

**R**OBODARSHAN

# IMAGE PROCESSING TUTORIALS

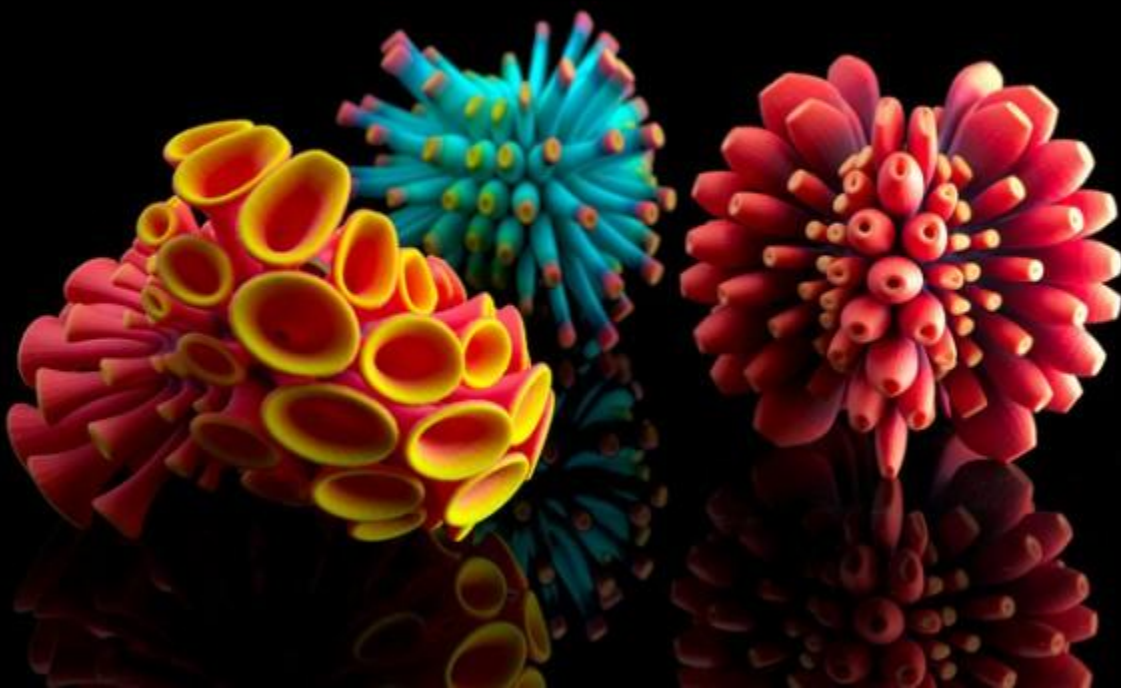
# Topics covered:

---

1. Different types of colour spaces.
2. Storing images as Matrices.
3. Working in Matlab.
4. Basic operations of Matrices in Matlab.
5. Extracting different layers of images.
6. Locating objects in images.
7. Using bounding boxes.
8. Identifying shapes.
9. Arduino Interfacing with Matlab.

You can watch an interesting video:

<https://www.youtube.com/watch?v=J8sl3nMIYxM>

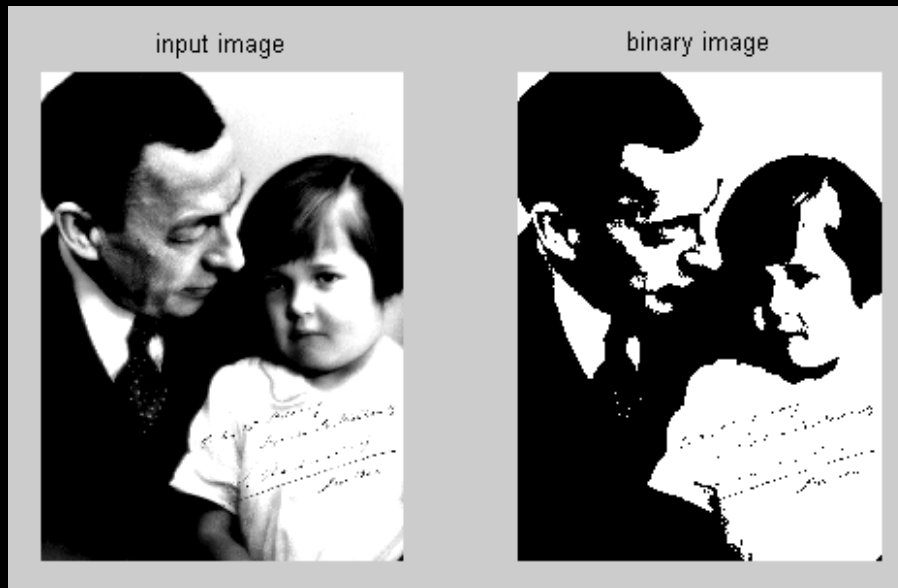


There are various types of images:

Binary – Having only black and white but nothing in the middle.

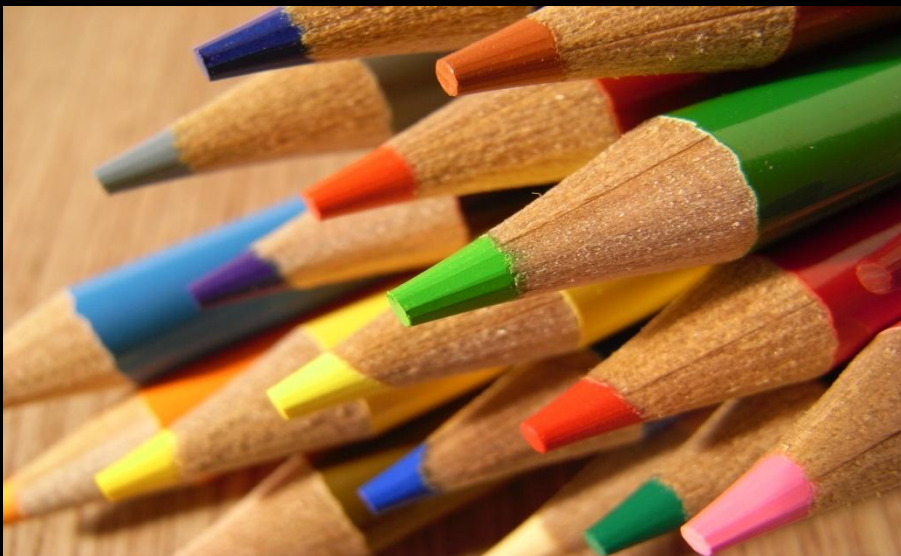
Grayscale – Having various shades of black and white.

Coloured – having all colours.



Grayscale

Binary



Coloured

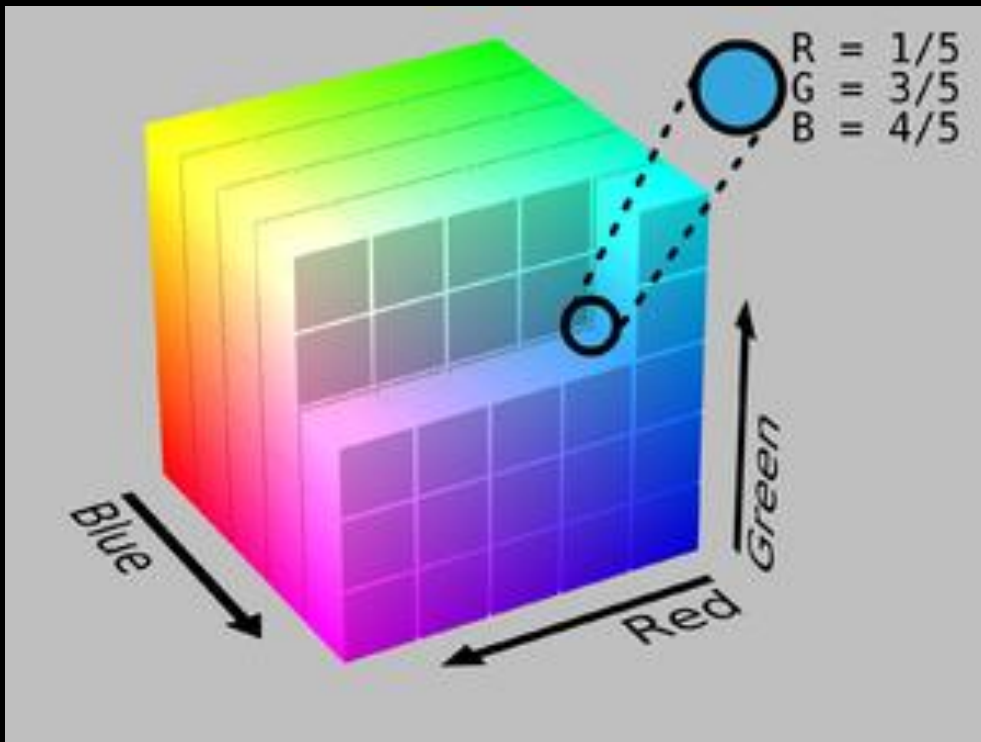
# COLOUR SPACES:

## RGB:

An **RGB color space** is any additive color space based on the RGB color model. A particular RGB color space is defined by the three chromaticities of the red, green, and blue additive primaries, and can produce any chromaticity that is the triangle defined by those primary colors. sRGB is by far the most commonly used RGB color space.

All 3 parameters can take values from 0-255.

**RGB** is an abbreviation for red–green–blue.

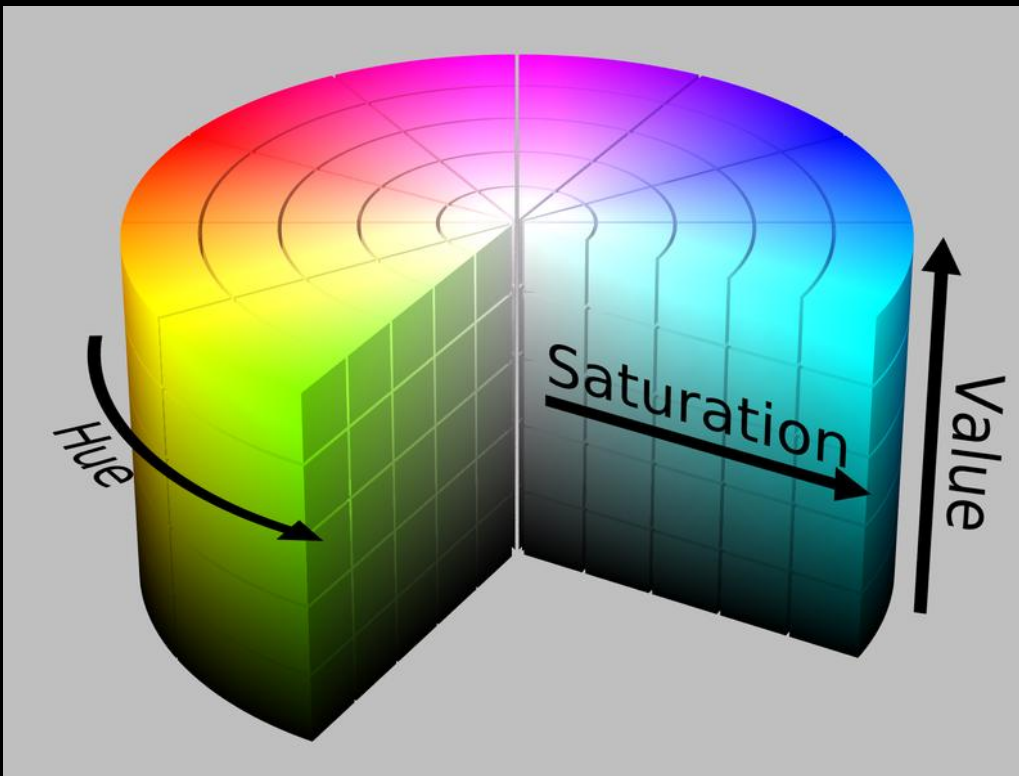


## HSV:

**HSL** and **HSV** are the two most common cylindrical coordinate representations of points in an RGB color model. The two representations rearrange the geometry of RGB in an attempt to be more intuitive and perceptually relevant than the Cartesian (cube) representation. Developed in the 1970s for computer graphics applications, HSL and HSV are used today in color pickers, in image editing software, and less commonly in image analysis and computer vision.

HSV stands for *hue*, *saturation*, and *value*, and is also often called **HSB** (*B* for *brightness*).

Each parameter can take values from 0-1.



## Storing Images:

As we all know that Digital images are stored as pixels, so, we can store images in matrices.

But memory is limited, so we tell how many pixels do we want to store.

There comes resolution, which is our enemy in image processing. If we store very small number of pixels then we will get pixelated images.

So you have to decide how many pixels you want to store.



High Res Image 300dpi



Low Res Image 72dpi



## For basic usage we will work on RGB images here.

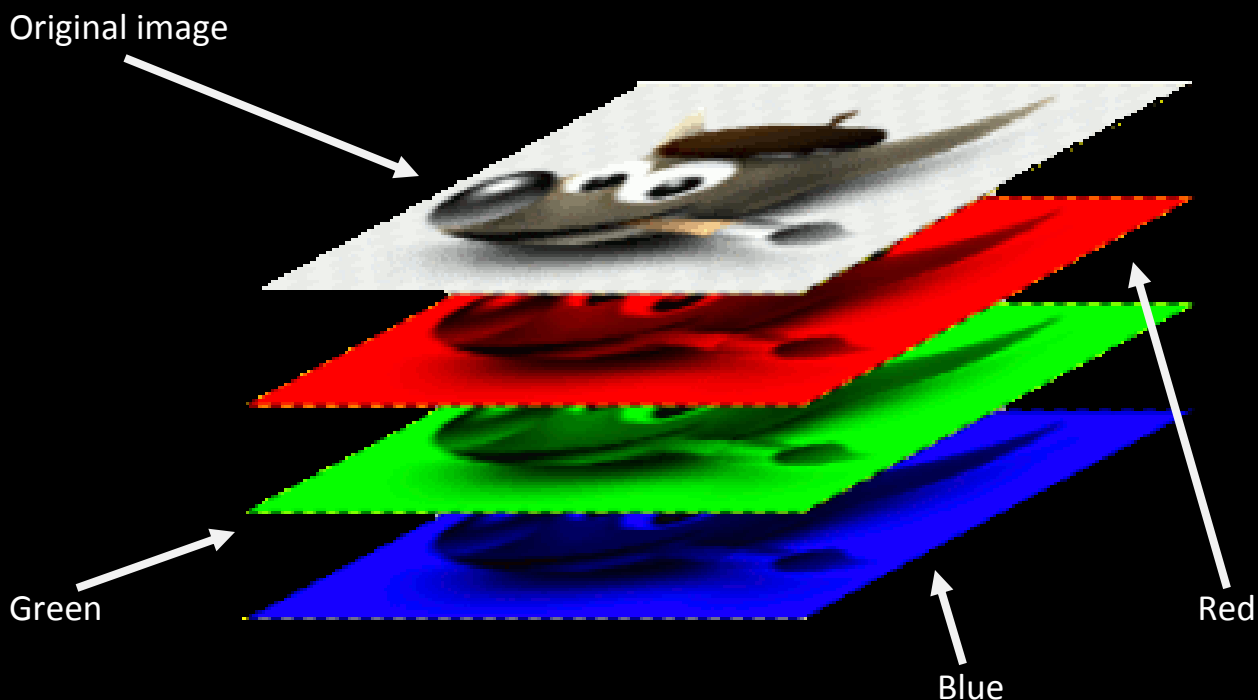
So, how are rgb images stored?

As you all know that rgb images have three components: red, green and blue. Thus, each pixel has three components, right? Thus for an  $M \times N$  resolution image with  $MN$  pixels, we need three  $M \times N$  matrices, each one for r, g, b respectively.

Each pixel can have values from 0-255.

Thus the representation for each pixel will be  $[X, Y, Z]$  (say).

Now, grayscale images will need only one matrix, so as for binary images where in each cell, either 0 or 1 will be stored.



## Working in MATLAB:

Working with Matlab is very easy. Even if you get stuck in anything type in the search above in the software.

You don't have to declare the data type but it will dynamically change with what you want to store.

If you put `a = 10`, it will store and display it.

## Basic matrix operations

As we know that digital images are basically matrices, we need to know about basic matrix operations in MATLAB to learn image processing.

Here, all basic operations are shown using examples.

Let's create a simple single row matrix or a vector.

```
>> a=[1,2,3]
```

```
a =
```

```
1    2    3
```

A simple vector has been created.

Now, let's create a single column matrix. Semicolons are used to separate rows of a matrix.

```
>> b=[1;2;3]
```

```
b =
```

```
1
```

```
2
```

```
3
```



A single column matrix has been created.

Now, let's create a 3X2 matrix.

```
>> c=[1,2;3,4;5,6]
```

c =

1 2

3 4

5 6

**Another way of creating a matrix:**

```
>> f=[1:4;6:9]
```

f =

1 2 3 4

6 7 8 9

But, g=[1:5,6:10] will create a simple vector.

g =

1 2 3 4 5 6 7 8 9 10

**To create a zero matrix:**

```
>> s=zeros(2,3)
```

s =

0 0 0

0 0 0

### To create an all-ones matrix:

```
>> x=ones(3,4)
```

```
x =
```

```
1  1  1  1
```

```
1  1  1  1
```

```
1  1  1  1
```

If we want to add a constant with every element of a matrix:

```
>> h=f+2
```

```
h =
```

```
3  4  5  6
```

```
8  9 10 11
```

### Matrix addition and subtraction:

```
>> k=[1,2,3;4,5,6]
```

```
k =
```

```
1  2  3
```

```
4  5  6
```

```
>> p=[4,8,7;6,4,9]
```

```
p =
```

```
4  8  7
```

```
6  4  9
```

```
>> u=k+p
```

```
u =
```

```
5 10 10
```

```
10  9  15
```

```
>> v=k-p
```

```
v =
```

```
-3  -6  -4
```

```
-2   1  -3
```

### **Matrix multiplication:**

```
>> l=[1,2,4;5,8,7;4,2,8]
```

```
l =
```

```
1   2   4
```

```
5   8   7
```

```
4   2   8
```

```
>> m=k*l
```

```
m =
```

```
23  24  42
```

```
53  60  99
```

### **For element wise multiplication:**

```
>> w=k.*p
```

```
w =
```

```
4  16  21
```

```
24  20  54
```

Note that, for element wise multiplication, the dimension of the matrices must be same.

### **Multiplication by a scalar:**

```
>> f*2
```

```
ans =
```

2   4   6   8

12   14   16   18

**For element wise division:**

```
>> z=w/2
```

z =

2.0000   8.0000   10.5000

12.0000   10.0000   27.0000

**Inverse of a matrix:**

Inverse of a matrix A is given by inv(A).

```
>> q=[1,4,8;5,4,9;7,6,4]
```

q =

1   4   8

5   4   9

7   6   4

```
>>inv(q)
```

ans =

-0.2533   0.2133   0.0267

0.2867   -0.3467   0.2067

0.0133   0.1467   -0.1067

**How to access elements of a matrix:**

```
>> mat=[1,4,5;5,8,9;4,7,3]
```

mat =

1   4   5

5   8   9

```
4 7 3
```

```
>> t=mat(2,3)
```

```
t =
```

```
9
```

```
>> mat(1,3)+mat(2,1)
```

```
ans =
```

```
10
```

Note that, in this case you have to use parentheses.

If you want all the elements of a dimension of a matrix, use colon.

Suppose, we have a three dimensional (2X3X3) matrix named mat3. mat3(:, :, 1) will give the elements of all the rows and the columns of the matrix.

```
>> mat3(:, :, 1)
```

```
ans =
```

```
1 4 5
```

```
8 7 3
```

```
>> mat3(:, :, 2)
```

```
ans =
```

```
5 4 6
```

```
1 8 4
```

```
>> mat3(:, :, 3)
```

```
ans =
```

```
8 7 2
```

```
9 5 1
```

mat3(:, 2, 3) will give all the elements of the second column of the third layer of the matrix.

```
>> mat3(:, 2, 3)
```

```
ans =
```

```
7
```

```
5
```

**How to know the dimensions of a matrix:**

```
>> [r,c,l]=size(mat3)
```

```
r =
```

```
2
```

```
c =
```

```
3
```

```
l =
```

```
3
```

If you want to ignore one dimension, e.g. number of layers:

```
>> [r,c,~]=size(mat3)
```

```
r =
```

```
2
```

```
c =
```

```
3
```

Note that `[r,c]=size(mat3)` will give wrong result here.

```
>> [r,c]=size(mat3)
```

```
r =
```

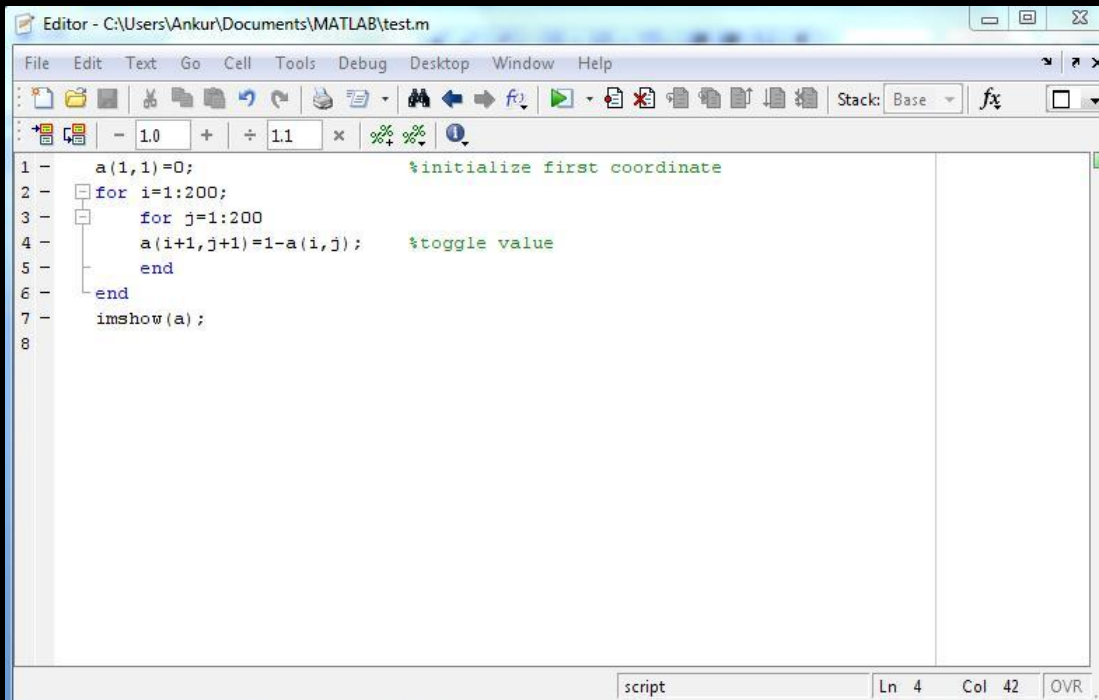
```
2
```

```
c =
```

```
9
```

## Making M Files & Functions in MATLAB

It is a provision in MATLAB where you can execute multiple commands using a single statement. Here the group of commands is stored as a MATLAB file (extension .m).

A screenshot of the MATLAB Editor window. The title bar reads 'Editor - C:\Users\Ankur\Documents\MATLAB\test.m'. The menu bar includes File, Edit, Text, Go, Cell, Tools, Debug, Desktop, Window, and Help. The toolbar contains various icons for file operations, editing, and execution. The script editor shows the following code:

```
1 - a(1,1)=0;           %initialize first coordinate
2 - for i=1:200;
3 -     for j=1:200
4 -         a(i+1,j+1)=1-a(i,j); %toggle value
5 -     end
6 - end
7 - imshow(a);
8
```

Comments are highlighted in green. The status bar at the bottom indicates 'script', 'Ln 4', 'Col 42', and 'OVR'.

Here we have saved the m-file by the name “test.m”. Now as you type  
>>test

And in MATLAB command window, all the above commands will execute.

**Comments:** As we have comments in C/C++/ Java using double slash (//), in MATLAB we use symbol % to write comments, i.e., statements that are not considered for execution. You can see comments in green in the snapshot above.

So, that’s the end of this matrix operation tutorial, let’s move on....



# Image processing in MATLAB

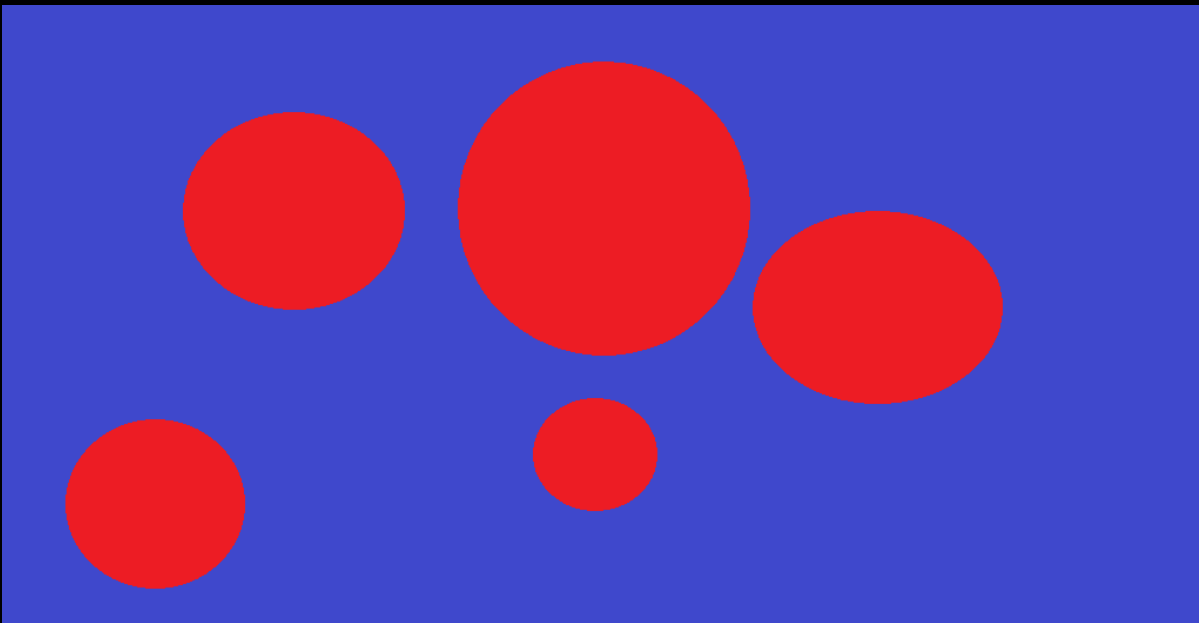
**Reading an image:**

```
>>img=imread('image.png');
```

**Display an image:**

```
>>imshow(img);
```

Let's take an image as example. For the rest of this tutorial, we will use this image as example.



As we know that all digital images are basically matrices, img is a three dimensional matrix, and we can apply all matrix operations on it.

Here, the origin of the co-ordinate system is in the upper left corner.

Now, to know the number of rows and columns of this image:

```
>> [row,col,~]=size(img)
```

```
row =
```

```
526
```

```
col =
```

```
1025
```

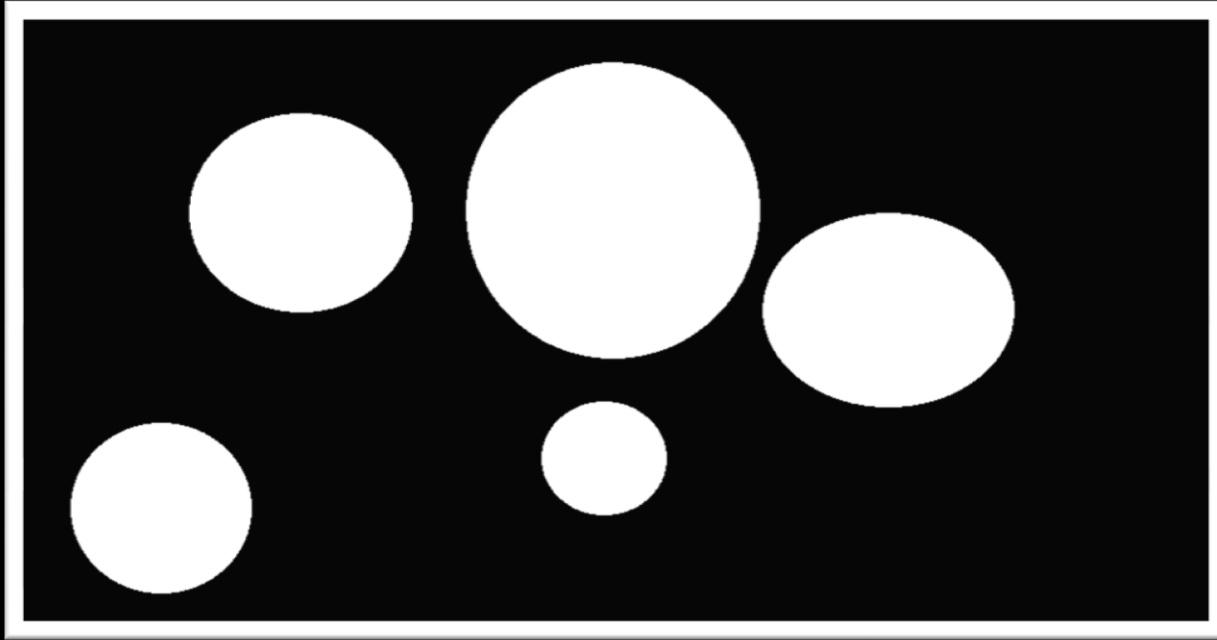
Now, to process this image, we must convert it into a binary image. Here, we will do it assuming RGB image.

To convert it into a binary image, we first create a zero matrix having same number of rows and columns as the image. Then, we check the R, G and B values of each pixel of the image. If the values are beyond a given threshold, we make the value of that pixel in the zero matrix equal to one.

*Example:*

```
img=imread('image.png');  
[row,col,~]=size(img);  
preimg=zeros(row,col);  
for i=1:row  
    for j=1:col  
        if((img(i,j,1)>200)&&(img(i,j,2)<100)&&(img(i,j,3)<100))  
preimg(i,j)=1;  
    end  
end  
end  
imshow(preimg);
```

The output of this script will be:



The threshold should be changed as required.

### **How to label connected components:**

To do this, we use the function `bwlabel()`.

*Syntax:*

```
[L,num] = bwlabel(BW)
```

Returns label matrix `L` that contains labels for the 8-connected objects found in `BW`. The label matrix, `L`, is the same size as `BW`. It also returns `num`, the number of connected objects found in `BW`.

*Example:*

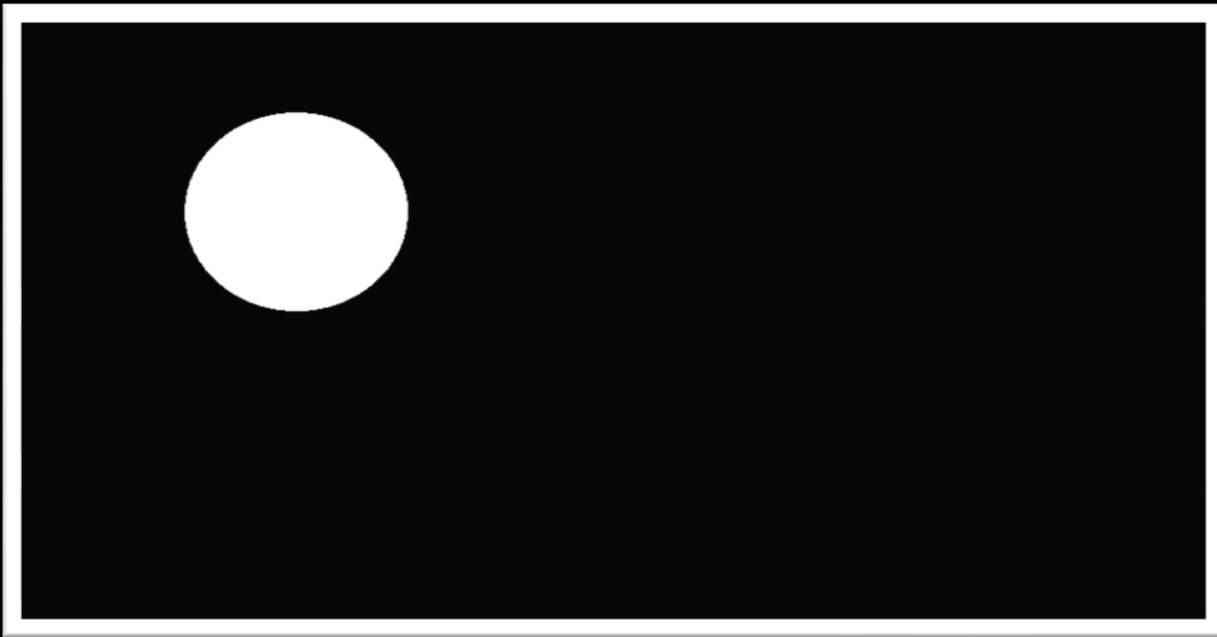
```
>> [Label,num]=bwlabel(preimg);
```

```
>> num
```

```
num =
```

```
5
```

```
>> imshow(Label==2);
```



### How to measure properties of region in image:

To do this, we use regionprops function

*Syntax:*

```
stats = regionprops(L,properties)
```

It returns measurements for the set of properties specified by `properties` for each labeled region in the label matrix `L`.

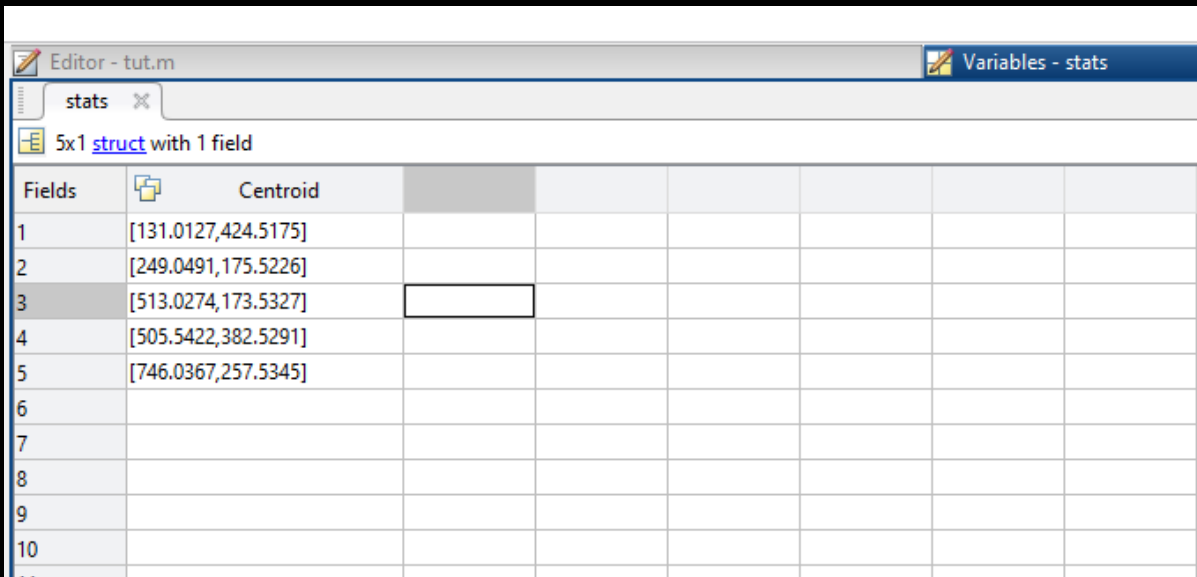
The properties which can be obtained by regionprops are:

'Area','BoundingBox','Centroid','ConvexArea','ConvexHull','ConvexImage','Eccentricity','EquivDiameter','EulerNumber','Extent','Extrema','FilledArea','FilledImage','Image','MajorAxisLength','MinorAxisLength','Orientation','Perimeter','PixelIdxList','PixelList','Solidity','SubarrayIdx'.

*Example:*

If you want to find the centroids of the white blobs in the binary image labelled in the last example:

```
>>stats=regionprops(Label,'Centroid');
```



Fields	Centroid
1	[131.0127,424.5175]
2	[249.0491,175.5226]
3	[513.0274,173.5327]
4	[505.5422,382.5291]
5	[746.0367,257.5345]
6	
7	
8	
9	
10	

As you can see, the centroids of the blobs have been stored in stats.

```
>>stats(1).Centroid
```

ans =

```
131.0127 424.5175
```

```
>>stats(2).Centroid(1)
```

ans =

```
249.0491
```

```
>>stats(2).Centroid(2)
```

ans =

```
175.5226
```

Similarly, if you want to get major axis length and minor axis length of every blob:

```
>>st=regionprops(Label,'MajorAxisLength','MinorAxisLength');
```

Editor - tut.m

Variables - st

stats st

5x1 struct with 2 fields

Fields	MajorAxisLength	MinorAxisLength								
1	152.9779	143.9133								
2	188.8724	167.9454								
3	249.8542	248.8448								
4	105.9327	95.9625								
5	212.8144	163.9366								
6										
7										
8										
9										
10										

### To find Area:

- The total number of 'ON' pixels in the image.

The number of ones in the matrix is 8.

### To find Centroid:

- Find the row and column having pixel value one. Eg.[row,column]=find(label==1)

Row=[ 1 2 2 2 3 1 2 3]

Column=[ 1 1 2 3 3 4 4 4]

- Find the mean of the row and column having pixel value one.

Mean of Row=2 and mean of column= 2.75

### To find the Bounding Box:

- We need 4 points, starting position(x,y) , length and breadth.
- Minimum value of row and column minus 0.5 gives starting position(x,y) respectively

- Minimum value of row= $1-0.5=0.5$
- Minimum value of column= $1-0.5=0.5$
- Maximum value of column – minimum value of column+1 gives breadth of the box
- Maximum value of column=4
- Max value-min value of column= $3+1$
- Maximum value of row- minimum value of row +1 gives length of the box
- maximum value of row=3
- Max value – Min value= $2+1$
- Bounding Box value for the given example: 0.5000   0.5000   4.0000   3.0000

### **To find the Perimeter**

- Find the boundary of the labeled component

Boundary pixels:

```

1  1
2  2
2  3
1  4
2  4
3  4
3  3
2  2
2  1
1  1

```



- Find the distance between the each adjoining pair of pixels around the border of the region.

- Use the distance formula:  $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

- For instance, calculate the distance between the two points (1,1) and (2,2).  
distance=sqrt((2-1).^2+(2-1).^2)=1.41

- Similarly, the distance is computed for all the pixel positions.

- The perimeter for the given example is 10.2426

### To find the Roundness:

- Roundness of an object can be determined using the formula:

$$\text{Roundness} = (4 * \text{Area} * \pi) / (\text{Perimeter}^2)$$

If the Roundness is greater than 0.90 then, the object is circular in shape.

$$\text{Result} = (4 * 8 * 3.14) / 10.2426^2 = 0.9582$$



## **MATLAB CODE:**

%Measure Basic Image Properties without using 'regionprops' function

%Measure Area, Perimeter, Centroid and Bounding Box

clc

%Read Original Image

I=imread('coins.png');

%Convert to Binary

B=im2bw(I);

%Fill the holes

C=imfill(B,'holes');

%Label the image

[Label,Total]=bwlabel(C,8);

%Object Number

num=4;

[row, col] = find(Label==num);

%To find Bounding Box

sx=min(col)-0.5;

sy=min(row)-0.5;

breadth=max(col)-min(col)+1;

len=max(row)-min(row)+1;

BBox=[sx sy breadth len];

display(BBox);

```
figure,imshow(I);  
  
hold on;  
  
x=zeros([1 5]);  
  
y=zeros([1 5]);  
  
x(:)=BBBox(1);  
  
y(:)=BBBox(2);  
  
x(2:3)=BBBox(1)+BBBox(3);  
  
y(3:4)=BBBox(2)+BBBox(4);  
  
plot(x,y);
```

```
%Find Area
```

```
Obj_area=numel(row);
```

```
display(Obj_area);
```

```
%Find Centroid
```

```
X=mean(col);
```

```
Y=mean(row);
```

```
Centroid=[X Y];
```

```
display(Centroid);
```

```
plot(X,Y,'ro','color','r');
```

```
hold off;
```

```
%Find Perimeter
```

```
BW=bwboundaries(Label==num);
```

```
c=cell2mat(BW(1));
```

```
Perimeter=0;
```

```
for i=1:size(c,1)-1
```

```
Perimeter=Perimeter+sqrt((c(i,1)-c(i+1,1)).^2+(c(i,2)-c(i+1,2)).^2);
```

```
end
```

```
display(Perimeter);
```

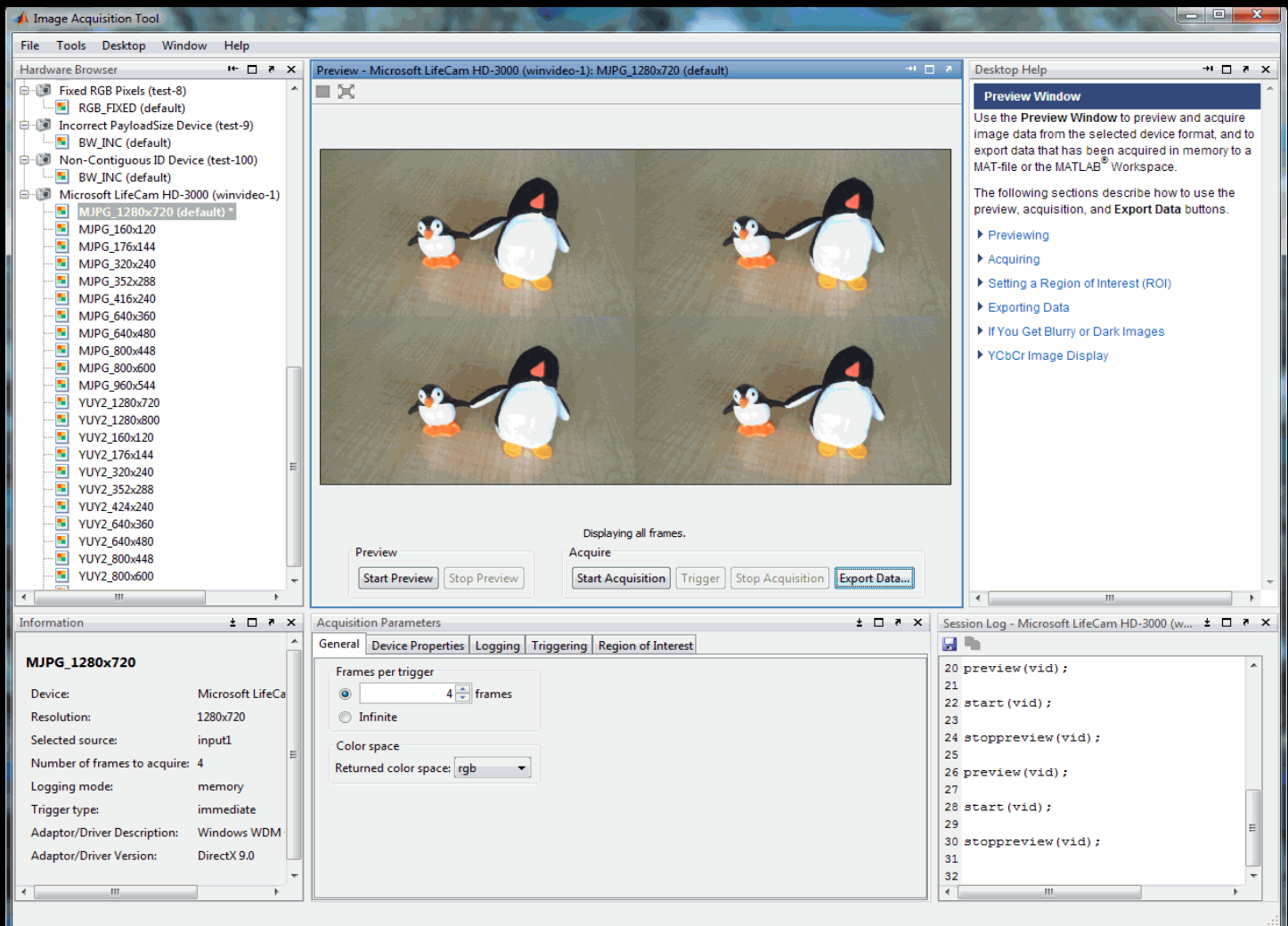
## TO SEE A COMPLETE LIST OF REGIONPROPS USAGE:

<https://in.mathworks.com/help/images/ref/regionprops.html>

## To get help in image acquisition in MATLAB:

<https://in.mathworks.com/help/imaq/>

<https://in.mathworks.com/help/imaq/the-image-acquisition-tool-desktop.html>



Please choose the ideal resolution or your code may run wrong or slow....

# Arduino Serial communication:

## Hardware

- Arduino Uno
- USB cable for Arduino or hc 05

Start MATLAB and create an m-file and write the following code:

```
% ARDUINO SERIAL COMMUNICATION CHECK WITH MATLAB
%this code needs your Arduino to send a 'y' when m is sent through the serial console...
a = arduino('com3', 'uno');    % create an arduino object
fopen(arduino);
%check connection with bot-----
fprintf(arduino,'%c','m');
serIn=fscanf(arduino,'%c');
if(serIn=='y')
    h=msgbox('Communication is working');
end
fclose(arduino);
clear a % end communication with arduino
```

## IF YOU WANT TO USE BLUETOOTH:

```
arduino= Bluetooth('HC-05',1);
```

```
fopen(arduino);
```

OR YOU CAN USE THE USB CABLE.

YOU HAVE TO EDIT THE THIRD LINE OF THE CODE AND GIVE THE CORRECT COM PORT FROM THE DEVICE MANAGER AND CORRECT BOARD MODEL.

# **THANK YOU !!**

- CONTACTS:-

- PUNYAJAY SAHA : 9968982501
- RANIT CHATERJEE : 9088179430
- SUPROTIK DEY : 8334826685
- SAYAN GANGULY : 9038731044
- VIVEK SHARMA : 9804023431
- SOHAM PODDAR : 9163642188
- ROHIT RAY : 8017894525

- VISIT US AT: <http://robodarshan.in/>  
<https://www.facebook.com/robodarshan/>

***There's nothing a bot can't do!!.....***

