

## 1.1 Computer Vision

### 1.1.1 Definition of Computer Vision

We humans, have a very robust visual system, which helps us to identify people and objects, play sports, perform operations, drive vehicles, read, and so on.

Although it might seem that we do not put any special effort to do most of these tasks, human visual system is fairly complex to replicate and implement.

Computer Vision, in the simplest terms, is the automation of such a visual system, so that computers or machines, in general, can obtain high level understanding of the environment from Digital Images & Videos.

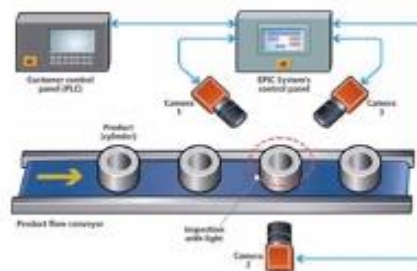
Let us understand a bit more about why is Computer Vision important, by looking at a few applications of Computer Vision.

### 1.1.2 Applications of Computer Vision

- Machine Vision

In the manufacturing sector, identifying defective products and ensuring quality and accuracy is of utmost importance.

Machine vision, is the technology used to provide imaging-based automatic inspection and analysis. It includes applications such as automatic inspection, process control, and robot guidance, usually in industry.



## ■ Object Identification & Tracking

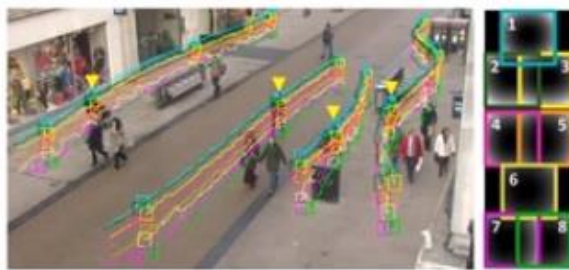
Object detection, deals with detecting instances of objects of a certain class, such as humans, buildings, or cars in digital images and videos.

Computer Vision is vital in implementing object detection from digital images.



## ■ Detection of Events

Detection of events or Activity recognition, aims to recognize the actions of one or more objects, based on a series of observations. For example, By automatically monitoring human activities, emergency health care services can be provided for people suffering from traumatic health concerns. It can also be used for security purposes like surveillance against intruders and/or animals.



## ■ Human–computer interaction Devices

Human–computer interaction (HCI) researches the design and use of computer technology, Computer Vision helps in creating such interfaces and making them as interactive as possible.



## ■ Mapping of Environments

An autonomous robot should be able to construct (or use) a map or floor plan and to localize itself and its recharging bases in it. Computer Vision is an important tool in understanding the environment and creating a visual map to enable this in the field of robotics.



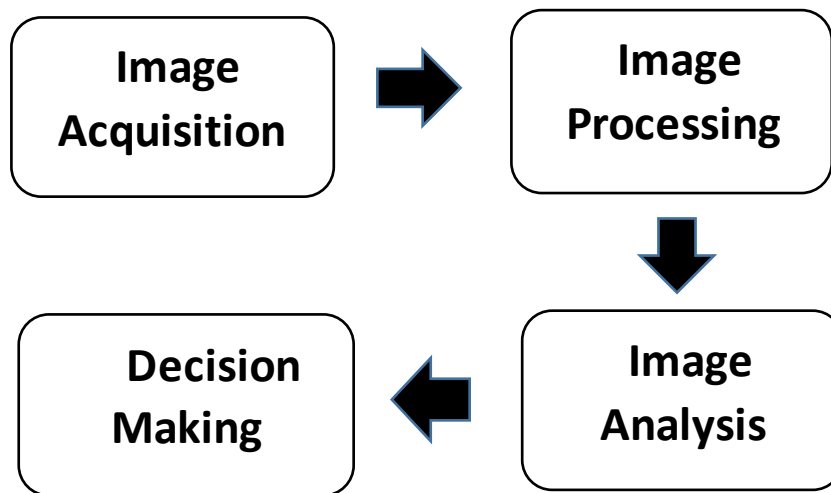
## ■ Autonomous Navigation

Self-driving cars are one of the most exciting upcoming technologies. The navigation of such autonomous driving systems uses computer vision algorithms. It also uses optical sensors including laser-based range finder and photo-metric cameras, to extract the visual features required for their mobility.



So, as you understand now, the applications of Computer Vision are wide and deep.

### 1.1.3 Computer Vision Pipeline



- Any computer vision application starts with Image acquisition. Image acquisition is the digital representation of the visual characteristics of the physical world. Image sensors are used to detect and capture the information required to make an image. Digital Cameras, Medical Imaging equipment, thermal imaging devices, RADAR, LIDAR, etc are some examples of image sensors.
- The images acquired are then processed in the next stage. In this step, the signals in the acquired images are filtered to remove the noise or any irrelevant frequencies. If needed the images are padded and transformed to a different space, so as to make them ready for the actual analysis.
- The processed images are then analyzed to extract useful information. This involves pattern identification, color recognition, object recognition, feature extraction, motion tracking, image segmentation, etc.

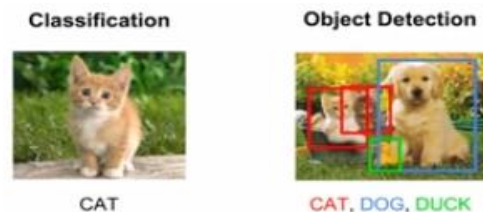
- Finally, the high dimensional data obtained from all the above steps is used to produce meaningful numerical information, which leads to making decisions.

In this project, you will follow the same pipeline to implement a computer vision driven project. Each stage will further be divided into smaller steps, based on the application and the project.

#### 1.1.4 Problems solved by Computer Vision

- **Object recognition & classification**

Object recognition and classification is where one or several pre-specified or learned objects can be recognized.



- **Pose estimation**

Pose estimation determines the position or orientation of a specific object relative to the camera.

An example application for this technique would be assisting a robot arm in retrieving objects from a conveyor belt in an assembly line situation or picking parts from a bin.



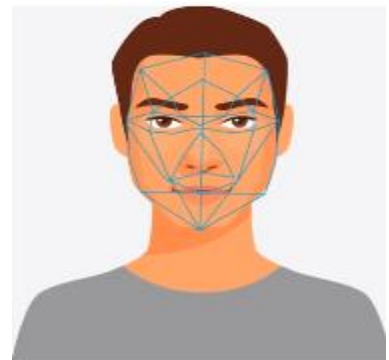
- **Optical character recognition**

Optical character recognition identifies characters in images of printed, or handwritten text, usually with an aim to make them readable and editable.



- **Facial recognition**

Facial recognition system is a Computer Vision based technology, capable of identifying or verifying a person from a digital image or a video frame from a video.



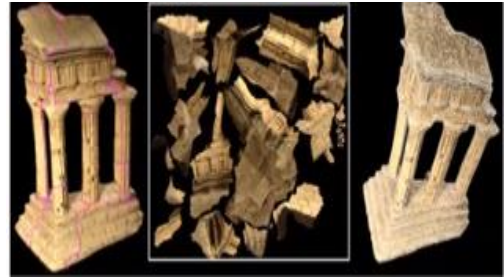
- **Motion Tracking**

Motion tracking is a process where an image sequence is analyzed to produce an estimate of the velocity of different objects in the image or in a 3D scene.



### ■ **scene reconstruction**

Given a few or more images of a scene, or a video, scene reconstruction aims at computing a 3D model of the scene. More sophisticated methods produce a complete 3D surface model.



### ■ **Image restoration**

Image restoration is the process of reconstructing lost or deteriorated parts of images and videos.



## **1.1.5 Tools used for Computer Vision**

### ■ **Open CV**

- Stands for 'Open Source Computer Vision'
- Library of functions for real-time Computer Vision
- Developed in C++
- Supports Python, Java, MATLAB and OCTAVE

## **1.2 Installation of Python 3 in Windows**

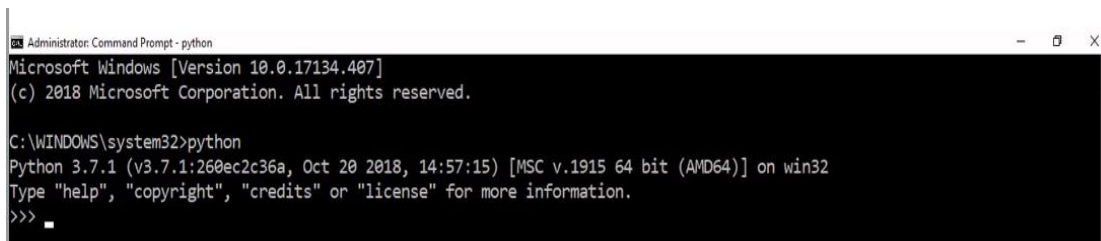
To install the python programming language on your computer, Open the official website of python, [python.org](https://python.org), Go to the tab of downloads in the website, and click on Windows. You will find many versions of python. You will use one of the Python 3 versions for this project. Choose the top most one, which is



usually the latest and the most stable release. You will have different types of installers available, for each version of Python.

I recommend you to download the executable installer, which will be a straight forward installation. Before selecting the file, you should know the architecture of your computer. If it is 64 bit or a 32 bit computer.

Once the installer is downloaded, open the file, by double clicking on the same to setup. And To verify the installation of the python, First, in the start menu, type CMD to search for the command prompt, then right click on it and select "Run as administrator", just type python and hit enter, then It will also display the version of python installed on your computer as shown below, If it throws an error, then the installation of python was not done properly.

A screenshot of a Windows Command Prompt window titled "Administrator: Command Prompt - python". The window shows the following text: "Microsoft Windows [Version 10.0.17134.407] (c) 2018 Microsoft Corporation. All rights reserved. C:\WINDOWS\system32>python Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:57:15) [MSC v.1915 64 bit (AMD64)] on win32 Type "help", "copyright", "credits" or "license" for more information. >>>". The prompt is at the end of the last line, indicating the command has been executed successfully.

```
Administrator: Command Prompt - python
Microsoft Windows [Version 10.0.17134.407]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>python
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:57:15) [MSC v.1915 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

### 1.3 Installation of Open CV & Other Packages for Image Processing

In addition to Python, you will need additional packages for the image processing project you will work on. You will install the Open CV, NumPy and Matplotlib libraries. These libraries are used for various applications in this project. That can be achieved by First, in the start menu, typing CMD to search for the command prompt, and right clicking on it and selecting "Run as administrator".

You will first update pip, which is a package management system, that will be used to install the rest of the packages. Just type the following command in the command:

```
python -m pip install -U pip
```



This will check for the previous version of PIP and will uninstall the file and will update to the latest version, ensure that you have internet connection,

```
C:\WINDOWS\system32>python -m pip install -U pip
Collecting pip
  Using cached https://files.pythonhosted.org/packages/c2/d7/90f34cb0d83a6c5631cf71dfe64cc1054598c843a92b400e55675cc2ac37/pip-18.1-py2.py3-none-any.whl
Installing collected packages: pip
  Found existing installation: pip 10.0.1
    Uninstalling pip-10.0.1:
      Successfully uninstalled pip-10.0.1
```

Next, you will install the numpy package. NumPy is a general-purpose array-processing package. It provides functions and tools to access and work with multidimensional array objects. It is the fundamental package for scientific computing with Python. You will use the same pip package to install the numpy package. Hence the syntax remains the same as before except replace the pip at the end of the command with numpy :

```
python -m pip install -U pip numpy
```

That command will install the numpy library in your machine.

```
C:\WINDOWS\system32>python -m pip install -U pip numpy
Requirement already up-to-date: pip in c:\program files\python37\lib\site-packages (18.1)
Collecting numpy
  Downloading https://files.pythonhosted.org/packages/00/0e/5a8c34adb97fc1cd6636d78050e575945e874c8516d501421d5a0f377a6c/numpy-1.15.4-cp37-none-win_amd64.whl (13.5MB)
    100% |#####| 13.5MB 766kB/s
Installing collected packages: numpy
  Found existing installation: numpy 1.15.3
    Uninstalling numpy-1.15.3:
      Successfully uninstalled numpy-1.15.3
  Successfully installed numpy-1.15.4
C:\WINDOWS\system32>
```

For plotting images and data on plots, you will be using the matplotlib package. Matplotlib is a Python 2D plotting library which has similar functionalities as the MATLAB software. By using the same command as earlier to install matplotlib, Replace numpy with matplotlib and hit enter.

```
Administrator: Command Prompt - python -m pip install -U pip matplotlib
C:\WINDOWS\system32>python -m pip install -U pip matplotlib
Requirement already up-to-date: pip in c:\program files\python37\lib\site-packages (18.1)
Collecting matplotlib
  Downloading https://files.pythonhosted.org/packages/5c/ee/efaf04efc763709f6840cd8d08865d194f7453f43e98d042c92755cdddec/matplotlib-3.0.2-cp37m-win_amd64.whl (8.9MB)
    100% |#####| 8.9MB 965kB/s
Requirement already satisfied, skipping upgrade: numpy>=1.10.0 in c:\program files\python37\lib\site-packages (from matplotlib) (1.15.4)
Requirement already satisfied, skipping upgrade: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in c:\program files\python37\lib\site-packages (from matplotlib) (2.3.0)
Requirement already satisfied, skipping upgrade: kiwisolver>=1.0.1 in c:\program files\python37\lib\site-packages (from matplotlib) (1.0.1)
Requirement already satisfied, skipping upgrade: cyclor>=0.10 in c:\program files\python37\lib\site-packages (from matplotlib) (0.10.0)
Requirement already satisfied, skipping upgrade: python-dateutil>=2.1 in c:\program files\python37\lib\site-packages (from matplotlib) (2.7.5)
Requirement already satisfied, skipping upgrade: setuptools in c:\program files\python37\lib\site-packages (from kiwisolver>=1.0.1->matplotlib) (39.0.1)
Requirement already satisfied, skipping upgrade: six in c:\program files\python37\lib\site-packages (from cyclor>=0.10->matplotlib) (1.11.0)
Installing collected packages: matplotlib
  Found existing installation: matplotlib 3.0.1
  Uninstalling matplotlib-3.0.1:
```

This will install the matplotlib library in your machine.

Finally you will install the Open CV library you can use the same command as above and replace with opencv hyphen python. This will install open cv a fresh or update any previous installation.

```
Successfully installed matplotlib-3.0.2
C:\WINDOWS\system32>python -m pip install -U pip opencv-python
Requirement already up-to-date: pip in c:\program files\python37\lib\site-packages (18.1)
Requirement already up-to-date: opencv-python in c:\program files\python37\lib\site-packages (3.4.3.18)
Requirement already satisfied, skipping upgrade: numpy>=1.14.5 in c:\program files\python37\lib\site-packages (from opencv-python) (1.15.4)
C:\WINDOWS\system32>exit_
```

You will now check whether the packages are installed properly, or not, by opening Python Shell from the Start Menu In the shell window displayed, trying to import the packages by typing the command import followed by the library name.

```
Python 3.7.1 Shell
File Edit Shell Debug Options Window Help
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:57:15) [MSC v.1915 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import numpy
>>> import matplotlib
>>> import cv2
>>> |
```

Once all the packages are tested, your python software is ready for programming for image processing.

## **1.4 Setting up the Working Directory for OpenCV**

First search for Python and open the Python IDLE. When you click on IDLE, the Python Shell will open. IDLE is Python's Integrated Development and Learning Environment. IDLE has two main window types, the Shell window and the Editor window. The shell window is used for writing and testing the code live. The code will not be saved for further use.

The editor is used for writing scripts that can be saved as python files, which can be run independently and repeatedly. You can open the editor by clicking on file menu and the new file option. Each of the windows have very simple and standard menus and menu items. Note that you will have to store the python script and the images or videos required to run the code in the working directory. Else, you will have to provide the entire path of the file, which you had stored.

Now, go ahead and save the empty python file you had opened in the working directory. Note that you will have to save the file opened in the editor and not the shell. Next, when you have to open the python file, for further usage, you will have to remember the following.

Double clicking on a python script file, will directly run the python file and execute the program written in it. Or open the file from the python shell file menu using the Open option.

## **1.5 Reading an Image using Open CV**

Open Python 3.7 that you had installed. Open a new file in the editor to write the python program for reading the image.

As we had learnt, you will use the Open CV to carry out all the image and video operations needed for this project. So, you will import the open CV library. You will write `import cv2`, which is the name of the open CV library. You will follow this with the name you want to give this library, which will be used to refer the functions and objects of this library. For simplicity, you can name it as `cv`. Note that this is optional, In case you do not do this, you will refer to the functions of the library using `cv2` as the prefix, instead of `cv`.

Similarly, you will import the numpy package of python and rename it as `np`, for simplicity. This package is important to conduct all the array operations on the images, which you will learn in further modules. You will now read the image with the name `hazard10 dot jpg`, which you have in your working folder. To read the image, you will use the **`imread`** function of the Open CV library. Note that to call any function of the open cv library, you will have to prefix it with the library name, which we renamed as `cv`, The input for the **`imread`** function, should be the file name, as a string, in case the file is in the same directory as the code. Else, you will have to provide the entire path for the file, as a string.

Here, the name of the file is `hazard 10 dot jpg`, which you have to include in quotes, to pass it as a string. You will assign the output of the read function to a variable, named as `img` here, to access the image for further operations. Now, use the function **`cv.imshow`** to display an image in a window. The window automatically fits to the image size. First argument in the function is a window name in which the image displayed this a string. The second argument is your image. You can create as many windows as you wish, but with different window names.

You will end the program by closing the image windows by pressing any key on the keyboard. **cv.waitKey()** is a keyboard binding function. Its argument is the time in milliseconds. The function waits for specified milliseconds for any keyboard event. If you press any key in that specified time, the program continues. If 0 is passed, it waits indefinitely for a key stroke. It can also be set to detect specific key strokes like, if key a is pressed etc which you will learn in further modules. **cv.destroyAllWindows()** simply destroys all the windows you had created. If you want to destroy any specific window, use the function **cv.destroyWindow()** where you pass the exact window name as the argument. Now, save the python file in the same folder as the one which has the image, hazard10.

That is the code :

```
import cv2 as cv
import numpy as np
img = cv.imread('C:\\Users\\pro\\Desktop\\hazard10.jpg')
cv.imshow("Image",img)
waitKey(0)
destroyAllWindows()
```

Run the script by pressing the F5 key or using the Run Module option in the Run Menu. The image will be displayed on your screen, as shown.

You can press any key on the keyboard, to destroy the image display window and stop the script. You have written the first python code using Open CV to read and display an image.



You will now use a second argument for the **imread** function, to specify the way the image should be read. If you pass the integer **-1**, as the flag, the image is loaded as it is including the alpha channel. If you pass the integer **1**, as the flag, the image is loaded as a color image and any transparency of the image will be neglected. Note that this is the default flag, if the second argument is not provided. Now, assign the loaded gray scale image to a new variable **img1**, Use the **imshow** function, to display the **img1** image, in a window named image1. You will also save this gray scale image.

The **imwrite** function is used to do the same. The First argument of the **imwrite** function is the file name and the second argument is the image you want to save in the file you specified.

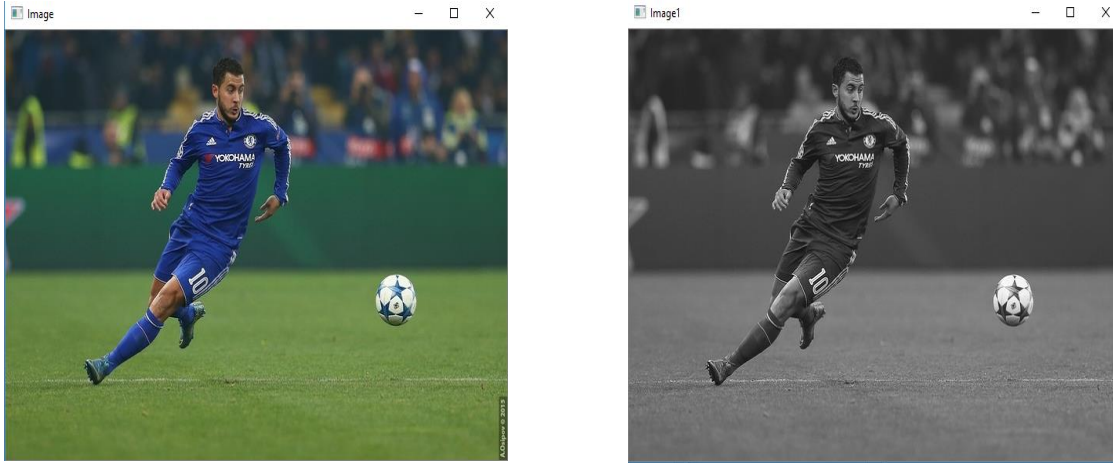
Here, you will save the grayscale image, stored in **img1**, in the file named as gray hazard dot jpg. This will save the image in JPG format in the working directory.

That is the code :

```
import cv2 as cv
import numpy as np
img = cv.imread('C:\\Users\\pro\\Desktop\\hazard10.jpg')
img1 = cv.imread('C:\\Users\\pro\\Desktop\\hazard10.jpg',0)
cv.imshow("Image",img)
cv.imshow("Image1",img1)
cv.imwrite('hazard10.png',img1)
cv.waitKey(0)
cv.destroyAllWindows()
```

Save the edited file and run it. You can see that now, two images, the original and the gray scale image are displayed, in 2 different windows, named image and image1, which are the labels you had specified. You can press any key on the keyboard, to destroy the display windows and stop the script. Also, you can see that in the working directory, a new file named gray hazard dot jpg has

appeared. Open the image file in your default picture application, and see that the gray scale image, which was displayed earlier, is saved in the file.



## 1.6 Saving Image using waitKey Function in Open CV

You had learned how to save an image, When any key on the keyboard is pressed. You will now see how that can be done, when a specific key is pressed. First import the CV - and numpy libraries as CV and NP respectively. You will read and display the hazard 10 image using the **imread** and **imshow** functions as you had done earlier. You will now assign the waitKey function to a variable name key. The wait key function gives out the Unicode value of the key pressed as the output.

The aim now is to close the display windows only if the Escape key is pressed. The unicode value of the escape key is **27**. So you will write an if conditional to verify if the key pressed is the escape key and destroy all windows in case it is true. Save and run the code



```
import cv2 as cv
import numpy as np
img =
cv.imread('C:\\Users\\pro\\Desktop\\hazard10.jpg')
cv.imshow("Image",img)
key = cv.waitKey(0)
if key == 27:
    cv.destroyAllWindows()
```

Save and run the code. Now try to press any key on your keypad other than the Escape key. You will see that the display window is not closed as in the earlier case. Now press the Escape key on your keyboard to close the window.

You will now write the code to display the gray scale image as well and save it if the key s is pressed. If the escape key is pressed the windows should be closed without saving. Read the image in the grayscale format using the flag zero in the imread function and assign it to a new variable img1. Display the same in a window named image one using the imshow function.

Now you will add an else--if structure after the if condition which you had written earlier. You will check if the key pressed is the key s of the keyboard. This is done by using the Ord function, which gives out the unicode value of the pressed key, When this condition is satisfied you will save the image using the **imwrite** function as you had done earlier You will then close all the windows.

That is the code:

```
import cv2 as cv
import numpy as np
img = cv.imread('C:\\Users\\pro\\Desktop\\hazard10.jpg')
img1 = cv.imread('C:\\Users\\pro\\Desktop\\hazard10.jpg',0)
## the flag,0, will load the image in grayscale mode
cv.imshow("Image",img)
cv.imshow("Image1",img1)
key = cv.waitKey(0)
if key == 27:
    cv.destroyAllWindows()
elif key == ord('s'):
    cv.imwrite('hazard1.jpg',img1)
    cv.destroyAllWindows()
```

Save and run the code. Now if you press the Escape key the display windows will just get closed. But if you press the key **s** the image will be saved as gray hazard1.jpg which is the file name you had provided earlier in the **imwrite** function. You can use the wait key function similarly for performing different operations based on the key pressed.

## 1.7 Computer Vision – Basic Concepts of a Image

### 1.7.1 What is an Image?

- Pixels → Picture Elements

An image is made of numerous pixels. Pixels are short for picture elements. All the pictures that you see, including the ones on your computer and mobile screens are made up of small picture elements. Numerous such pixels form a picture.

- 2D Function →  $F(x, y)$
- $x, y$  → Spatial Coordinates
- $F$  → Intensity/Color of the image at  $(x, y)$

So, an image can be known as a 2 dimensional function of  $x$  and  $y$ , where  $x$  and  $y$  determine the spatial coordinates of a pixel and the function determines the intensity or the color of the

pixel. So, reading, understanding and manipulating an image comes to down to accessing these pixels and reading and changing the function.

### **1.7.2 Pixel Reference**

Let us now see how these pixels are organized in an image and how to access and read them. Any digital image can be understood as an array of pixels arranged as a combination of rows and columns.

Now, let us see how they can be referred to. Unlike the traditional x and y axes, the axes of an image are drawn. The x-axis is along the columns, increasing from left to right. And, the y-axis is along the rows, increasing from top to bottom. The number of pixels on the x-axis determines the width of the image. Whereas the number of pixels on the y-axis determines the height of the image.

The origin of an image is at the top left center of the image, and not at the bottom left, as is the case in traditional coordinate system. So, to refer to a particular pixel, you will refer to the row number and the column number of a pixel. Note that the index starts at 0 and not at 1. So, an image is essentially a 2D spatial array, of pixels.

And, each array element stores the value of the pixel, which is either the intensity function or the color function. Now that you understood the spatial configuration of the pixels, \let us understand the function or the value of each pixel.

### **1.7.3 Color Spaces in Images**

Each pixel has a combination of Red, Green and Blue values, which correspond to the color of the pixel. So the 2D array of the image can be split into 3 channels, with each channel corresponding to one color. For a particular pixel at a particular

location, there will be 3 channels or elements. Each channel corresponds to an individual color. And the combination of these colors will determine the color of the pixel. So, in essence, a colored image can be thought of as a 3D array, with 3 channels of 2D arrays corresponding to individual pixels. Each color is in turn represented by an 8 bit integer, which lies between 0 to 255.

An image on your computer screen is made of multiple pixels. Take, for example, a pixel that is yellow colored. As mentioned, the yellow color will be obtained by the combination of red, green and blue colors. Yellow is a combination of red and green. So, the pixel, which has 3 channels corresponding to each color will have the following 8 bit data.

The red channel has the value 255, the maximum integer that can be stored in 8 bits. Similarly, the green channel has the value of 255. Whereas, the blue channel has the value of 0, stored in it. This combination of values, results in the yellow colored pixel. Similarly, different combinations of different values of each color channel, will result in different colors for the pixels. Let us now understand about color models, which will be essential in composing and manipulating an image.

#### **1.7.4 Color Models**

- Description of Colors and their representation

A color model is an abstract mathematical model describing the way colors can be represented as tuples of numbers, typically as three or four values or color components.

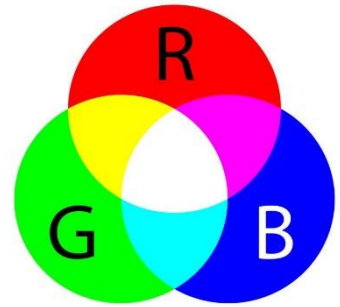
- Tuples of numbers

Ex: (R, G, B), (H, S, V), (C, M, Y, K), etc.

There are numerous color models designed for varied applications.

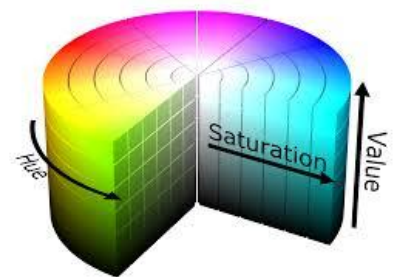
## RGB Color Model

- Red-Green-Blue Color Model
- Primary Colors → Cover most of human Color Space
- Colored Screens use RGB Model
- Additive Color Model → Colors created by mixing shades of different colors
- Represented by (R, G, B) → 3 channels  
Each color represented by 8 bits (0-255)



## HSV Color Model

- Hue-Saturation-Value/Brightness Color Model
  - Hue → Actual Color
  - Saturation → Amount of Grey added to the color
  - Value → Amount of White added to the color
- Useful for Color Based Operations & Color Manipulation
- Cylindrical Color Model
  - Hue → Angle (0 to 360)
  - Saturation → Absolute (0% - 100%)
  - Value → Absolute (0% - 100%)
- Represented by (H, S, V) → 3 channels



## 2. Basic Image and Video Operations using OpenCV

### 2.1 *Properties & Pixels of an Image - Basic Image Operations using OpenCV*

you will know how to read and print the shape and size of the image. You will also access individual pixels and change their properties.

Open a new file in Python IDLE to write the python program. Make sure the image that will be used for testing is present in the working folder. As usual, you will first import the cv2 and the numpy libraries, as cv and np, respectively. You will now read the image with the name hazard10 dot jpg, which you have in your working folder, using the cv.imread function. Now, use the function cv.imshow to display the image in a window. You will end the program with the cv.waitKey function, by passing 0 as the argument, which will wait indefinitely for a key press on the keyboard. You will follow up the waitkey function, with the destroy all windows function, to kill all the windows opened while performing testing.

you will write to find the properties of the image like shape, size and its datatype. It is known that a colored image is a 3 dimensional array, with the first 2 dimensions representing the size of the image and the 3rd dimension representing the number of color channels. We will use the print() function, which prints the specified message to the screen, or other standard output device. The message can be a string, or any other object, the object will be converted into a string before written to the screen.

You will print the shape and size of the image. Shape of image is accessed by img.shape function. It returns a tuple of number of rows, columns and channels (if image is color). The total number of array elements in an image is accessed by the function img.size.

Similarly, the type of values stored in each element of the array, is given by the img.dtype function. Save the file and Run the Python Code.

The code:

```
import cv2 as cv
import numpy as np
img = cv.imread('hazard10.jpg')
cv.imshow("Image",img)
print('The Shape of the Image is ')
print(img.shape)
print('The Size of the Image is ')
print(img.size)
print('The Datatype of the pixel values in the Image is ')
print(img.dtype)
cv.waitKey(0)
cv.destroyAllWindows()
```

The result:

The image will be displayed on the screen, along with the properties of the image on the shell window as shown.



The Shape of the image is a tuple with the number of rows, columns and channels. The 3 in place of channels represents the 3 color channel, which is BGR. Each pixel of an image, is an element



of the array represented by the image. So, you can access a pixel value by its row and column coordinates. For the default BGR image, it returns an array of Blue, Green, and Red values, in the same order.

Let us say, you want to access the pixel at 100, 100 and find its color properties and change them. You can select the pixel like any other element of an array. So, you will access the 100, 100 pixel by writing the array name, `img`, followed by the row and column numbers in square brackets. You can select the pixel 100 x 100 this way and assign it to a variable `px`. This will display the value of the image at the particular pixels in B G R format.

```
px = img[100, 100]
print('The Value of the pixel at 100x100 is ')
print(px)
```

If you wish to access only the blue color of the pixel, you will have to add the index of the third dimension, while accessing the pixel.

The blue color value will be the 0th element of the particular pixel. So, create a variable, `blue`, and assign it to `img` followed by 100, 100 and zero in square brackets, to access the blue color of the pixel at 100, 100.

```
blue = img[100,100,0]
print('The Value of the blue color in pixel at 100x100 is ')
print(blue)
```

For applying white, you have to update the array with 255 for all the color channels. Select the pixel from the image, using the array accessing you had done earlier, and assign the BGR color array, `[255,255,255]`, which corresponds to the white color.

```
img[100,100]=[255,255,255]
```

In the way we knew how to access the shape, size and individual pixels of an image.

## 2.2 Drawing Geometric Shapes using OpenCV

We will learn how to draw the basic geometric shapes on an image. To draw the geometric shapes, We will first create a black image as background for better visibility.

It is known that an image isn't three-dimensional array with the pixels as rows and columns and the channels as the third dimension. So you will create an array using the zeros function of the numpy library. The zeroes function takes two arguments the shape of the array as a tuple of integers and the desired datatype of the array elements.

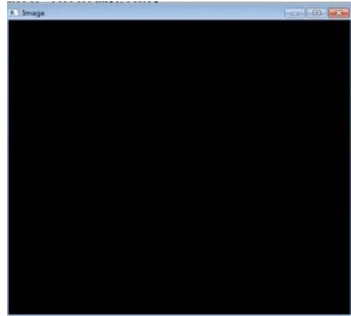
It returns an array of zeros with the given shape and datatype. An array of zeros is essentially an image with every pixel colored in black. We will assign the output of a zeros function to a variable named as IMG here to access the image for further operations, Now use the function NP zeros to create a black image in a window. First argument in the function is a tuple with the size of the images the first two elements. The number of channels is the third element. The second argument is the datatype of the returned array.

Now use the function C V dot imshow to display the image you created in a window. As you remember the first argument in the function is the window name, The second argument is your image. You will end the program by closing the image windows by pressing any key on the keyboard. You will use the CV dot waitkey function

```
import cv2 as cv
import numpy as np
#Create a black image
img = np.zeros((512,512,),np.uint8)
cv.imshow("Image",img)
cv.waitKey(0)
cv.destroyAllWindows()
```

which is a keyboard binding function. You see the dot destroy all windows that simply destroys all the windows you had created.

The result:



Now using this black image as the background you will draw some of the basic geometric shapes. Let's begin with drawing a simple colored line.

Ensure that you include the code for drawing the geometric shapes above the CV imshow function or else the imshow function will display just the black image. You will use the function CV dot line to draw a line by mentioning the two endpoints of the line.

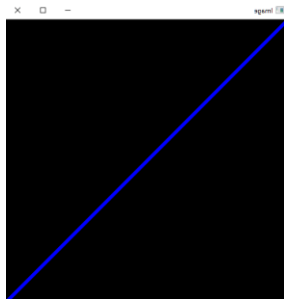
The first argument is the image on which the line is to be drawn. You are drawing a diagonal line. So the pixels must be the pixels on the left top corner and the right bottom corner which will be (0, 0) and (511, 511). So the next two arguments of the function are the Starting point of the line, which is (0, 0) and the ending point of the line, which is (511, 511).

The fourth argument is the color of the line in the BGR format. By giving the value of B as 255 and the remaining colors as 0 you will draw a blue colored line. You can edit the BGR combination and apply any other color to your line. The fifth argument is the thickness of the line in pixels.

```
import cv2 as cv
import numpy as np
# draw a black image
imag = np.zeros((512,512,3),np.uint8)
# draw a diagonal blue line with thickness of 5px by giving 2 end points
cv.line(imag,(0,0),(511,511),(255,0,0),5)
cv.imshow('Image',imag)
cv.waitKey(0)
cv.destroyAllWindows()
```

The result:

You can see the black background image with a blue line passing diagonally. Congratulations, you now know how to draw simple line on an image.

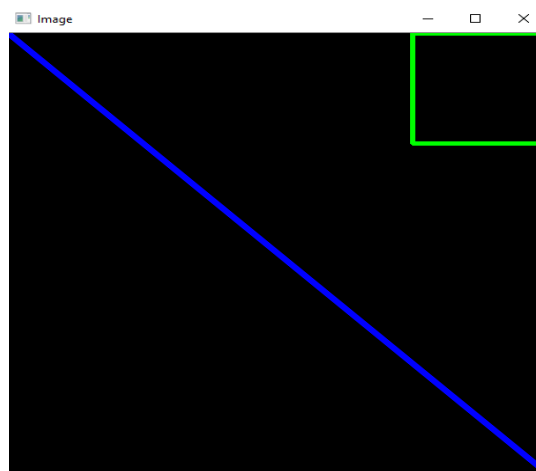


Let us now try to draw a rectangle, You will use the function `cv.rectangle` to draw a rectangle using the two opposite corners of a rectangle.

The parameters of this function are similar to that of the line function. Like the line function the first argument is the image file on which the rectangle is to be drawn. You will then mention the two opposite corners of the rectangle, the top left one and the bottom right one as the arguments. The fourth argument is the color of the rectangle edges in the BGR format. By giving the value of G as 255 and the remaining colors as 0 you will draw a green colored rectangle. By giving the value of G as 255 and the remaining colors as 0 you will draw a green colored rectangle. The fifth argument is for the thickness of the rectangle edges.

```
import cv2 as cv
import numpy as np
# draw a black image
imag = np.zeros((512,512,3),np.uint8)
# draw a diagonal blue line with thickness of 5px by giving 2 end points
cv.line(imag,(0,0),(511,511),(255,0,0),5)
# draw a green rectangle with thickness of 3px by giving 2 opposite corners
cv.rectangle(imag,(384,0),(510,128),(0,255,0),3)
cv.imshow('Image',imag)
cv.waitKey(0)
cv.destroyAllWindows()
```

The result:



## 2.3 Color Spaces in Image Processing - Change Color Space of an Image in Open CV

Open CV reads any image in the BGR format, by default.

We will now convert the image into a different colorspace. There are more than 150 color-space conversion methods available in OpenCV but we will see the basic ones. For color conversion, we use the `cv.cvtColor` function. The first argument for the `cvtColor` function is the image file that needs to be converted. The second argument is the code for the conversion type. There are numerous color spaces that OpenCV can handle, and therefore, quite a few conversion codes.

We will first convert the image from BGR to gray scale. So we will use the code, `cv.COLOR_BGR2GRAY`. The output of the `cvtColor` function is an image, which we will assign to a variable, named as `img_gray`. The shape of the grayscale image will have only height and width, because it does not have 3 channel of colors.

We will now convert the image from BGR to HSV where HSV stands for Hue, Saturation, and Value. The code to convert BGR to HSV is `COLOR_BGR2HSV` and we will assign the output of the `cvtColor` function to a variable, named as `img_hsv`

```
import cv2 as cv
import numpy as np

img = cv.imread("C:\\Users\\pro\\Desktop\\1image1.jpg")
cv.imshow('cHANGEcOLOR',img)
print('the shape of image is')
print(img.shape)
print('the soze of the image is')
print(img.size)

img_gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY) #CONVERSION FROM BGR TO GRAY
cv.imshow('grayiMAGE',img_gray)

img_hsv = cv.cvtColor(img,cv.COLOR_BGR2HSV) #CONVERSION FROM BGR TO HSV
cv.imshow('hvs image',img_hsv)

cv.waitKey(0)
cv.destroyAllWindows()
```



## 2.4 Plot an Image in Python using matplotlib

Matplotlib is a plotting library for the python programming language, and its numerical mathematical extension numpy. We will import the pyplot module from the matplotlib library, which will be useful for plotting the image. we will follow this with the name you want to give this library, which will be used to refer the functions and objects of this library. For simplicity, we can name it as plt.

Open CV reads an image in the BGR colorspace by default. But, the matplotlib refers to an image in the RGB colorspace. So, we will first convert the image into RGB colorspace, by passing the original image as an argument to the cvt Color function and using the cv.COLOR\_BGR2RGB color code for conversion.

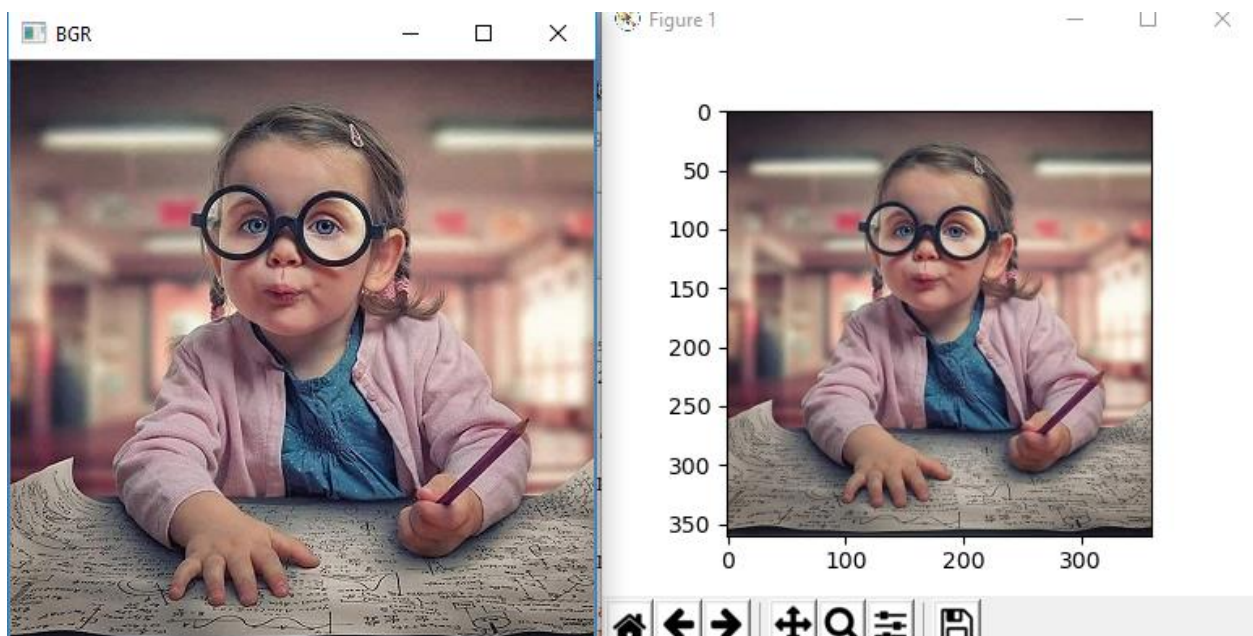
Note that the imshow function of the pi plot module works differently compared to the imshow function of the Open CV library. The imshow function of the pi plot takes the image to be drawn as an input. Also, unlike the imshow function of Open CV it does not display the image on the screen. It only draws the image on a plot. To display the plot, with the image drawn on it, we will have to use the show function of the pi plot module. So, we will write plt.show, to actually display the plot with the image on it.



```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img = cv.imread("C:\\Users\\pro\\Desktop\\1image.jpg")
cv.imshow('BGR',img)
img_RGB = cv.cvtColor(img,cv.COLOR_BGR2RGB)
plt.imshow(img_RGB)
plt.show()
cv.waitKey(0)
cv.destroyAllWindows()
```

Two windows will be displayed on your screen with the original image in each of them. The first window, with the title BGR, will be the typical opencv display window with the image. The other window, will have the image displayed on what resembles a plot, with the axis and the axis ticks set.

Displaying an image as a plot using matplotlib has its own advantages, as you will learn in further modules. We can also display multiple images in the same window, and compare them.



## 2.5 Region of Interest in an Image - Basic Image Operations using OpenCV

We will know how to access a region of interest and update it or change it.

Now, we are going to access a particular region of interest in an image. This is of interest in many operations of image processing, as a specific object or a feature of an image might be focused in only a specific area of the image. In the case of using the image of football player, we are going to find the region of the ball in the image displayed.

To access the particular region of the image, we should know the range of the pixels in which the football is present. The default display window of the image, does not provide the location of each pixel. So, you will use matplotlib library to show the image, so that you can find the pixels of the region of interest, by simply hovering on it. For finding the region of the image we should display the image using the plot function of the matplotlib library.

Note that matplotlib reads images in the RGB format, whereas open CV reads them in the BGR format. So, we will first convert the image from BGR to RGB. For color conversion, you will use the `cv.cvtColor` function.

The function `plt.axis`, provides access to the the axis properties, when anything is plotted using matplotlib. If it is passed the 'on' string, it turns on the display of the axis lines, and labels, and if it is provided with 'off', then it turns them off axis lines and labels.

So, to access a region of interest in an image, we will access the elements in a particular range, from the array. Now for selecting the particular region of the image, we use the image, and give the starting point and the ending point of the x and y coordinates of the pixels, corresponding to the region of interest. In this case, we noted that the ball lies between the pixels 235 and 295 in the x-direction And, 465 to 530 in the y-direction. We will write this region as the image array, `img`, followed by the x-range and y-range in

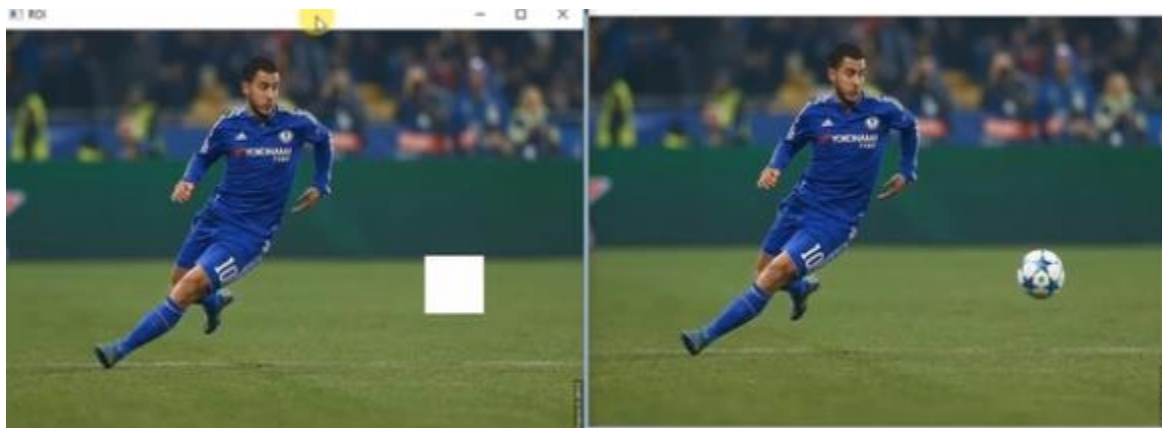
```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt

img = cv.imread("hazard10.jpg")
cv.imshow('image',img)

## use Matplotlib to find the pixel range of the pixel range of the ball
img_RGB = cv.cvtColor(img,cv.COLOR_BGR2RGB)
plt.axis('off')
plt.imshow(img_RGB)
ball = img[235:295,465:530]

## convert the ball region into white
ball = ([255,255,255])
img[235:295,465:530]= ball
cv.imshow('region of interest',img)
cv.waitKey(0)
cv.destroyAllWindows()
```

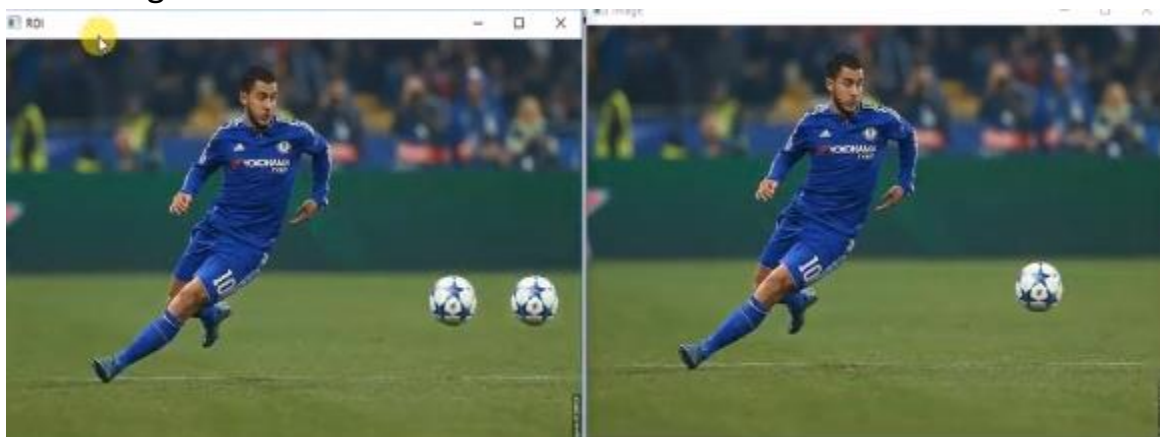
We can see that the ROI has turned to white, instead of having the football, as in the original image.



We can also use the ROI to replicate the ball in another location of the image. To do that, comment out the code to change the ROI to white and Assign the region of the ball to a different pixel range on the image. Here we are selecting the pixels from 235 to 295 in X-axis and 555 to 620 in the Y-axis, where you be replacing the pixels with the pixels of the ball. Note that the target pixel range size and the size of the ROI should be the same. Else, the program will throw an error.

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img = cv.imread("hazard10.jpg")
cv.imshow('image',img)
img[235:295, 555:620] = img[235:295,465:530]
cv.imshow('region of interest',img)
cv.waitKey(0)
cv.destroyAllWindows()
```

You will see that a replicated image of the ball has appeared on the original image. We have essentially replaced the pixels that were originally present there, with the pixels from the ROI. We have now learnt how to access and change a ROI in an image.



## Basic Concepts of a Video

We will know more about the characteristics of a video, video codecs and the video codec formats in fourcc.

As we know, video is a collection and representation of multiple visual images. Video systems vary in frame rate, aspect ratio, display resolution, refresh rate, color capabilities and other qualities. Frame rate, expressed in frames per second or fps, is the frequency at which consecutive images called frames appear on a display. FPS ranges from six or eight frames per second for old mechanical cameras to 120 or more frames per second for new professional cameras. But, typically, most of the videos available are shot in the range of 24 to 60 fps.

Aspect ratio describes the proportional relationship between the width and height of a video and the screens that play the video. All popular video formats are rectangular. The ratio width to height for a traditional television screen is 4:3. High definition televisions use an aspect ratio of 16:9.

A video codec is a software that compresses or decompresses digital video. It converts uncompressed video to a compressed format or vice versa. In the context of video compression, "codec" is a concatenation of "encoder" and "decoder". A program that only compresses is typically called an encoder, and one that only decompresses is a decoder. The compression is typically lossy, meaning that the compressed video lacks some information present in the original video. A consequence of this is that decompressed video has lower quality than the original, uncompressed video because there is insufficient information to accurately reconstruct the original video. There are a variety of codec software based on the application and the transport median type. Some of the commonly used codecs are "x264", "xvid", "divx", and "wmv".

A FourCC, which means four-character code, is a sequence of four bytes used to uniquely identify data and file formats. Four-byte identifiers are useful because they can be made up of four human-readable characters which can easily be remembered. They also fit into the four-byte memory space typically allocated for integers in 32-bit systems. Thus, the codes can be used efficiently in program code as integers. One of the most well-known uses of FourCCs is to identify the video codec.

## Capture Video in Open CV

We will capture a simple video, and display it as it is captured. To capture a video, we need to create a video capture object.

Its argument can be either the device index, or the name of a video file. Device index is just the number to specify the camera that a Computer is Interfaced with.

### ***Video Capture Function***

Syntax: `cv.VideoCapture(Argument)`

Argument – Device index(0,1,2 which depend on the number of the device) or the name of the video file

Normally, one camera is connected, and you can specify that camera, by simply passing 0. You can select the second camera by passing 1, and so on. After that, you can capture the video, frame-by-frame.

Remember that, a video is a collection of multiple frames shown at a high speed. Each frame is essentially an image. We will use an infinite loop, while true, to capture, and display the video continuously. We will then use the read function of the video capture object, to read the images from the video being captured.

The function gives two outputs. It returns a boolean, ret, which is true, when the frame is read correctly. It also gives out the image captured, as an output. The captured image is stored in the variable, frame. Finally, when everything is done, don't forget to release the capture. We can do that by using the release function of the video capture object.

```
import cv2 as cv
import numpy as np

cam = cv.VideoCapture(0)

while(True) :

    # capture frame-by-frame
    ret, frame = cam.read()

    # Display the resulting frame
    cv.imshow('frame', frame)

    key = cv.waitKey(10)

    if key == 27: #wait for Esc key to exit
        break

# when everything is done, release the capture
cam.release()

cv.destroyAllWindows()
```

The video being captured by your webcam, will be displayed.

## Playing Video using OpenCV

We will learn how to read a video file and then play the video from it. Playing a video is very similar to capturing video from a camera attached to the computer.

For reading an existing video, you will pass the name of the video file, as a string.

```
import cv2 as cv
import numpy as np

vid = cv.VideoCapture('vtest.avi')
while(vid.isOpened()) :
    ret, frame = vid.read()
    cv.imshow('frame', frame)

    key = cv.waitKey(100)

    if key == 27: #wait for Esc key to exit
        break

# when everything is done, release the capture
cam.release()

cv.destroyAllWindows()
```

In the case of **vid = cv.VideoCapture('vtest.avi')** , it will read and play the video, vtest.avi, present in the working directory.

We can control with the speed of displaying the video by changing the time given to the wait key function. As the time increase, the video is displayed slowly, and as the time decrease, the video speed is higher.

## **Saving Video as a File using Open CV**

For images, it is very simple, We you just use the cv2.imwrite().But in case of the video a little more work is required. Now for saving the video, we have to specify the video codec. A video codec is a software that compresses or decompresses digital video. It converts uncompressed video to a compressed format or vice versa. We will be using fourCC code to specify the video codec. FourCC is a sequence of four bytes used to uniquely identify data formats.

We will use XVID codec library, which runs very fast as it is optimized for the latest CPUs. We will use the VideoWrite\_fourcc function and pass the string XVID preceded by the asterisk to denote variable number of arguments that can be passed to the function. We will use a VideoWriter function to create the output video object.



The first argument of the function is the name of the output file.

The second argument is the fourCC codec type, which you had specified earlier using the `videowriter_fourcc` function.

The third argument is the frame rate of the video. You can give a frame rate of anything above 20 fps for a decent video.

The fourth argument is the size of the frame.

#### ***cv.VideoWriter***

Syntax: `cv.VideoWriter(filename, fourcc, fps, frame size)`

Parameters:

`filename` – the name of the output file

`fourcc` - video codec for the output file in fourcc format

`fps` - frame rate of the video

`frame size` – tuple with width and height of each frame of the video

Output: Creates a video object

```
import cv2 as cv
import numpy as np

cam = cv.VideoCapture(0)

#Define the codec and create videoWriter object
fourcc = cv.VideoWriter_fourcc(*'XVID')
output = cv.VideoWriter('Output.avi',fourcc,20.0,(640,480))

while(cam.isOpened()):
    ret, frame = cam.read()
    if ret == True:
        cv.imshow('frame',frame)
        output.write(frame)
        if cv.waitKey(1) == ord('q'):
            print('saves video')
            break
    else:
        print('error in capturing video')
        break

cam.release() #release everything if job is finished
output.release()
cv.destroyAllWindows()
```

We can see the video being captured by our webcam. Pressing the q key to save the video in the desired location. If any error text, make sure camera is working fine using any other camera application.

### 3.1 Advanced Image Operations in OpenCV

We will learn about Thresholding and its types. We will then learn about Image Smoothing. And, finally about Canny Edge Detection.

Image Thresholding is a technique to convert a colored image into a binary image, based on some criteria. It is primarily used in image segmentation - to segment an image into different areas based on specific criteria and have only those areas visible in the image for further analysis.

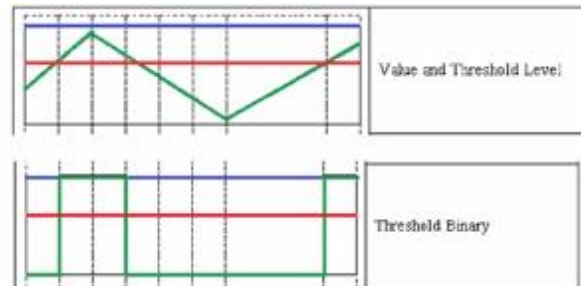
As we had learnt, every pixel of an image has some intensity associated with it. Now, for thresholding, you will set a predefined value called threshold. The simplest thresholding methods replace each pixel in an image with a white pixel if the image intensity is greater than the threshold  $T$ , or a black pixel if the image intensity is less than that constant. In the example image on the right, this results in the man and the camera becoming completely black, and the rest of the objects becoming completely white.



Let us now see the different types of thresholding. In binary thresholding, the pixels with values above the threshold value are set to 255, or the white pixel. And the pixels with values less than the threshold value, are set to zero, or to the black pixels. This is shown in the picture where an image is represented as a signal.

#### Binary Thresholding

- $I_{i,j} > T \rightarrow$  Replace with maximum value
- $I_{i,j} < T \rightarrow$  Replace with 0 (black pixel)

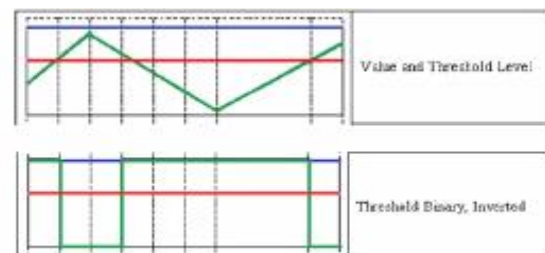


The green line represents the intensity values of different pixels in an image. The red line represents the threshold value. The pixels with intensity values above the threshold value, are replaced with the maximum value, which is 255. And, the pixels with intensity values, less than the threshold value, are replaced with the minimum value, which is 0.

The next thresholding method is Inverted Binary Thresholding. In inverted binary thresholding, the pixels with values above the threshold value are set to zero. And the pixels with values less than the threshold value, are set to the maximum value. So, this is the exact inversion of binary thresholding.

#### Inverted Binary Thresholding

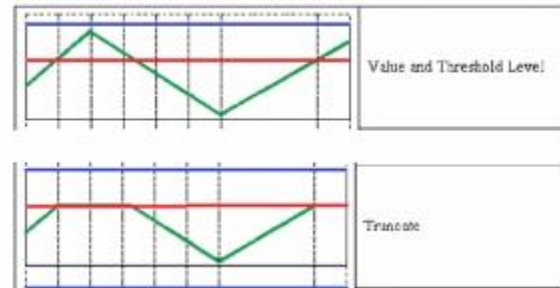
- $I_{i,j} > T \rightarrow$  Replace with 0 (black pixel)
- $I_{i,j} < T \rightarrow$  Replace with maximum value



Next up is truncate thresholding. In truncate thresholding, the pixels with values above the threshold value are set to the threshold value itself. And the pixels with values less than the threshold value, are left to remain as they are. So, this is essentially truncating, or chopping the image, at the threshold value.

### Truncate

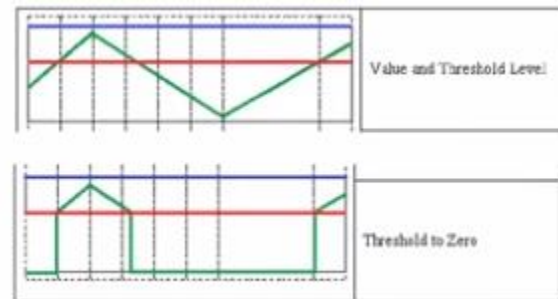
- $I_{i,j} > T \rightarrow$  Replace with  $T$  (Threshold value)
- $I_{i,j} < T \rightarrow$  Keep same as  $I_{i,j}$



In thresholding to zero, the pixels with values above the threshold value are left to remain as they are. And the pixels with values less than the threshold value, are set to zero. So, this is essentially removing the pixels, which have values less than the threshold.

### Threshold to Zero

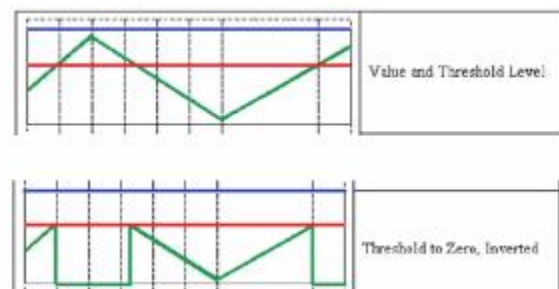
- $I_{i,j} > T \rightarrow$  Keep same as  $I_{i,j}$
- $I_{i,j} < T \rightarrow$  Replace with 0 (black pixel)



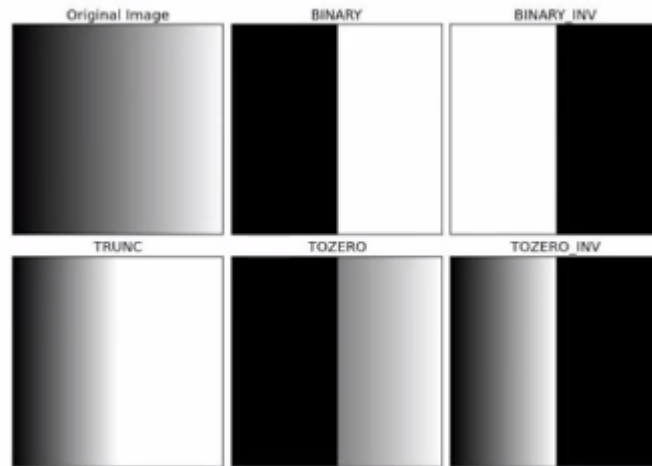
Finally, the inverted thresholding to zero method. In inverted thresholding to zero, the pixels with values above the threshold value are set to zero. And the pixels with values less than the threshold value, are left as they are. So, this is essentially the inversion of the thresholding to zero. In essence, all the pixels above the threshold value are blacked out, and the pixels below the threshold remain as they are.

### Threshold to Zero - Inverted

- $I_{i,j} < T \rightarrow$  Keep same as  $I_{i,j}$
- $I_{i,j} > T \rightarrow$  Replace with 0 (black pixel)



We can see the effect of different types of thresholding in this image. A gradient image with varying intensities is applied the different simple thresholding methods



In the simple thresholding methods, a single global value is used as the threshold value. But that may not be good enough in all the conditions, especially when an image has different lighting conditions in different areas. In that case, you will use adaptive thresholding. In this, the thresholding algorithm calculates different thresholds for multiple small regions of the image. So we get different thresholds for different regions of the same image, and that generates better results for images with varying illumination.

There are 2 types of adaptive method, which will decide how the threshold value is calculated. `cv.ADAPTIVE_THRESH_MEAN_C` calculates the threshold value as the mean of neighbourhood area.

`cv.ADAPTIVE_THRESH_GAUSSIAN_C`, calculates the threshold value as the weighted sum of the neighbourhood values, where the weights are a gaussian window. We can see the effect of simple thresholding and adaptive thresholding on an image with variable lighting in different regions.

Thresholding with a global threshold value removes all the pixels with intensity less than the threshold. This results in the loss of significant data.

Whereas, you will be able to see the numbers, when adaptive thresholding technique is applied.

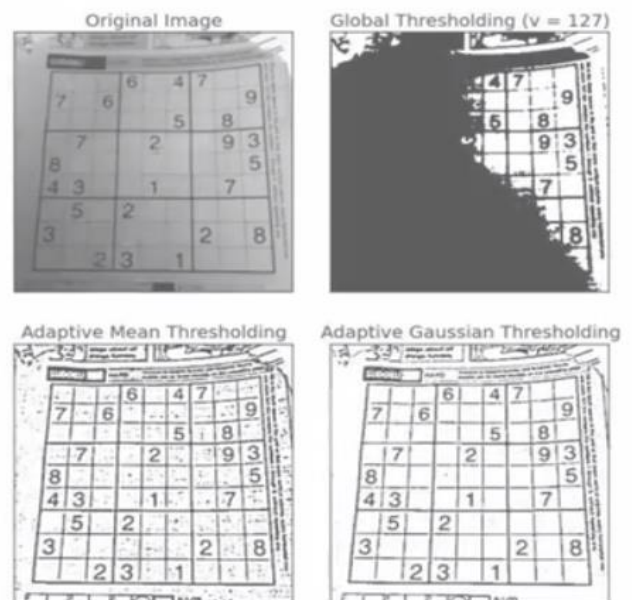
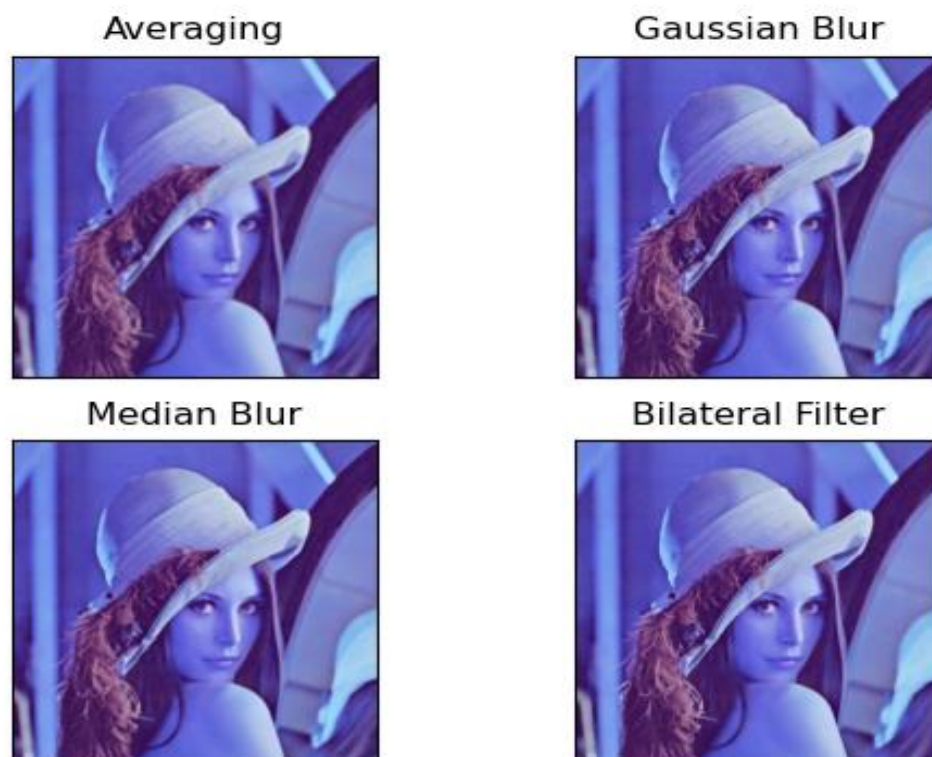


Image smoothing or blurring is useful for removing noise from the images. It actually removes the high frequency content like noise and edges from the image. It is also effective in reducing pixellation of an image.

In essence, smoothing replaces a pixel value with the average of the values of the pixels in its neighborhood. The average is calculated in different techniques, which leads to different outcomes in smoothing. It uses a kernel to do the same. Kernel is a smaller matrix, compare to the image matrix, which determines the size of the pixel neighborhood to be considered while smoothing.

Smoothing is divided to main four types provided with **open cv** library, the first type is Averaging that replaces central pixel with average, the second is Gaussian Blur that replaces central pixel with weighted average with Gaussian weights and smoothens an image using a Gaussian filter, the third is Median Blur that replaces central pixel with Median and is highly effective against salt-and-pepper noise in the images, and finally Bilateral Filtering is a 2 dimensional Gaussian Filter (space & Intensity) and is highly effective in noise removal while keeping the edges sharp.

The next figure shows the smoothing types :



Canny Edge Detection is a popular edge detection algorithm, and the next figure shows that:



Canny Edge Detection is a multi-stage algorithm with 4 stages as shown in the next figure:



Since edge detection is susceptible to noise in the image, the first step is to remove the noise in the image with a 5x5 Gaussian filter. Finding Intensity Gradient of the Image. The Smoothened image is then filtered with another technique, to get the first derivative of the pixel intensity in the horizontal direction and the vertical direction.

Gradient direction is always perpendicular to edges. It is rounded to one of four angles representing vertical, horizontal and two diagonal directions.

After getting gradient magnitude and direction, a full scan of image is done to remove any unwanted pixels which may not constitute the edge. For this, at every pixel, pixel is checked if it is a local maximum in its neighborhood in the direction of the gradient.

Hysteresis Thresholding, This stage decides which are all edges are really edges and which are not. For this, two threshold values are used, minVal and maxVal. Any edges with intensity gradient more than maxVal are sure to be edges and those below minVal are sure to be non-edges, and hence discarded. Those which lie between these two



thresholds are classified edges or non-edges based on their connectivity with other edges.

If they are connected to "sure-edge" pixels, they are considered to be part of edges. Otherwise, they are also discarded. So what we finally get is strong edges in the image.

### 3.2 Smoothing an Image - Basic Image Operations using OpenCV

Image blurring is useful for removing noise from the images. It actually removes the high frequency content like noise and edges from the image. OpenCV provides mainly four types of blurring techniques.

Now let's start with a basic averaging option for the above image. It simply takes the average of all the pixels under a specified kernel area and replaces the central element with the average. This is done by the function `cv.blur()`.

The first argument is the image source. The second argument is the size of the kernel.

It specifies the number of pixels to be considered, along the width and height of an image, for averaging, and replacing every central pixel element.

#### ***blur()***

Syntax: `cv.blur(image, ksize)`

Parameters:

image – input image

ksize - blurring kernel size or size of neighborhood.

Output: the function smoothen an image using the kernel

Now, we will use gaussian blur to the image. The Gaussian blur Blurs an image using a Gaussian filter. So we will use the function `cv.GaussianBlur` to apply the gaussian filter on the image.

The first argument is the image.

The second argument is the kernel size of the gaussian filter.

The third argument is the gaussian blur standard deviation value, which can be set to 0 by default, to have a central gaussian filter.

#### ***GaussianBlur()***

Syntax: `cv. GaussianBlur(img, size, sigma)`

Parameters:

img – input image

size - size of the Gaussian Filter.

Sigma – Gaussian kernel standard deviation.

Output: the function helps to blur the image using a Gaussian filter

We will now use median blurring, which takes the median of all the pixels under a specified kernel area, and replaces the central element with this median value. This is highly effective against salt-and-pepper noise in the images. The interesting thing is that, in the previous filters, the central element is a newly calculated value, which may be a pixel value in the image, or a new value.



But in median blurring, central element is always replaced by some pixel value in the image. It reduces the noise effectively. Its kernel size should be a positive odd integer. We will use the function `cv.medianBlur` to apply the median filter.

The first argument is the image file. The second argument is the size of the kernel, which should be a positive odd integer.

#### ***medianBlur()***

Syntax: `cv.medianBlur(img, size)`

Parameters:

`img` – input image

`size` - size of the neighborhood. Positive and odd integer

Output: the function helps to blur the image using a median filter

Finally, we will be using a bilateral filter on an image. Bilateral Filtering is highly effective in noise removal while keeping edges sharp. So, we will use the function `cv.bilateralFilter` to apply the filter on the image.

The first argument is the image. The second argument is the size of the pixel neighborhood. The third and fourth arguments are the sigma values. For simplicity, you can set the 2 sigma values to be the same. If they are small, that is less than 10, the filter will not have much effect. Whereas if they are large, which is greater than 150, they will have a very strong effect, making the image look "cartoonish". SO, choose the values somewhere in the middle.

#### ***bilateral()***

Syntax: `cv.bilateralFilter(img, size, sigma values)`

Parameters:

`img` – input image

`size` - size of the pixel neighborhood.

Sigma values – set the same. Set between 10 and 150

Output: the function helps to blur the image using a bilateral filter

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img = cv.imread("lena.jpg")
blur= cv.blur(img,(5,5)) #used cv.blur for smoothing
gauss_blur = cv.GaussianBlur(img,(5,5),0) #gaussian blur blurs the image with a gaussian filter
median = cv.medianBlur(img,5)
bilateral = cv.bilateralFilter(img,9,75,75)
plt.subplot(121)
plt.imshow(img)
plt.title('original')
plt.xticks([])
plt.yticks([])
plt.subplot(122)
titles = ['Averaging', 'Gaussian Blur', 'Median Blur', 'Bilateral Filter']
images = [blur, gauss_blur, median, bilateral]
for i in range(4):
    plt.subplot(2,2,i+1),plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
cv.waitKey(0)
cv.destroyAllWindows()
```



### 3.3 Simple Thresholding of Images using OpenCV

We will learn to do the thresholding of images. Thresholding is used to create binary images from grayscale images, which can further be used for object detection and identification.

Simple thresholding of Images is pretty straight forward. If the pixel value is greater than a threshold value, it is assigned a predefined value, which may be white, else it is assigned another value which may be black.

As mentioned, in simple thresholding operation, the pixels whose values are greater than a specified threshold value, are assigned with a standard value. We can perform the simple thresholding operation on an image, using the function `cv.threshold()`.

First argument of the function is the source image, which should be a grayscale image.

Second argument is the threshold value, which is used to classify the pixel values. We will use 127, which is exactly the midpoint value between 0 to 255, which will be the range of the pixel values in a grayscale image.

Third argument is the maxVal, which represents the standard value to be assigned, if the pixel value is more than, or sometimes less than the threshold value. We will use 255 here, which will be the maximum value that can be given to a pixel. This will turn the pixel to a white pixel.

The fourth Argument represents the type of Threshold. OpenCV provides different styles of thresholding and it is decided by the fourth parameter of the function. Each differ from the other, based on what operation is implemented on the pixels classified using thresholding value.

What is done with the pixels, which have values more than the threshold and, what is done with the pixels, which have values less than the threshold.

First, you will start with basic Binary Threshold. This can be set by passing the flag, **cv.THRESH\_BINARY** as the fourth argument of the function. In binary thresholding, the pixels with values above the threshold value are set to maxval. And the pixels with values less than the threshold value, are set to zero.

The next thresholding method is Inverted Binary Thresholding. This can be set by passing the flag, **cv.THRESH\_BINARY\_INV** as the fourth argument of the function. In inverted binary thresholding, the pixels with values above the threshold value are set to zero. And the pixels with values less than the threshold value, are set to the maxVal. So, this is the exact inversion of binary thresholding.

Next, we will create another image, with the truncate thresholding. This can be set by passing the flag, **cv.THRESH\_TRUNC** as the fourth argument of the function. In truncate thresholding, the pixels with values above the threshold value are set to the threshold value itself. And the pixels with values less than the threshold value, are left to remain as they are. So, this is essentially truncating, or chopping the image, at the threshold value.

#### ***threshold()***

Syntax: cv.threshold(image,thresh, maxVal, type)

Parameters:

image – the name of the image

thresh - threshold value.

maxVal - standard value to be given. If the pixel value is more than the thresh value.

type - threshold type.

Output: thresholded binary image

Next, we will create another image, with the thresholding to zero method. This can be set by passing the flag, **cv.THRESH\_TOZERO** as the fourth argument of the function. In thresholding to zero, the pixels with values above the threshold value are left to remain as they are. And the pixels with values less than the threshold value, are set to zero. So, this is essentially removing the pixels, which have values less than the threshold.

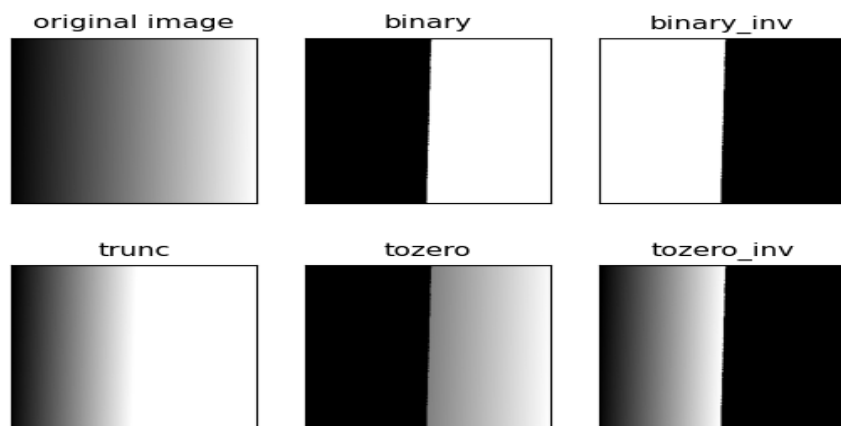
Finally, we will create another image, thresh5, with the inverted thresholding to zero method. This can be set by passing the flag, **cv.THRESH\_TOZERO\_INV** as the fourth argument of the function. In inverted thresholding to zero, the pixels with values above the threshold value are set to zero. And the pixels with values less than the threshold value, are left as they are. So, this is essentially the inversion of the thresholding to zero. In essence, all the pixels above the threshold value are blacked out, and the pixels below the threshold remain as they are.

```

import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt

img = cv.imread("gradient.png",0)
ret,thresh1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
ret,thresh2 = cv.threshold(img,127,255,cv.THRESH_BINARY_INV)
ret,thresh3 = cv.threshold(img,127,255,cv.THRESH_TRUNC)
ret,thresh4 = cv.threshold(img,127,255,cv.THRESH_TOZERO)
ret,thresh5 = cv.threshold(img,127,255,cv.THRESH_TOZERO_INV)
#CHANGE THE IMAGE IN THE FOLLOWING STATEMENT AFTER EACH THRESHOLDING
titles = ['original image', 'binary', 'binary_inv' , 'trunc' , 'tozero' , 'tozero_inv']
images = [img, thresh1 , thresh2 , thresh3 , thresh4 , thresh5]
for i in range(6):
    plt.subplot(2,3,i+1),plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([],plt.yticks([]))
plt.show()
cv.waitKey(0)
cv.destroyAllWindows()

```



You will now compare all the simple thresholding methods,

**In the Binary thresholding**, the pixels with values less than the threshold value, are set to zero. Which is approximately half of the highest intensity, are turned to white color. And, the pixels with values less than 127, are turned to black.

**In the Inverted Binary thresholding**, there is a clear demarcation created in the image. All the pixels, with values more than 127, are turned to black color. And, the pixels with values less than 127, are turned to the maxVal, which is white.

**In the Truncate thresholding**, we can note that after the midpoint level, all the values above the threshold value, have been replaced with the same value as the threshold value. All the pixels, with values more than 127, are turned 127. And, the pixels with values less than 127, are as they are.

**In thresholding to zero**, we can note that before the midpoint level, all the values less than the threshold value, have been replaced with the black pixels. All the pixels with values more than 127, remain with the original values. And, the pixels with values less than 127, are set to zero, or as black pixels.

**In the inverted thresholding to zero**, we can note that after the midpoint level, all the values more than the threshold value, have been replaced with the black pixels. All the pixels, with values more than 127, are set to zero, or as black pixels. And, the pixels with values less than 127, remain with the original values.

### 3.4 Adaptive Thresholding of Images using OpenCV

In simple thresholding, you used a single global value as threshold value. But that may not be good enough in all the conditions, especially when an image has different lighting conditions in different areas. In that case, we will use adaptive thresholding.

In this, the thresholding algorithm calculates different thresholds for multiple small regions of the image. So we get different thresholds for different regions of the same image, and that generates better results for images with varying illumination.

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img = cv.imread('sudoku.png',0)
img = cv.medianBlur(img,5)
ret,th1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
cv.imshow('Image - Threshold' , th1)
th2 = cv.adaptiveThreshold(img,255,cv.ADAPTIVE_THRESH_MEAN_C,cv.THRESH_BINARY,11,2)
th3 = cv.adaptiveThreshold(img,255,cv.ADAPTIVE_THRESH_GAUSSIAN_C,cv.THRESH_BINARY,11,2) #THE BEST
titles = ['Original Image', 'Global Image - Threshold' , 'Image_Adaptive Mean Thresholding' , 'Image_Adaptive Gaussian Thresholding']
images = [img,th1 , th2 , th3]
for i in range(4):
    plt.subplot(2,2,i+1),plt.imshow(images[i],'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
cv.waitKey(0)
cv.destroyAllWindows()
```



As the image has noise in it, in varying degrees, we will smoothen it using the median blur function. We will also use simple binary thresholding to the image.

We can see all the 4 images on the plot with the title and the threshold output, and observe the differences.

Original Image



Global Image - Threshold



Image\_Adaptive Mean Thresholding



Image\_Adaptive Gaussian Thresholding



## Edge Detection using Canny Algorithm

We will learn to detect the edges in an image using the canny algorithm.

Candy algorithm is one of the most robust techniques for detection of edges in an image. OpenCV is equipped with a single function that performs the entire candy algorithm named canny.

We will use the same function to perform the edge detection of an image

First argument is the input image. The second and third arguments are the minimum and maximum values of intensity gradients respectively which will be considered for classifying edges.

Any edge with intensity gradient more than the max value is sure to be an edge and any edge with gradient less than min value is discarded from being an edge. The ones in between are checked for connectivity with other edges and then determined to be edges or not. The canny function is an image with the edges detected.

We will now apply the Canny algorithm for a live video captured on your computer.

#### ***cv.Canny()***

Syntax: `cv.Canny(img,threshold1,threshold2)`

Parameters:

img – the name of the image

threshold1 - minimum value of intensity Gradient.

Threshold2 - maximum value of intensity Gradient

Output: returns the edges in the image

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
cam = cv.VideoCapture(0)
while(1):
    ret,frame = cam.read()
    frame = cv.cvtColor(frame,cv.COLOR_BGR2GRAY)
    edges = cv.Canny(frame,100,200)
    cv.imshow('frames',frame)
    cv.imshow('edges',edges)
    key = cv.waitKey(20)
    if key == 27:
        break
cam.release()
cv.destroyAllWindows()
```

We will see that the video being captured by the camera is displayed in its grayscale format, and also edges are being detected from the live video.

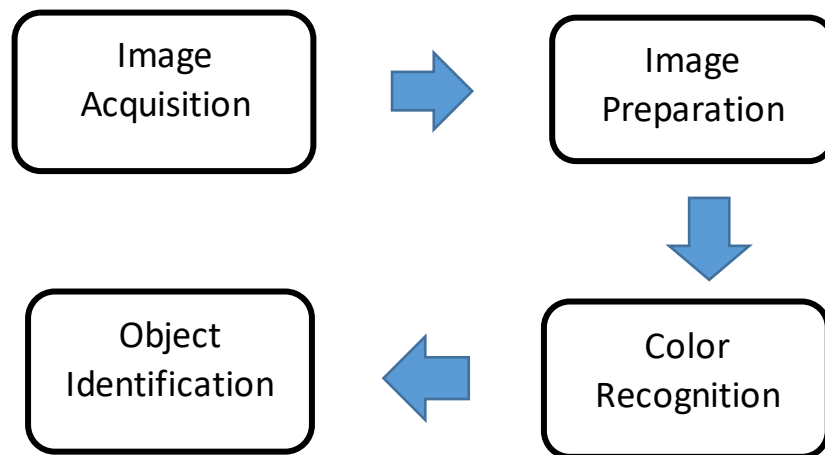
## 4 Object Tracking using OpenCV

### 4.1 Programming Logic for Tracking a Colored Object using OpenCV

In this section, you will learn about the pipeline, for tracking colored objects. You will also learn about the functions used in each stage of the pipeline.

---

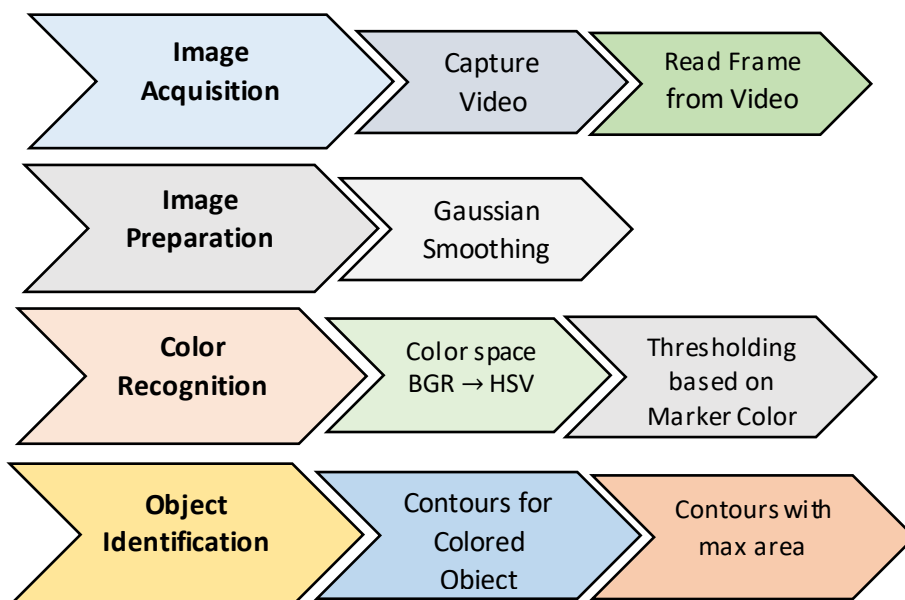
#### Pipeline for Object Tracking



---

The pipeline for object tracking has 4 stages. They are image acquisition, which involves acquiring the images, that will have the objects that need to be tracked. The next is to prepare the raw images. We will next recognize the colored objects in the image. And, finally, We will identify the specific object.

Let us now see the steps in each stage:



You are essentially going to track an object, from a live video being captured by your webcam. So, you will first capture video from the webcam.

As you had learnt, a video is a collection of frames. You will then read the frames from the video, so that you can apply the image processing techniques, on the individual frames.

Now that you have obtained the image, you will prepare the image, appropriately, for the further process. You will remove any noise from the image, using Gaussian Smoothing. Remember that, you are going to track the objects, based on their specific color. The default images in Open CV, are in the B G R color space. But, it is easier, and better, to identify colored objects, in the H S V space. So, you will first convert the image, from the B G R, to the H S V color space. Finally, you will separate the objects of a specific color, or a color range, from the image. You will do a thresholding operation, to identify the objects of the specific color range.

Once all the objects of the specific color are separated, using color recognition, you will identify the object of our interest. You will first find the contours for all the colored objects. And, then identify your object, as the object with the maximum contour area. Finally, you will use contour features, to track the object in the live video.

Let us now see the functions you will use, in each step, of each stage, of the object tracking pipeline.

To capture a video, you need to create a Video Capture object. Its argument can be either the device index, or the name of a video file. Device index, is just the number to specify which camera connected to your computer is being used for video capture. Normally one camera will be connected. So you simply pass 0. You can select the second camera by passing 1, and so on.

#### *Video Capture*

- ***Cam = cv2.VideoCapture(index)***
- Creates Video Object
- Index → Index of Camera connected to computer or name of the video file
- cam → Video Capture object

After that, you can capture frame-by-frame. You will then use the read function, of the video capture object, to read the image from the video being captured. The function gives two outputs. It returns a boolean, ret, which is true, when the frame is read correctly. It also gives out the frame from the video captured, as an output. The captured image, is stored in the variable, frame. You can now perform the usual image functions on the variable, frame.

#### *Read frame from Video*

- ***ret, frame = cam.read()***
- Reads frame from the video
- ret → boolean output. True, if frame is read correctly. False otherwise
- frame → image with the frame of the video

In the stage of Image preparation, Image smoothing is useful for removing noise from the images. It actually removes the high frequency content, like noise, and edges, from the image. Open CV provides mainly four types of blurring techniques. You will use the most robust, the Gaussian blurring technique.

The Gaussian blur, Blurs an image using a Gaussian filter. You will use the function `cv2.gaussianBlur`, to apply the gaussian filter

on the image. The first argument, is the image.

The second argument, is the kernel size of the gaussian filter. The third argument, is the gaussian blur standard deviation value, which can be set to 0 by default, to have a central gaussian filter. Gaussian blur, gives a better smoothed image.

#### *Gaussian smoothing*

- `img_blur = cv2.GaussianBlur(img, ksize, sigma)`
- Gaussian Smoothing of Image
- `img` → Original Image
- `ksize` → size of neighborhood around the pixel to be considered
- `sigma` → standard deviation (assign 0, by default)
- `img_blur` → Output image

In the stage of Color Recognition, You will now convert the image, into a different color space. There are more than 150 color-space conversion methods, available in Open CV, but you will see the basic ones.

For color conversion, you will use the `cv.cvtColor` function. The first argument for the `cv.cvtColor` function, is the image file that needs to be converted. The second argument, is the code for the conversion type. There are numerous color spaces that Open CV can handle, and therefore, quite a few conversion codes. You will convert the image from BGR, to HSV, to ensure better operations based on color. So you will use the code, `cv.COLOR_BGR2HSV`.

The output of the `cv.cvtColor` function, is an image, in the HSV color space.

#### *Color space BGR → HSV*

- `img_hsv = cv2.cvtColor(img, conversion_code)`
- convert one image from one color space to the other color space
- `img` → input image
- `conversion_code` → color conversion code based on color spaces.  
For ex: `cv.COLOR_BGR2HSV`
- `img_hsv` → converted image

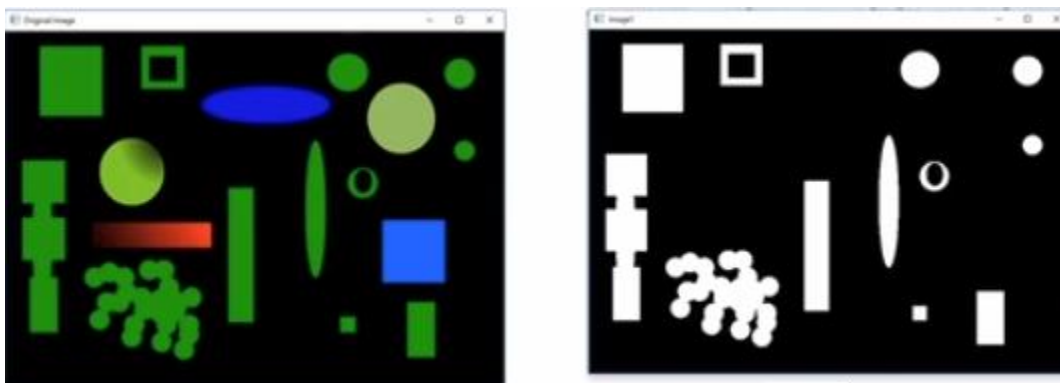
Now, to extract only the pixels of a particular color range, you will use the `inRange` function. You will first determine the range of HSV values, that your colored object lies in - the lower bound, and the upper bound. You will then use the `inRange` function, to convert the HSV image, to a binary image, which has only the pixels of a particular color.

The in range function, takes three inputs: the original array, the lower bound, and, the upper bound. The function gives an array with the same shape as the original array the elements, corresponding to the elements of the original array, the elements, corresponding to the elements of the original array, lying within the bounds, are given the value of 1 and, the elements, corresponding to the elements of the original array, beyond the bounds, are assigned the value of 0.

#### *Thresholding based on Object Color*

- `img_threshold = cv2.inRange(img, lower_bound, upper_bound)`
- convert pixels based on their values
- `lower_bound < pixel value (in img) < upper_bound` → pixel value = 255
- Else, pixel value = 0

So, as you can see, the in range function, applied for green color, picks only the objects in green color, and removes the rest of the objects.



In the stage of Object Identification, Now that you converted the original image into a binary image, you will find the contours of the colored objects, from the binary image. To find the contours, you will use the cv.find contours function. There are three arguments in cv.find Contours function. The first one, is the source image in a binary form, from which the contours need to be found.

The second argument, is the con tour retrieval mode, which you had learnt about earlier.

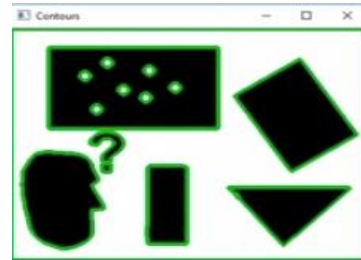
And, the third, is the contour approximation method.

The find con tours function, outputs a modified image, the contours, and the hierarchy. Contours, is a Python list of all the contours in the image. Each individual contour, is a Numpy array of x, y coordinates, of boundary points of the object. Hierarchy, is the representation of the

#### *Contours for Colored Object*

- `cv.findContours(img, mode, method)`
- Finds contours in an image
- `img` → source image
- `mode` → contour retrieval mode
- `method` → contour approximation method
- Returns the following:
  - `Img2` → resultant image with contours
  - `contours` → Array of detected contours
  - `method` → relationship between contours

relationships between different contours, found from an image. The contours identified, can be displayed on an image, using the drawContours function, as shown.



Finally, you will find the contour with the maximum area, as the contour of interest, and identify it as the object. You will ignore the smaller contours, as they might be due to some error in setting the bounds, or some extraneous pixels that do not belong to the specific object. You can then use contour features, like centroid, bounding rectangle, to track the motion of the object. This finishes the programming logic, on how to track a colored object.

#### ***Find Contours with max Area***

- Ignore the smaller contours as errors

#### ***Calculate and use Features of contour with Max Area***

- Centroid
- Bounding Rectangle

## **4.2 Contours in Open CV - Finding & Drawing Contours**

We will learn to find, and draw the contours in an image. Contours can be explained simply as the curves joining all the continuous points, having the same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition.

In OpenCV, finding contours is like finding white objects from a black background. So, remember, the objects to be found should be white, and the background should be black.

Before you go ahead and find contours in the image, you will first convert the color image, with 3 channels, into a binary image.

So, convert the default BGR image to a grayscale image, using the cv.cvtColor function. Pass the original image and the color conversion code, cv.COLOR\_BGR2GRAY, as the arguments to the function.

You will then use simple binary thresholding to convert the grayscale image to a binary image. In simple thresholding operation the pixels whose values are greater than the specified threshold value, are assigned with a standard maximum value. You will use the threshold function, to do the same.

The first argument in the function, is the image file, on which the thresholding should be done. You will then pass the threshold value, beyond which, all the pixels will be converted to the maximum value, and below which, all the pixel values will be changed to 0. The next argument will be the maximum value, which has to be assigned to the pixels, which are beyond the threshold value. The fourth argument represents the type of thresholding to be used. You will do the binary thresholding, by assigning 0 as the argument.

You will assign the result of the function to the variable `thresh` and the boolean value `ret`. Now that you converted the original image into a binary image, you will find the contours on the same. To find the contours, you will use the `cv.findContours` function.

An important note before you continue. With the recent update in the OpenCV library, we need to make some changes to the `findContour` function. As you saw before, we are using 3 variables. But this will throw an error as a result of the recent updation to the library. The error you will get if continued without any changes is, value error: not enough values to unpack, expected 3, got 2. Earlier the `findContour` function returns three values. However, now it returns only 2 values that can be stored in two variable namely `contours` and `hierarchy`. Hence, we have to remove the `img2` variable.

There are three arguments in `cv.findContours()` function. The first one is the source image in a binary form, from which the contours need to be found. The second argument is the contour retrieval mode. And, the third is the contour approximation method.

The `find contours` function outputs a modified image, the contours, and the hierarchy. Contours is a Python list of all the contours in the image.

Each individual contour, is a Numpy array of (x,y) coordinates of boundary points of the object.

Now, what is heirarchy? You will use the `find contours` function to detect objects in an image. The objects are in different locations in an image. But in some cases, some shapes are inside other shapes, just like nested figures. In this case, we call the outer contour as a parent and the inner one as the child. This way, contours in an image have some relationship with each other. And you can specify how one contour is connected to other contours, For example, is it child of some other



contour, or is it a parent etc. Representation of these relationships is called the Hierarchy.

Let us now understand the retrieval mode argument based on the concept of hierarchy, which you have learnt. There are multiple retrieval modes that can be used for finding contours.

For example, the list retrieval mode, represented as `RETR_LIST`, is the simplest of the modes. It simply retrieves all the contours, without creating any parent-child relationship. Parents and kids are equal under this rule. and they are just contours. ie they all belong to the same hierarchy level. Whereas, `RETR_TREE` retrieves all the contours and creates a full family hierarchy list.

Let us now understand the contour approximation methods. AS mentioned that contours are the boundaries of a shape with the same intensity. It stores the (x,y) coordinates of the boundary of a shape. But does it store all the coordinates? That is specified by this contour approximation method.

If you pass `cv.CHAIN_APPROX_NONE`, all the boundary points are stored. But actually do we need all the points? For example, lets say you found the contour of a straight line. Do you need all the points on the line to represent that line? No, you need just two end points of that line.

This is what `cv.CHAIN_APPROX_SIMPLE` does. It removes all redundant points and compresses the contour, thereby saving memory.

Now that you found the contours of an image, you will represent them on the image, by drawing the contours. You will use the `cv.drawContours` function, to draw the contours on the image.

The first argument is the image, on which the contours need to be drawn. The second argument is the array of contour, which you had obtained from the `findContours` function. The third argument is the contour index, which is a parameter indicating the contour to draw.

If it is negative, all the contours are drawn. The fourth argument is the color to be used for drawing the contours, in the BGR format. The fifth argument is the thickness of the contour, in pixels.

```

import cv2 as cv

import numpy as np

img = cv.imread('pic1.png')

img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

ret, thresh = cv.threshold(img_gray, 127, 255, 0)

contours, hierarchy = cv.findContours(thresh, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)

cv.drawContours(img, contours, -1, (0, 255, 0), 3)

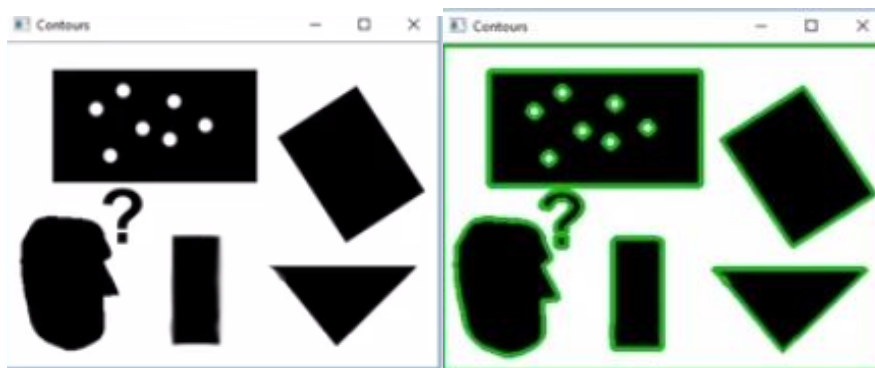
cv.imshow('Contours', img)

cv.waitKey(0)

cv.destroyAllWindows()

```

You can see that the original image is displayed with all the contours, marked in a green color, which is the color you had specified.



You will now draw only one contour, instead of all, say the 2nd contour. So, you can just replace the -1 with the contour number of the contour you are interested in. So, you can just replace the -1 with the contour number of the contour you are interested in, which will be 1, in this case.

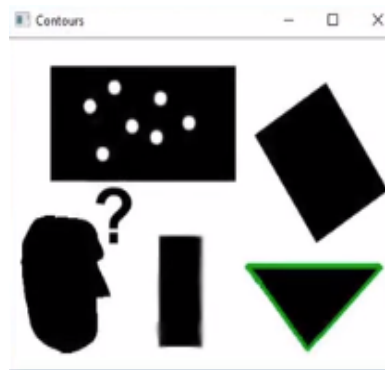
```

#cv.drawContours(img, contours, -1, (0, 255, 0), 3)

cv.drawContours(img, contours, 1, (0, 255, 0), 3)

```

You will observe that the contour is displayed for only one object, as shown.



#### 4.3 Contours in Open CV - Calculation of Features

We will learn to find the features of the contours which you had identified earlier. You will use the same pic one.PNG image which you had used to identify and draw the contours.

You will now calculate some features of the facial contour. you will first calculate the center of mass of the facial contour using the moments function. The function `cv.moments` gives a dictionary of all moment values calculated for the image. From these moments you can extract useful data like area, centroid etc.

Centroid is given by the relations,  $cx = \text{int}(M['m10']/M['m00'])$

And  $cy = \text{int}(M['m01']/M['m00'])$  Then, print the centroid of the facial contour.

Similarly, you can calculate and print the area and the perimeter of the contour, using the `contourArea` and the `arcLength` functions.

You will now draw a rectangle that encompasses a contoured object. Such a rectangle is called the bounding rectangle. This kind of operation is useful for object tracking in a video. The bounding rectangle is found by the function `cv.boundingRect()`. The function takes the contour of interest, as the input argument. The output of the function is `x` and `y`, which are the coordinates of the top left corner of the rectangle, and `w` and `h`, which will be the width and height of the rectangle. You can then draw this rectangle on the image, using the `cv.rectangle` function, which you had used earlier for drawing geometric shapes on an image. Remember that, the rectangle function, needs the following arguments. The image, on which the rectangle needs to be drawn. 2 opposite

vertices - which can be obtained as shown, as you know the width and height of the rectangle. The color of the rectangle to be drawn. And, the thickness, in pixels, of the rectangle.

```
import cv2 as cv
import numpy as np

img = cv.imread('pic1.png')
img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
ret, thresh = cv.threshold(img_gray, 127, 255, 0)

contours, hierarchy = cv.findContours(thresh, cv.RETR_TREE,
cv.CHAIN_APPROX_SIMPLE)

cnt = contours[6]
cv.drawContours(img, [cnt], 0, (0,255,0), 3)

M = cv.moments(cnt)
cx = int(M['m10']/M['m00'])
cy = int(M['m01']/M['m00'])
print('centriod of the Face is ', (cx,cy))

area = cv.contourArea(cnt)
print('Area of the Face is', area)

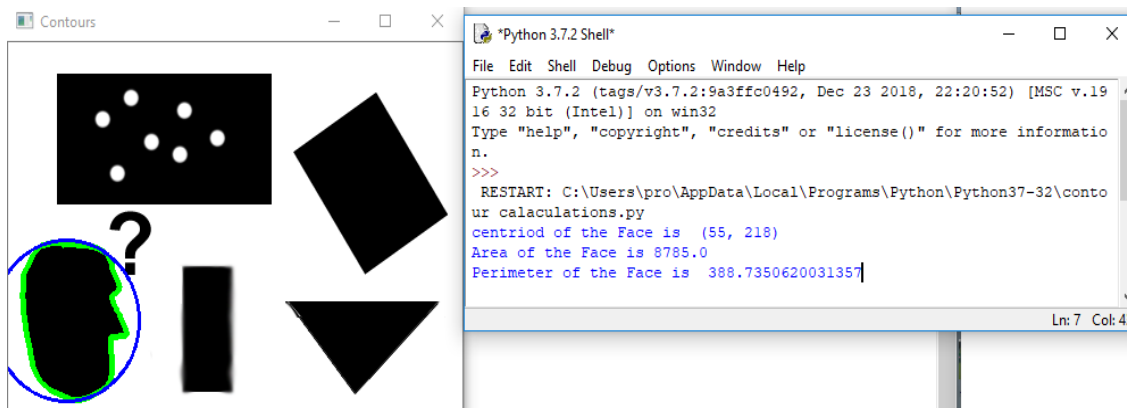
perimeter = cv.arcLength(cnt, True)
print('Perimeter of the Face is ', perimeter)

##x,y,w,h = cv.boundingRect(cnt)  #for drawing rectangle around the contour
##cv.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)

(x,y),raduis = cv.minEnclosingCircle(cnt)
center = (int(x),int(y))
raduis = int(raduis)

cv.circle(img,center,raduis,(255,0,0),2)  #for drawing circle around the contour
cv.imshow('Contours',img)
cv.waitKey(0)
cv.destroyAllWindows()
```

Similarly, you will find the circumcircle of an object, using the function `cv.minEnclosingCircle()`. It is a circle which completely covers the object with minimum area. The function takes the contour as an input and outputs the center and the radius of the circle. You can then use the same, to draw the circle, using the circle function. You will see that the facial contour has been enclosed in a circle.



#### 4.4 Tracking of a Colored Object using OpenCV

To start a video capture if you did any error text make sure your camera is working fine using any other camera application. You will now track a color object, yellow in this case. You can use an object with a color of your choice. You will first try to define the HSV value of the colored object. To find the HSV value you need the image to be plotted in RGB and use any online tool to convert the RGB. By this method you will be able to find the upper and lower range of the yellow color.

Now for plotting the image on the plot will import the Pyplot module from the Matplotlib Library and name it as `plt` for simplicity. We can plot an image only in RGB format. So first we have to convert the image into RGB from the default BGR color space. And using `plt.imshow` function to plot the RGB image of the plot. Finally, use that function `plt.show` to display the plot.

Now that you have to have the HSV values of the yellow color, even take a rough range of lower and upper bounds to detect yellow. You will now find the yellow colored object by extracting only the pixels which lie in a color range.

You had to find the range of color using the lower and upper bounds of HSV values. So first you will convert this smooth image into HSV color format using the `cv.cvtColor` function. We will assign the Function output to variable `image_hsv`.

Now to extract all the pixels in a particular region, you will use the `inRange` function. The `inRange` function takes 3, inputs the original array, the lower bound and the upper bound. The function gives an array with the same shape as the original array. The elements corresponding to the elements of the original array line within the bounds are given. The value of one. And the elements corresponding to the elements of the original array beyond the bounds are assigned a value of 0.

#### ***inRange()***

Syntax: `cv.inRange(img,lowerboundary,upperboundary)`

Parameters:

`src` – first input array

`lowerboundary` - inclusive lower boundary array or a scalar.

`upperboundary` - inclusive upper boundary array or a scalar.

Output:

Function returns '1' when the values lies in between lower and upper boundaries, else '0'

```

import cv2 as cv
import numpy as np
cam = cv.VideoCapture(0)
lower_yellow = np.array([20,100,100])
upper_yellow = np.array([40,255,255])
while(True) :

    # capture frame-by-frame
    ret, frame = cam.read()

    #smoothen the image
    image_smooth = cv.GaussianBlur(frame,(7,7),0)

    # threshold the image for yellow color
    image_hsv = cv.cvtColor(image_smooth, cv.COLOR_BGR2HSV)
    img_threshold = cv.inRange(image_hsv, lower_yellow, upper_yellow)

    # Display the resulting frame
    cv.imshow('frame', img_threshold)

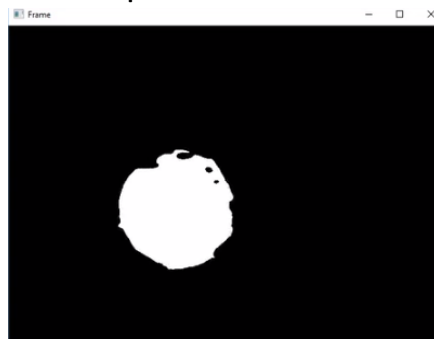
    key = cv.waitKey(10)

    if key == 27:      #wait for Esc key to exit
        break

cam.release()      # when everything is done, release the capture
cv.destroyAllWindows()

```

As expected, only the yellow object is being shown in white, while the rest of the frame is completely black. The output will be as shown:



#### 4.5 Program to Determine Location of an Object

We will write a program, for detecting the location (relative position) of a colored object. You will use the code you had written for the object tracking, and then modify it, and build on top of it.

We had earlier used a yellow object for object tracking. In this case, we will use a red colored object. So, change the values based on the color of the object. You then had written an infinite loop, to read frames, from the video captured by your webcam. After you read the frame, using the `cam.read` function, you will flip the frame, about the vertical axis.

This is done, as the webcam, by default, shows the mirror image, and so the marker motion, is also flipped. This will make the cursor, move in the opposite direction. To avoid this, and make the interface more intuitive, you will flip the frame, using the `cv.flip` function.

The first argument, is the source image. And the second argument, determines the axis about which, the image has to be flipped. The flag, 0, flips the image about the x-axis. The flag 1, flips it about the y-axis. And, -1, flips the image about both the axes. You will be flipping the frame in y axis alone. Hence, you will be using 1 in the function.

The output will be assigned to the same variable, frame. As you had learnt, you will be using a part of the video frame, to record the marker object, and determine its relative location. So, after smoothening the image using Gaussian Blur, you will define a Region of Interest, and track the motion of the marker only in that region. You will define a mask, to select the region of interest.

First, you will create a completely black mask, by creating a zeros array, of the same size as the frame captured by the webcam. You will use the `np.zeros_like` function, to do the same. You will then change the black pixels in a square region, say from 50, to 350, in either directions, to white pixels. So, your mask, in essence, looks like this. You will then get the region of interest of your frame, by performing a bit wise and operation, between the original frame, and the mask. This function, Calculates the per-element, bit-wise conjunction, of two arrays.

The first argument, is the smooth image, and the second argument, is the mask image. As you know, according to the AND logic table, if both the inputs are high,



the output will be high. IF any one of the inputs is low, the output will be low. In this case, the mask image in the square region, has all white pixels, and the rest are all black pixels, which are equivalent to zeros. So, only the pixels of the frame, which lie in the region, are shown as an output. And the rest of the pixels are blacked out. Assign the output of the function, to the variable image\_roi.

Now, draw a rectangle, and a grid, on the actual frame, to indicate this R O I. So, you will use the cv.rectangle function, to draw a rectangle on the image, frame. The top left vertex of the rectangle is 50, 50. The bottom right is at 350, 350. You will assign the rectangle to be drawn in red color, with a thickness of 2 pixels.

You will also draw 4 thin lines on this rectangle, 2 vertical, and 2 horizontal, to create the 3x3 grid. The end points of the grid lines, can be easily obtained. You will next have to convert the masked image, into the HSV color space. So, change the argument for the cvt Color function, to image\_roi. Keep the rest of the pipeline as the same. It will detect the largest yellow colored object, in the Region of Interest.

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt

cam = cv.VideoCapture(0)

lower_red = np.array([0,125,125])
upper_red = np.array([10,255,255])

while(True):
    ret, frame = cam.read()
    frame = cv.flip(frame,1)
    w = frame.shape[1]
    h = frame.shape[0]
    #smothen the image
    image_smooth = cv.GaussianBlur(frame,(7,7),0)
```

```

#Define Region of Interest
mask = np.zeros_like(frame)
mask[50:350, 50:350] = [255,255,255]
image_roi= cv.bitwise_and(image_smooth, mask)
cv.rectangle(frame, (50,50), (350,350), (0,0,255), 2)
cv.line(frame, (150,50), (150,350), (0,0,255), 1)
cv.line(frame, (250,50), (250,350), (0,0,255), 1)
cv.line(frame, (50,150), (350,150), (0,0,255), 1)
cv.line(frame, (50,250), (350,250), (0,0,255), 1)
#Threshold the image for red color
#img_hsv = cv.cvtColor(image_smooth, cv.COLOR_BGR2HSV)
img_hsv = cv.cvtColor(image_roi, cv.COLOR_BGR2HSV)
image_threshold = cv.inRange(img_hsv, lower_red, upper_red)
#find contours
contours, heirarchy= cv.findContours(image_threshold, \
                                     cv.RETR_TREE, \
                                     cv.CHAIN_APPROX_NONE)
#find the index of the largest contour
if(len(contours)!=0):
    areas= [cv.contourArea(c) for c in contours]
    max_index= np.argmax(areas)
    cnt = contours[max_index]
    #x_bound, y_bound, w_bound, h_bound = cv.boundingRect(cnt)
    #cv.rectangle(frame, (x_bound, y_bound), (x_bound + w_bound, y_bound + h_bound), (255,0,0), 2)
    ##Pointer on Video
    M = cv.moments(cnt)
    if(M['m00'] != 0):
        cx = int(M['m10']/M['m00'])
        cy = int(M['m01']/M['m00'])
        cv.circle(frame, (cx, cy), 4, (0,255,0), -1)

```

```
#Cursor Motion

if cx in range(150,250):
    if cy <150:
        print("Upper Middle")
    elif cy > 250:
        print("Lower Middle")
    else:
        print("Center")
if cy in range(150,250):
    if cx <150:
        print("Left Middle")
    elif cx >250:
        print("Right Middle")
    else:
        print("Center")

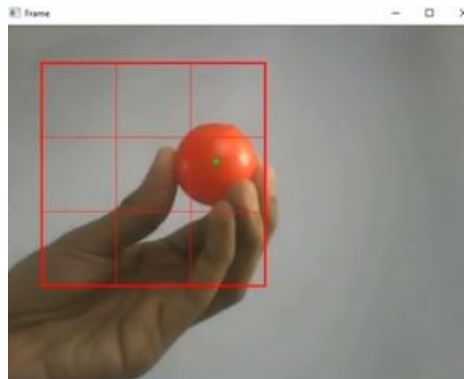
#cv.imshow('Frame', frame)

#cv.imshow('Frame', image_smooth)
#cv.imshow('Frame', image_threshold)
cv.imshow('Frame', frame)

#key = cv.waitKey(10)
key = cv.waitKey(100)
if key == 27:
    break

#img_RGB= cv.cvtColor(frame, cv.COLOR_BGR2RGB)
#plt.imshow(img_RGB)
#plt.show()
cam.release()
cv.destroyAllWindows()
```

Now, you can see that there is a rectangle, with the grid lines on the flipped frame. You can also see that whenever a red object enters the region, it is being tracked.



You will now track the motion of the marker. Earlier, you had used the bounding rectangle feature of the contours. You will now need to use the centroid of the contour, to track the motion of the marker. So, remove the code for the bounding rectangle. And, include the code, to find the centroid of the red object contour, as shown. You will have to use the moments function, to do the same. Now, draw a small circle, with  $c_x$  and  $c_y$ , as the center of the circle, and a small radius of 4 pixels, on the frame. This will represent the position of the marker. Now, you will determine the position of the marker, relative to the grid. If the value of  $c_x$ , lies between 150, and 250, then the pointer lies in the middle region.

Now, we will check the value of  $c_y$ . If the value of  $c_y$  is less than 150, then the marker lies in the upper sector. If the value is more than 250, then the marker is in the lower sector. In case the value of  $c_y$  is not in either of those ranges, then the marker will be in the center. So, print the relative location of the marker based on these values. Repeat similar steps, using the if conditions, to determine the relative x position of the marker.

If you move the ball out of the grid, the relative position will not be printed.



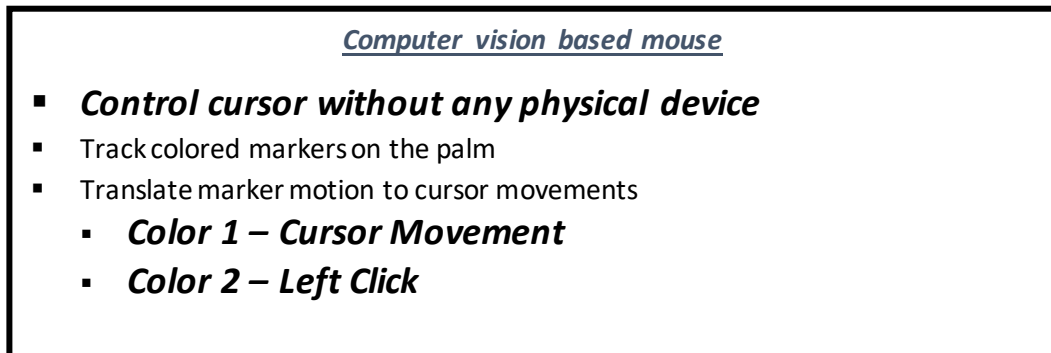
## 5 Computer Vision Based Mouse

### 5.1 Programming logic for computer vision based mouse.

Learning Outcomes:

- Pipeline for Computer Vision based Mouse
- Tracking Markers and their motion
- Translate Marker Motion to Cursor Motion

A computer vision based mouse, is a system to control the cursor of your computer, without any physical device. Even a mouse! You will essentially have colored markers on your palm. You will capture the video of the motion of your palm, using the webcam of your computer. You will track the colored markers, and use their motion, to control the cursor of your computer! You can use 3 colors, for the 3 typical actions of the mouse.



Let us now learn about the pipeline, for implementing the Computer Vision based Mouse. The pipeline for CV based mouse, has 6 stages. They are:

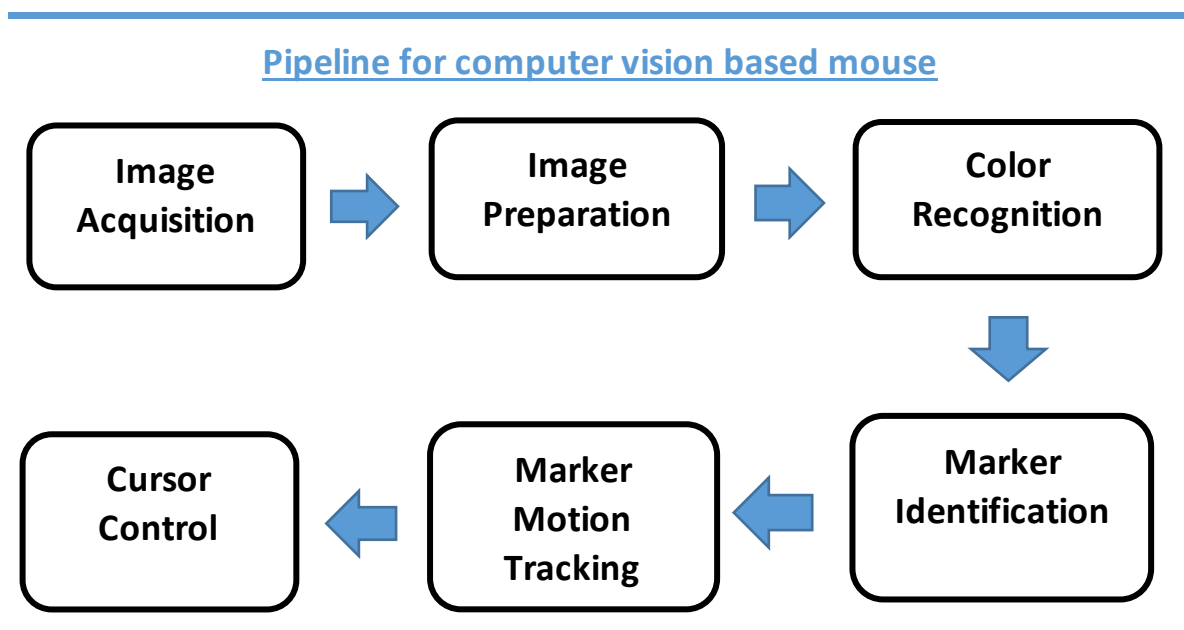
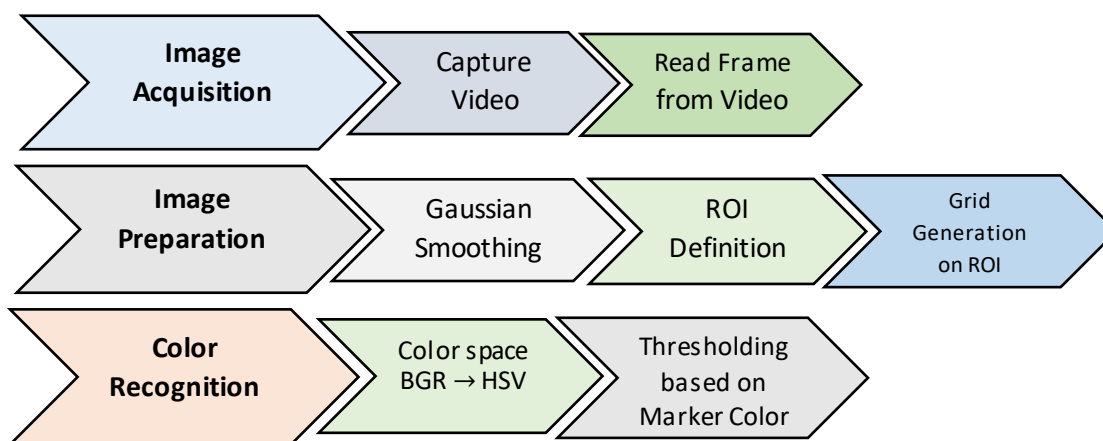
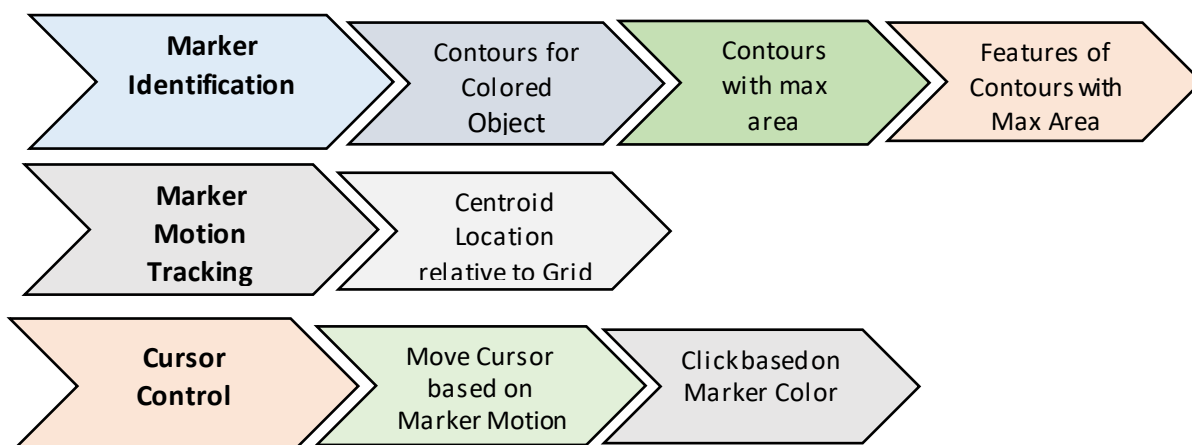


Image Acquisition which involves acquiring the images, that will have the markers on your palm, that need to be tracked, The next is to prepare the raw images, You will next recognize the colored objects in the image, You will then identify the markers, from all the objects recognized in the image, Once you identify the markers, in each frame, you will track their motion in different frames. Finally you will translate this marker motion, to the control of the mouse cursor on your computer. Note that, the first 4 stages of the CV based Mouse, are equivalent to the Colored Object Tracking, which you had implemented earlier.

Let us now look at the steps in each stage. These are the steps in the first 3 stages of the pipeline. Most of the steps, are the same as the ones in Object Tracking.



There are just 2 additional steps to be done, while preparing the image. As you will have the control on the markers, you can place them in a specific region, So as to improve the accuracy of their identification and tracking. So, while preparing the image, you will define a region of interest. You will also create a grid, to later identify the location of the markers. Now, the next 3 stages have the steps shown.



Again, the 4th stage of marker identification, is similar to the object identification in object tracking. Once the marker is identified in Stage 4, you will track the motion of the marker in different frames, by locating the centroid of the marker, in the Region of Interest. Finally, you will move the cursor, based on the marker motion. You will also click the mouse, based on the color of the marker, as mentioned earlier. This finishes the detailed pipeline, for implementing a Computer Vision based Mouse.

In the stage of Image Preparation, after acquiring the image, and removing the noise, you will create a region of interest, and a grid, in that region. This will increase the accuracy of the marker identification, and tracking. The grid will help you in translating the position of the marker, to cursor movement. Let us now see how the motion of the marker is tracked.

In the stage of Marker motion tracking, you will use the contour features, to find the contour around the markers on your palm. You will then find the centroid of the marker. Finally, you will find the location of the centroid, in the grid that you had created. It can fall in any of the 9 regions as shown.

If it is in the top left region, the cursor should move towards the top left part of your screen. If the marker is in the top center region of the grid, the cursor should move towards the left of your screen, and so on. If you hold the marker in the central region, the cursor should not move, and stay still. This is how you can control the cursor, based on the marker motion. Now, how do you actually control the cursor?

▪ ***Find Location of Marker Relative to Grid***

- Top - Left
- Top – Center
- Top – Right
- Mid- Left
- Mid – Center
- Mid – Right
- Bottom - Left
- Bottom – Center
- Bottom – Right

In the stage of Cursor control, you will use the **pyautogui** library of python, to control the mouse cursor, and also click, based on the color of the marker.

## **5.2 Installation of pyautogui Library for Windows**

PyAutoGUI is a Python module for pro-grammatically controlling the mouse and keyboard of your computer. The installation of the PyAutoGUI library is similar to the installation of the openCV and numpy packages you have done earlier. Open the command prompt as the administrator. Now type the command `pip space install space pyautogui`, which will install the python auto GUI library as an additional python package. Ensure that your computer is connected to the internet, for this to happen.



```
Administrator: Command Prompt - pip install pyautogui
C:\WINDOWS\system32>pip install pyautogui
Collecting pyautogui
  Downloading https://files.pythonhosted.org/packages/69/81/a8f44c4b613717c25e0cdabf405e942fc7c7bcdedf3198c58c79fdbababc0/PyAutoGUI-0.9.38.tar.gz (47kB)
    100% |#####| 51kB 103kB/s
Collecting pymsgbox (from pyautogui)
  Downloading https://files.pythonhosted.org/packages/b6/65/86379ede1db26c40e7972d7a41c69cdf12cc6a0f143749aabf67ab8a41a1/PyMsgBox-1.0.6.zip
Collecting PyTweening>=1.0.1 (from pyautogui)
  Downloading https://files.pythonhosted.org/packages/b9/f8/c32a58d6e4dff8aa5c27e907194d69f3b57e525c2e4af96f39c6e9c854d2/PyTweening-1.0.3.zip
Collecting Pillow (from pyautogui)
  Downloading https://files.pythonhosted.org/packages/55/ea/305f61258278790706e69f01c53e107b0830ea5a4a69aa1f2c11fe605ed3/Pillow-5.3.0-cp37m-win_amd64.whl (1.6MB)
    100% |#####| 1.6MB 849kB/s
Collecting pyscreeze (from pyautogui)
  Downloading https://files.pythonhosted.org/packages/1c/80/ba95b654c92264675a8e67646d0f80066e0285c2a3ccd466126035c3fbbf/PyScreeze-0.1.18.tar.gz
Installing collected packages: pymsgbox, PyTweening, Pillow, pyscreeze, pyautogui
  Running setup.py install for pymsgbox ... done
  Running setup.py install for PyTweening ... done
  Running setup.py install for pyscreeze ... done
  Running setup.py install for pyautogui ... done
Successfully installed Pillow-5.3.0 PyTweening-1.0.3 pyautogui-0.9.38 pymsgbox-1.0.6 pyscreeze-0.1.18
```

Once the package is installed, you will check whether it is installed properly, or not. Open Python Shell from the Start Menu In the shell window displayed, try to import the package by typing the command `import` followed by the library name. Type `import space pyautogui`. If the command moves to the next prompt without throwing an error, the package was installed properly.

## 5.3 Cursor Control using pyautogui

We will know how to control the mouse cursor using the pyautogui library. First, open the python shell from the windows menu. Import the pyautogui library in the python shell using the `import` command. The prompt will be return to the next line, once the library is imported.

In this section, you'll learn how to move the mouse and track its position on the screen using PyAutoGUI, but first you need to understand how pyAutoGUI works with coordinates.

The mouse functions of PyAutoGUI use x- and y-coordinates of the pixels on the screen, just like any image. So, the resolution of a screen, which determines how many pixels wide and tall your screen is, is a significant parameter. The `pyautogui.size()` function returns a two-integer tuple of the screen's width and height in pixels. Depending on your screen's resolution, your return value may be different. You can store the width and height from `pyautogui.size()` in variables like `width` and `height`, to access them in your programs, if needed. You can also determine the mouse's current position by calling the `pyautogui.position()` function. The function will return a tuple of the mouse cursor's x and y positions at the time of the function call. Of course, the return values will vary depending on where your mouse cursor is.

Now, that you understand how to get the resolution of your screen and get the current mouse cursor position, let's move the mouse pro-grammatically.

Open a new file in Python IDLE. Import the pyautogui library using the import command. Before you jump in to automating your mouse controls, you should know how to escape problems that may arise. Python can move your mouse at an incredible speed. In fact, it might be too fast for other programs to keep up with. Also, if something goes wrong but your program keeps moving the mouse around, it will be hard to tell what exactly the program is doing or how to recover from the problem. Stopping the program can be difficult if the mouse is moving around on its own, preventing you from clicking the IDLE window to close it. Fortunately, there are several ways to prevent or recover from GUI automation problems. You can tell your script to wait after every function call, giving you a short window to take control of the mouse and keyboard if something goes wrong.

To do this, set the pyautogui.PAUSE variable to the number of seconds you want it to pause.

For example, after setting pyautogui.PAUSE= 1.5, every PyAutoGUI function call will wait one and a half seconds after performing its action. Non-PyAutoGUI instructions will not have this pause.

**PAUSE:**

Syntax: pyautogui.PAUSE = 1

Output:

Pause after each pyautogui call

PyAutoGUI also has a fail-safe feature. Moving the mouse cursor to the upper-left corner of the screen, will cause PyAutoGUI to raise an exception. The fail-safe feature will stop the program if you quickly move the mouse as far up and left as you can. You can disable this feature by setting pyautogui.FAILSAFE = False.

**FAILSAFE:**

Syntax: pyautogui.FAILSAFE = False

Output:

Pause after each pyautogui call

Let us now, finally, control the motion of the mouse cursor. The pyautogui.moveTo() function will instantly move the mouse cursor to a specified position on the screen. Integer values for the x- and y-coordinates make up the function's first and second arguments, respectively. An optional duration argument specifies the number of seconds it should take to move the mouse to the destination. If you leave it out, the default is 0 for instantaneous movement. Note that all of the duration keyword arguments in PyAutoGUI functions are optional.

**moveTo:**

Syntax: pyautogui.moveTo(x, y)

Output:

Current mouse cursor position.

You will now write for loop to move the mouse cursor clockwise in a square pattern for a total of ten times. As you had passed a duration as 0, to the pyautogui.moveTo() function, the mouse cursor would instantly teleport from one point to the next point. Now, replicate this for 3 more times and change the location of the pixels, that the cursor should move to.

```
import pyautogui
pyautogui.PAUSE = 1
pyautogui.FALLSAFE = False
## Move mouse cursor in a square pattern within the 4 coordinates provided
for i in range(10):
    pyautogui.moveTo(300, 300, duration = 0)
    pyautogui.moveTo(400, 300, duration = 0)
    pyautogui.moveTo(400, 400, duration = 0)
    pyautogui.moveTo(300, 400, duration = 0)
```

You can see that the mouse cursor makes a clockwise square pattern at a particular location, for 10 times, as you had a for loop that runs for 10 times. Note that the mouse cursor moves from one point to the next, instantaneously, as expected.



If you changed the duration to 0.25 seconds in the moveTo functions, which you had written. Notice that each movement of the mouse cursor now takes a quarter of a second, as specified by the duration=0.25 keyword argument. Note that even if you forcefully move the cursor away and leave, it will come back to the next point, which you had specified and continue the cursor motion, which you had programmed.

The `pyautogui.moveRel()` function moves the mouse cursor relative to its current position. The following piece of code moves the mouse in the same square pattern, except it begins the square from wherever the mouse happens to be on the screen when the code starts running. `pyautogui.moveRel()` also takes three arguments: How many pixels to move horizontally to the right, how many pixels to move vertically downward, and (optionally) how long it should take to complete the movement. A negative integer for the first or second argument will cause the mouse to move left or upward, respectively.

```
import pyautogui
pyautogui.PAUSE = 1
pyautogui.FALLSAFE = False
## Move mouse cursor in a square pattern, relative to its current position
for i in range(10):
    pyautogui.moveRel(100, 0, duration=0.25)
    pyautogui.moveRel(0, 100, duration=0.25)
    pyautogui.moveRel(-100, 0, duration=0.25)
    pyautogui.moveRel(0, -100, duration=0.25)
```

The mouse cursor moves in the square pattern, relative to the current cursor position, unlike in the previous case. You can increase the duration of the function, to make the mouse cursor move slowly.

## 5.4 Designing the Program for the Computer Vision based Mouse

Most of the stages in the pipeline of the Computer Vision based Mouse are the same, or are very similar to the Object Tracking pipeline. So, we will use the code we had written for the object tracking, and then modify it, and build on top of it.

Now, for controlling the mouse, you will have to import the `pyautogui` library. Earlier, we had mentioned how to control the cursor on your screen, programmatically. In this project, you will learn to control the cursor based on markers on your palm, which will be captured using the camera. We will be using two different colors - Yellow and Green, where, yellow will move the cursor pointer and green will help in clicking. We had earlier used a yellow object for object tracking. So, the range defined for that object, can be re-used to track the yellow marker.

```
lower_yellow = np.array([20,100,100]) # Move Cursor Pointer
upper_yellow = np.array([40,255,255])
```

In case your choice of color for the object is different from the marker you will use for this project, then change the values based on the color of the marker. Now, we will do the same for the other marker. You will assign a range for the green color, by finding the RGB values first, and then finding the HSV values from any online color conversion tool. You then had written an infinite loop to read frames from the video captured by your webcam. After you read the frame using the `cam.read()` function, you will flip the frame about the vertical axis. This is done, as the webcam, by default shows the mirror image, and so the marker motion is also flipped. This will make the cursor move in the opposite direction. To avoid this and make the interface more intuitive, you will flip the frame using the `cv.flip()` function.

The first argument is the source image, and the second argument determines the axis about which the image has to be flipped. The flag, 0 flips the image about the x-axis, The flag 1 flips it about the y-axis and -1 flips the image about both the axes. You will be flipping the frame in y axis alone, hence you will be using 1 in the function. The output will be assigned to the same variable, frame.

#### ***Flip()***

Syntax: `cv.flip(src,flipcode)`

Parameter:

src – input image

flipcode - A flag specify how to flip

0 - flipping x- axis

1 - flipping y- axis

-1 - flipping x and y axis

Output:

The function flips a 2D array vertical, horizontal or both axis

As we had learnt, we will be using a part of the video frame, to record the marker and control the cursor. So, after smoothening the image using Gaussian Blur, we will define a Region of Interest and track the motion of the marker only in that region. We will define a mask to select the region of interest. First, you will create a completely black mask, by creating a zeros array of the same size as the frame captured by the webcam. We will use the `np.zeros_like` function, to do the same. We will then change the black pixels in a square region, say from 50 to 350 in either directions, to white pixels. We will then get the region of interest of the frame, by performing a BIT-WISE-AND operation between the original frame and the mask.

#smothen the image

```
image_smooth=cv.GaussianBlur(frame,(7,7),0)
```

#Definr Region Of Interest

```
mask=np.zeros_like(frame)
```

```
mask[50:350, 50:350]=[255, 255, 255]
```

```
image_roi=cv.bitwise_and(image_smooth, mask)
```

This function calculates the per-element bit-wise conjunction of two arrays. The first argument is the smooth image, and the second argument is the mask image. As we know, according to the AND logic table, if both the inputs are high, the output will be high. If any one of the inputs is low, the output will be low. In this case, the mask image in the square region has all white pixels, and the rest are all black pixels, which are equivalent to 0s. So, only the pixels of the frame, which lie in the region are shown as an output, and the rest of the pixels are blacked out. Assign the output of the function, to the variable image\_roi.

#### ***bitwise\_and()***

Syntax: cv.bitwise\_and(src1,mask)

Parameter:

Src1 – first image array

mask - mask image array

Output:

The function calculate the pre-element bit-wise conjunction of two arrays.

Now, we will draw a rectangle and a grid on the actual frame, to indicate this ROI. So, we will use the cv.rectangle() function, to draw a rectangle on the image, frame. The top left vertex of the rectangle is 50, 50. The bottom right is at 350, 350. we will assign the rectangle to be drawn in red color, with a thickness of 2 pixels. You will also draw 4 thin lines on this rectangle - 2 vertical and 2 horizontal, to create the 3x3 grid. The end points of the grid lines, can be easily obtained as the ones shown here. It will detect the largest yellow colored object, in the Region of Interest.

## Drawing a rectangle on the image

```
cv.rectangle(frame, (50,50), (350,350), (0,0,255), 2)
cv.line(frame, (150,50), (150,350), (0,0,255), 1)
cv.line(frame, (250,50), (250,350), (0,0,255), 1)
cv.line(frame, (50,150), (350,150), (0,0,255), 1)
cv.line(frame, (50,250), (350,250), (0,0,255), 1)
```

We will next have to convert the masked image, into the H S V Color space. So, we have to change the argument for the cvtColor function, to image\_roi. And Keeping the rest of the pipeline as the same.

##Threshold the image for yellow color

```
image_hsv = cv.cvtColor(image_roi, cv.COLOR_BGR2HSV)
image_threshold = cv.inRange(image_hsv, lower_yellow,
```

Earlier, we had used the bounding rectangle feature of the contours. We will now need to use the centroid of the contour, to track the motion of the marker. So, we will remove the code for the bounding rectangle. And, include the code, to find the centroid of the yellow object contour.

We will have to use the moments function, to do the same. Also, we will determine the position of the marker, relative to the grid. If the value of  $c_x$ , is less than 150, then the marker is in the left sector. If the value is more than 250, then the marker is in the right sector. And, if the value lies between 150, and 250, the marker is in the middle sector. And, if the pointer is in the left sector, the cursor should move towards the left. So, we will set the relative distance for the mouse motion, in the horizontal axis, to be -20 pixels. Similarly, when the pointer is in the right sector, the cursor should move towards the right. We will set the distance, for the cursor motion, to be 20 pixels. Finally, set it to 0, if the marker is in the central zone. By Repeat similar steps, using the if conditions, to determine the relative y position of the marker. We will now use the moveRel function, of the pautogui library, to move the cursor, based on the distances you had assigned earlier.

```
## pointer on video

M = cv.moments(cnt)

if (M['m00']!=0):

    cx = int(M['m10']/M['m00'])

    cy = int(M['m01']/M['m00'])

    cv.circle(frame, (cx,cy), 4, (0,255,0), -1)

    #Cursor Motion

    if cx < 150:

        dist_x = -20

    elif cx > 250:

        dist_x = 20

    else:        ## the marker is in the central zone.

        dist_x = 0

    if cy < 150:

        dist_y = -20

    elif cy > 250:

        dist_y = 20

    else:

        dist_y = 0

    pyautogui.moveRel(dist_x, dist_y, duration=0.25)
```

Now for adding the clicking option, we will use a green colored object. We will use the same pipeline, that you had used for yellow, to detect the green marker. We will use the in range function, to find the range in the h s v, between the lower green, and upper green bounds, which we had provided earlier. We will assign the output of the function

to a variable, image\_threshold\_green. By using the find contour function, you will find the contour of the image for green threshold.

We will check if the contours list is not empty, which means, a green colored contour is actually detected. If the condition is true, then it will call the click function, of the pautogui library, which is by default the left click. And By Giving a delay for 1000 milliseconds, to wait at least 1 second, between detecting consecutive clicks.

As the same as the green color, we will use the red color for right click, by using the click function and passing the parameter button='right'.

We can control the mouse using the yellow marker, and we can just flash the green marker, whenever you need to left click, and we can just flash the red marker, whenever you need to right click. By Making sure there is no interference of any background object, with the marker colors.

```
import cv2 as cv
import numpy as np
import pyautogui
from matplotlib import pyplot as plt
cam = cv.VideoCapture(0)
lower_yellow = np.array([20,100,100]) ##Yellow_Move Cursor Pointer
upper_yellow = np.array([40,255,255])
lower_green = np.array([50,100,100]) ##Green_for Clicking(left)
upper_green = np.array([80,255,255])
lower_red = np.array([0,125,125]) ##red_for Clicking(right)
upper_red = np.array([10,255,255])
while(True):
    ret, frame = cam.read() ##in usual the cursor move in the opposite direction
    frame = cv.flip(frame, 1) ## that to avoid that problem
    #smothen the image
    image_smooth = cv.GaussianBlur(frame,(7,7),0)
```



```
#Definr Region Of Interest
```

```
mask = np.zeros_like(frame)
```

```
mask[50:350, 50:350] = [255, 255, 255]
```

```
image_roi = cv.bitwise_and(image_smooth, mask)
```

```
## Drawing a rectangle on the image
```

```
cv.rectangle(frame, (50,50), (350,350), (0,0,255), 2)
```

```
cv.line(frame, (150,50), (150,350), (0,0,255), 1)
```

```
cv.line(frame, (250,50), (250,350), (0,0,255), 1)
```

```
cv.line(frame, (50,150), (350,150), (0,0,255), 1)
```

```
cv.line(frame, (50,250), (350,250), (0,0,255), 1)
```

```
##Threshold the image for yellow color
```

```
#img_hsv = cv.cvtColor(image_smooth, cv.COLOR_BGR2HSV)
```

```
image_hsv = cv.cvtColor(image_roi, cv.COLOR_BGR2HSV)
```

```
image_threshold = cv.inRange(image_hsv, lower_yellow, upper_yellow)
```

```
#find contours
```

```
contours, heirarchy = cv.findContours(image_threshold, \
```

```
cv.RETR_TREE, \
```

```
cv.CHAIN_APPROX_NONE)
```

```
#find the index of the largest contour
```

```
if(len(contours)!=0):
```

```
    areas = [cv.contourArea(c) for c in contours]
```

```
    max_index = np.argmax(areas)
```

```
    cnt = contours[max_index]
```

## #Cursor Motion

```
if cx < 150:    ##if the pointer is in the left sector, the cursor should move towards the left.
    dist_x = -20

elif cx > 250:    ##when the pointer is in the right sector, the cursor should move towards the right.
    dist_x = 20

else:    ## the marker is in the central zone.
    dist_x = 0

if cy < 150:
    dist_y = -20

elif cy > 250:
    dist_y = 20

else:
    dist_y = 0

pyautogui.moveRel(dist_x, dist_y, duration=0.25)

## Check for (left)click

image_threshold_green = cv.inRange(image_hsv, lower_green, upper_green)
contours_green, heirarchy = cv.findContours(image_threshold_green, \
                                             cv.RETR_TREE, \
                                             cv.CHAIN_APPROX_NONE)

if (len(contours_green) != 0): ##check if the contours list is not empty, which means, a green colored contour is actually detected.
    pyautogui.click()
    cv.waitKey(1000)

## Check for click(right)

image_threshold_red = cv.inRange(image_hsv, lower_red, upper_red)
contours_red, heirarchy = cv.findContours(image_threshold_red, \
                                           cv.RETR_TREE, \
                                           cv.CHAIN_APPROX_NONE)

if (len(contours_red) != 0): ##check if the contours list is not empty, which means, a red colored contour is actually detected.
    pyautogui.click(button='right')
    cv.waitKey(1000)
```

```
cv.imshow('Frame', frame)

#key = cv.waitKey(10)
key = cv.waitKey(100)

if key == 27:
    break

#img_RGB= cv.cvtColor(frame, cv.COLOR_BGR2RGB)
#plt.imshow(img_RGB)
#plt.show()

cam.release()

cv.destroyAllWindows()
```

The results as shown:

