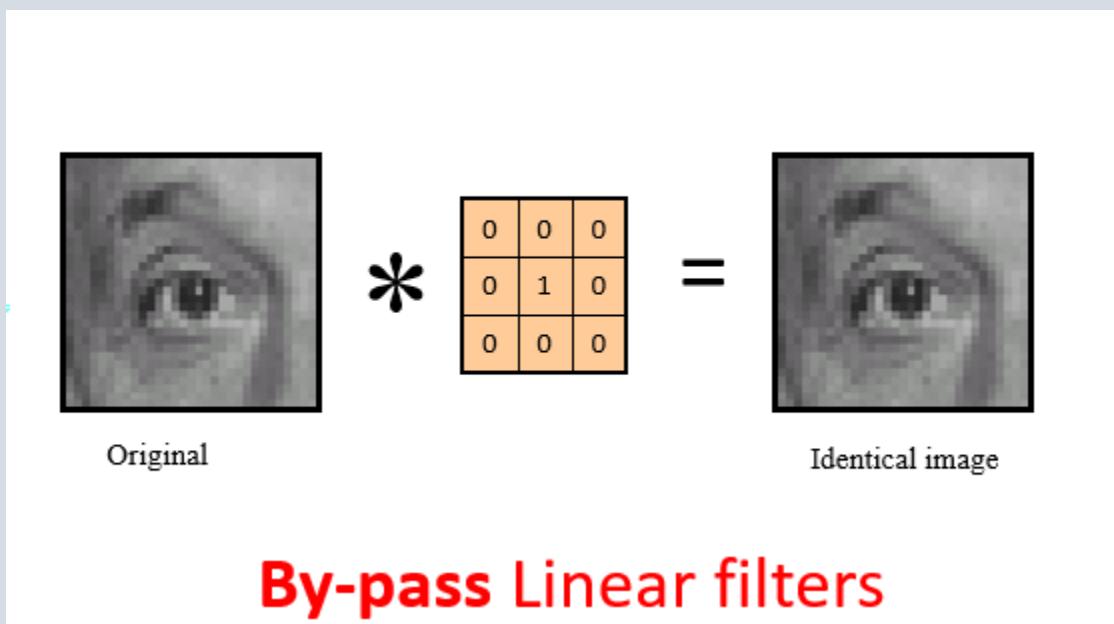


Blurring filter Mask → Low Pass Filters:

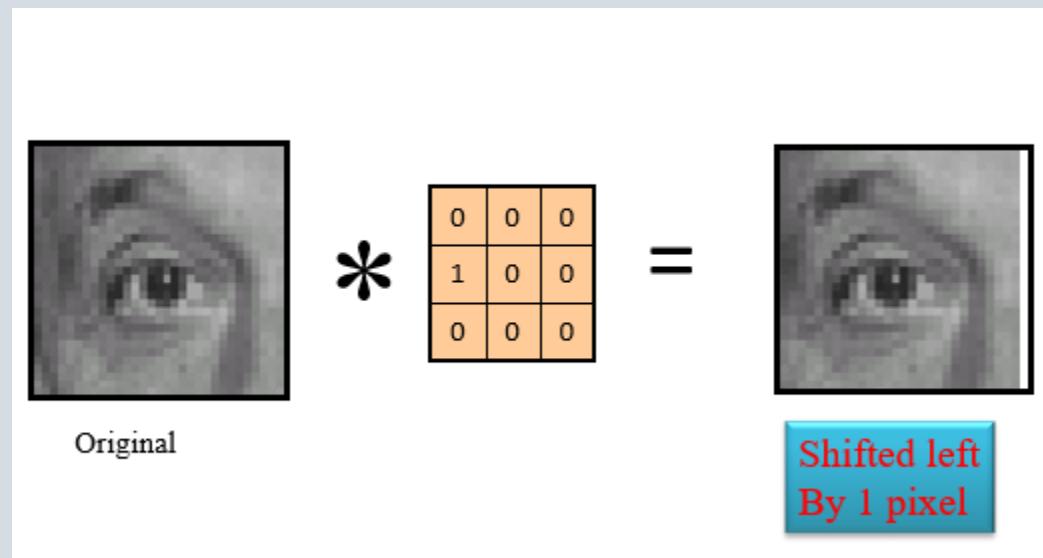
- 1- Mean Filter (Average)



2- Linear filters:



By-pass Linear filters



LESSON#13 IMAGE PROCESSING & COMPUTER VISION

The diagram illustrates the convolution of an "Original" image with a "Blur (with a mean filter)" result. On the left is a grayscale image of an eye. In the center is a multiplication symbol (*) followed by a fraction $\frac{1}{9}$ and a 3x3 kernel matrix where every element is 1. To the right is an equals sign (=) followed by a blurred grayscale image of the same eye.

Original

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

Blur (with a mean filter)

The diagram illustrates the convolution of an "Original" image with a "Sharpening filter (accentuates edges)" result. On the left is a grayscale image of an eye. In the center is a multiplication symbol (*) followed by a subtraction operation ($-$). The first term is a 3x3 kernel matrix with values 0, 0, 0; 0, 2, 0; and 0, 0, 0. The second term is a fraction $\frac{1}{9}$ multiplied by a 3x3 kernel matrix where every element is 1. To the right is a sharper grayscale image of the same eye.

Original

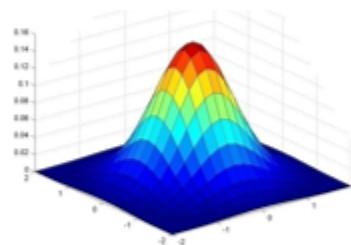
$\ast \left(\begin{matrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{matrix} - \frac{1}{9} \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} \right) =$

Sharpening filter
(accentuates edges)

3- Gaussian Kernel:

LESSON#13 IMAGE PROCESSING & COMPUTER VISION

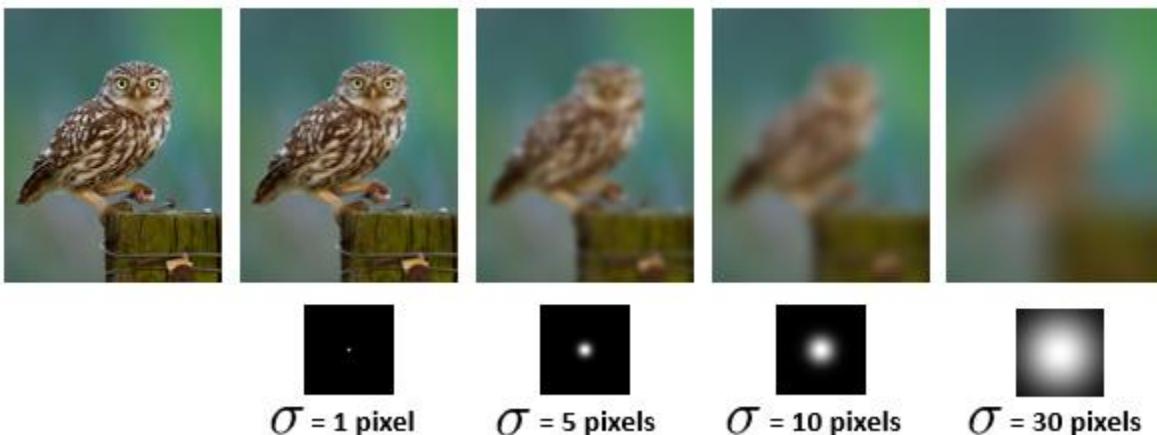
$$G_{\sigma} = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$



| | | | | |
|-------|-------|-------|-------|-------|
| 0.003 | 0.013 | 0.022 | 0.013 | 0.003 |
| 0.013 | 0.059 | 0.097 | 0.059 | 0.013 |
| 0.022 | 0.097 | 0.159 | 0.097 | 0.022 |
| 0.013 | 0.059 | 0.097 | 0.059 | 0.013 |
| 0.003 | 0.013 | 0.022 | 0.013 | 0.003 |

5 x 5, $\sigma = 1$

- Constant factor at front makes volume sum to 1 (can be ignored, as we should re-normalize weights to sum to 1 in any case)

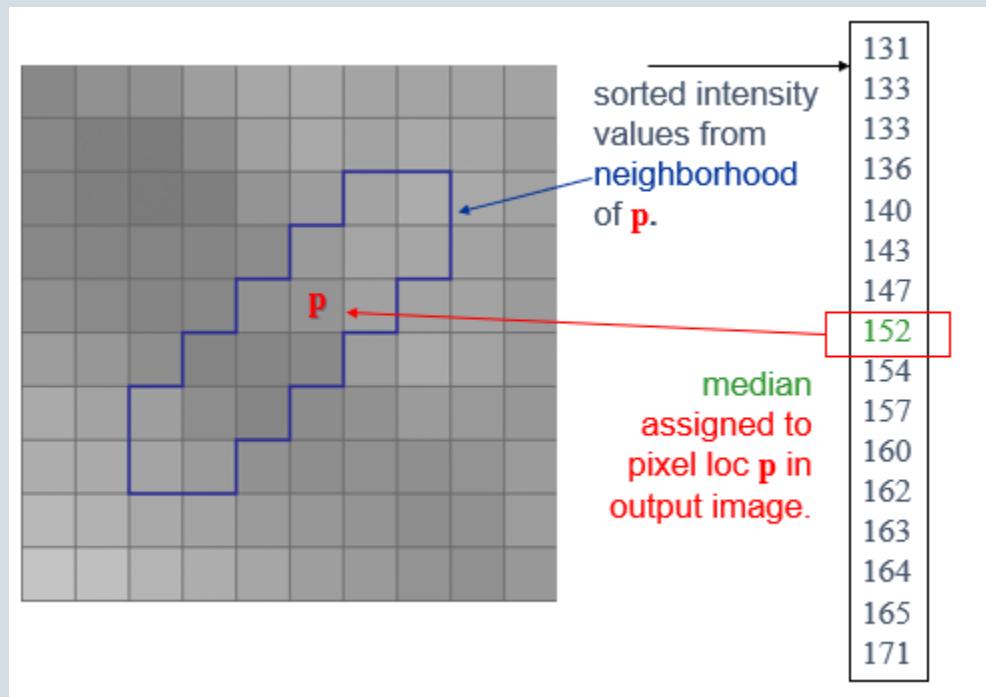
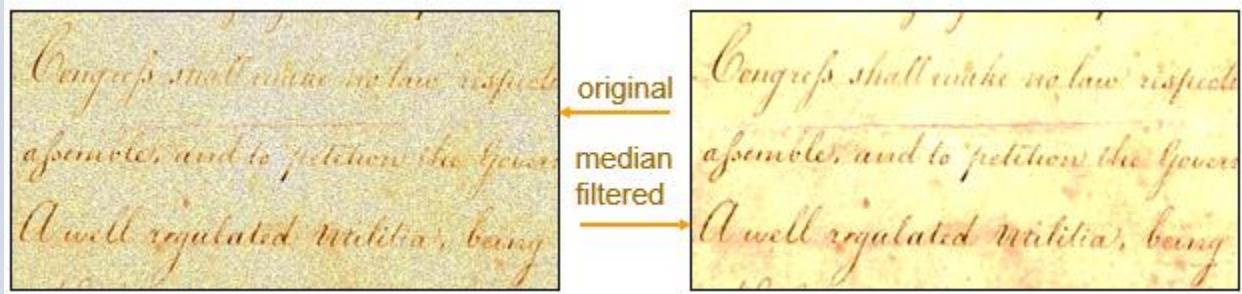


4- The Median Filter:

- Returns the median value of the pixels in a neighborhood
- Is non-linear
- Is a morphological filter
- Is similar to a uniform blurring filter which returns the mean value of the pixels in a neighborhood of a pixel

LESSON#13 IMAGE PROCESSING & COMPUTER VISION

- Unlike a mean value filter the median tends to preserve step edges



The goal of this lab

It is the knowledge how to use convolution with images in MATLAB.

Filter in MATLAB

LESSON#13 IMAGE PROCESSING & COMPUTER VISION

To make filter to image we need first padding the image by zeros. Insert two columns and two rows, as shown below:

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | | | | | | 0 |
| 0 | | | | | | 0 |
| 0 | | | | | | 0 |
| 0 | | | | | | 0 |
| 0 | | | | | | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

After padding (7x7)

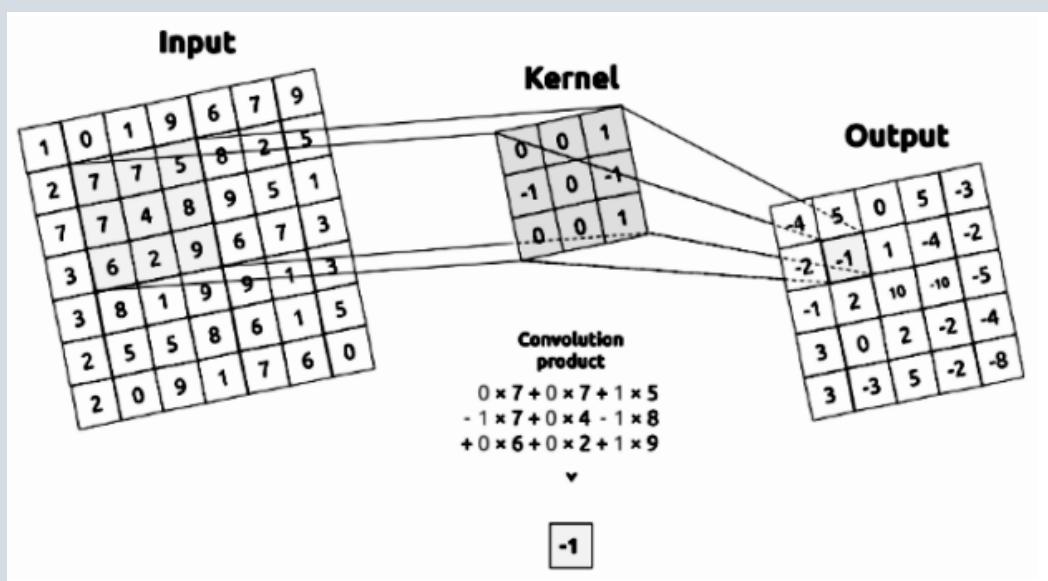
| | | | | |
|--|--|--|--|--|
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

Original Image (5x5)

Next, we need to choose the type and size of the filter that we will use. In this lab we use 3x3 filter and different types of filters.

How we built filter in MATLAB without built in function?

We need to pass through all of the pixels in the image so in order to pass all of the pixels we will use a for loop.



LESSON#13 IMAGE PROCESSING & COMPUTER VISION

In MATLAB

Part of the image

| | | |
|---------|--------|----------|
| i-1,j-1 | i-1, j | i-1, j+1 |
| i,j-1 | i,j | i,j+1 |
| i+1,j-1 | i+1, j | i+1,j+1 |

MATLAB

```
A(i,j)*w(1,1)+A(i,j+1)*w(1,2)+A(i,j+2)*w(1,3) +  
A(i+1,j)*w(2,1)+A(i+1,j+1)*w(2,2)+A(i+1,j+2)*  
w(2,3) +A(i+2,j)* w(3,1)+ A(i+2,j+1) *w(3,2) +  
A(i+2,j+2)*w(3,3)
```

Kernel =

| | | |
|-----|-----|-----|
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |

These equations are placed in a instead loop.

Procedure

LESSON#13 IMAGE PROCESSING & COMPUTER VISION

1- Mean Filter

```
%mean filter by for loop (without function)
clc;
clear all
close all
img=imread('peppers.png');
img=rgb2gray(img);
figure
imshow(img)

[m,n]=size(img);
%Average filter
w=1/9*[1 1 1; 1 1 1; 1 1 1];
%paddind matrix by zeros
A=zeros(m+2,n+2);
for i=1:m
    for j=1: n
        A(i+1,j+1)=img(i,j);
    end
end

%make filter by instead loop
for i=1:m
    for j=1:n
        img
        (i,j)=A(i,j)*w(1,1)+A(i,j+1)*w(1,2)+A(i,j+2)*w(1,3) ...
            +
        A(i+1,j)*w(2,1)+A(i+1,j+1)*w(2,2)+A(i+1,j+2)*w(2,3) ...
            +
        A(i+2,j)*w(3,1)+

        A(i+2,j+1)*w(3,2)+A(i+2,j+2)*w(3,3);
    end
end

% convert image values to int numbers
A=uint8(A);
figure()
imshow(A)
%image after filter
figure('color','white')
imshow(img);
```

LESSON#13 IMAGE PROCESSING & COMPUTER VISION

The Result



Original image





Repeats these steps with color image.

Built Filter bu using MATLAB Function

Imfilter → N-D filtering of multidimensional images

B = imfilter(A,h) filters the multidimensional array A with the multidimensional filter h and returns the result in B.

```
%Average Filter by imfilter & special (True)
clc
clear all
close all
originalRGB = imread('peppers.png');
imshow(originalRGB)
%Create a motion-blur filter using the fspecial
%function.
h = fspecial('average', [3,3]);
%Apply the filter to the original image to create an
%image with motion blur
filteredRGB = imfilter(originalRGB, h);
figure, imshow(filteredRGB)
```

LESSON#13 IMAGE PROCESSING & COMPUTER VISION

The Result



Original Image



After filter

Report

- 1- Open the new script in MATLAB program and make the special filter in two ways (with and without function) but use the 5x5 filter.
- 2- what is happen when we increase the α in Gaussian filter. (Discuss it with show result image)

Lesson#14 Image Processing & COMPUTER VISION

**Image Restoration part1 – Introduction and Noise
Reduction**

Eng. Elaf A.Saeed
Email: elafe1888@gmail.com

LAB Content

Noise Reduction from Grayscale image in MATLAB:

Reduce different Noise from RGB image in MATLAB:

Table of Functions

| Function Name | Description |
|--|---|
| Imfilter N-D filtering of multidimensional images | - N-D filtering of multidimensional images. - $B = \text{imfilter}(A,h)$ filters the multidimensional array A with the multidimensional filter h and returns the result in B . |
| medfilt2 $J = \text{medfilt2}(I)$ $J = \text{medfilt2}(I,[m\ n])$ | 2-D median filtering. |
| imnoise $J = \text{imnoise}(I,\text{'gaussian'})$ $J = \text{imnoise}(I,\text{'gaussian'},m)$ | Add noise to image. |

Theory

Image Restoration

Introduction

Image restoration attempts to reconstruct or recover an image that has been degraded by a degradation phenomenon. Thus, restoration techniques are oriented toward modeling the degradation and applying the inverse process in order to recover the original image. As in image enhancement, the ultimate goal of restoration techniques is to improve an image in some predefined sense.

Image Restoration vs. Image Enhancement

| Image Restoration | Image Enhancement |
|---|--|
| Is an object process. | Is a subject process. |
| Formulation a criterion of goodness that will yield an optimal estimate of the desired result. | Involves heuristic procedures designed to manipulate an image in order to satisfy the human visual system. |
| Techniques include noise removal and deblurring (removal of image blur). | Techniques include contrast stretching. |

Like enhancement techniques, restoration techniques can be performed in the spatial domain and frequency domain. For example, noise removal is applicable using spatial domain filters whereas deblurring is performed using frequency domain filters because image blur is difficult to approach in the spatial domain using small masks.

A model of Image Degradation & Restoration

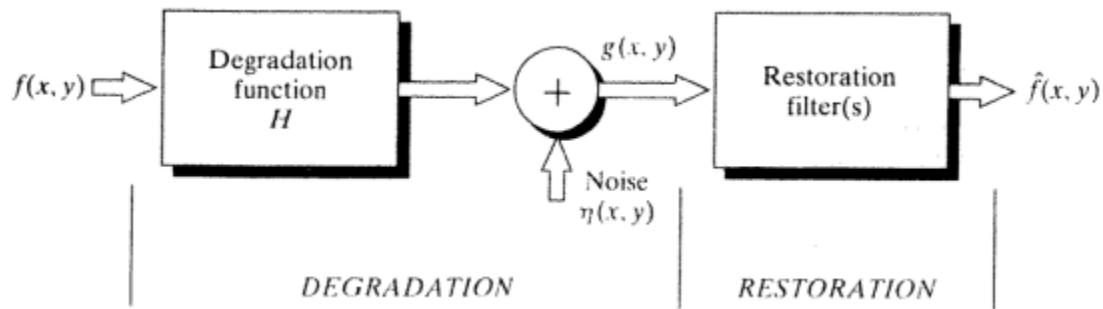
LESSON#14 IMAGE PROCESSING & COMPUTER VISION

As shown in the next figure, image degradation is a process that acts on an input image $f(x,y)$ through a degradation function H and an additive noise $\eta(x,y)$. It results in a degraded image $g(x,y)$ such that:

$$g(x,y) = h(x,y) * f(x,y) + \eta(x,y)$$

where $h(x,y)$ is the spatial representation of the degradation function and the symbol “ $*$ ” indicates convolution.

Note that we only have the degraded image $g(x,y)$. The objective of restoration is to obtain an estimate $\hat{f}(x,y)$ of the original image. We want the estimate to be as close as possible to the original input image and, in general, the more we know about H and η , the closer $\hat{f}(x,y)$ will be to $f(x,y)$.



In the frequency domain, this model is equivalent to:

$$G(u,v) = H(u,v)F(u,v) + N(u,v)$$

The approach that we will study is based on various types of image restoration filters. We assume that H is the identity operator, and we deal only with degradations due to noise.

A noise and its Characteristic

LESSON#14 IMAGE PROCESSING & COMPUTER VISION

Noise in digital images arises during:

- Acquisition: environmental conditions (light level & sensor temperature), and type of cameras
- and/or transmission – interference in the transmission channel

To remove noise we need to understand the spatial characteristics of noise and its frequency characteristics (Fourier spectrum).

Generally, spatial noise is assumed to be independent of position in an image and uncorrelated to the image itself (i.e. there is no correlation between pixel values and the values of noise components). Frequency properties refer to the frequency content of noise in the Fourier sense.

Noise Model

Spatial noise is described by the statistical behavior of the gray-level values in the noise component of the degraded image. Noise can be modeled as a random variable with a specific probability distribution function (PDF). Important examples of noise models include:

1. Gaussian Noise
2. Rayleigh Noise
3. Gamma Noise
4. Exponential Noise
5. Uniform Noise
6. Impulse (Salt & Pepper) Noise

Gaussian Noise

The PDF of Gaussian noise is given by

$$p(z) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(z-\mu)^2/2\sigma^2}$$

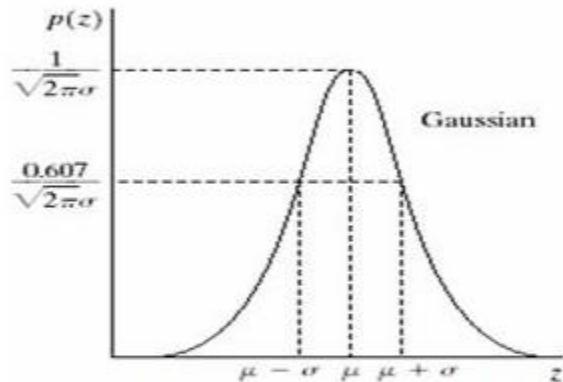


Figure 10.2 Gaussian noise PDF

where z is the gray value, μ is the mean and σ is the standard deviation.

Rayleigh Noise

The PDF of Rayleigh noise is given by

$$p(z) = \begin{cases} (2/b)/(z-a)e^{-(z-a)^2/b} & \text{for } z \geq a \\ 0 & \text{for } z < a \end{cases}$$

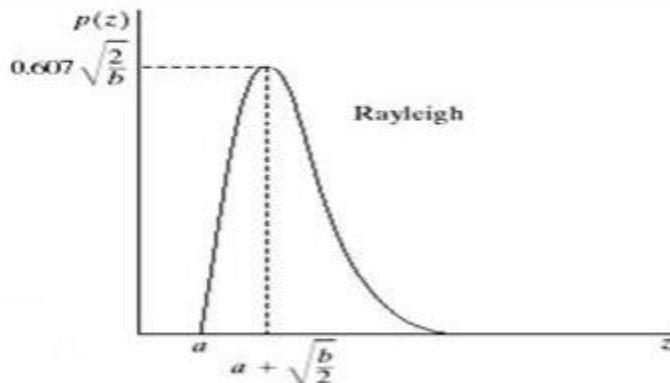


Figure 10.3 Rayleigh noise PDF

Impulse (Salt & Pepper) Noise

The PDF of impulse noise is given by

$$p(z) = \begin{cases} P_a & \text{if } z = a \\ P_b & \text{if } z = b \\ 0 & \text{otherwise} \end{cases}$$

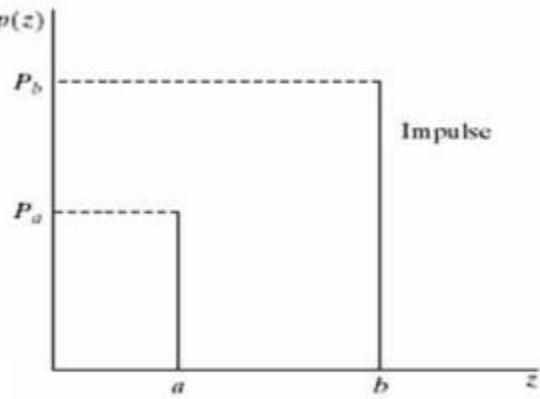


Figure 10.4 Impulse noise PDF

If $b > a$, then gray level b appears as a light dot (salt), otherwise the gray level a appears as a dark dot (pepper).

Determining noise models

The simple image below is well-suited test pattern for illustrating the effect of adding various noise models.



Figure 10.5 Test pattern used to illustrate the characteristics of the noise models

The next figure shows degraded (noisy) images resulted from adding the previous noise models to the above test pattern image.

LESSON#14 IMAGE PROCESSING & COMPUTER VISION

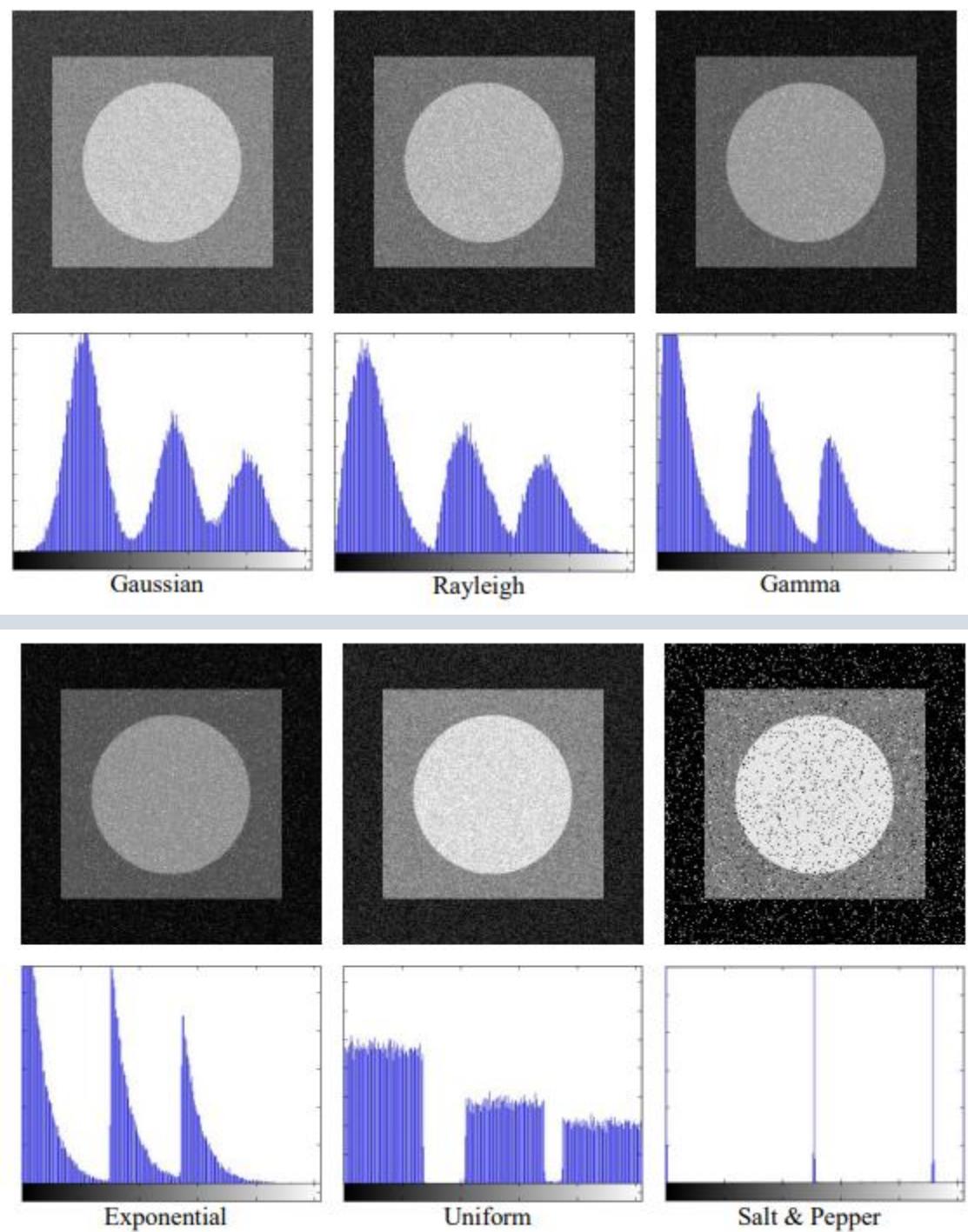


Figure 10.6 Images and histograms from adding Gaussian, Rayleigh, Gamma, Exponential, Uniform, and Salt & Pepper noise.

LESSON#14 IMAGE PROCESSING & COMPUTER VISION

To determine the noise model in a noisy image, one may select a relatively small rectangular sub-image of relatively smooth region. The histogram of the sub-image approximates the probability distribution of the corrupting model of noise. This is illustrated in the figure below.

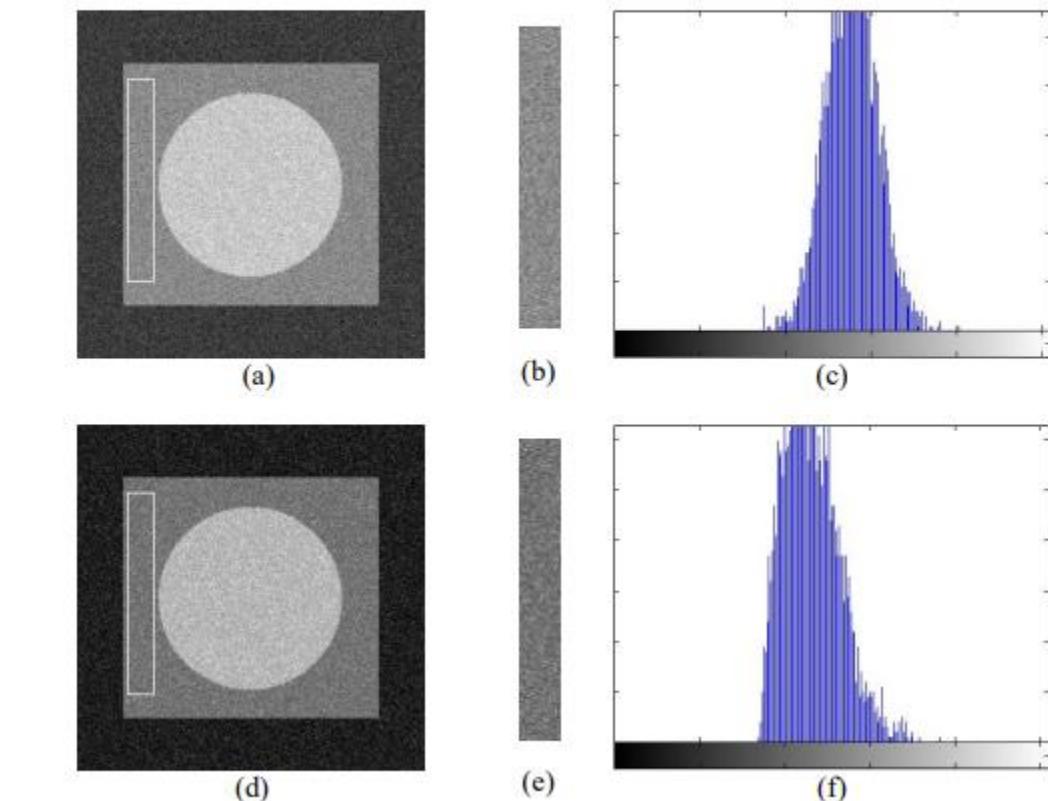


Figure 10.7 (a) Gaussian noisy image. (b) sub-image extracted from a. (c) histogram of b
(d) Rayleigh noisy image. (e) sub-image extracted from d. (f) histogram of e

Image restoration in the presence of Noise Only

When the only degradation present in an image is noise, the degradation is modeled as:

$$g(x, y) = f(x, y) + \eta(x, y)$$

and

$$G(u, v) = F(u, v) + N(u, v)$$

LESSON#14 IMAGE PROCESSING & COMPUTER VISION

Spatial filtering is the method of choice in situations when only additive noise is present. Spatial filters that designed to remove noise include:

1. *Order Statistics Filters*: e.g. Min, Max, & Median
2. *Adaptive Filters*: e.g. adaptive median filter

Order-Statistics Filters

We have used one of these filters (i.e. median) in the image enhancement. We now use additional filters (min and max) in image restoration.

Min filter

This filter is useful for finding the darkest points in an image. Also, it reduces salt noise as a result of the min operation.

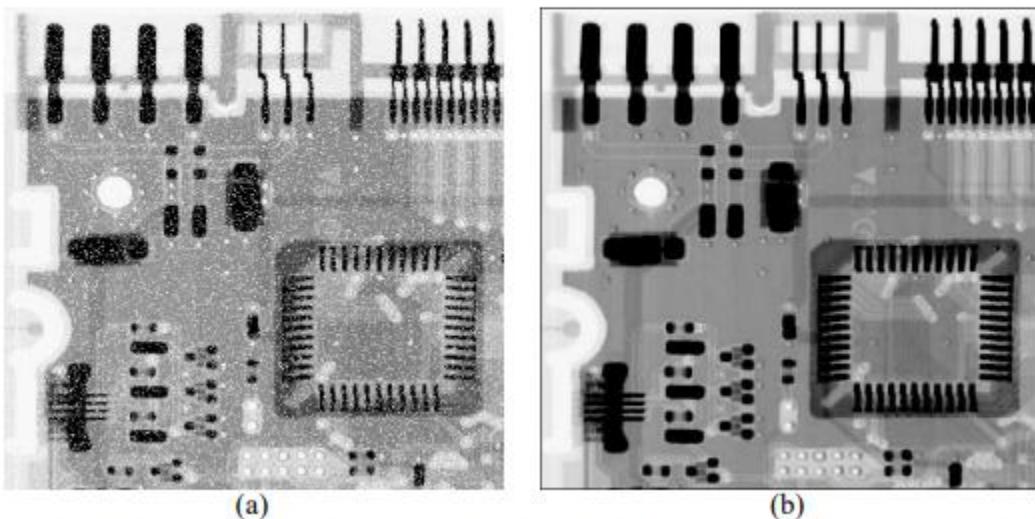


Figure 10.8 (a) image corrupted by salt noise. (b) Result of filtering (a) with a 3×3 min filter.

Max filter

This filter is useful for finding the brightest points in an image. Also, because pepper noise has very low values, it is reduced by this filter as a result of the max operation.

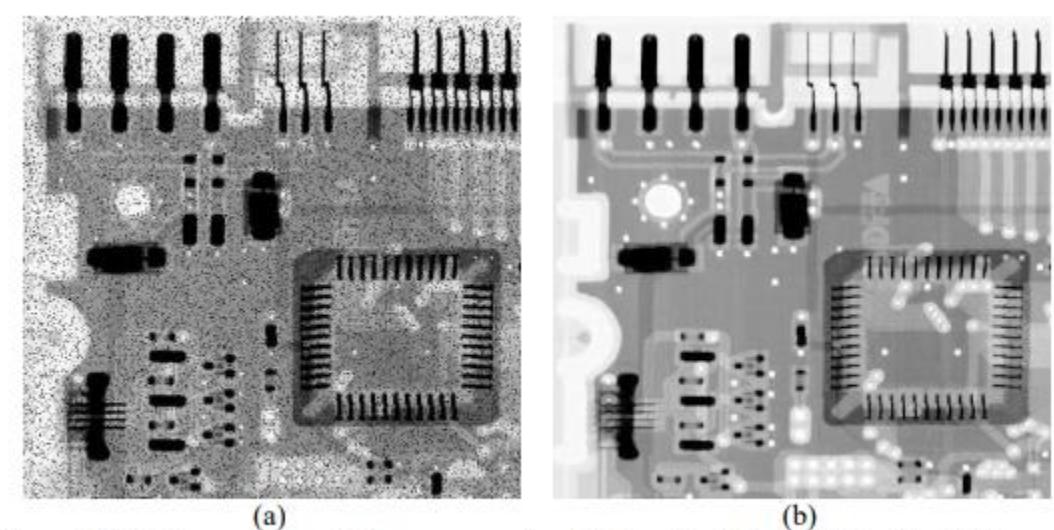


Figure 10.9 (a) image corrupted by pepper noise. (b) Result of filtering (a) with a 3×3 max filter.

Adaptive Filters

The previous spatial filters are applied regardless of local image variation. Adaptive filters change their behavior using local statistical parameters in the mask region. Consequently, adaptive filters outperform the non-adaptive ones.

LESSON#14 IMAGE PROCESSING & COMPUTER VISION

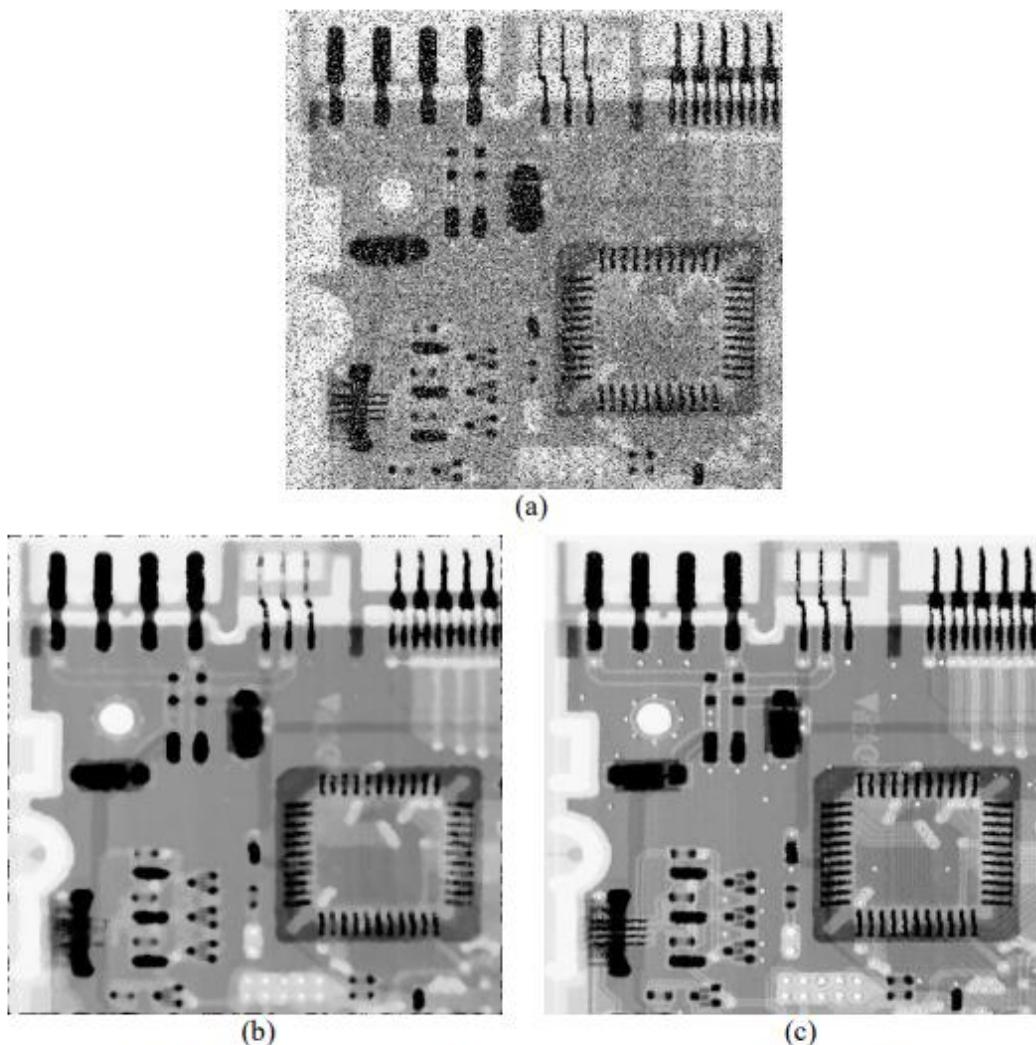


Figure 10.10 (a) Image corrupted by salt&pepper noise with density 0.25. (b) Result obtained using a 7×7 median filter. (c) Result obtained using adaptive median filter with $S_{max} = 7$.

From this example, we find that the adaptive median filter has three main purposes:

1. to remove salt-and-pepper (impulse) noise.
2. to provide smoothing of other noise that may not be impulsive.
3. to reduce distortion, such as excessive thinning or thickening of object boundaries.

The goal of this lab

It is the knowledge the restoration and add noise to image images in MATLAB.

Procedure

- Noise Reduction from Grayscale image in MATLAB:

1. save the image in the current folder.
2. Open new script file and write the following:

```
clc
clear all
close all
im = imread('cameraman.tif');
imshow(im);
% To add Noise
noisy_image=imnoise(im, 'salt & pepper', 0.02);
%image filter
img_filtered = medfilt2(noisy_image); %noise removal

subplot(221), imshow(noisy_image); subplot(222);
imshow(img_filtered)
```

The Result



With Gaussian Noise:

```
clc
clear all
close all
im = imread('cameraman.tif');
imshow(im);
% To add Noise - Gaussian
```

LESSON#14 IMAGE PROCESSING & COMPUTER VISION

```
noisy_image=imnoise(im, 'gaussian', 0.02);  
%image filter  
img_filtered = medfilt2(noisy_image); %noise removal  
  
subplot(221), imshow(noisy_image); subplot(222);  
imshow(img_filtered)
```

The Result



- Reduce different Noise from RGB image in MATLAB:

- 1- save the image in the current folder.
- 2- Open new script file and write the following:

```
clc  
clear all  
close all  
im = imread('gift.jpg');  
subplot(221), imshow(im);  
noise = input('Noise I want to add: ')  
noisy_image=imnoise(im,'salt & pepper',noise);  
subplot(222), imshow(noisy_image);  
  
%average filter  
mat = ones(5,5)/25;  
img1 = imfilter(noisy_image,mat);  
subplot(223), imshow(img1);  
  
%mean filter  
for k=1:3  
    img2(:,:,k) = medfilt2(noisy_image(:,:,k), [3,3]);  
end  
subplot(224), imshow(img2);
```

LESSON#14 IMAGE PROCESSING & COMPUTER VISION

The Result



Report

- 1- Open the new script in MATLAB program to make restoration to image by using different filters. First to remove the pepper noise and then to remove the salt noise.

Lesson#15 Image Processing & COMPUTER VISION

High Pass Filter (HPF) – Edge Detection and Sharpening

Eng. Elaf A.Saeed
Email: elafe1888@gmail.com

LAB Content

Edge Detection

Sharpening image

Table of Functions

| Function Name | Description |
|---|--|
| Edge | <p>Find edges in intensity image.</p> <p><code>BW = edge(I)</code> returns a binary image BW containing 1s where the function finds edges in the grayscale or binary image I and 0s elsewhere. By default, edge uses the Sobel edge detection method.</p> <p><code>BW = edge(I,method)</code> detects edges in image I using the edge-detection algorithm specified by method.</p> |
| imsharpen <code>B = imsharpen(A)</code> <code>B = imsharpen(A,Name,Value)</code> | <p>Sharpen image using unsharp masking.</p> <p><code>B = imsharpen(A)</code> sharpens the grayscale or truecolor (RGB) input image A by using the unsharp masking method.</p> <p><code>B = imsharpen(A,Name,Value)</code> uses name-value pairs to control aspects of the unsharp masking.</p> |

Theory

Concept of Edge Detection

What are edges

We can also say that sudden changes of discontinuities in an image are called as edges. Significant transitions in an image are called as edges.

Types of edges

Generally, edges are of three types:

- Horizontal edges
- Vertical Edges
- Diagonal Edges

Why detects edges

Most of the shape information of an image is enclosed in edges. So first we detect these edges in an image and by using these filters and then by enhancing those areas of image which contains edges, sharpness of the image will increase and image will become clearer.

Here are some of the masks for edge detection that we will discuss in the upcoming tutorials.

- Prewitt Operator
- Sobel Operator
- Robinson Compass Masks
- Krisch Compass Masks
- Laplacian Operator.

Above mentioned all the filters are Linear filters or smoothing filters.

Prewitt Operator

Prewitt operator is used for detecting edges horizontally and vertically.

LESSON#15 IMAGE PROCESSING & COMPUTER VISION

Sobel Operator

The sobel operator is very similar to Prewitt operator. It is also a derivate mask and is used for edge detection. It also calculates edges in both horizontal and vertical direction.

Robinson Compass Masks

This operator is also known as direction mask. In this operator we take one mask and rotate it in all the 8 compass major directions to calculate edges of each direction.

Kirsch Compass Masks

Kirsch Compass Mask is also a derivative mask which is used for finding edges. Kirsch mask is also used for calculating edges in all the directions.

Laplacian Operator

Laplacian Operator is also a derivative operator which is used to find edges in an image. Laplacian is a second order derivative mask. It can be further divided into positive laplacian and negative laplacian.

All these masks find edges. Some find horizontally and vertically, some find in one direction only and some find in all the directions. The next concept that comes after this is sharpening which can be done once the edges are extracted from the image

Sharpening

Sharpening is opposite to the blurring. In blurring, we reduce the edge content and in Sharpening, we increase the edge content. So in order to increase the edge content in an image, we have to find edges first.

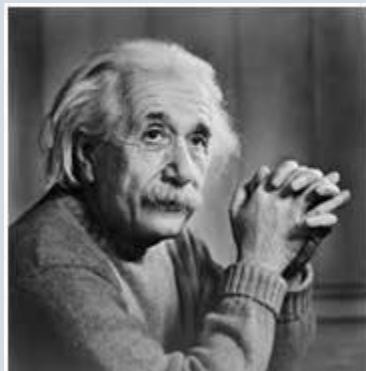
Edges can be find by one of the any method described above by using any operator. After finding edges, we will add those edges on an image and thus the image would have more edges, and it would look sharpen.

This is one way of sharpening an image.

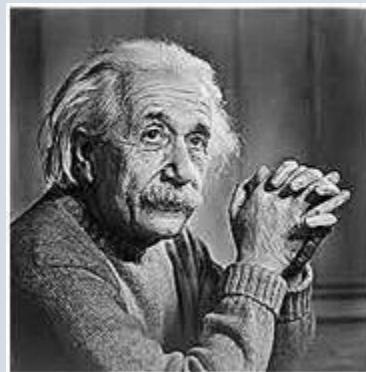
The sharpen image is shown below.

LESSON#15 IMAGE PROCESSING & COMPUTER VISION

Original Image



Sharpen Image



Prewitt Operator

Prewitt operator is used for edge detection in an image. It detects two types of edges

- Horizontal edges
- Vertical Edges

Edges are calculated by using difference between corresponding pixel intensities of an image. All the masks that are used for edge detection are also known as derivative masks. Because as we have stated many times before in this series of labs that image is also a signal so changes in a signal can only be calculated using differentiation. So that's why these operators are also called as derivative operators or derivative masks.

All the derivative masks should have the following properties:

LESSON#15 IMAGE PROCESSING & COMPUTER VISION

- Opposite sign should be present in the mask.
- Sum of mask should be equal to zero.
- More weight means more edge detection.

Prewitt operator provides us two masks one for detecting edges in horizontal direction and another for detecting edges in a vertical direction.

Vertical direction

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

Above mask will find the edges in vertical direction and it is because the zeros column in the vertical direction. When you will convolve this mask on an image, it will give you the vertical edges in an image.

How it works

When we apply this mask on the image it prominent vertical edges. It simply works like as first order derivate and calculates the difference of pixel intensities in an edge region. As the center column is of zero so it does not include the original values of an image but rather it calculates the difference of right and left pixel values around that edge. This increase the edge intensity and it became enhanced comparatively to the original image.

Horizontal Direction

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

Above mask will find edges in horizontal direction and it is because that zeros column is in horizontal direction. When you will convolve this mask onto an image it would prominent horizontal edges in the image.

How it works

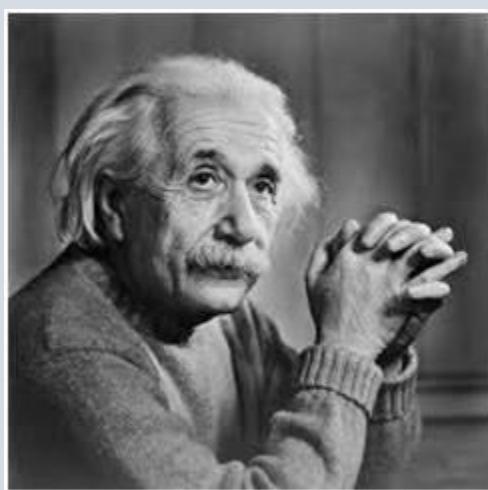
LESSON#15 IMAGE PROCESSING & COMPUTER VISION

This mask will prominent the horizontal edges in an image. It also works on the principle of above mask and calculates difference among the pixel intensities of a particular edge. As the center row of mask is consist of zeros so it does not include the original values of edge in the image but rather it calculates the difference of above and below pixel intensities of the particular edge. Thus, increasing the sudden change of intensities and making the edge more visible. Both the above masks follow the principle of derivate mask. Both masks have opposite sign in them and both masks sum equals to zero. The third condition will not be applicable in this operator as both the above masks are standardizing and we can't change the value in them.

Now it's time to see these masks in action:

Sample Image

Following is a sample picture on which we will apply above two masks one at time.



After applying Vertical Mask

After applying vertical mask on the above sample image, following image will be obtained. This image contains vertical edges. You can judge it more correctly by comparing with horizontal edges picture.



After applying Horizontal Mask

After applying horizontal mask on the above sample image, following image will be obtained.



Comparison

As you can see that in the first picture on which we apply **vertical mask**, all the vertical edges are more visible than the original image. Similarly, in the second picture we have applied the **horizontal mask** and in result all the horizontal edges are

LESSON#15 IMAGE PROCESSING & COMPUTER VISION

visible. So, in this way you can see that we can detect both horizontal and vertical edges from an image.

Sobel Operator

The sobel operator is very similar to Prewitt operator. It is also a derivate mask and is used for edge detection. Like Prewitt operator sobel operator is also used to detect two kinds of edges in an image:

- Vertical direction
- Horizontal direction

Difference with Prewitt Operator

The major difference is that in sobel operator the coefficients of masks are not fixed and they can be adjusted according to our requirement unless they do not violate any property of derivative masks.

Following is the vertical Mask of Sobel Operator:

$$\begin{array}{ccc} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{array}$$

This mask works exactly same as the Prewitt operator vertical mask. There is only one difference that is it has “2” and “-2” values in center of first and third column. When applied on an image this mask will highlight the vertical edges.

How it works

When we apply this mask on the image it prominent vertical edges. It simply works like as first order derivate and calculates the difference of pixel intensities in an edge region.

As the center column is of zero so it does not include the original values of an image but rather it calculates the difference of right and left pixel values around that edge. Also, the center values of both the first and third column is 2 and -2 respectively.

LESSON#15 IMAGE PROCESSING & COMPUTER VISION

This give **more weight** age to the pixel values around the **edge region**. This increase the edge intensity and it became enhanced comparatively to the original image.

Following is the horizontal Mask of Sobel Operator

$$\begin{array}{ccc} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{array}$$

Above mask will find edges in horizontal direction and it is because that zeros column is in horizontal direction. When you will convolve this mask onto an image it would prominent horizontal edges in the image. The only difference between it is that it has 2 and -2 as a center element of first and third row.

How it works

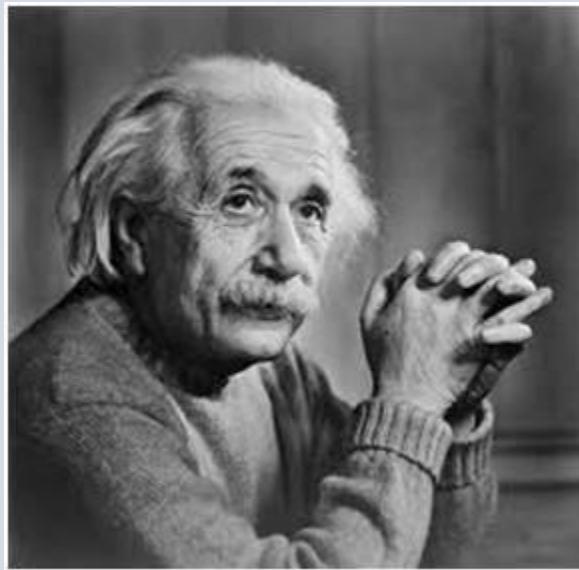
This mask will prominent the horizontal edges in an image. It also works on the principle of above mask and calculates difference among the pixel intensities of a particular edge. As the center row of mask is consist of zeros so it does not include the original values of edge in the image but rather it calculates the difference of above and below pixel intensities of the particular edge. Thus, increasing the sudden change of intensities and making the edge more visible.

Now it's time to see these masks in action:

Sample Image

Following is a sample picture on which we will apply above two masks one at time.

LESSON#15 IMAGE PROCESSING & COMPUTER VISION



After applying Vertical Mask

After applying vertical mask on the above sample image, following image will be obtained.



After applying Horizontal Mask

After applying horizontal mask on the above sample image, following image will be obtained



Comparison

As you can see that in the first picture on which we apply vertical mask, all the vertical edges are more visible than the original image. Similarly in the second picture we have applied the horizontal mask and in result all the horizontal edges are visible.

So in this way you can see that we can detect both horizontal and vertical edges from an image. Also if you compare the result of sobel operator with Prewitt operator, you will find that sobel operator finds more edges or make edges more visible as compared to Prewitt Operator.

This is because in sobel operator we have allotted more weight to the pixel intensities around the edges.

Applying more weight to mask

Now we can also see that if we apply more weight to the mask, the more edges it will get for us. Also as mentioned in the start of the lab that there is no fixed coefficients in sobel operator, so here is another weighted operator

$$\begin{array}{ccc} -1 & 0 & 1 \\ -5 & 0 & 5 \\ -1 & 0 & 1 \end{array}$$

If you can compare the result of this mask with of the Prewitt vertical mask, it is clear that this mask will give out more edges as compared to Prewitt one just because we have allotted more weight in the mask.

Robinson Compass Mask

Robinson compass masks are another type of derivate mask which is used for edge detection. This operator is also known as direction mask. In this operator we take one mask and rotate it in all the 8 compass major directions that are following:

- North
- North West
- West
- South West
- South
- South East
- East
- North East

There is no fixed mask. You can take any mask and you have to rotate it to find edges in all the above-mentioned directions. All the masks are rotated on the bases of direction of zero columns.

For example, let's see the following mask which is in North Direction and then rotate it to make all the direction masks.

LESSON#15 IMAGE PROCESSING & COMPUTER VISION

North Direction Mask

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

North West Direction Mask

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

West Direction Mask

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

South West Direction Mask

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

South Direction Mask

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

South East Direction Mask

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

East Direction Mask

| | | |
|----|----|----|
| -1 | -2 | -1 |
|----|----|----|

LESSON#15 IMAGE PROCESSING & COMPUTER VISION

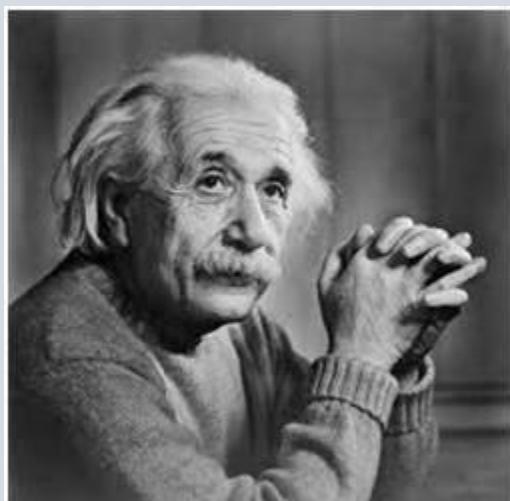
$$\begin{matrix} 0 & 0 & 0 \\ 1 & 2 & 1 \end{matrix}$$

North East Direction Mask

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

As you can see that all the directions are covered on the basis of zeros direction. Each mask will give you the edges on its direction. Now let's see the result of the entire above masks. Suppose we have a sample picture from which we have to find all the edges. Here is our sample picture:

Sample Picture



Now we will apply all the above filters on this image and we get the following result.

North Direction Edges

LESSON#15 IMAGE PROCESSING & COMPUTER VISION



North West Direction Edges



West Direction Edges



South West Direction Edges

LESSON#15 IMAGE PROCESSING & COMPUTER VISION



South Direction Edges



South East Direction Edges



East Direction Edges

LESSON#15 IMAGE PROCESSING & COMPUTER VISION



North East Direction Edges



As you can see that by applying all the above masks you will get edges in all the direction. Result is also depending on the image. Suppose there is an image, which do not have any North East direction edges so then that mask will be ineffective.

Kirsch Compass Mask

Kirsch Compass Mask is also a derivative mask which is used for finding edges. This is also like Robinson compass find edges in all the eight directions of a compass. The only difference between Robinson and kirsch compass masks is that in Kirsch we have a standard mask but in Kirsch we change the mask according to our own requirements.

With the help of Kirsch Compass Masks we can find edges in the following eight directions.

- North
- North West
- West
- South West

LESSON#15 IMAGE PROCESSING & COMPUTER VISION

- South
- South East
- East
- North East

We take a standard mask which follows all the properties of a derivative mask and then rotate it to find the edges.

For example let's see the following mask which is in North Direction and then rotate it to make all the direction masks.

North Direction Mask

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

North West Direction Mask

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

West Direction Mask

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

LESSON#15 IMAGE PROCESSING & COMPUTER VISION

South West Direction Mask

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

South Direction Mask

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

South East Direction Mask

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

East Direction Mask

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

North East Direction Mask

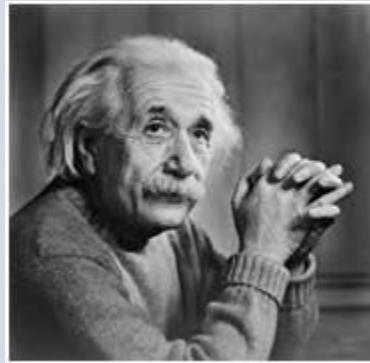
| | | |
|----|----|----|
| -3 | -3 | -3 |
|----|----|----|

LESSON#15 IMAGE PROCESSING & COMPUTER VISION

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

As you can see that all the directions are covered and each mask will give you the edges of its own direction. Now to help you better understand the concept of these masks we will apply it on a real image. Suppose we have a sample picture from which we have to find all the edges. Here is our sample picture:

Sample Picture



Now we will apply all the above filters on this image and we get the following result.

North Direction Edges



LESSON#15 IMAGE PROCESSING & COMPUTER VISION

North West Direction Edges



West Direction Edges



South West Direction Edges



LESSON#15 IMAGE PROCESSING & COMPUTER VISION

South Direction Edges



South East Direction Edges



East Direction Edges



LESSON#15 IMAGE PROCESSING & COMPUTER VISION

North East Direction Edges



As you can see that by applying all the above masks you will get edges in all the direction. Result is also depending on the image. Suppose there is an image, which do not have any North East direction edges so then that mask will be ineffective.

Laplacian Operator

Laplacian Operator is also a derivative operator which is used to find edges in an image. The major difference between Laplacian and other operators like Prewitt, Sobel, Robinson and Kirsch is that these all are first order derivative masks but Laplacian is a second order derivative mask. In this mask we have two further classifications one is Positive Laplacian Operator and other is Negative Laplacian Operator.

Another difference between Laplacian and other operators is that unlike other operators Laplacian didn't take out edges in any particular direction but it take out edges in following classification.

- Inward Edges
- Outward Edges

Let's see that how Laplacian operator works.

Positive Laplacian Operator

In Positive Laplacian we have standard mask in which center element of the mask should be negative and corner elements of mask should be zero.

0

1

0

LESSON#15 IMAGE PROCESSING & COMPUTER VISION

| | | |
|---|----|---|
| 1 | -4 | 1 |
| 0 | 1 | 0 |

Positive Laplacian Operator is use to take out outward edges in an image.

Negative Laplacian Operator

In negative Laplacian operator we also have a standard mask, in which center element should be positive. All the elements in the corner should be zero and rest of all the elements in the mask should be -1.

| | | |
|----|----|----|
| 0 | -1 | 0 |
| -1 | 4 | -1 |
| 0 | -1 | 0 |

Negative Laplacian operator is use to take out inward edges in an image

How it works

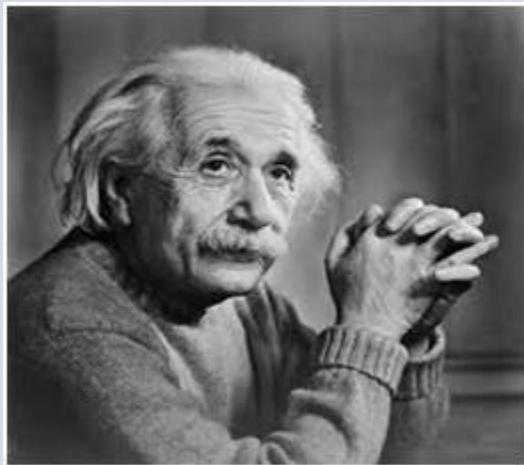
Laplacian is a derivative operator; its uses highlight gray level discontinuities in an image and try to deemphasize regions with slowly varying gray levels. This operation in result produces such images which have grayish edge lines and other discontinuities on a dark background. This produces inward and outward edges in an image

The important thing is how to apply these filters onto image. Remember we can't apply both the positive and negative Laplacian operator on the same image. we have to apply just one but the thing to remember is that if we apply positive Laplacian operator on the image then we subtract the resultant image from the original image to get the sharpened image. Similarly, if we apply negative Laplacian operator then we have to add the resultant image onto original image to get the sharpened image.

Let's apply these filters onto an image and see how it will get us inward and outward edges from an image. Suppose we have a following sample image.

LESSON#15 IMAGE PROCESSING & COMPUTER VISION

Sample Image



After applying Positive Laplacian Operator

After applying positive Laplacian operator we will get the following image.



After applying Negative Laplacian Operator

After applying negative Laplacian operator, we will get the following image.



The goal of this lab

It is the knowledge the **edge detection** with images in MATLAB.

Procedure

- Edge Detection

1. save the image in the current folder.
2. Open new script file and write the following:

Edge Detection with Noise Image

```
close all  
%Clear current figure window  
clf  
I = imread('cameraman.tif');  
imshow(I)  
subplot(4,2,1),imshow(I),title('original image')  
%0.02 == Noise density for salt and pepper noise,  
specified as a numeric scalar.  
im_noise=imnoise(I,'salt & pepper',0.02);
```

LESSON#15 IMAGE PROCESSING & COMPUTER VISION

```
subplot(4,2,2),imshow(im_noise),title('noise  
image')  
%Edge detection before filtering  
im_edge_pre=edge(im_noise,'sobel',0.2);  
subplot(4,2,3),imshow(im_edge_pre),title('before  
filtering')  
% Image Filtering  
im_filter=medfilt2(im_noise);  
subplot(4,2,4),imshow(im_filter),title('Image  
Filtering')  
%Edge detection after filtering  
%Sobel Operator  
im_edge=edge(im_filter,'sobel',0.2);  
subplot(4,2,5),imshow(im_edge),title('Sobel')  
%Prewitt Operator  
im_edge=edge(im_filter,'prewitt',0.2);  
subplot(4,2,6),imshow(im_edge),title('prewitt')  
%canny Operator  
im_edge=edge(im_filter,'canny',0.2);  
subplot(4,2,7),imshow(im_edge),title('canny')
```

The Result

original image



noise image



before filtering



Image Filtering



Sobel



prewitt



canny



LESSON#15 IMAGE PROCESSING & COMPUTER VISION

Edge Detection without MATLAB Function - Sobel method

```
clc
clear all
close all

a= imread('gift.jpg');
b= rgb2gray(a);
c= double(b);
[m n]= size(b);

for i=1:m-2
    for j=1:n-2
        gx = (c(i+2,j)+2*c(i+2)+c(i+2,j+2))-
(c(i,j)+2*c(i,j+1)+c(i,j+2));
        gy = (c(i,j+2)+2*c(i+1,j+2)+c(i+2,j+2))-
(c(i,j)+2*c(i+1,j)+c(i+2,j));
        b(i,j)=sqrt(gx.^2+gy.^2);
    end
end
subplot(1,2,1)
imshow(a)
title('original');

subplot(1,2,2);
imshow(b);
```

The Result



LESSON#15 IMAGE PROCESSING & COMPUTER VISION

- Sharpening image

- 1- save the image in the current folder.
- 2- Open new script file and write the following:

```
%Sharpen Function image (RGB image)
clc
clear all
close all
x=imread('peppers.png');
figure;
imshow(x);
title('Original Image');
I_sharpen=imsharpen(x,'amount',3);
figure;
imshow(I_sharpen);
title('Sharpened Image');
```

The Result

Original Image



LESSON#15 IMAGE PROCESSING & COMPUTER VISION

Sharpened Image



Sharpening the result of edge detection:

```
clc
clear all
close all

a= imread('gift.jpg');
b= rgb2gray(a);
c= double(b);
[m n]= size(b);

for i=1:m-2
    for j=1:n-2
        gx = (c(i+2,j)+2*c(i+2)+c(i+2,j+2))-
(c(i,j)+2*c(i,j+1)+c(i,j+2));
        gy = (c(i,j+2)+2*c(i+1,j+2)+c(i+2,j+2))-
(c(i,j)+2*c(i+1,j)+c(i+2,j));
        b(i,j)=sqrt(gx.^2+gy.^2);
    end
end
subplot(1,3,1)
imshow(a)
```

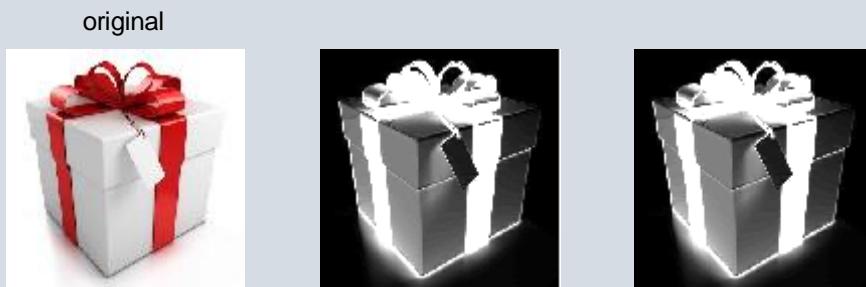
LESSON#15 IMAGE PROCESSING & COMPUTER VISION

```
title('original');

subplot(1,3,2);
imshow(b);

I_sharpen=imsharpen(a, 'amount', 3);
subplot(1,3,3);
imshow(b);
```

The Result



Report

- 1- Write the MATLAB program to sharping image without built in function.
- 2- Write the MATLAB program to down sampling the 1024x1024 image to 512x512, 256x256, and 128x128.
- 3- Write the MATLAB program to add Gaussian noise to image and remove it.

Lesson#16 Image Processing & COMPUTER VISION

Frequency Domain – Filter in Frequency Domain

Eng. Elaf A.Saeed
Email: elafe1888@gmail.com

LAB Content

Fourier Transformation without Mask filter

Bandworth Lowpass filter in frequency Domain

Table of Functions

| Function Name | Description |
|---|--|
| fft2 Y = fft2(X) | 2-D fast Fourier transform. Y = fft2(X) returns the two-dimensional Fourier transform of a matrix using a fast Fourier transform algorithm, which is equivalent to computing <code>fft(fft(X).').'</code> . If X is a multidimensional array, then fft2 takes the 2-D transform of each dimension higher than 2. The output Y is the same size as X. |
| Fftshift Y = fftshift(X) | Shift zero-frequency component to center of spectrum. Y = fftshift(X) rearranges a Fourier transform X by shifting the zero-frequency component to the center of the array. |
| ifft2 X = ifft2(Y) | 2-D inverse fast Fourier transform. X = ifft2(Y) returns the two-dimensional discrete inverse Fourier transform of a matrix using a fast Fourier transform algorithm. If Y is a multidimensional array, then ifft2 takes the 2-D inverse transform of each dimension higher than 2. The output X is the same size as Y. |

Theory

Introduction to Frequency domain

Frequency domain analysis

Till now, all the domains in which we have analyzed a signal , we analyze it with respect to time. But in frequency domain we don't analyze signal with respect to time, but with respect of frequency.

Difference between spatial domain and frequency domain

In spatial domain, we deal with images as it is. The value of the pixels of the image change with respect to scene. Whereas in frequency domain, we deal with the rate at which the pixel values are changing in spatial domain.

For simplicity, Let's put it this way.

Spatial domain



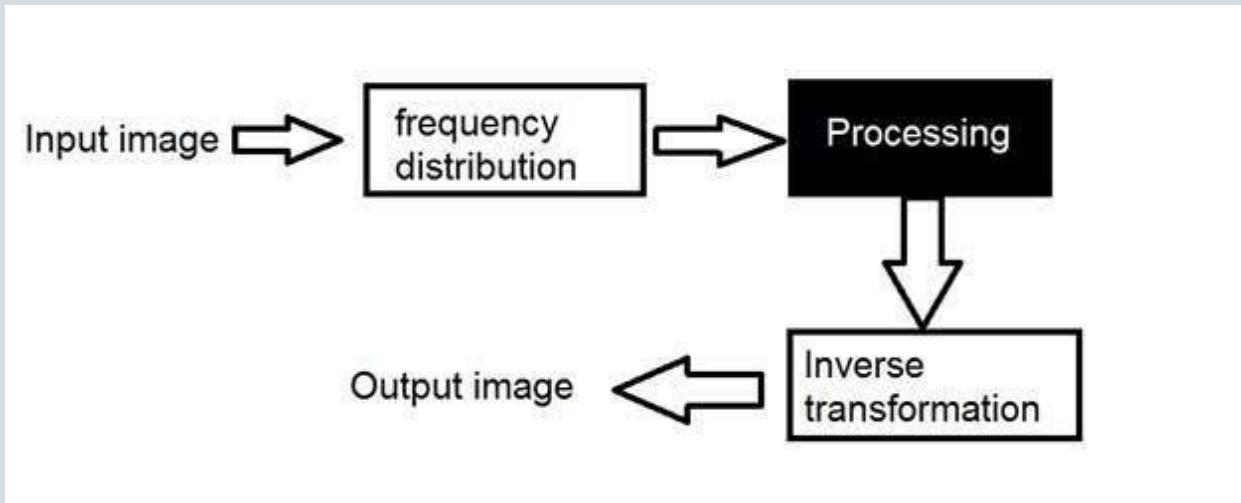
In simple spatial domain, we directly deal with the image matrix. Whereas in frequency domain, we deal an image like this.

Frequency Domain

We first transform the image to its frequency distribution. Then our black box system performs whatever processing it has to performed, 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

LESSON#16 IMAGE PROCESSING & COMPUTER VISION



Here we have used the word transformation. What does it actually mean?

Transformation

A signal can be converted from time domain into frequency domain using mathematical operators called transforms. There are many kind of transformation that does this. Some of them are given below.

- Fourier Series
- Fourier transformation
- Laplace transform
- Z transform

Out of all these, we will thoroughly discuss Fourier series and Fourier transformation in our next tutorial.

Frequency components

Any image in spatial domain can be represented in a frequency domain. But what do these frequencies actually mean.

We will divide frequency components into two major components.

High frequency components

High frequency components correspond to edges in an image.

Low frequency components

Low frequency components in an image correspond to smooth regions.

Fourier

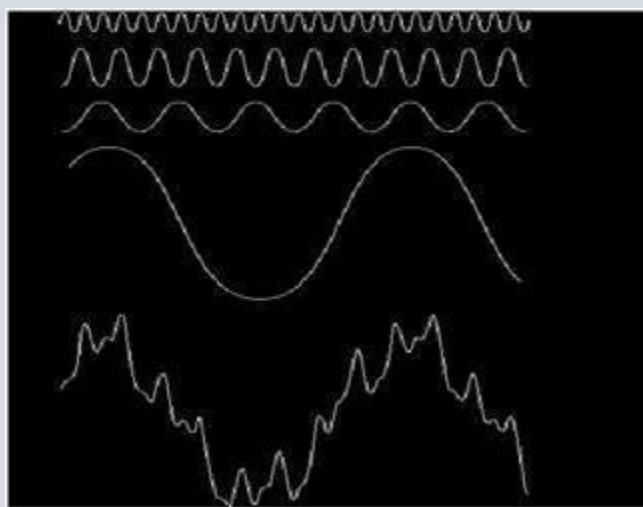
Fourier was a mathematician in 1822. He give Fourier series and Fourier transform to convert a signal into frequency domain.

Fourier Series

Fourier series simply states that, periodic signals can be represented into sum of sines and cosines when multiplied with a certain weight. It further states that periodic signals can be broken down into further signals with the following properties.

- The signals are sines and cosines
- The signals are harmonics of each other

It can be pictorially viewed as



In the above signal, the last signal is actually the sum of all the above signals. This was the idea of the Fourier.

How it is calculated

Since as we have seen in the frequency domain, that in order to process an image in frequency domain, we need to first convert it using into frequency domain and we have to take inverse of the output to convert it back into spatial domain. That's why both Fourier series and Fourier transform has two formulas. One for conversion and one converting it back to the spatial domain.

Fourier series

The Fourier series can be denoted by this formula.

LESSON#16 IMAGE PROCESSING & COMPUTER VISION

$$F(u, v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) e^{-j2\pi(ux+vy)} dx dy$$

The inverse can be calculated by this formula.

$$f(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(u, v) e^{j2\pi(ux+vy)} du dv.$$

Fourier transform

The Fourier transform simply states that that the non-periodic signals whose area under the curve is finite can also be represented into integrals of the sines and cosines after being multiplied by a certain weight.

The Fourier transform has many wide applications that include, image compression (e.g JPEG compression), filtering and image analysis.

Difference between Fourier series and transform

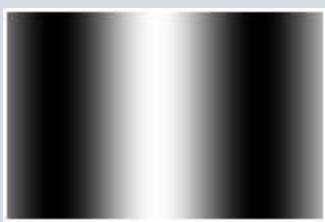
Although both Fourier series and Fourier transform are given by Fourier, but the difference between them is Fourier series is applied on periodic signals and Fourier transform is applied for non-periodic signals

Which one is applied on images

Now the question is that which one is applied on the images, the Fourier series or the Fourier transform. Well, the answer to this question lies in the fact that what images are. Images are non – periodic. And since the images are non-periodic, so Fourier transform is used to convert them into frequency domain.

Discrete fourier transform

Since we are dealing with images, and in fact digital images, so for digital images we will be working on discrete fourier transform



LESSON#16 IMAGE PROCESSING & COMPUTER VISION

Consider the above Fourier term of a sinusoid. It includes three things.

- Spatial Frequency
- Magnitude
- Phase

The spatial frequency directly relates with the brightness of the image. The magnitude of the sinusoid directly relates with the contrast. Contrast is the difference between maximum and minimum pixel intensity. Phase contains the color information.

The formula for 2-dimensional discrete Fourier transform is given below.

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

The discrete Fourier transform is actually the sampled Fourier transform, so it contains some samples that denotes an image. In the above formula $f(x,y)$ denotes the image, and $F(u,v)$ denotes the discrete Fourier transform. The formula for 2 dimensional inverse discrete Fourier transform is given below.

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

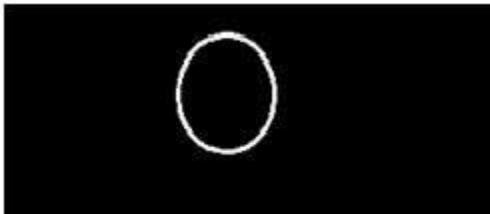
The inverse discrete Fourier transform converts the Fourier transform back to the image

Consider this signal

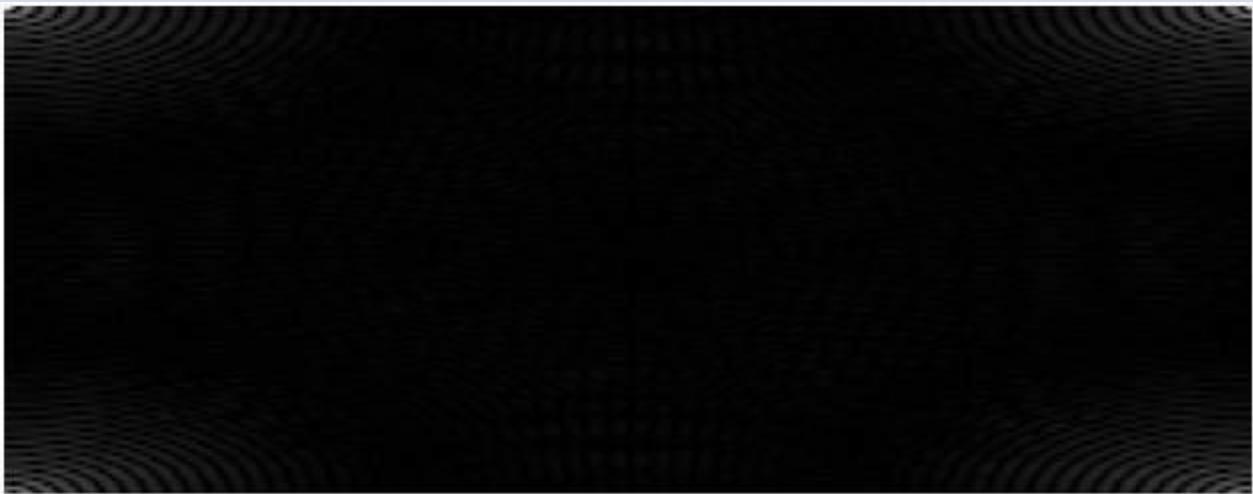
Now we will see an image, whose we will calculate FFT magnitude spectrum and then shifted FFT magnitude spectrum and then we will take Log of that shifted spectrum.

LESSON#16 IMAGE PROCESSING & COMPUTER VISION

Original Image



The Fourier transform magnitude spectrum

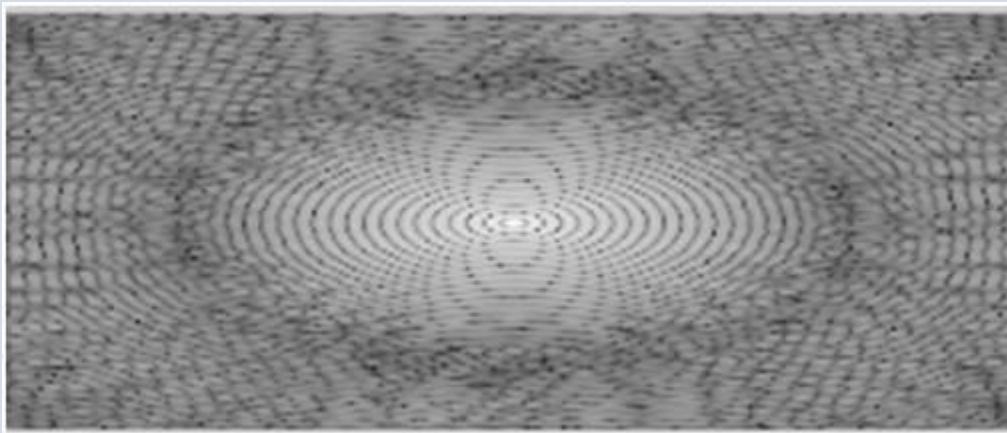


The Shifted Fourier transform



LESSON#16 IMAGE PROCESSING & COMPUTER VISION

The Shifted Magnitude Spectrum

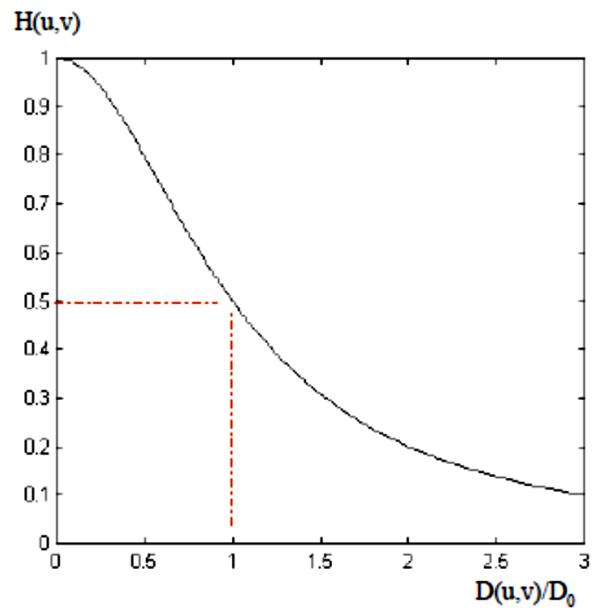


Butterworth lowpass filter

- The transfer function of a Butterworth lowpass filter (BLPF) of order n with cutoff frequency D_0 is given by

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

- where $D(u, v) = [u^2 + v^2]^{1/2}$
- For this smooth transition filter, a cutoff frequency locus is chosen such that $D(u, v)$ is a certain percentage of its maximum
- Designed such that at $D(u, v) = D_0$ $H(u, v) = 0.50$ (50% of its maximum value)



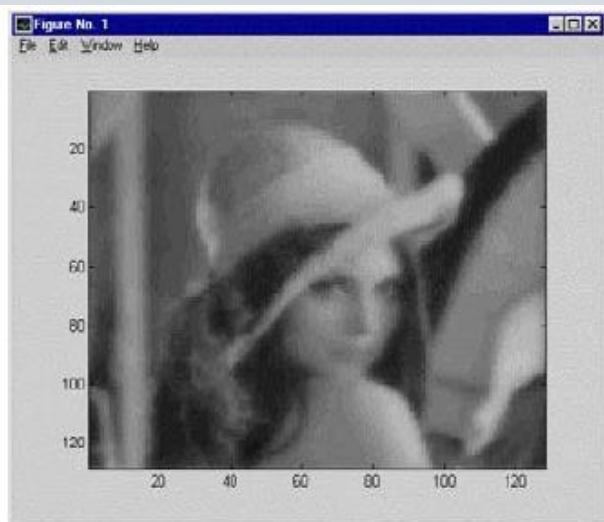
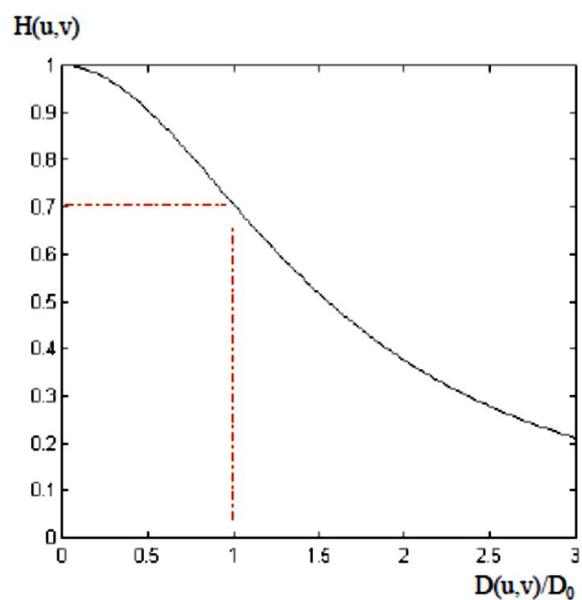
LESSON#16 IMAGE PROCESSING & COMPUTER VISION

- Another transfer function of a Butterworth lowpass filter (BLPF) of order n with cutoff frequency D_0 is given by

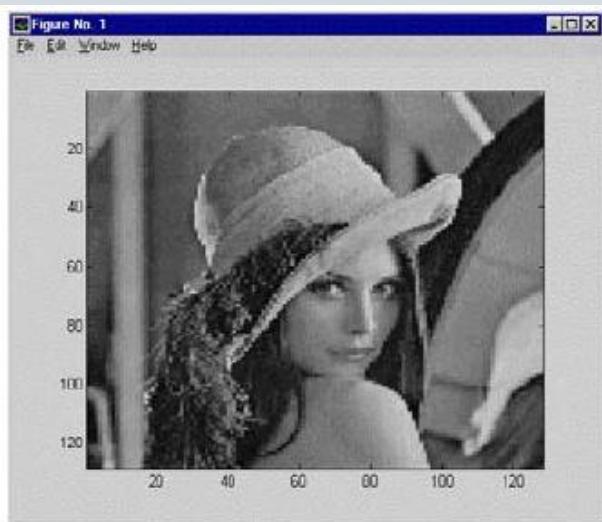
$$H(u,v) = \frac{1}{1 + [\sqrt{2} - 1][D(u,v)/D_0]^{2n}}$$

- Designed such that at $D(u,v)=D_0$

$$H(u,v) = \frac{1}{\sqrt{2}}$$

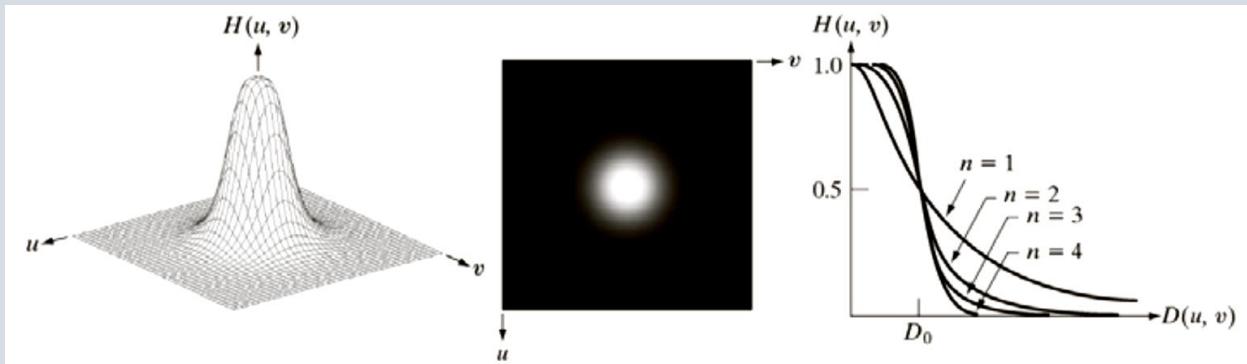


- Cutoff frequency=10



- Cutoff frequency=100

LESSON#16 IMAGE PROCESSING & COMPUTER VISION



a b c

FIGURE 4.44 (a) Perspective plot of a Butterworth lowpass-filter transfer function. (b) Filter displayed as an image. (c) Filter radial cross sections of orders 1 through 4.

Gaussian lowpass filter

- The transfer function of a Gaussian lowpass filter (GLPF) is given by

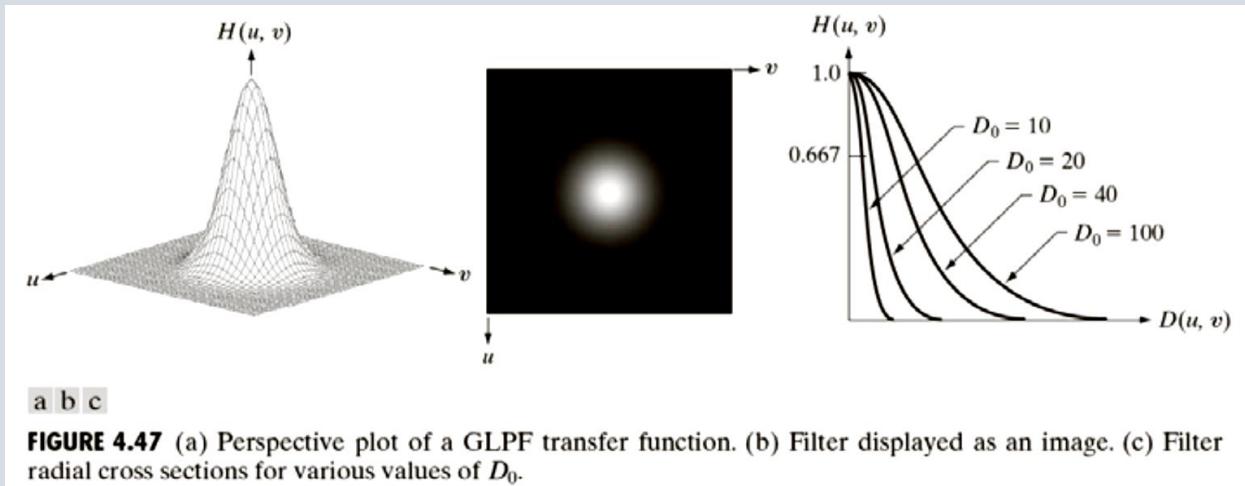
$$H(u, v) = e^{-D^2(u,v)/2\sigma^2}$$

- Here, σ is a measure of spread about the center
- Let $\sigma = D_0$, then

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

- where D_0 is the cutoff frequency

LESSON#16 IMAGE PROCESSING & COMPUTER VISION



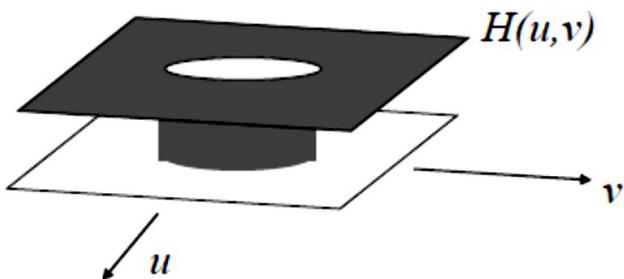
a b c

FIGURE 4.47 (a) Perspective plot of a GLPF transfer function. (b) Filter displayed as an image. (c) Filter radial cross sections for various values of D_0 .

Ideal highpass filter (IHPF)

- A transfer function for a 2-D ideal highpass filter (IHPF) is given as
- $$H(u, v) = \begin{cases} 0 & \text{if } D(u, v) \leq D_0 \\ 1 & \text{if } D(u, v) > D_0 \end{cases}$$
- where D_0 is a stated nonnegative quantity (the cutoff frequency) and $D(u, v)$ is the distance from the point (u, v) to the center of the frequency plane

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



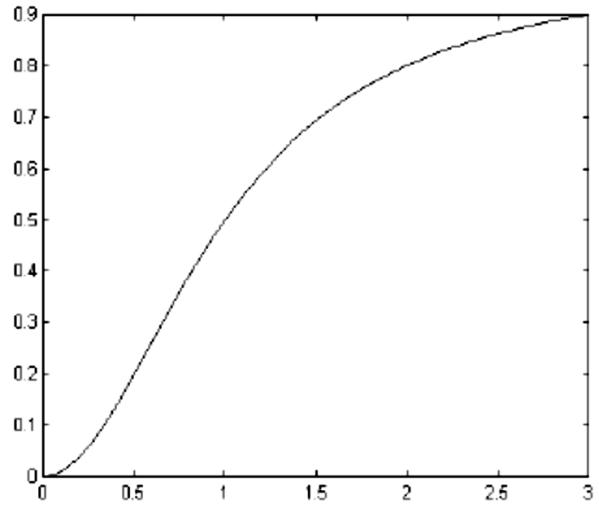
Butterworth highpass filter

LESSON#16 IMAGE PROCESSING & COMPUTER VISION

- The transfer function of a Butterworth highpass filter (BHPF) of order n with cutoff frequency D_0 is given by

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

- where $D(u, v) = [u^2 + v^2]^{1/2}$
- For this smooth transition filter, a cutoff frequency locus is chosen such that $D(u, v)$ is a certain percentage of its maximum
- Designed such that at $D(u, v) = D_0$, $H(u, v) = 0.50$ (50% of its maximum value)



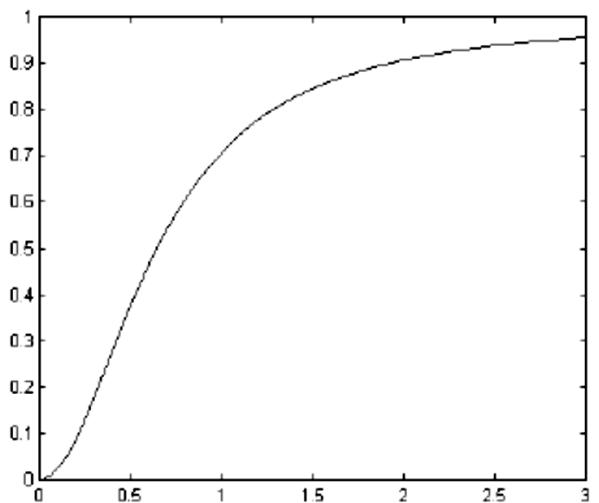
Butterworth highpass filter

- Another transfer function of a Butterworth highpass filter (BHPF) of order n with cutoff frequency D_0 is given by

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

- Designed such that at $D(u, v) = D_0$

$$H(u, v) = \frac{1}{\sqrt{2}}$$



Gaussian highpass filter

LESSON#16 IMAGE PROCESSING & COMPUTER VISION

- The transfer function of a Gaussian highpass filter (GHPF) is given by

$$H(u, v) = 1 - e^{-D^2(u,v)/2\sigma^2}$$

- Here, σ is as in the Gaussian lowpass case
- Let $\sigma = D_0$, then

$$H(u, v) = 1 - e^{-D^2(u,v)/2D_0^2}$$

- where D_0 is the cutoff frequency

Highpass filter functions

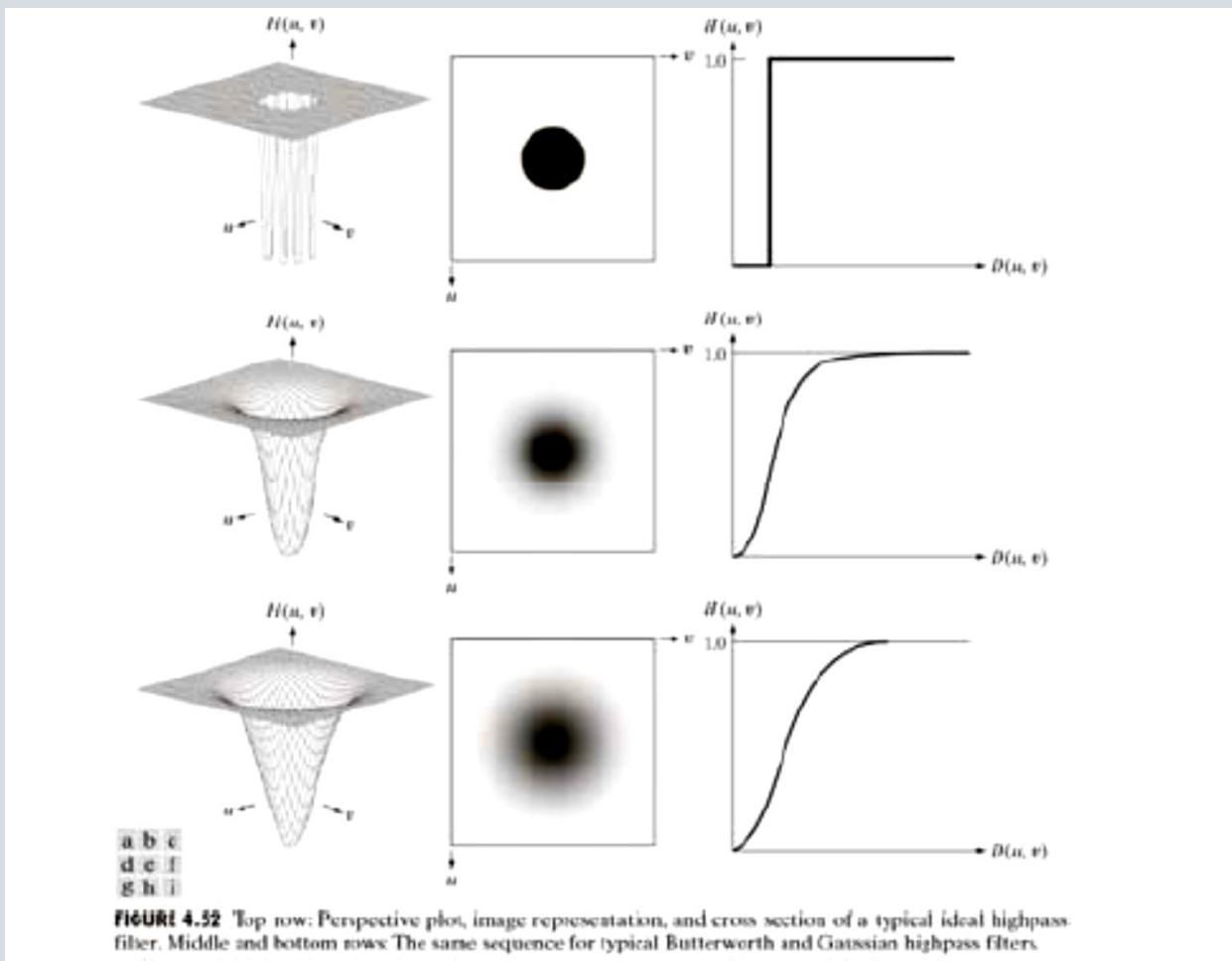


FIGURE 4.52 Top row: Perspective plot, image representation, and cross section of a typical ideal highpass filter. Middle and bottom rows: The same sequence for typical Butterworth and Gaussian highpass filters.

The goal of this lab

It is the knowledge the **Filter in frequency domain** with images in MATLAB.

Procedure

The program

fft2 → 2-D fast Fourier transform

Y = fft2(X) returns the two-dimensional Fourier transform of a matrix using a fast Fourier transform algorithm, which is equivalent to computing `fft(fft(X).')'`. If X is a multidimensional array, then fft2 takes the 2-D transform of each dimension higher than 2. The output Y is the same size as X.

fftshift → Shift zero-frequency component to center of spectrum.

ifft2 → 2-D inverse fast Fourier transform.

Fourier Transformation without Mask filter

```
%Fourier Transformation without Mask filter(true)
clc
clear all
close all
a1=imread('peppers.png');
a=rgb2gray(a1);
figure(1)
imshow(a);
title('original image')
%Fourier Transformation
af=fft2(a);
mat2gray(log(1+abs(af)));
h=ones(3,3)/9;
h(size(a,1),size(a,2))=0; %pad filter to be same size
of image
H=fft2(h);
log(abs(fftshift(H)));
%Convolution product
G=af.*H;
g=real(ifft2(G));
figure(2)
```

LESSON#16 IMAGE PROCESSING & COMPUTER VISION

```
imshow(g, [])
```

The Result

original image

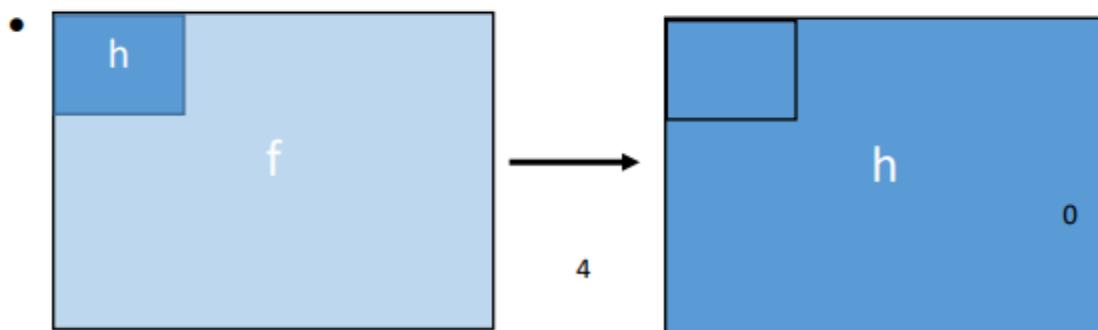


LESSON#16 IMAGE PROCESSING & COMPUTER VISION

NOTE

Need to pad filter to be same size as image

- To do this by setting the point in the lower right corner
 $h(\text{size}(a1,1), \text{size}(a1,2))=0;$
- Where $\text{size}(a1)$ is the size (#row,#cols) of the image.
- MATLAB expands the filter and fills new values to zero.



Bandwidth Lowpass filter in frequency Domain

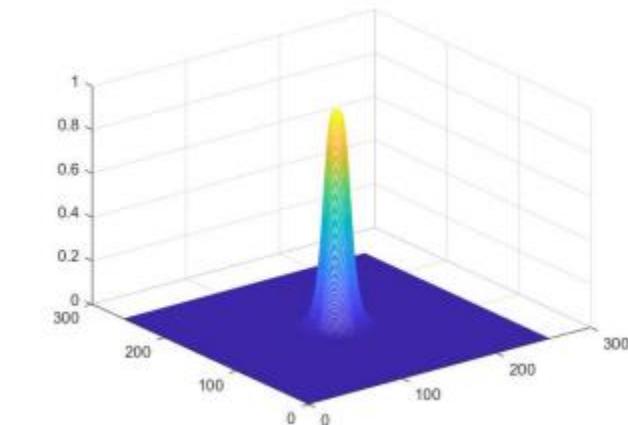
```
%Bandwidth Lowpass filter in frequency Domain
function [g] = blpf(f,order,cutoff)
%UNTITLED2 Summary of this function goes here
% Detailed explanation goes here
F=fft2(f);
F=fftshift(F);
[uMax, vMax]=size(F);
for u=1:uMax
    for v=1:vMax
        H(u,v)=1/(1+(sqrt(2)-1)*(sqrt(((uMax/2-
(u_1)).^2+(vMax/2-(v-1)).^2))/cutoff).^(2*order));
    end
end
figure
mesh(H)
G=H.*F;
G=ifft2(G);
g=sqrt(real(G).^2+imag(G).^2);
figure
imshow(G, [])
```

LESSON#16 IMAGE PROCESSING & COMPUTER VISION

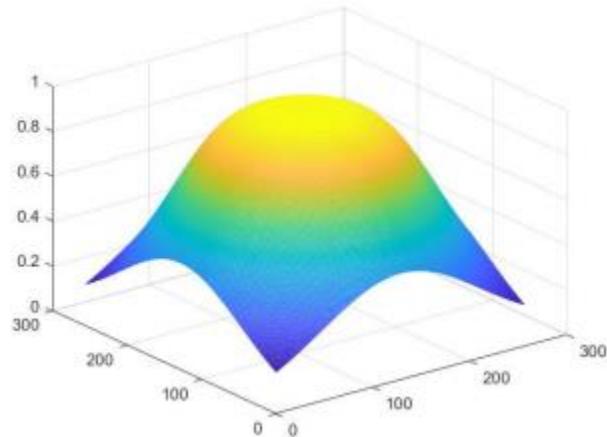
```
end  
%to Call function in command window:  
% blpf(imread('cameraman.tif'),2,10)  
%when the order increase the image sharpness
```

The Result

```
blpf(imread('cameraman.tif'),2,10)
```



```
blpf(imread('cameraman.tif'),2,100)
```



Report

- 1- Open the new script in MATLAB program and make the frequency domain filter by using Fourier transform. Use the following filters by using the specific equation of each filter:
 - a- Gaussian lowpass filter

LESSON#16 IMAGE PROCESSING & COMPUTER VISION

- b- Butterworth highpass filter
- c- Gaussian highpass filter
- d- Butterworth Bandreject filters

*(Display the filter image and mesh display for filter

Lesson#17 Image Processing & COMPUTER VISION

**Image Restoration – Deblurring Image by using
Bilateral Filter – Inverse Filter – Winner Filter**

Eng. Elaf A.Saeed
Email: elafe1888@gmail.com

LAB Content

Linear Filtering with Gaussian Blur (GB)

Nonlinear Filtering with Bilateral Filter (BF)

Inverse Filter

Pseudo-Inverse Filtering

Winner Filter

Table of Functions

| Function Name | Description |
|------------------|--|
| deconvwnr | <p>Deblur image using Wiener filter.</p> <p>J = deconvwnr(I,psf,nsr) deconvolves image I using the Wiener filter algorithm, returning deblurred image J. psf is the point-spread function (PSF) with which I was convolved. nsr is the noise-to-signal power ratio of the additive noise. The algorithm is optimal in a sense of least mean square error between the estimated and the true images.</p> |

Theory

From Gaussian Blur to Bilateral Filter

Linear Filtering with Gaussian Blur (GB)

Convolution by a positive kernel is the basic operation in linear image filtering. It amounts to estimate at each position a local average of intensities and corresponds to low-pass filtering. One defines the Gaussian blur (GB) filtered image by:

$$GB[I]_p = \sum_{q \in S} G_\sigma(\|p - q\|) I_q,$$

Where $G_\sigma(x)$ denotes the two-dimensional Gaussian kernel

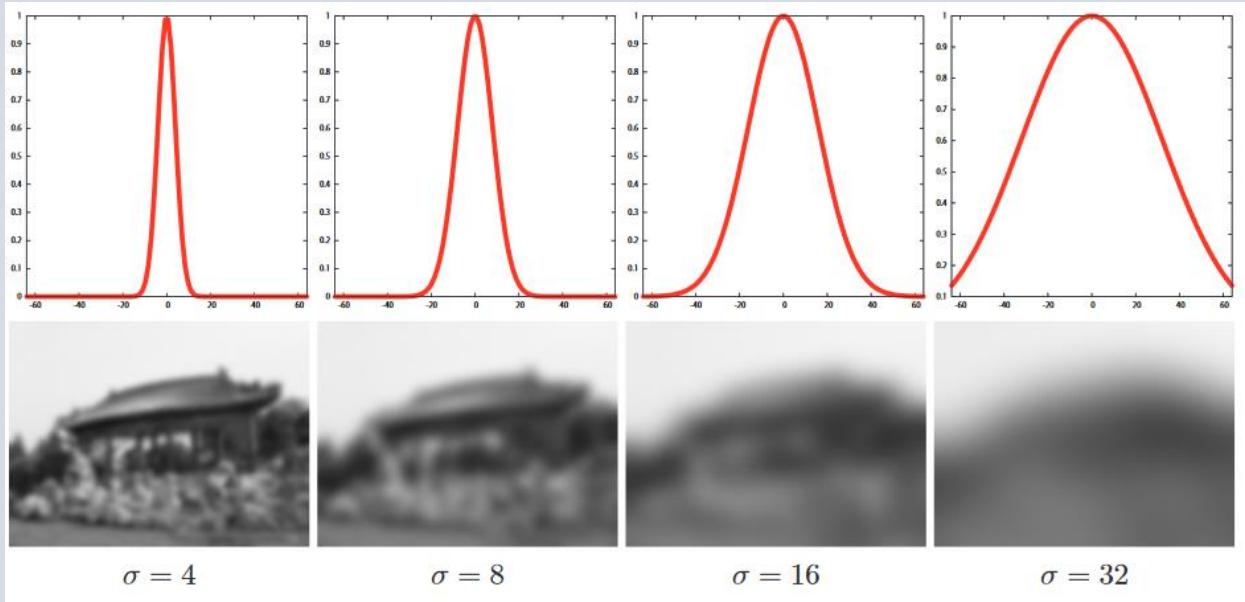
$$G_\sigma(x) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2}{2\sigma^2}\right).$$

So, Gaussian filtering is a weighted average of the intensity of the adjacent positions with a weight decreasing with the spatial distance to the center position. This distance is defined by $G_\sigma(\|p - q\|)$, where σ is a parameter defining the extension of the neighborhood.

- ❖ *Since the filter action is independent of the image content, only the distances between positions matter.*

Remark: An advantage of linear filters is that they can be implemented efficiently using various techniques such as fast Fourier transform. Unfortunately, these acceleration techniques do not apply to nonlinear filters such as the bilateral filter. Nonetheless, fast numerical schemes have been developed specifically for the bilateral filter.

LESSON#17 IMAGE PROCESSING & COMPUTER VISION



Example of Gaussian linear filtering with different σ . Top row show the profile of a 1D Gaussian kernel and bottom row the result obtained by the corresponding 2D Gaussian blur filtering. Edges are lost with high values of σ since more averaging is performed.

Nonlinear Filtering with Bilateral Filter (BF)

Similarly, to the Gaussian convolution, the bilateral filter is also defined as a weighted average of pixels. The difference is that the bilateral filter takes into account the variation of intensities to preserve edges. The rationale of bilateral filtering is that two pixels are close to each other not only if they occupy nearby spatial locations but also if they have some similarity in the photometric range.

The bilateral filter, denoted by $BF[\cdot]$, is defined by:

$$BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(I_p - I_q) I_q$$

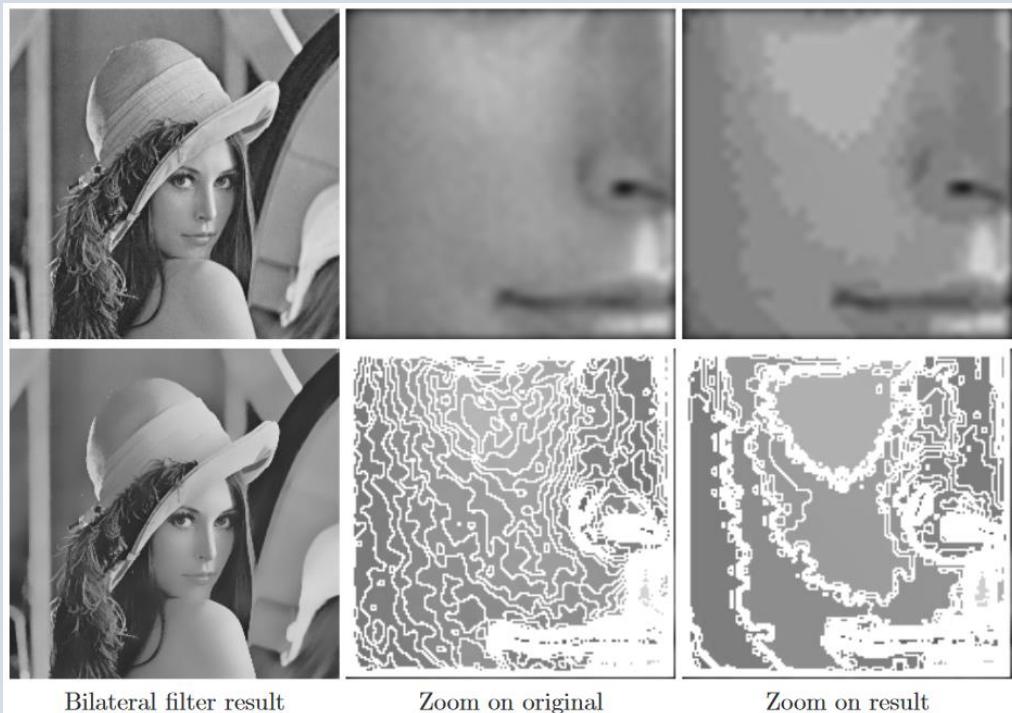
Where W_p is a normalization factor:

$$W_p = \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(I_p - I_q)$$

Parameters σ_s and σ_r will measure the amount of filtering for the image I . The equation is a normalized weighted average where G_{σ_s} is a spatial Gaussian that

LESSON#17 IMAGE PROCESSING & COMPUTER VISION

decreases the influence of distant pixels, for a range Gaussian that decreases the influence of pixels' q with an intensity value different from I_p . Note that the term **rang** qualifies quantities related to pixel values, by opposition to **space** which refers to pixel location. Figure 4 shows a sample output of the bilateral filter and Figure 5 illustrates how the weights are computed on a simple example.



Example of result obtained with the Bilateral filter. First columns shown the original image (top) and the smoothed image (bottom). Second column is a zoom on one part of the image (top) with corresponding level lines below. Third column is the same illustration but for the result.

Parameters The bilateral filter is controlled by two parameters: σ_s and σ_r .

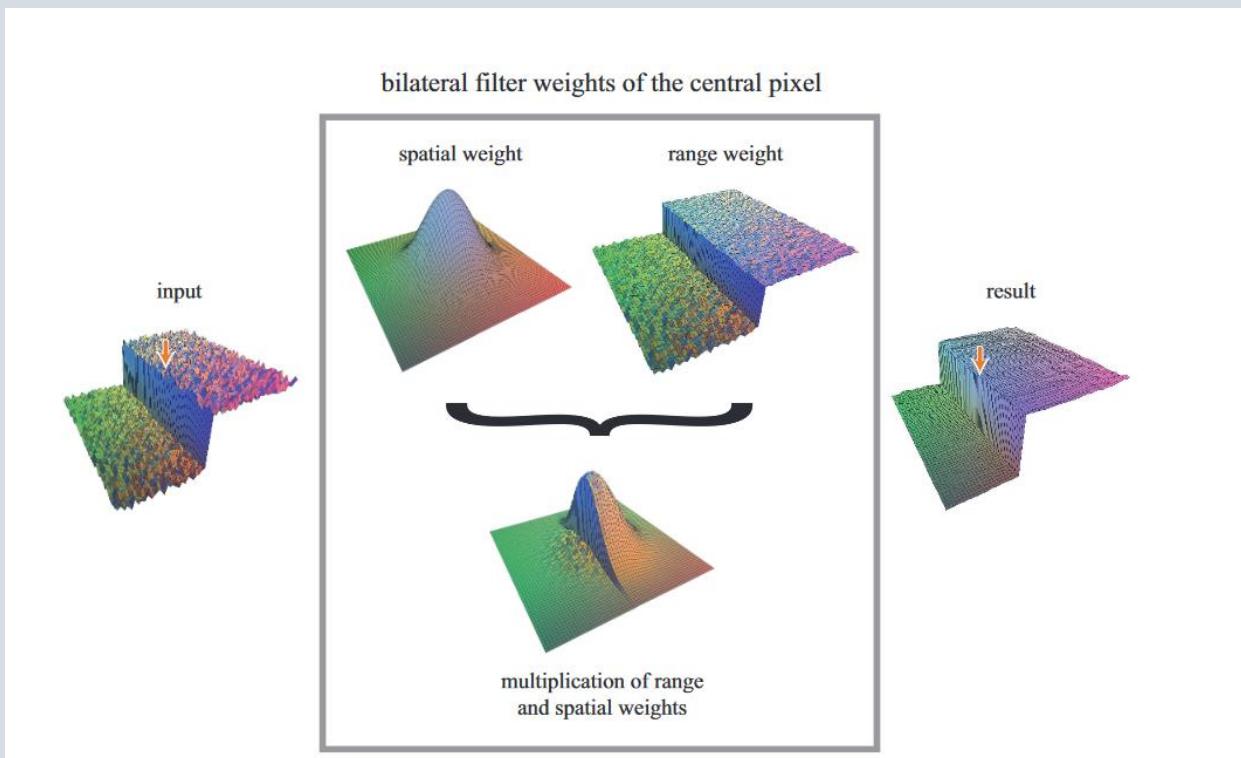
- As the range parameter σ_r increases, the bilateral filter becomes closer to Gaussian blur because the range Gaussian is flatter i.e., almost a constant over the intensity interval covered by the image.
- Increasing the spatial parameter σ_s smooths larger features.

An important characteristic of bilateral filtering is that the weights are multiplied, which implies that as soon as one of the weight is close to 0, no smoothing occurs.

LESSON#17 IMAGE PROCESSING & COMPUTER VISION

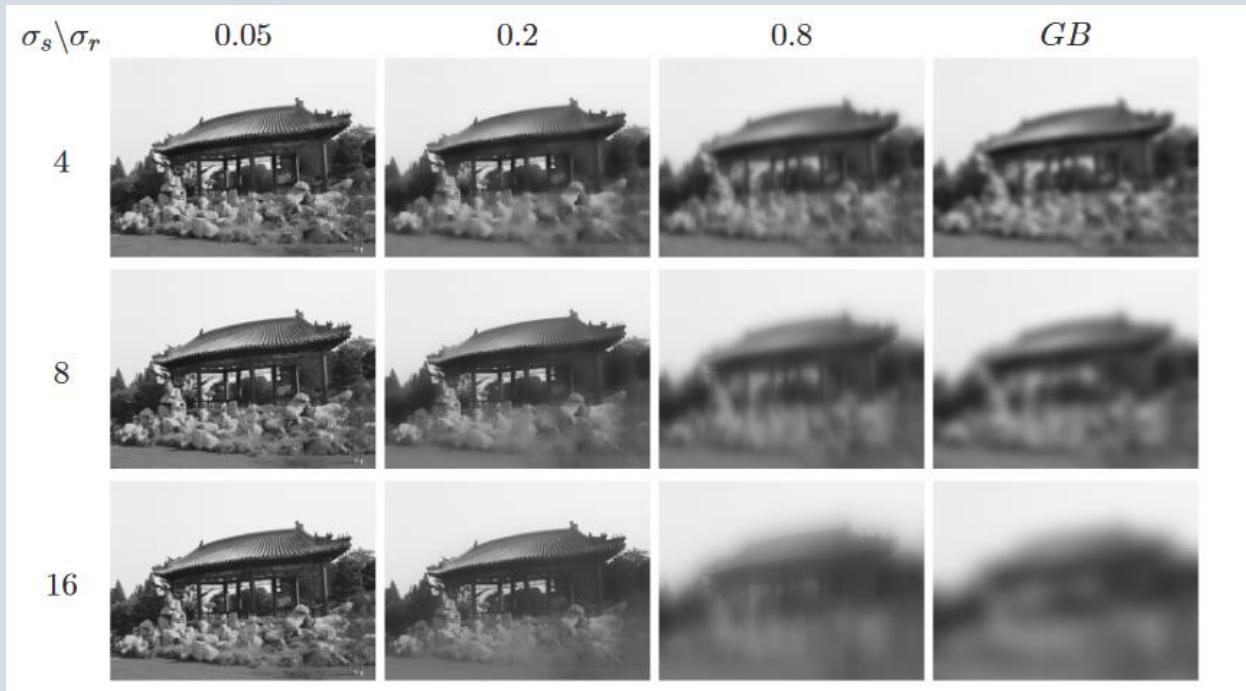
As an example, a large spatial Gaussian coupled with narrow range Gaussian achieves a limited smoothing although the filter has large spatial extent. The range weight enforces a strict preservation of the contours.

Iterations The bilateral filter can be iterated. This leads to results that are almost piece wise constant as shown on Figure below. This type of effect is desirable for applications such as stylization, while computational photography techniques.



The bilateral filter smooths an input image while preserving its edges. Each pixel is replaced by a weighted average of its neighbors. Each neighbor is weighted by a spatial component that penalizes distant pixels and range component that penalizes pixels with a different intensity. The combination of both components ensures that only nearby similar pixels contribute to the final result. The weights are represented for the central pixel (under the arrow).

LESSON#17 IMAGE PROCESSING & COMPUTER VISION



Effects of the range and spatial parameters, and comparison with Gaussian blur. As soon as one of the weight is close to 0, no smoothing occurs. As a consequence, increasing the spatial sigma has no consequence on an edge as long as the range sigma is less than its amplitude. For instance, the contour of the roof is unaffected for small range values, independently of the spatial setting. The range values are given considering that the intensities span [0,1].

Inverse Filtering

In image restoration the goal is to recover an image that has been corrupted or degraded. The more information we have of the degradation process, the better off we are. This is known as a priori knowledge. There are several techniques in image restoration, some use frequency domain concepts, others attempt to model the degradation and apply the inverse process. The modeling approach requires determining a criterion of "goodness" that will yield an "optimal" solution.

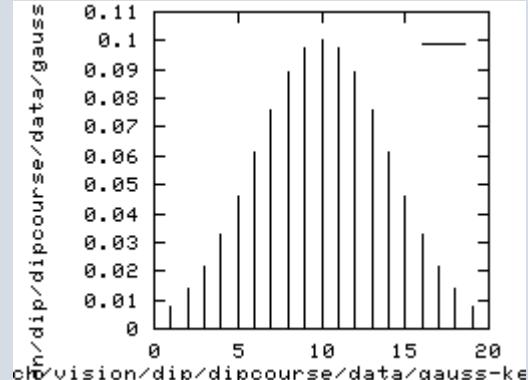
In this experiment we demonstrate the simplest concept in image restoration when the actual spatial convolution filter used to degrade the image is known.

LESSON#17 IMAGE PROCESSING & COMPUTER VISION

Shown below is the blurred image that is the result of convolving a Gaussian filter in the width direction with the original image. This effect is similar to the one



a)



b)

Let f be the original image, h the blurring kernel, and g the blurred image. The idea in inverse filtering is to recover the original image from the blurred image.

From the convolution theorem, the DFT of the blurred image is the product of the DFT of the original image and the DFT of the blurring kernel. Thus, dividing the DFT of the blurred image by the DFT of the kernel, we can recover the original image. Then the inverse frequency filter, $R(u)$, to be applied is $1/H(u)$. We are assuming no noise is present in the system. Observe the difficulties we encounter when $H(u)$ is very small or equal to zero.

$$g(x) = f(x) * h(x) \Leftrightarrow G(u) = F(u)H(u)$$

$$F(u) = \frac{1}{H(u)}G(u)$$

$$F(u) = R(u)G(u)$$

$$R(u) = \frac{1}{H(u)}$$

The formulas correspond to the 1D case and can be easily extended to the 2D domain.

$$P(u,v) = H(u,v) Q(u,v),$$

LESSON#17 IMAGE PROCESSING & COMPUTER VISION

where

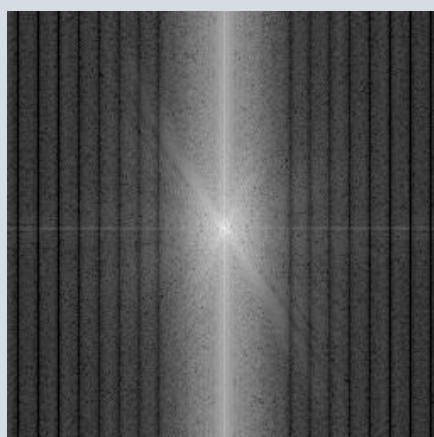
P(u,v) is the degraded image,

H(u,v) is the degradation transfer function, and

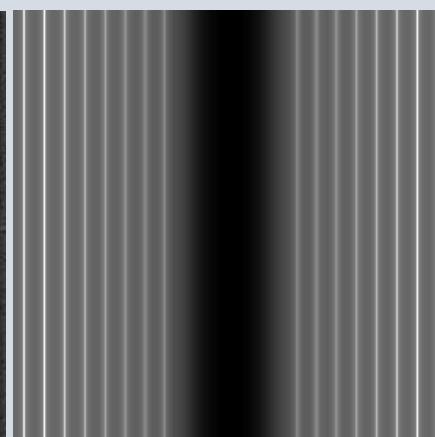
Q(u,v) is the original image.

The inverse filtering process is then

$$Q(u,v) = P(u,v) / H(u,v).$$

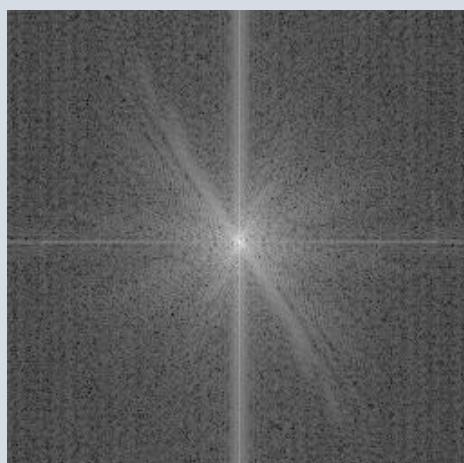


a)



b)

a)spectrum of the blurred image; b)spectrum of $R(u) = 1/H(u)$



a)



b)

a)restored DFT; b)inverse DFT of restored DFT

Pseudo-Inverse Filtering

LESSON#17 IMAGE PROCESSING & COMPUTER VISION

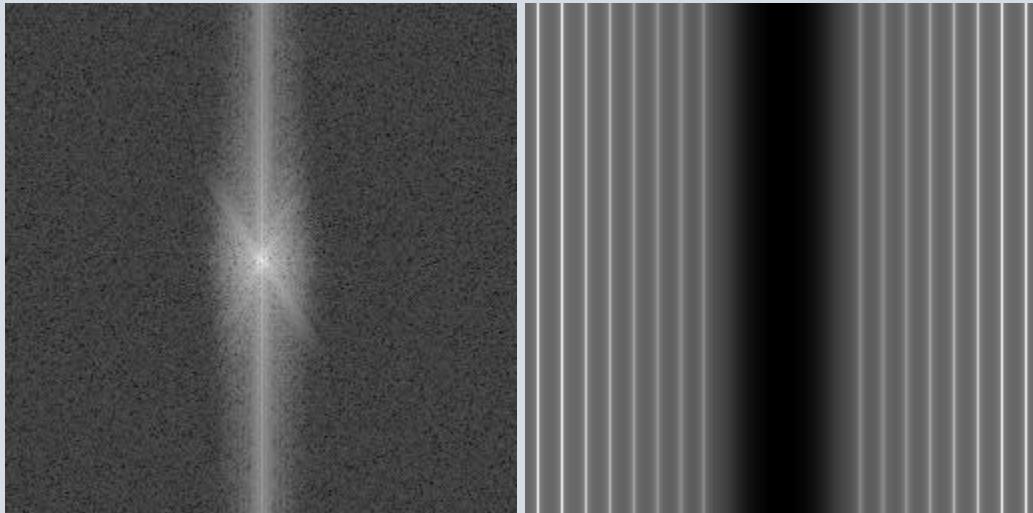
In the previous Inverse Filtering experiment, we assumed an ideal situation that rarely exists. In this lesson we will use a more practical approach to image restoration.

In the Inverse Filtering lesson, the blurred image pixels are floating point. If that image is converted to unsigned byte data type, the consequence is similar to subtracting a constant distribution of noise with values from 0 to 1. Although these differences do not cause any problem with our visual perception of the image, it plays an important role in image restoration by the inverse filtering procedure.



a)

b)



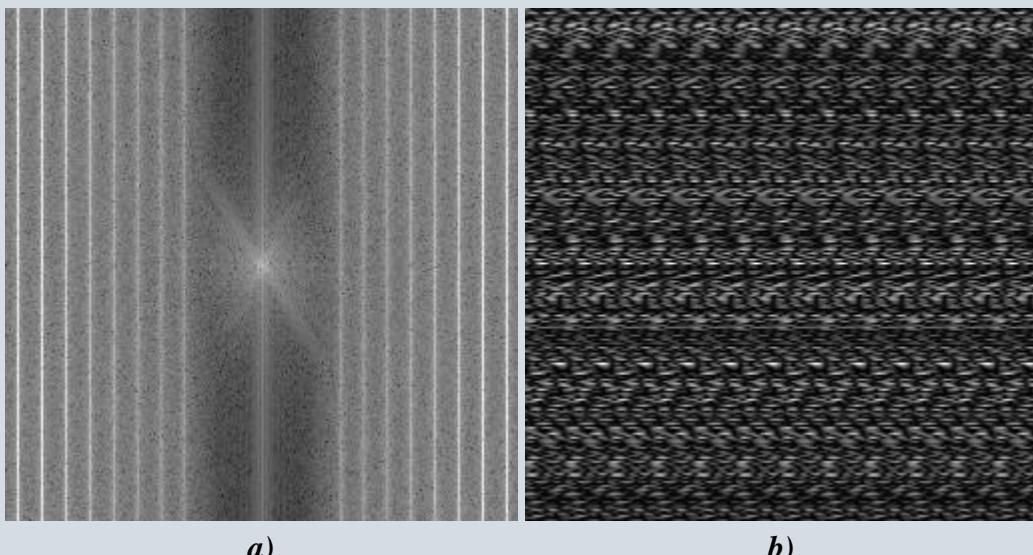
LESSON#17 IMAGE PROCESSING & COMPUTER VISION

a)

b)

a)spectrum of the blurred byte image; b)spectrum of the inverse filter

Notice that the DFT of the blurred byte image does not show any vertical stripes of low values. This is because the DFT of the noise was added to the spectrum. When using the inverse filtering technique, the problem encountered is that in the region of vertical high value stripes the noise is amplified and corrupts the restored image.



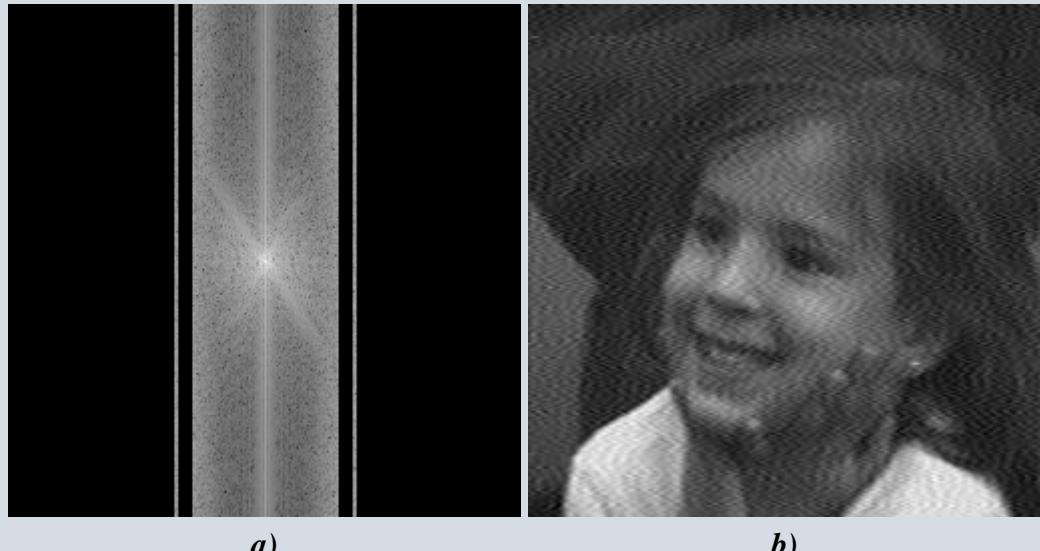
a)

b)

a) division of blurred byte DFT by the kernel DFT; b) restored image

A practical solution to this problem is to define a threshold value in the magnitude of inverse kernel DFT where the division will take place. This can be depicted in the results shown below.

$$R(u) = \begin{cases} \frac{1}{H(u)} & \text{if } |H(u)| > T \\ 0 & \text{if } |H(u)| \leq T \end{cases}$$



a) pseudo-inverse division of the blurred byte DFT by the kernel DFT; b) restored image

The Wiener filter

The Wiener filter is the MSE-optimal stationary linear filter for images degraded by additive noise and blurring. Calculation of the Wiener filter requires the assumption that the signal and noise processes are second-order stationary (in the random process sense). gif for this description, only noise processes with zero mean will be considered (this is without loss of generality).

Wiener filters are usually applied in the frequency domain. Given a degraded image $x(n,m)$, one takes the Discrete Fourier Transform (DFT) to obtain $X(u,v)$. The original image spectrum is estimated by taking the product of $X(u,v)$ with the Wiener filter $G(u,v)$:

$$\hat{S}(u, v) = G(u, v)X(u, v)$$

The inverse DFT is then used to obtain the image estimate from its spectrum. The Wiener filter is defined in terms of these spectra:

LESSON#17 IMAGE PROCESSING & COMPUTER VISION

- $H(u, v)$ Fourier transform of the point-spread function (PSF)
- $P_s(u, v)$ Power spectrum of the signal process, obtained by taking the Fourier transform of the signal autocorrelation
- $P_n(u, v)$ Power spectrum of the noise process, obtained by taking the Fourier transform of the noise autocorrelation

The Wiener filter is:

$$G(u, v) = \frac{H^*(u, v)P_s(u, v)}{|H(u, v)|^2P_s(u, v) + P_n(u, v)}$$

Dividing through by P_s makes its behavior easier to explain:

$$G(u, v) = \frac{H^*(u, v)}{|H(u, v)|^2 + \frac{P_n(u, v)}{P_s(u, v)}}$$

The term $\frac{P_n}{P_s}$ can be interpreted as the reciprocal of the signal-to-noise ratio.

Where the signal is very strong relative to the noise, $\frac{P_n}{P_s} \approx 0$ and the Wiener filter becomes $H^{-1}(u, v)$ - the inverse filter for the PSF. Where the signal is very weak $P_n/P_s \rightarrow \infty$, and $G(u, v) \rightarrow 0$.

For the case of additive white noise and no blurring, the Wiener filter simplifies to:

$$G(u, v) = \frac{P_s(u, v)}{P_s(u, v) + \sigma_n^2}$$

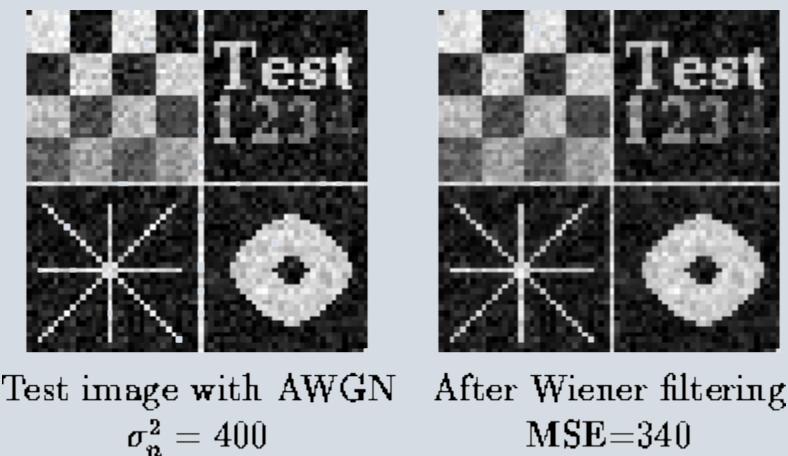
Where σ_n^2 is the noise variance.

Wiener filters are unable to reconstruct frequency components which have been degraded by noise. They can only suppress them. Also, Wiener filters are unable to restore components for which $H(u, v)=0$. This means they are unable to undo blurring caused by band limiting of $H(u, v)$. Such band limiting occurs in any real-world imaging system.

LESSON#17 IMAGE PROCESSING & COMPUTER VISION

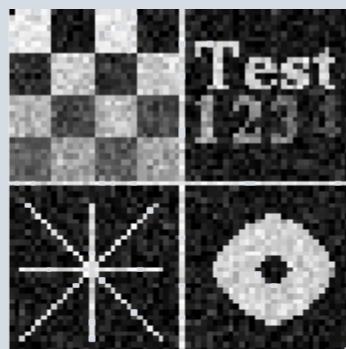
Obtaining p_s can be problematic. One can assume that p_s has a parametric shape, for example exponential or Gaussian. Alternately, p_s can be estimated using images representative of the class of images being filtered.

For Wiener results presented in this thesis p_s , was calculated from image to be filtered: p_s was assumed to be radially symmetric, i.e. $p_s(u, v) = p_s(p)$ and was estimated by averaging over 30 radial frequency bands. Linear interpolation was used to give p_s a smooth shape.

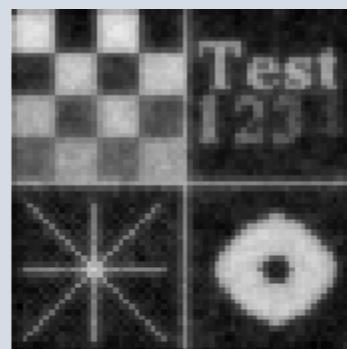


shows a Wiener filter result. The small test image has very strong high-frequency components, so the Wiener filter leaves lots of residual noise. If the test image, which is 64x64, is centered in a 256x256 empty image, the relative power of those high-frequency components is diminished by the large amounts of empty space. The Wiener filter then elects to attenuate high-frequency components to reduce noise in the empty regions. This results in blurring over the small 64x64 subimage. Although the MSE over the 256x256 image is quite small, the MSE over the 64x64 test region increases from 400 to 1232. This illustrates an important point about using MSE as a criteria for global filtering: regions are given priority for restoration according to how large they are, rather than their visual importance.

LESSON#17 IMAGE PROCESSING & COMPUTER VISION



Test image with AWGN
 $\sigma_n^2 = 400$
but centered inside a
256x256 empty image



After Wiener filtering
MSE=121 (256x256 image)
MSE=1232 (portion shown)

Wiener filters are comparatively slow to apply, since they require working in the frequency domain. To speed up filtering, one can take the inverse FFT of the Wiener filter $G(u,v)$ to obtain an impulse response $g(n,m)$. This impulse response can be truncated spatially to produce a convolution mask. The spatially truncated Wiener filter is inferior to the frequency domain version, but may be much faster.

Procedure

1. save the image in the current folder.
2. Open new script and write the following:

Bilateral Filter

```
clc
clear all
close all
%read the image in the workspace and display it
a=imread('peppers.png');
imshow(a)
b=size(a);
if size(b,2)==3
    a1=rgb2gray(a);
end
figure(2)
```

LESSON#17 IMAGE PROCESSING & COMPUTER VISION

```
imshow(a1)

%Add Noise
a3=imnoise(a1,'gaussian',0,0.003998); %0=>mean 0.003998 =>variance
a2=double(a3);
figure(3)
imshow(a3)

%initialize the parameters
n=11;
n1=ceil(n/2);
vars=50;
varr=25;
c=0;
c1=0;

%Bilateral filter loop
for i=n1:b(1)-n1 %b(1) => rows
    for j=n1:b(2)-n1 %b(1) => cols
        for k=1:n
            for m=1:n
                c = c+gs(sqrt((-n1+k)^2 +(-n1+m)^2), 0, vars)*...
                    gs(a2(i-n1+k, j-n1+m), a2(i,j), varr)* a2(i-n1+k, j-n1+m);

                c1=c1+gs(sqrt((-n1+k)^2 +(-n1+m)^2),0,vars)*...
                    gs(a2(i-n1+k, j-n1+m), a2(i,j), varr);
            end
        end
        d(i-n1+1,j-n1+m)=c/c1;
        c=0;
        c1=0;
    end
end

d1 = uint8(d);
% plotting the image
figure(4)
subplot(1,2,1)
imshow((a3));
```

LESSON#17 IMAGE PROCESSING & COMPUTER VISION

```
title('Noisy image')
%figure(4)
subplot(1,2,2)
imshow(d1);
title( 'bilateral filter output image')
```

The gs function file:

```
function out = gs(p,q,var)
    out=1/((2*pi)*var^2)*exp(-(p-q)...
        (p-q)'/(2*var^2));
return
```

The Result

In figure below that shows the image when we can read the index value of each.

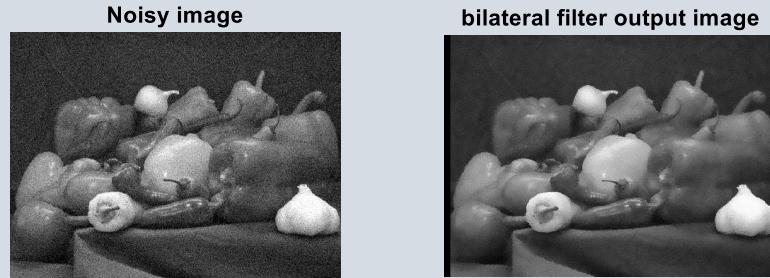


Original

LESSON#17 IMAGE PROCESSING & COMPUTER VISION



LESSON#17 IMAGE PROCESSING & COMPUTER VISION



Inverse Filter

1. save the image in the current folder.
2. Open new script and write the following:

```
clc
clear all
close all
%read the image
I=imread('cameraman.tif');
I1=im2double(I);
imshow(I)
[r,c]=size(I);

%Blurring image
h=fspecial('gaussian',15,2);
blurred=imfilter(I1,h,'circular');
noise=0.001*randn(size(I));
g=blurred+noise;
figure(2),imshow(g)

%Blurred image frequency domain
G=fftshift(fft2(g));
figure(3);
imshow(log(abs(G)));

%Frequency domain filter creation
h=ifftshift(fspecial('gaussian',size(I),2));
figure(4),mesh(h)
H=fftshift(fft2(h));
figure(5),imshow(abs(H),[]);
```

LESSON#17 IMAGE PROCESSING & COMPUTER VISION

```
F=zeros(size(I));
R=40;

for u=1:size(I,2)
    for v=1:size(I,1)
        %offset to the original point (0,0)
        du=u-size(I,2)/2;
        dv=v-size(I,2)/2;
        if du^2 + dv^2<=R^2
            F(v,u)=(G(v,u))/(H(v,u));
        end
    end
end
figure(6),imshow(log(abs(F)),[])
frestored=real(ifft2(ifftshift(F)));
figure(7),imshow(frestored)
```

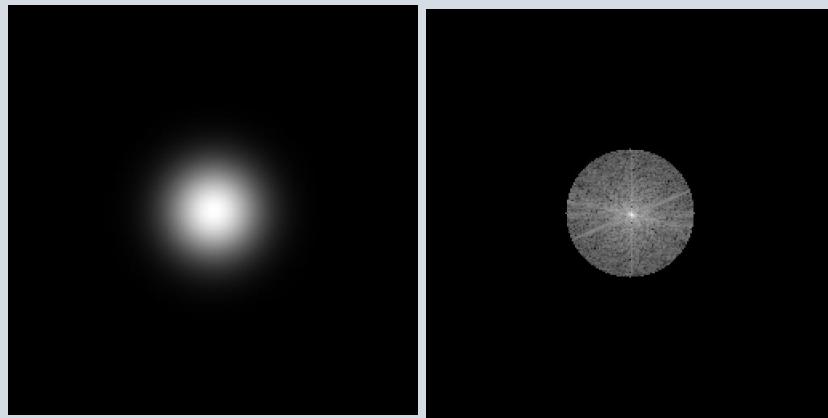
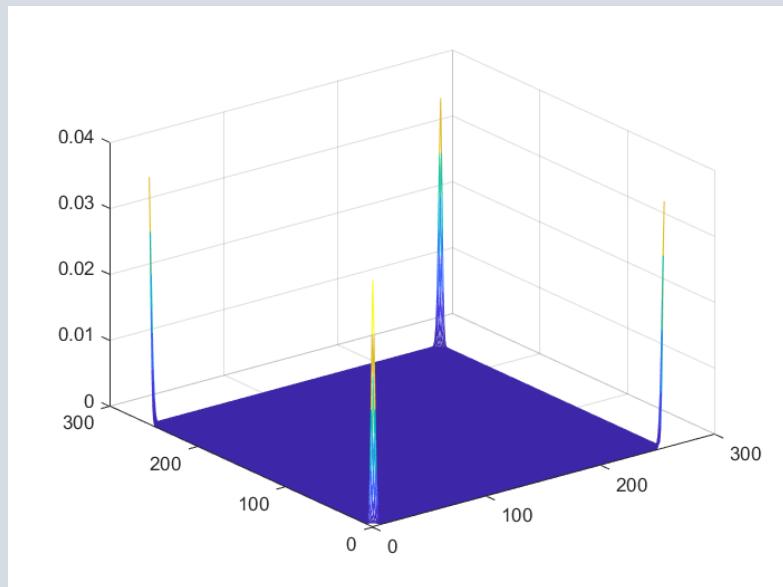
The Result

In figure below that shows the image when we can read the index value of each.



Original

LESSON#17 IMAGE PROCESSING & COMPUTER VISION





Winner Filter

```
clc
clear all
close all
%1- read the image into the workspace and display it
I=im2double(imread('cameraman.tif'));
imshow(I);
title('Original Image(courtesy of MIT)');
% 2- Simulate a motion blur
len=21;%Linear motion of camera, specified as a numeric
scalar, measured in pixels.default (9)
theta=11; %Angle of camera motion. default (0)
PSF=fspecial('motion',len,theta);
blurred=imfilter(I,PSF,'conv','circular');
figure, imshow(blurred)

% 3- Simulate Additive Noise
noise_mean=0; %Mean of Gaussian noise. default (0)
noise_var=0.0001; %Variance of Gaussian noise. default
(0.01)
blurred_noisy=imnoise(blurred,'gaussian',noise_mean,noi
se_var);
figure, imshow(blurred_noisy)
title('Simulate Blur and Noise')

%Try restoration assuming
```

LESSON#17 IMAGE PROCESSING & COMPUTER VISION

```
estimate_nsr=0;
wnr2=deconvwnr(blurred_noisy,PSF,estimate_nsr);
figure, imshow(wnr2)
title('Restoration of Blurred, Noisy Image Using
NSR=0')

%Try restoration using a better estimate of the noise-
to-signal-power ratio
estimate_nsr=noise_var/var(I(:));
wnr3=deconvwnr(blurred_noisy,PSF,estimate_nsr);
figure, imshow(wnr3)
title('Restoration of Blurred, Noisy Image Using
Estimate NSR')
```

The Result

In figure below that shows the image when we can read the index value of each.



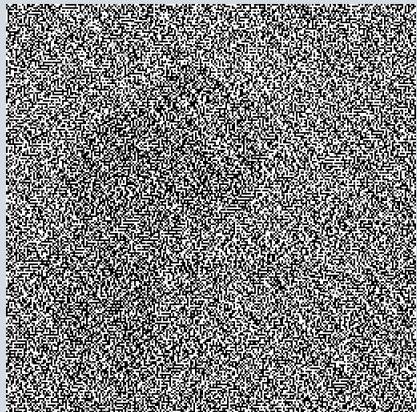
Original

Simulate Blur and Noise



LESSON#17 IMAGE PROCESSING & COMPUTER VISION

Restoration of Blurred, Noisy Image Using NSR=0



Restoration of Blurred, Noisy Image Using Estimate NSR



Report

1. Explain Why we use bilateral filter? With an illustrative example given by using MATLAB.
2. What happen when the σ_s & σ_r increases? explain it with an example by using MATLAB.
3. what happen if the value of H is very small and equal to zero? Write the MATLAB program to solve this problem and what the filter is used.
4. what the advantage of winner filter?

LESSON#17 IMAGE PROCESSING & COMPUTER VISION

5. Why we use the winner filter?

Lesson#18 Image Processing & COMPUTER VISION

Image Segmentation - Threshold

Eng. Elaf A.Saeed
Email: elafe1888@gmail.com

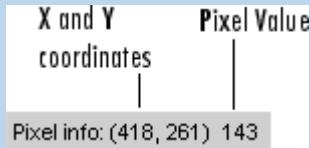
LAB Content

Threshold without built-in function

Threshold with built-in Threshold function

Adaptive Threshold

Table of Functions

| Function Name | Description |
|--|--|
| impixelinfo | <p>Use the <code>impixelinfo</code> function to create a Pixel Information tool. The Pixel Information tool displays information about the pixel in an image that the pointer is positioned over. If the figure contains multiple images, the tool displays pixel information for all the images. For more information about the tool</p>  |
| graythresh <code>T = graythresh(I)</code> <code>[T,EM] = graythresh(I)</code> | <p>Global image threshold using Otsu's method.</p> <p><code>T = graythresh(I)</code> computes a global threshold T from grayscale image I, using Otsu's method [1]. Otsu's method chooses a threshold that minimizes the intraclass variance of the thresholded black and white pixels. The global threshold T can be used with <code>imbinarize</code> to convert a grayscale image to a binary image.</p> |

Theory

What Is Image Segmentation?

Image segmentation is a commonly used technique in digital image processing and analysis to partition an image into multiple parts or regions, often based on the characteristics of the pixels in the image. Image segmentation could involve separating foreground from background, or clustering regions of pixels based on similarities in color or shape. For example, a common application of image segmentation in medical imaging is to detect and label pixels in an image or voxels of a 3D volume that represent a tumor in a patient's brain or other organs.

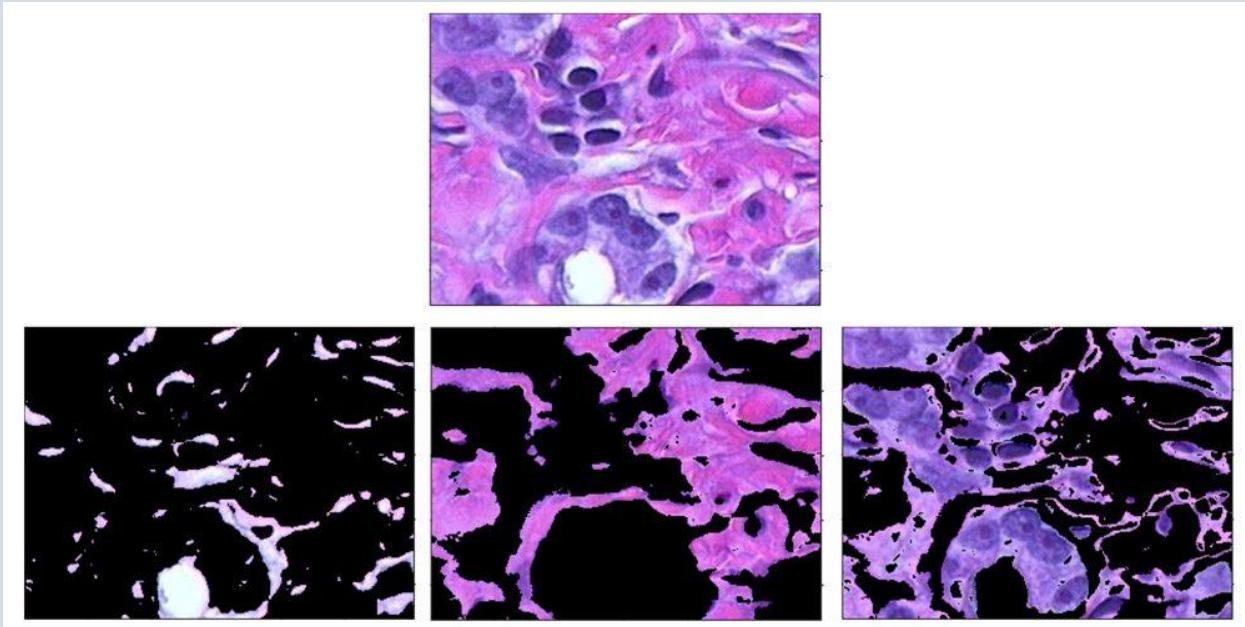
Why Image Segmentation Matters

Several algorithms and techniques for image segmentation have been developed over the years using domain-specific knowledge to effectively solve segmentation problems in that specific application area. These applications include medical imaging, automated driving, video surveillance, and machine vision.

Medical Imaging

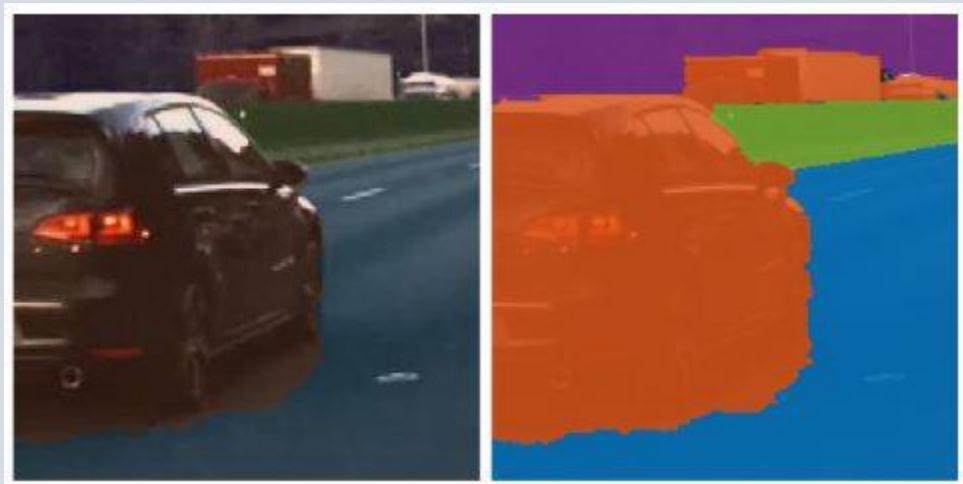
During medical diagnosis for cancer, pathologists stain body tissue with hematoxylin and eosin (H&E) to distinguish between tissue types. They then use an image segmentation technique called clustering to identify those tissue types in their images. Clustering is a method to separate groups of objects in a scene. The K-means clustering algorithm finds separations such that objects within each cluster are as close to each other as possible, and as far from other objects in other clusters as possible.

LESSON#18 IMAGE PROCESSING & COMPUTER VISION



Autonomous Driving

When designing perception for autonomous vehicles, such as self-driving cars, semantic segmentation is popularly used to help the system identify and locate vehicles and other objects on the road.



How Image Segmentation Works

Image segmentation involves converting an image into a collection of regions of pixels that are represented by a mask or a labeled image. By dividing an image into segments, you can process only the important segments of the image instead of processing the entire image.

LESSON#18 IMAGE PROCESSING & COMPUTER VISION

A common technique is to look for abrupt discontinuities in pixel values, which typically indicate edges that define a region.

What Is Image Filtering in the Spatial Domain?

Filtering is a technique for modifying or enhancing an image. For example, you can filter an image to emphasize certain features or remove other features. Image processing operations implemented with filtering include smoothing, sharpening, and edge enhancement.

Filtering is a neighborhood operation, in which the value of any given pixel in the output image is determined by applying some algorithm to the values of the pixels in the neighborhood of the corresponding input pixel. A pixel's neighborhood is a set of pixels, defined by their locations relative to that pixel. (See Neighborhood or Block Processing: An Overview for a general discussion of neighborhood operations.) Linear filtering is filtering in which the value of an output pixel is a linear combination of the values of the pixels in the input pixel's neighborhood.

Convolution

Linear filtering of an image is accomplished through an operation called convolution. Convolution is a neighborhood operation in which each output pixel is the weighted sum of neighboring input pixels. The matrix of weights is called the convolution kernel, also known as the filter. A convolution kernel is a correlation kernel that has been rotated 180 degrees.

For example, suppose the image is

$$A = \begin{bmatrix} 17 & 24 & 1 & 8 & 15 \\ 23 & 5 & 7 & 14 & 16 \\ 4 & 6 & 13 & 20 & 22 \\ 10 & 12 & 19 & 21 & 3 \\ 18 & 22 & 25 & 23 & 9 \end{bmatrix}$$

What Is Image Filtering in the Spatial Domain?

Filtering is a technique for modifying or enhancing an image. For example, you can filter an image to emphasize certain features or remove other features. Image processing operations implemented with filtering include smoothing, sharpening, and edge enhancement.

Filtering is a neighborhood operation, in which the value of any given pixel in the output image is determined by applying some algorithm to the values of the pixels in the neighborhood of the corresponding input pixel. A pixel's neighborhood is a set of pixels, defined by their locations relative to that pixel. (See Neighborhood or Block Processing: An Overview for a general discussion of neighborhood operations.) Linear filtering is filtering in which the value of an output pixel is a linear combination of the values of the pixels in the input pixel's neighborhood.

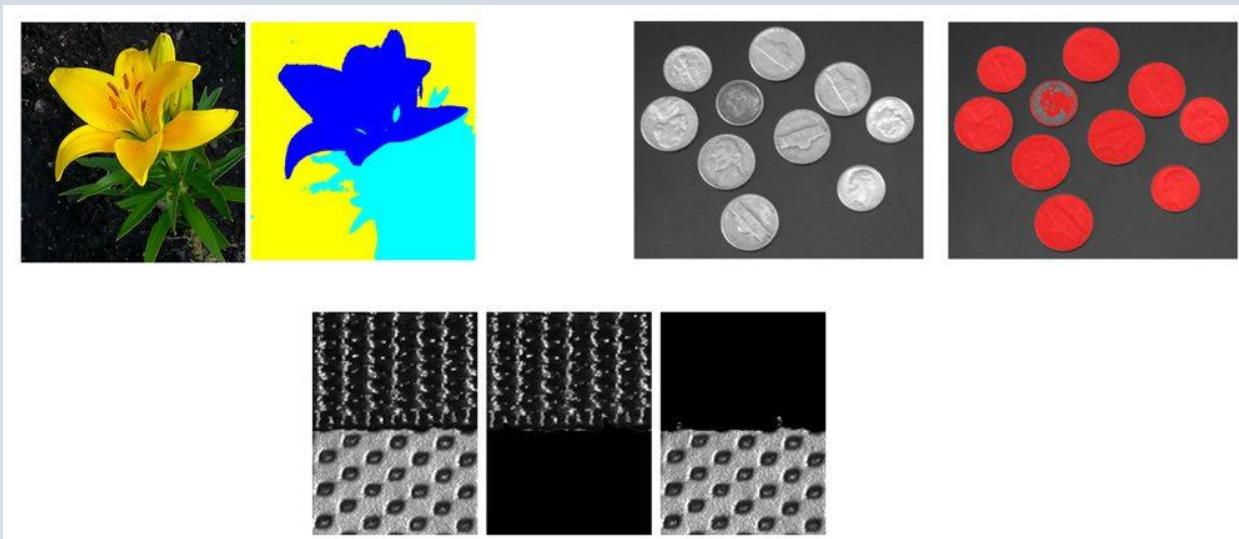
Convolution

Linear filtering of an image is accomplished through an operation called convolution. Convolution is a neighborhood operation in which each output pixel is the weighted sum of neighboring input pixels. The matrix of weights is called the convolution kernel, also known as the filter. A convolution kernel is a correlation kernel that has been rotated 180 degrees.

For example, suppose the image is

$$A = \begin{bmatrix} 17 & 24 & 1 & 8 & 15 \\ 23 & 5 & 7 & 14 & 16 \\ 4 & 6 & 13 & 20 & 22 \\ 10 & 12 & 19 & 21 & 3 \\ 18 & 22 & 25 & 23 & 9 \end{bmatrix}$$

Another common approach is to detect similarities in the regions of an image. Some techniques that follow this approach are region growing, clustering, and thresholding.



A variety of other approaches to perform image segmentation have been developed over the years using domain-specific knowledge to effectively solve segmentation problems in specific application areas.

Image Segmentation with MATLAB

With MATLAB®, you can:

- Use apps to interactively explore different segmentation techniques
- Simplify image analysis workflows using built-in image segmentation algorithms

LESSON#18 IMAGE PROCESSING & COMPUTER VISION

- Perform deep learning for image segmentation

Using Apps to Interactively Threshold Images

Image Segmente App

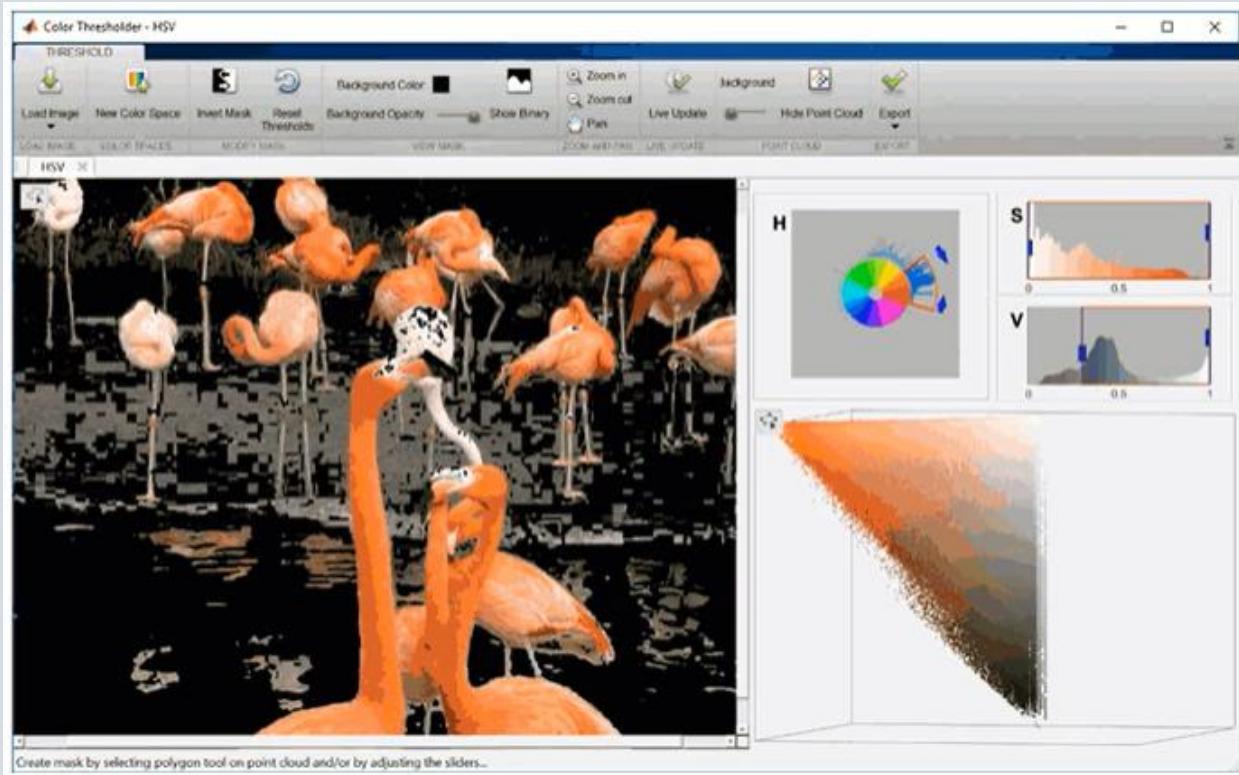
Using the interactive **Image Segmente** app, you can iteratively try several methods to segment an image before achieving the desired result. For example, you can use the app to segment and further refine the results of an MRI image of a knee with different methods.



Color Thresholder App

This Color Thresholder app lets you apply thresholding to color images by manipulating the color of the images interactively, based on different color spaces. For example, you can use the Color Thresholder app to create a binary mask using point cloud controls for a color image.

LESSON#18 IMAGE PROCESSING & COMPUTER VISION

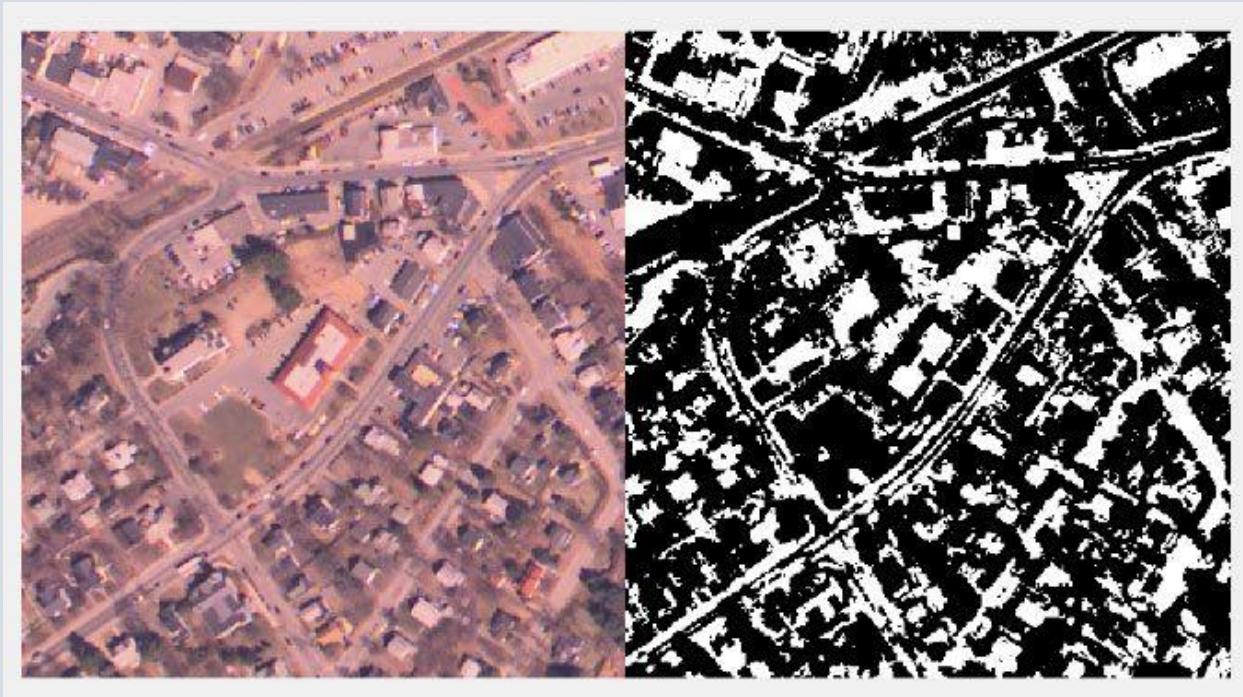


Using a Variety of Image Segmentation Techniques

With functions in MATLAB and Image Processing Toolbox™, you can experiment and build expertise on the different image segmentation techniques, including thresholding, clustering, graph-based segmentation, and region growing.

Thresholding

Using **Otsu's method**, `imbinarize` performs thresholding on a 2D or 3D grayscale image to create a binary image. To produce a binary image from an RGB color image, use `rgb2gray` to first convert it to a grayscale image.



Global Thresholds:

Adaptive Threshold:

Adaptive Thresholding: It is an algorithm for calculating the threshold of a small area in the image, so we will get several thresholds for several regions in the same image, and this will give better results for images in different lighting conditions.

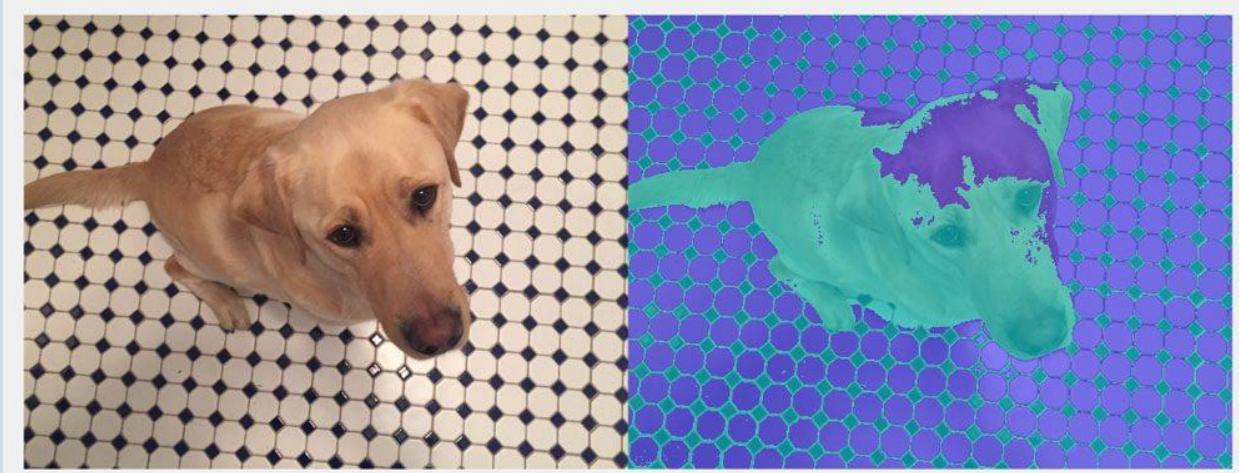
Otsu's Binarization

In simple thresholding we chose a random value for the threshold, then experimentally we determine the ideal values for the arrangement, but if we have a bimodal image (meaning the binary image is the image that has two values in its graph), we will take an intermediate threshold value between these two values and this is what Enable Otsu Conversion.

Clustering

This technique lets you create a segmented labeled image using a specific clustering algorithm. Using K-means clustering-based segmentation, `imsegkmeans` segments an image into K number of clusters.

LESSON#18 IMAGE PROCESSING & COMPUTER VISION

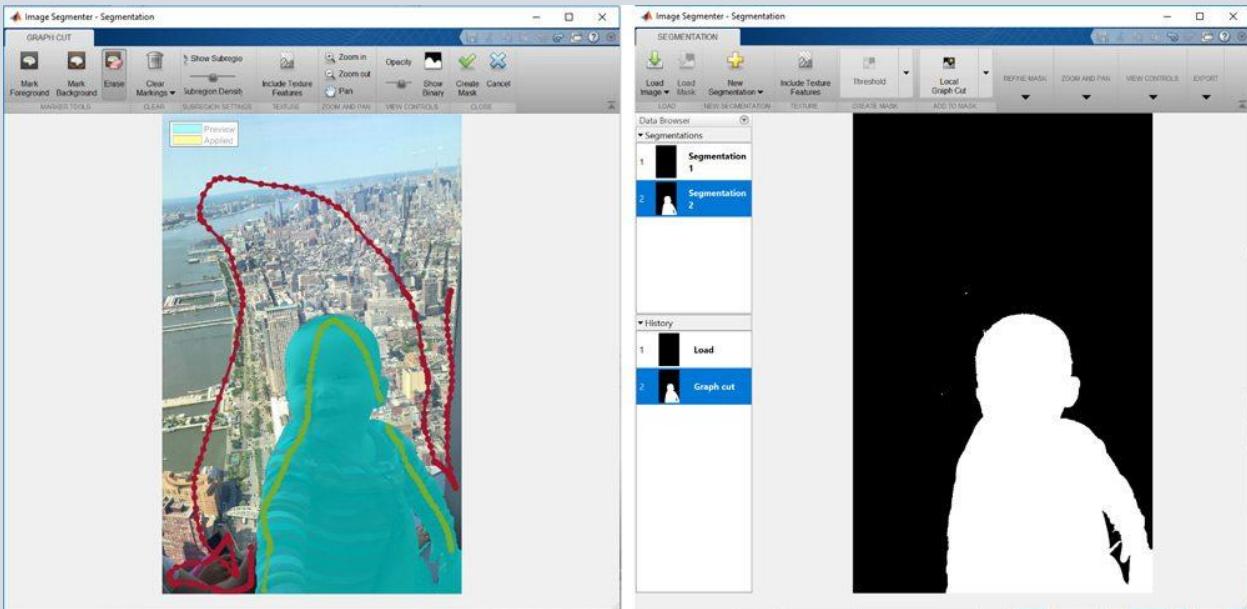


Graph-Based Segmentation

Graph-based segmentation techniques like lazy-snapping enable you to segment an image into foreground and background regions. MATLAB lets you perform this segmentation on your image either programmatically ([lazysnapping](#)) or interactively using the Image Segmenter app.



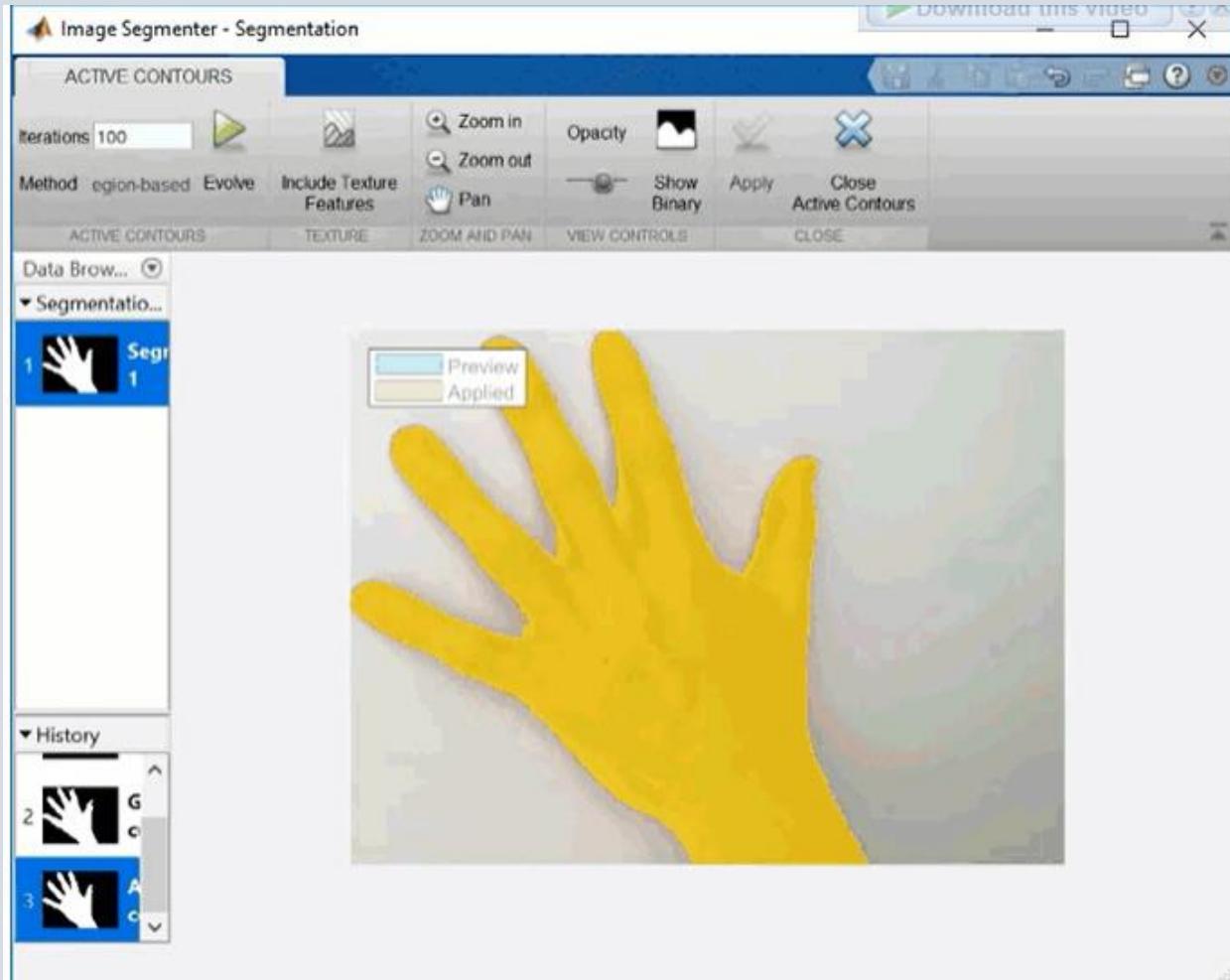
LESSON#18 IMAGE PROCESSING & COMPUTER VISION



Region Growing

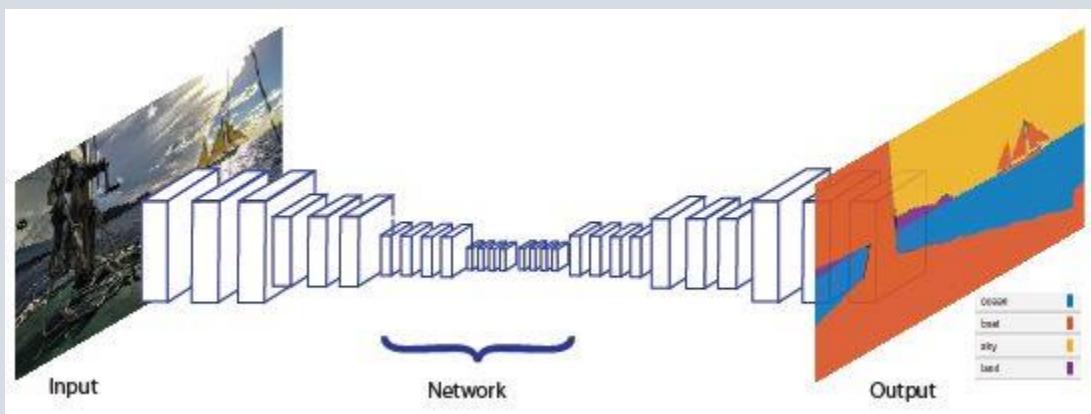
Region growing is a simple region-based (also classified as a pixel-based) image segmentation method. A popularly used algorithm is [activecontour](#), which examines neighboring pixels of initial seed points and determines iteratively whether the pixel neighbors should be added to the region. You can also perform this segmentation on images using the Image Segmenter app.

LESSON#18 IMAGE PROCESSING & COMPUTER VISION



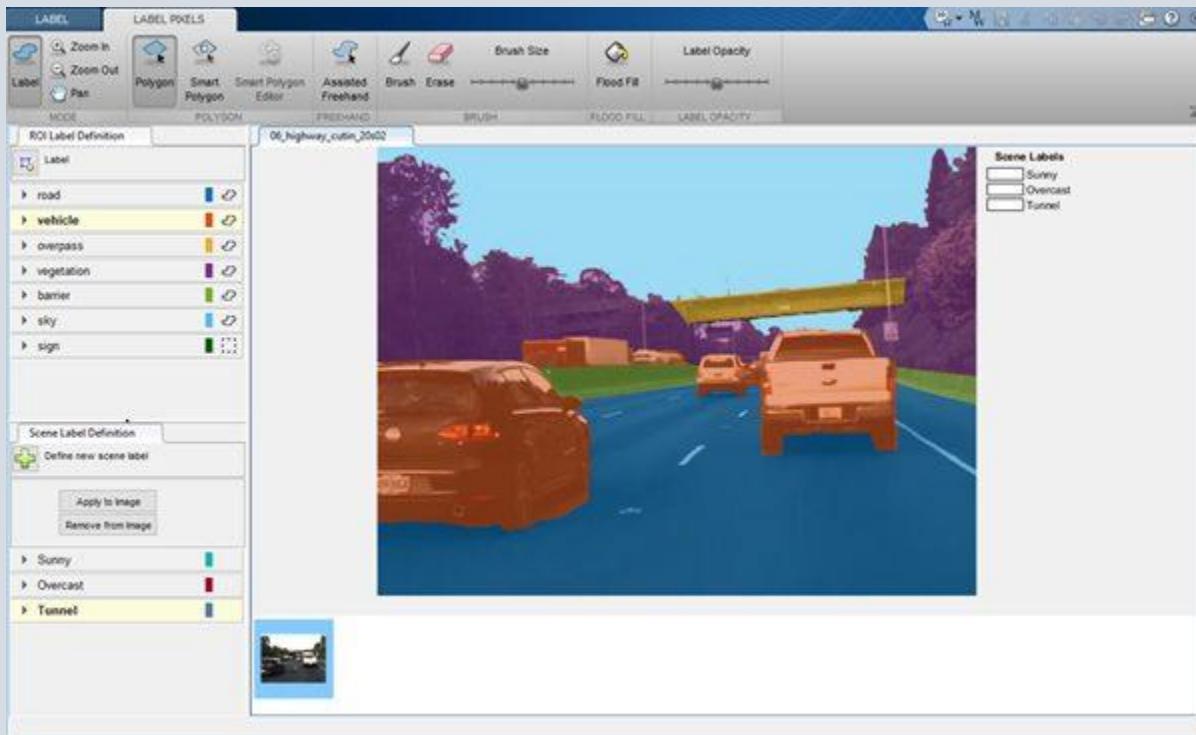
Deep Learning for Image Segmentation

Using convolutional neural networks (CNNs), a deep learning technique called semantic segmentation lets you associate every pixel of an image with a class label. Applications for semantic segmentation include autonomous driving, industrial inspection, medical imaging, and satellite image analysis.



LESSON#18 IMAGE PROCESSING & COMPUTER VISION

Using MATLAB, you can design and train semantic segmentation networks with a collection of images and their corresponding labeled images, and then use the trained network to label new images. To label the training images, you can use the Image Labeler, Video Labeler, or Ground Truth Labeler apps.



The goal of this lab

It is the knowledge the **segmentation** with images in MATLAB.

Procedure

1. save the image in the current folder.
2. Open new script file and write the following:
 - **Threshold without built-in function**

```
%Threshold without built-in function
clc
clear all
close all

x= imread('gift.jpg');
```

LESSON#18 IMAGE PROCESSING & COMPUTER VISION

```
figure, imshow(x)

xg=rgb2gray(x);
figure, imshow(xg), impixelinfo

[r,c]=size(xg);
for i=1:r
    for j=1:c
        if xg(i,j)>= 190 && xg(i,j) <=215
            xg(i,j)=255;
        else
            xg(i,j)=0;
        end
    end
end
figure, imshow(xg), impixelinfo
```

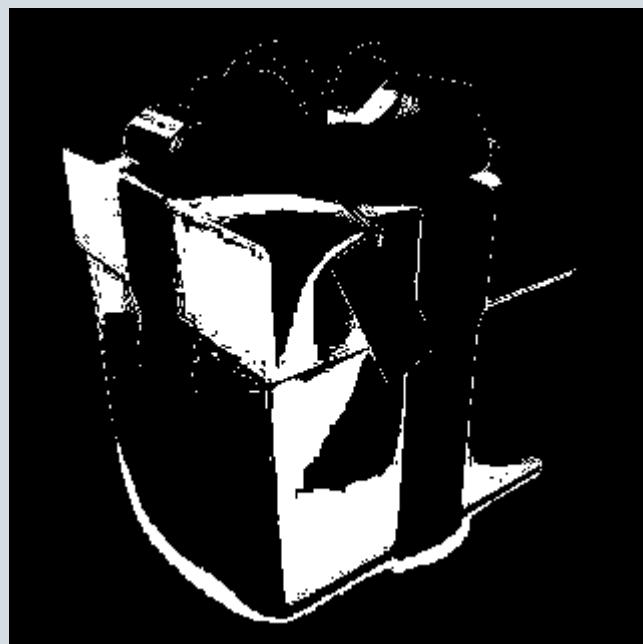
The Result



LESSON#18 IMAGE PROCESSING & COMPUTER VISION



Pixel info: (279, 55) 255



Pixel info: (X, Y) Intensity

- Threshold with built-in Threshold function

```
clc  
clear all
```

LESSON#18 IMAGE PROCESSING & COMPUTER VISION

```
close all
image = imread('bird.jpg');
%global threshold
thresh = im2bw(image);
%Gray level Threshold - otsu
threshold_otsu = graythresh(image);
thresh_image = im2bw(image,threshold_otsu);
subplot(1,3,1), imshow(thresh);
subplot(1,3,2), imshow(thresh_image);
subplot(1,3,3), imshow(image);
```

The Result



• Adaptive Threshold

```
clc
clear all
close all
image = imread('bird.jpg');
mean_filter=imfilter(image, fspecial('average',
[15,15]),'replicate');
subtract= image-(mean_filter+20);
black_white=im2bw(subtract,0);
subplot(1,2,1); imshow(black_white); title('threshold
image');
subplot(1,2,2); imshow(image); title('original
image');
```

LESSON#18 IMAGE PROCESSING & COMPUTER VISION

The Result

threshold image



original image



Report

- 1- Open the new script in MATLAB program to make segmentation to this image by using the appropriate thresholding.



Lesson#19 Image Processing & COMPUTER VISION

Morphology Process

Eng. Elaf A.Saeed
Email: elafe1888@gmail.com

LAB Content

Dilation Process

Open & Close Process to remove noise

Table of Functions

| Function Name | Description |
|--|--|
| Imdilate <code>J = imdilate(I,SE)</code> | Dilate image. <code>J = imdilate(I,SE)</code> dilates the grayscale, binary, or packed binary image I, returning the dilated image, J. SE is a structuring element object or array of structuring element objects, returned by the <code>strel</code> or <code>offsetstrel</code> functions. |
| Imopen <code>J = imopen(I,SE)</code> <code>J = imopen(I,nhood)</code> | Morphologically open image. <code>J = imopen(I,SE)</code> performs morphological opening on the grayscale or binary image I, returning the opened image, J. SE is a single structuring element object returned by the <code>strel</code> or <code>offsetstrel</code> functions. The morphological open operation is an erosion followed by a dilation, using the same structuring element for both operations. |
| Imclose <code>J = imclose(I,SE)</code> | Morphologically close image. <code>J = imclose(I,SE)</code> performs morphological closing on the grayscale or binary image I, returning the closed image, J. SE is a single structuring element object returned by the <code>strel</code> or <code>offsetstrel</code> functions. The morphological close operation is a dilation followed by an erosion, using the same structuring element for both operations. |

Theory

What Is Image Segmentation?

Morphological Transformations are some simple image operations that are usually performed on binary images. The basic processes are erosion, expansion, opening, closing, and gradient.

We'll see how to apply these operations to the next image.



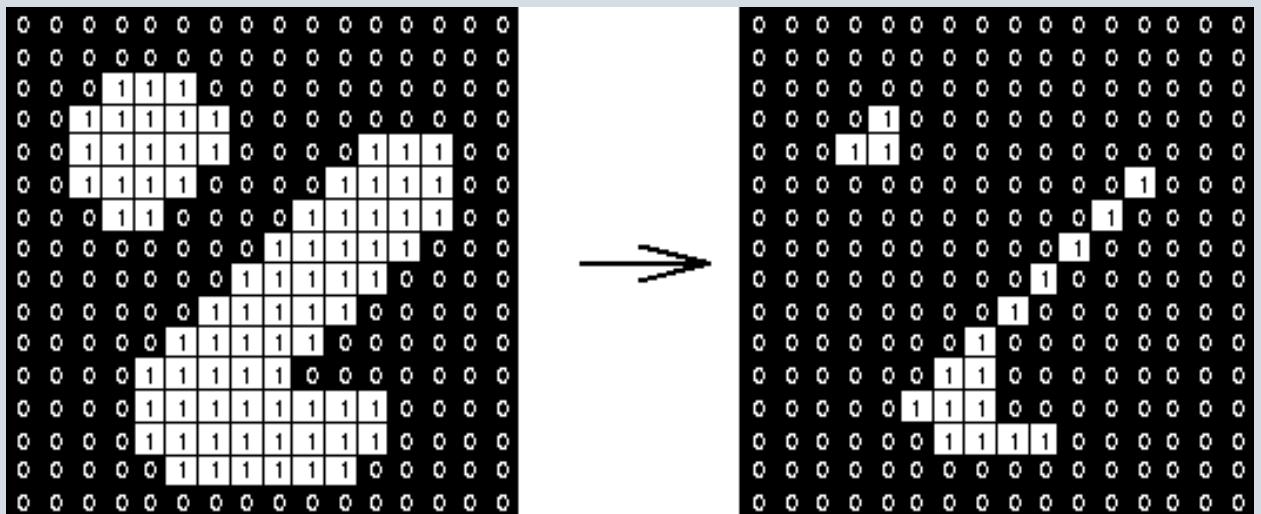
Erosion Operation

The basic idea of erosion is just like soil erosion, when applying the erosion process to the shape, it softens the front surface boundaries.

How is the Erosion process done?!

- The pixel in the original image takes two values, either 0 or 1 (black or white).
- The pixel in the original image will be considered 1 only if all the pixels are under the mask 1, otherwise it will be ignored, meaning that it takes the value 0.
- What happens is that every pixel close to the boundary will be ignored. The thickness of the border to be eaten depends on the size of the mask.

LESSON#19 IMAGE PROCESSING & COMPUTER VISION

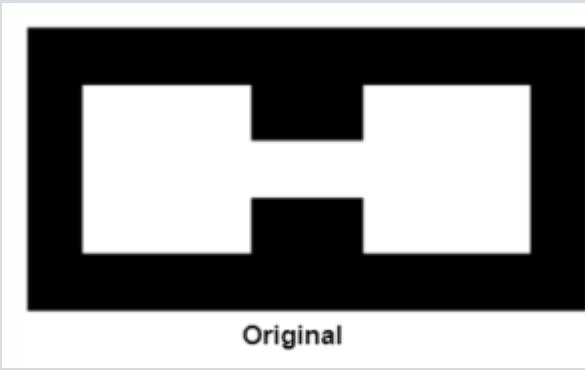


The diagram illustrates a dilation operation. On the left, a 5x5 input matrix is shown with a 3x3 kernel applied to it. The kernel is defined by the following binary values:

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |

The input matrix contains mostly zeros, with some ones representing noise or features. The result of the dilation operation is shown on the right, where the kernel has been shifted across the input and its maximum value (1) has been placed at the center of each kernel's footprint. This results in a larger, more connected white region.

This process helps us remove noise like small white dots, and it also helps us separate two linearly connected objects.

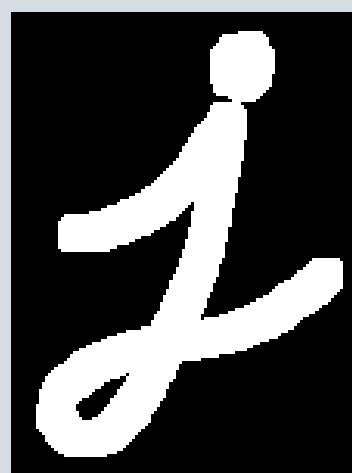
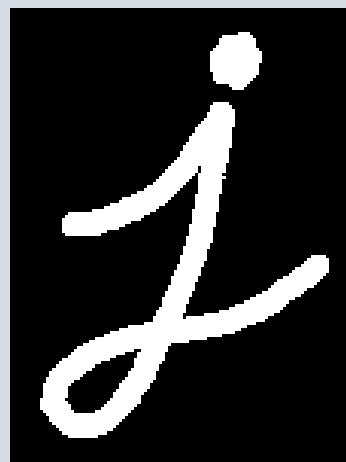


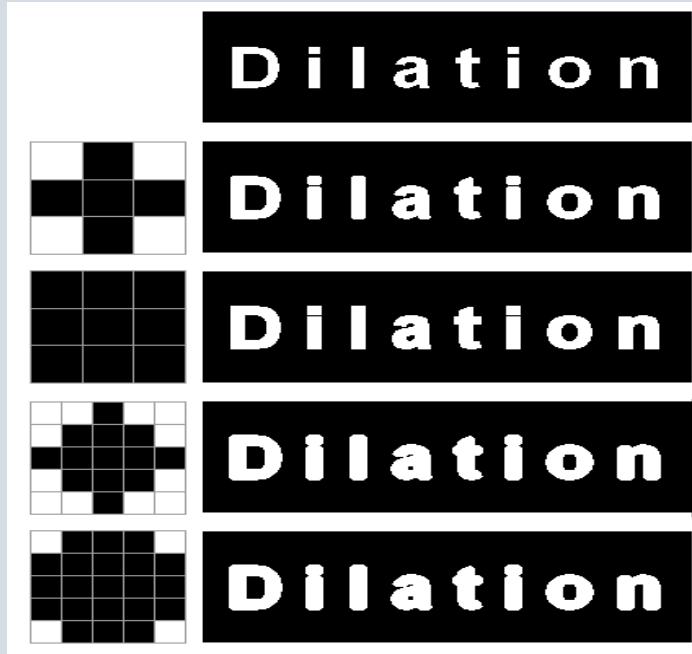
2- Dilation Operation

LESSON#19 IMAGE PROCESSING & COMPUTER VISION

The Dilation process reverses the erosion process. Here, pixel elements will have a value of 1 if one of its elements are under the mask at least 1, and as a result the size of the white area increases.

Usually when applying the erosion process to remove the noise of the white area in the image, we notice that the area of the body has been reduced, for this reason we follow the process of erosion with the stretching process in order to increase the area of the body that is being eaten. The process of dilation is useful for restoring Corroded parts of the body.





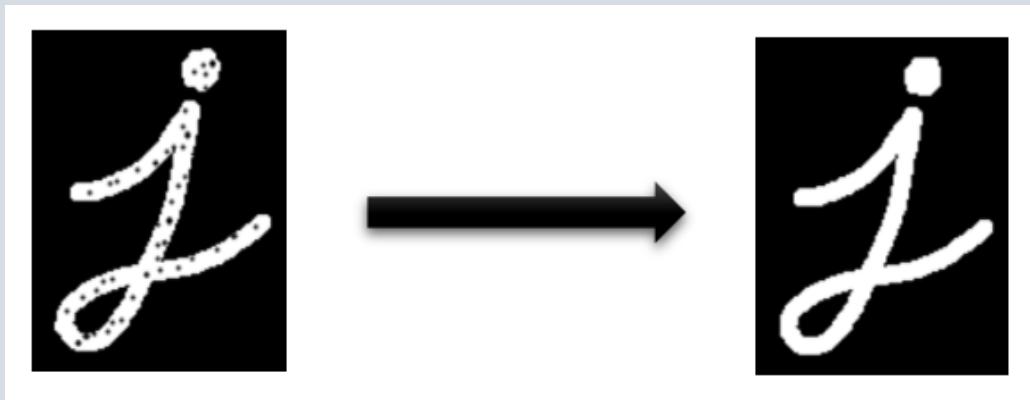
3- Opening Operation

The opening process is an erosion + Dilation process. This process is useful for removing noise



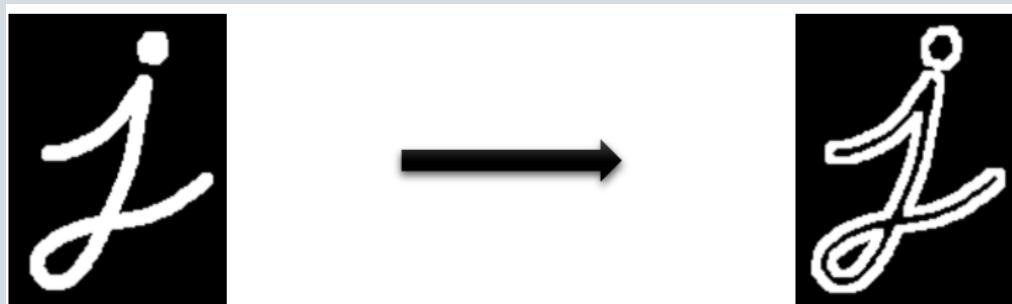
4- Closing Operation

Reverse the opening process, we first carry out the dilation process, then the erosion process. This process is useful for close the holes in the body and useful for closing the small black spots on the body.



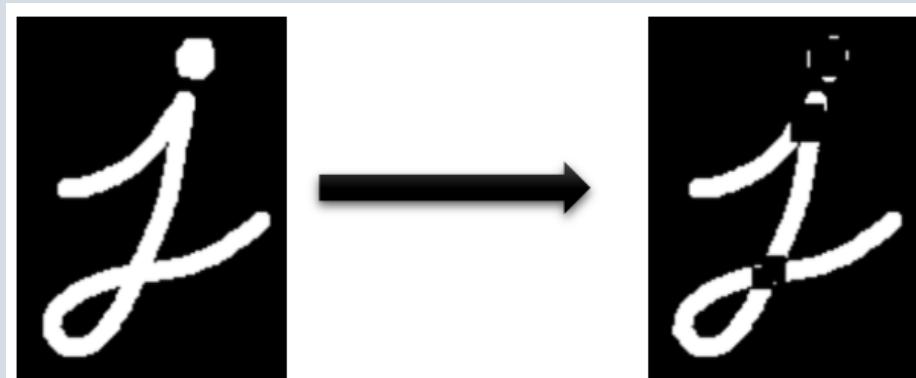
5- Morphological Gradient

It is the difference between the process of erosion and dilation, as it will appear as the boundaries of the body.



6- Top Hat

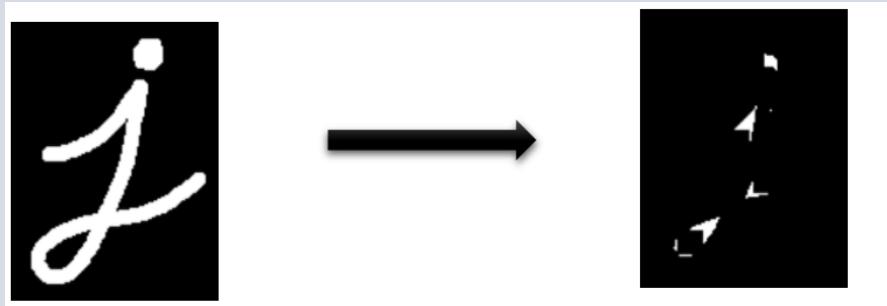
The difference between the image opening process and the image itself.



7- Black Hat

It is the difference between closing the image and the original image.

LESSON#19 IMAGE PROCESSING & COMPUTER VISION



Different Mythology Function in MATLAB

| | |
|----------------------|--|
| imerode | Erode image |
| imdilate | Dilate image |
| imopen | Morphologically open image |
| imclose | Morphologically close image |
| imtophat | Top-hat filtering |
| imbothat | Bottom-hat filtering |
| imclearborder | Suppress light structures connected to image border |
| imfill | Fill image regions and holes |
| bwhitmiss | Binary hit-miss operation |
| bwmorph | Morphological operations on binary images |
| bwmorph3 | Morphological operations on binary volume |
| bwperim | Find perimeter of objects in binary image |
| bwskele | Reduce all objects to lines in 2-D binary image or 3-D binary volume |
| bwulterode | Ultimate erosion |

The goal of this lab

It is the knowledge the **Morphology** with images in MATLAB.

Procedure

1. save the image in the current folder.
2. Open new script file and write the following:

- **Dilation Process**

```
%Dilation
clc
clear all
close all

originalBW= imread('j.png');
se = strel('line',9,9);
dilatedBW= imdilate(originalBW,se)
figure, subplot(121), imshow(originalBW),
subplot(122), imshow(dilatedBW)
```

The Result



- **Open & Close Process to remove noise**

```
clc
clear all
```

LESSON#19 IMAGE PROCESSING & COMPUTER VISION

```
close all
warning off
x= rgb2gray(imread('gift.jpg'));
imshow(x), title('original image')
figure;
se=ones(7,7);
imshow(imopen(x,se)), title('Before-open process')
figure;
imshow(imclose(x,se)), title('Before-close process')
%add noise to image
I=imnoise(x, 'salt & pepper', 0.5);
figure;
imshow(I), title('Noise Image')
figure;
imshow(imopen(x,se)), title('after-open process')
figure;
imshow(imclose(x,se)), title('after-close process')
figure;
imshow(imclose(imopen(x,se), se)), title('after-open &
close process')
```

The Result

LESSON#19 IMAGE PROCESSING & COMPUTER VISION

original image



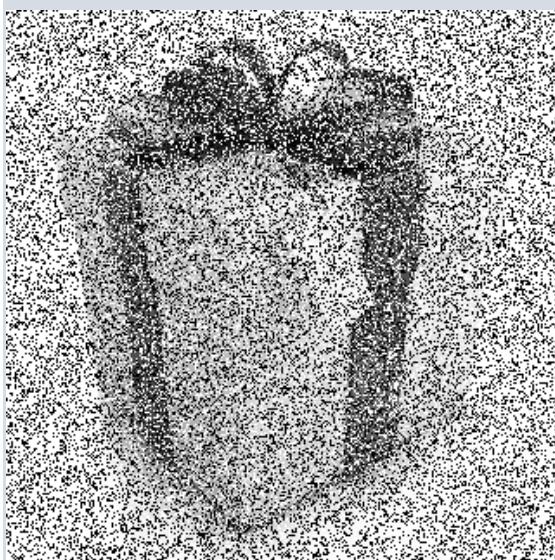
Before-open process



Before-close process



Noise Image



LESSON#19 IMAGE PROCESSING & COMPUTER VISION

after-open process



after-close process



after-open & close process



Report

1-

LESSON#19 IMAGE PROCESSING & COMPUTER VISION

Email: elafe1888@gmail.com

linkden: www.linkedin.com/in/elaf-a-saeed-97bbb6150

Facebook: <https://www.facebook.com/profile.php?id=100004305557442>

twitter: <https://twitter.com/ElafASaeed1>

GitHub: <https://github.com/ElafAhmedSaeed>

YouTube: https://youtube.com/channel/UCE_RiXkyqREUdLAiZcbBqSg

SlideShare: <https://www.slideshare.net/ElafASaeed>

Slide player: <https://slideplayer.com/search/?q=Elaf+A.Saeed>

Google

Scholar:

<https://scholar.google.com/citations?user=VIpVZKkAAAAJ&hl=ar&gmla=AJsN-F7PIgAjWJ44Hzb18fwPqJaaUmG0XzbLdzx09>

Lesson#20 Image Processing & COMPUTER VISION

Image Pyramids

Eng. Elaf A.Saeed
Email: *elafe1888@gmail.com*

LAB Content

Subsampling Image without built-in function.

Subsampling Image without built-in function.

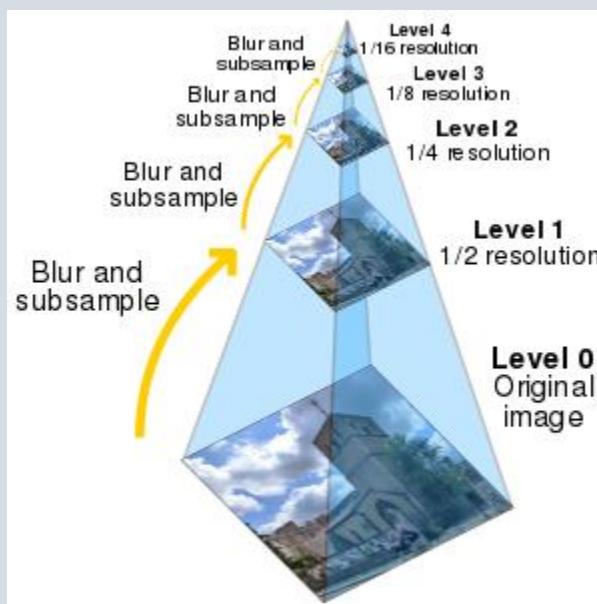
Table of Functions

| Function Name | Description |
|---|--|
| impyramid B = impyramid(A,direction) | Image pyramid reduction and expansion. B = impyramid(A,direction) computes a Gaussian pyramid reduction or expansion of A by one level. direction determines whether impyramid performs a reduction or an expansion. |

Theory

Pyramid (image processing)

Pyramid, or pyramid representation, is a type of multi-scale signal representation developed by the computer vision, image processing and signal processing communities, in which a signal or an image is subject to repeated smoothing and subsampling. Pyramid representation is a predecessor to scale-space representation and multiresolution analysis.



Pyramid generation

There are two main types of pyramids: lowpass and bandpass.

A lowpass pyramid is made by smoothing the image with an appropriate smoothing filter and then subsampling the smoothed image, usually by a factor of 2 along each coordinate direction. The resulting image is then subjected to the same procedure, and the cycle is repeated multiple times. Each cycle of this process results in a smaller image with increased smoothing, but with decreased spatial sampling density (that is, decreased image resolution). If illustrated graphically, the entire multi-scale representation will look like a pyramid, with the original image on the bottom and each cycle's resulting smaller image stacked one atop the other.

A bandpass pyramid is made by forming the difference between images at adjacent levels in the pyramid and performing image interpolation between adjacent levels of resolution, to enable computation of pixelwise differences.

Pyramid generation kernels

A variety of different smoothing kernels have been proposed for generating pyramids. Among the suggestions that have been given, the binomial kernels arising from the binomial coefficients stand out as a particularly useful and theoretically well-founded class. Thus, given a two-dimensional image, we may apply the (normalized) binomial filter (1/4, 1/2, 1/4) typically twice or more along each spatial dimension and then subsample the image by a factor of two. This operation may then proceed as many times as desired, leading to a compact and efficient multi-scale representation. If motivated by specific requirements, intermediate scale levels may also be generated where the subsampling stage is sometimes left out, leading to an oversampled or hybrid pyramid. [10] With the increasing computational efficiency of CPUs available today, it is in some situations also feasible to use wider supported Gaussian filters as smoothing kernels in the pyramid generation steps.

Gaussian pyramid

In a Gaussian pyramid, subsequent images are weighted down using a Gaussian average (Gaussian blur) and scaled down. Each pixel containing a local average corresponds to a neighborhood pixel on a lower level of the pyramid. This technique is used especially in texture synthesis.

Laplacian pyramid

A Laplacian pyramid is very similar to a Gaussian pyramid but saves the difference image of the blurred versions between each level. Only the smallest level is not a difference image to enable reconstruction of the high-resolution image using the difference images on higher levels. This technique can be used in image compression.

Steerable pyramid

A steerable pyramid, developed by Simoncelli and others, is an implementation of a multi-scale, multi-orientation band-pass filter bank used for applications including image compression, texture synthesis, and object recognition. It can be thought of as an orientation selective version of a Laplacian pyramid, in which a bank of steerable filters is used at each level of the pyramid instead of a single Laplacian or Gaussian filter.

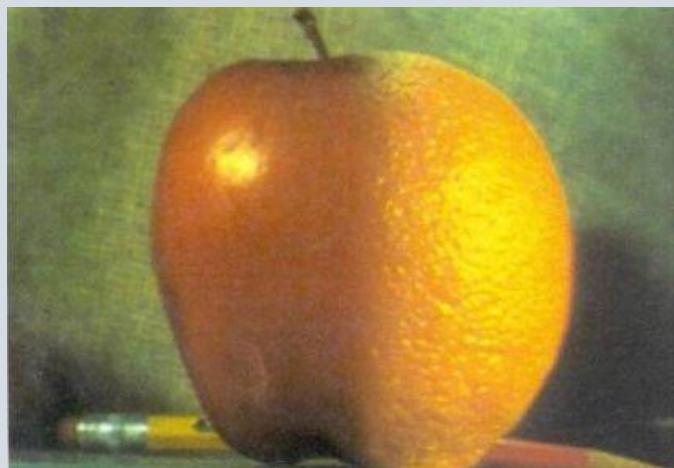
LESSON#20 IMAGE PROCESSING & COMPUTER VISION

Image Pyramid Application

One of the applications of the pyramids is to blending images.

The goal of this lab

It is the knowledge the **Image Pyramids** in MATLAB.



Note:

Aliasing happen; when downsampling by a factor of two.

- Subsampling with Gaussian pre-filtering

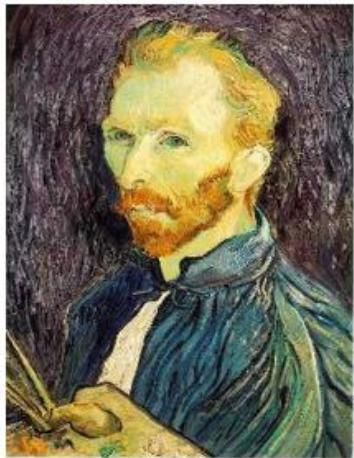


Gaussian 1/2

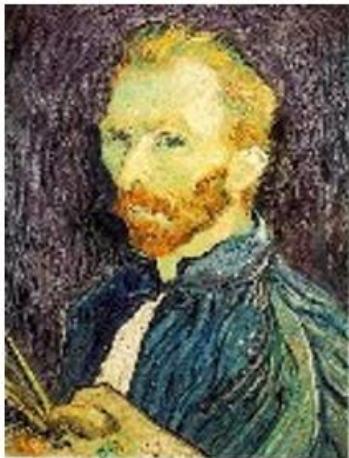
G 1/4

G 1/8

Compare with...



1/2



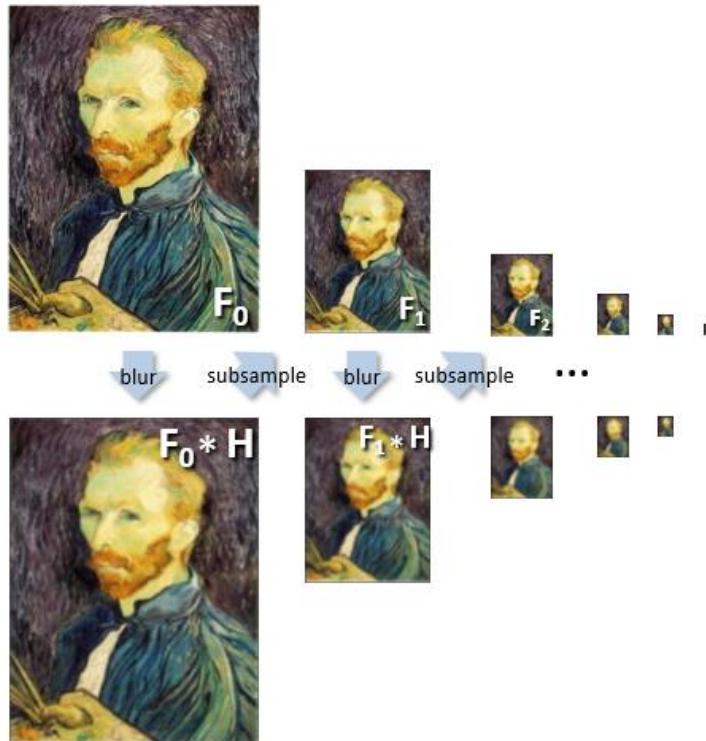
1/4 (2x zoom)



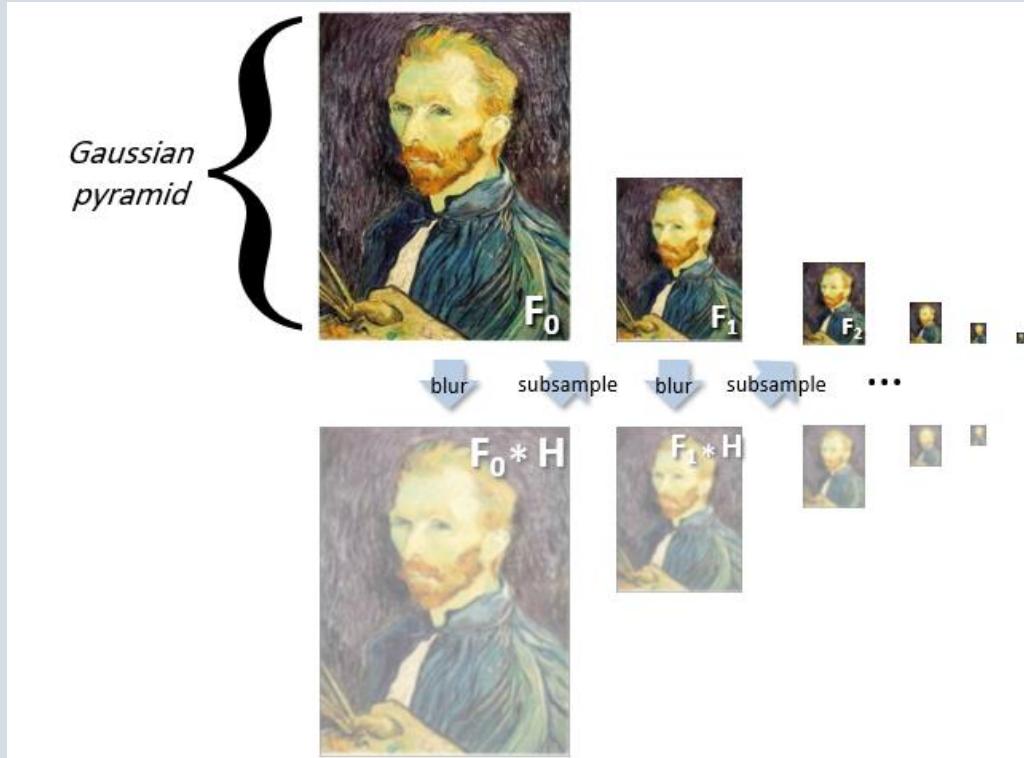
1/8 (4x zoom)

Gaussian pre-filtering

- Solution: filter the image, *then* subsample



LESSON#20 IMAGE PROCESSING & COMPUTER VISION



Procedure

1. save the image in the current folder.
2. Open new script file and write the following:

Subsampling Image without built-in function

```
%Subsampling Image
clc
clear all
close all
im1=imread('cameraman.tif');
figure, imshow(im1), title('original image')
[hight,width,depth]=size(im1);
%convert RGB image to gray image
if depth==1
    im2=im1;
else
    im2=im1(:,:,1);
end
%downsampling no prefiltering
```

LESSON#20 IMAGE PROCESSING & COMPUTER VISION

```
M=4;  
im3=im2(1:M:hight , 1:M:width);  
figure,imshow(im3)  
title(['Downsampled by' num2str(M) ':No  
prefiltering']);  
%Reconstruct to original size  
im4=imresize(im3,[hight,width],'bilinear');  
figure,imshow(im4)  
title('Reconstructed from downsampled image: No  
prefiltering')  
%downsampled by a factor of M, after prefiltering  
%use gaussian filter  
im5=imfilter(im2,fspecial('gaussian',[9  
9],1),'symmetric','same');  
im6=im5(1:M:hight , 1:M:width);  
figure,imshow(im6)  
title(['Downsampled by' num2str(M) ':After  
prefiltering']);  
im7=imresize(im6,[hight,width],'bilinear');  
figure,imshow(im7)  
title('Reconstructed from downsampled image: with  
prefiltering')
```

The Result

original image



Downsampled by 4: No prefiltering



LESSON#20 IMAGE PROCESSING & COMPUTER VISION

Reconstructed from downsampled image: No prefiltering



Downsampled by 4: After prefiltering



Reconstructed from downsampled image: with prefiltering



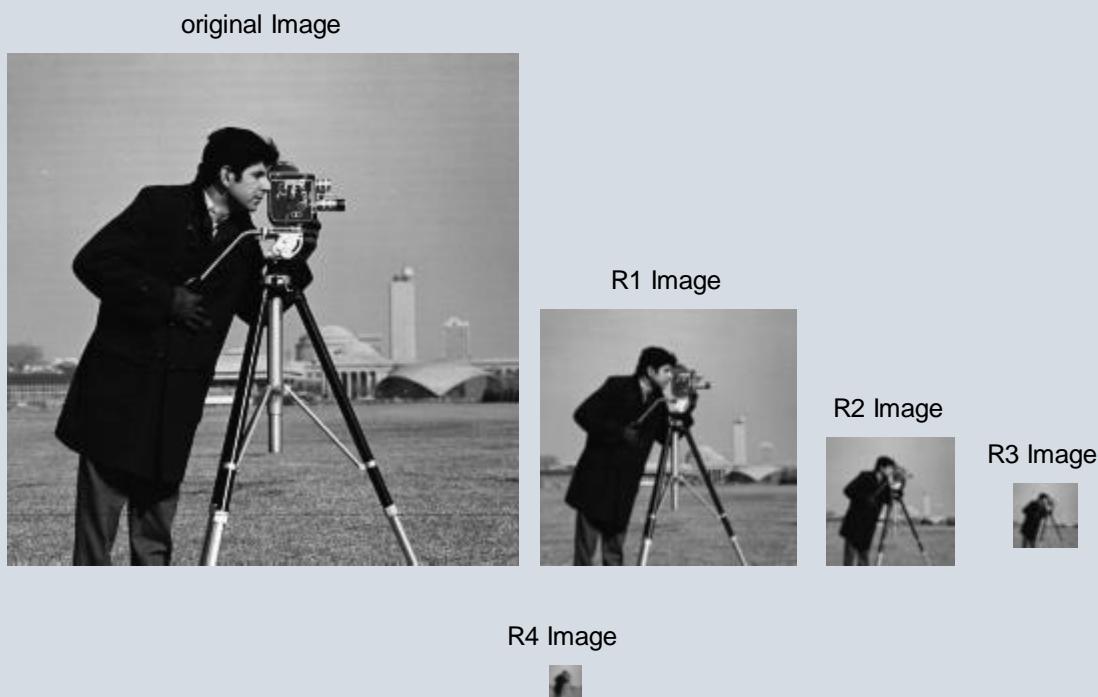
Subsampling Image without built-in function

```
% Subsampling Image - impyramid Function  
clc  
clear all  
close all  
h=imread('cameraman.tif');  
R1=impyramid(h, 'reduce');  
R2=impyramid(R1, 'reduce');  
R3=impyramid(R2, 'reduce');  
R4=impyramid(R3, 'reduce');  
  
imshow(h)  
title('original Image');
```

LESSON#20 IMAGE PROCESSING & COMPUTER VISION

```
figure;  
imshow(R1)  
title('R1 Image');  
figure;  
imshow(R2)  
title('R2 Image');  
figure;  
imshow(R3)  
title('R3 Image');  
figure;  
imshow(R4)  
title('R4 Image');
```

The Result



Report

- 1- Write the MATLAB program to sharping image without built in function.
- 2- Write the MATLAB program to down sampling the 1024x1024 image to 512x512, 256x256, and 128x128.
- 3- Write the MATLAB program to add Gaussian noise to image and remove it.



Elaf A.Saeed is a systems and control engineer at the University of Al-Nahrain, college of information engineering, Iraq. she works in the fields of control, embedded systems, web design and has a talent of programming. she was ranking as the first student of her stage for all years in B.Sc . you can contact them via e-mail elafe1888@gmail.com

linkden: www.linkedin.com/in/elaf-a-saeed-97bbb6150

Facebook:

<https://www.facebook.com/profile.php?id=100004305557442>

Twitter: <https://twitter.com/ElafASaeed1>

GitHub: <https://github.com/ElafAhmedSaeed>

YouTube:

https://youtube.com/channel/UCE_RiXkyqREUdLAiZcbBqSg

SlideShare: <https://www.slideshare.net/ElafASaeed>

SlidePlayer:

<https://slideplayer.com/search/?q=Elaf+A.Saeed>

Google

<https://scholar.google.com/citations?user=VIpVZKkA AAAJ&hl=ar&gmla=AJsN-F7PIgAjWJ44Hzb18fwPqJaaUmG0XzbLdzx09>

Scholar: