MIT-ADT UNIVERSITY PUNE, INDIA A leap towards World Class Education	**MIT School of Engineering, Pune** **Dept. of Information Technology** **TY IT - APL-CV Lab; VIth Semester** **Academic Year : 2019-20**

List of Assignments

Assn. No.	**Title**
	Based on UNIT 1
1.	Demonstrate working of various scan conversion algorithms for Line and Circle Drawing.
	Based on UNIT 2
2.	Perform following transformations with all cases on 2D Objects - Triangle and Rectangle : Translation, Rotation, Scaling, Reflection and Shearing
	Based on UNIT 3
3.	1. Working with basic operations on image: ● Reading and Writing on images ● Blending of images and geometric shapes on images ● Performing geometric, affine and perspective transformation on images. 2. Capturing Video through Webcam or Analysing Static Video (Reading, Writing and Displaying)
	Based on UNIT 4
4.	1. Working with various preprocessing operations on an image: ● Image Thresholding, Gradient and Histograms ● Edge Detection on Image ● Morphological operations on an Image 2. Motion detection and estimation using contour-based segmentation.
	Based on UNIT 5
5.	1. Face and Eye Detection from Image and Video 2. Object detection and tracking using template matching.

Practical Examples of Unit - 1 & 2

APL-CV-Lab Assignments - Part 1: Graphics Programming

1. Demonstrate working of Line Drawing and Circle Drawing Algorithms
2. Perform following transformations with all cases on 2D Objects - Triangle and Rectangle Translation, Rotation, Scaling, Reflection and Shearing

Basics of Graphics Programming:

1. Download graphics.py from https://mcsp.wartburg.edu/zelle/python/
2. Copy graphics.py in C://Python/Lib/Site-Packages//
3. Open Pycharm and create new python file
4. Write following lines in Sample.py
5. from graphics import *
6. win = GraphWin()

Sample Programs : Basic Graphics Programming with graphics.py

- Hello.py

```
#print("Hello World")
from graphics import *
win = GraphWin()
pt = Point(100, 100)
pt.draw(win)
pt.setFill('blue')
cir = Circle(pt, 25)
cir.draw(win)
cir.setOutline('red')
cir.setFill('yellow')
line = Line(pt, Point(80, 90))
line.draw(win)
line.setFill('green')
rect = Rectangle(Point(150, 150), pt)
rect.draw(win)
rect.setFill('pink')
win.getMouse()
win.close()
```

- Sample_Color.py

```
from graphics import *
import random, time
def main():
  win = GraphWin("Random Circles", 300, 300)
  for i in range(75):
        r = random.randrange(256)
        b = random.randrange(256)
        g = random.randrange(256)
        color = color_rgb(r, g, b)
        radius = random.randrange(3, 40)
        x = random.randrange(5, 295)
        y = random.randrange(5, 295)
        circle = Circle(Point(x, y), radius)
        circle.setFill(color)
        circle.draw(win)
        time.sleep(.05)
main()
```

- Smiley.py

```
from graphics import *
def main():
  win = GraphWin('Face', 200, 150) # give title and dimensions
  head = Circle(Point(40,100), 25) # set center and radius
  head.setFill("yellow")
  head.draw(win)
  eye1 = Circle(Point(30, 105), 5)
  eye1.setFill('blue')
  eye1.draw(win)
  eye2 = Line(Point(45, 105), Point(55, 105)) # set endpoints
  eye2.setWidth(3)
  eye2.draw(win)
  mouth = Oval(Point(30, 90), Point(50, 85)) # set corners of bounding box
  mouth.setFill("red")
  mouth.draw(win)
  label = Text(Point(100, 120), 'A face')
  label.draw(win)
  message = Text(Point(win.getWidth()/2, 20), 'Click anywhere to quit.')
  message.draw(win)
  win.getMouse()
  win.close()
main()
```

Problem 1.1 : Demonstrate working of Line Drawing Algorithms(DDA and Bresenhams')

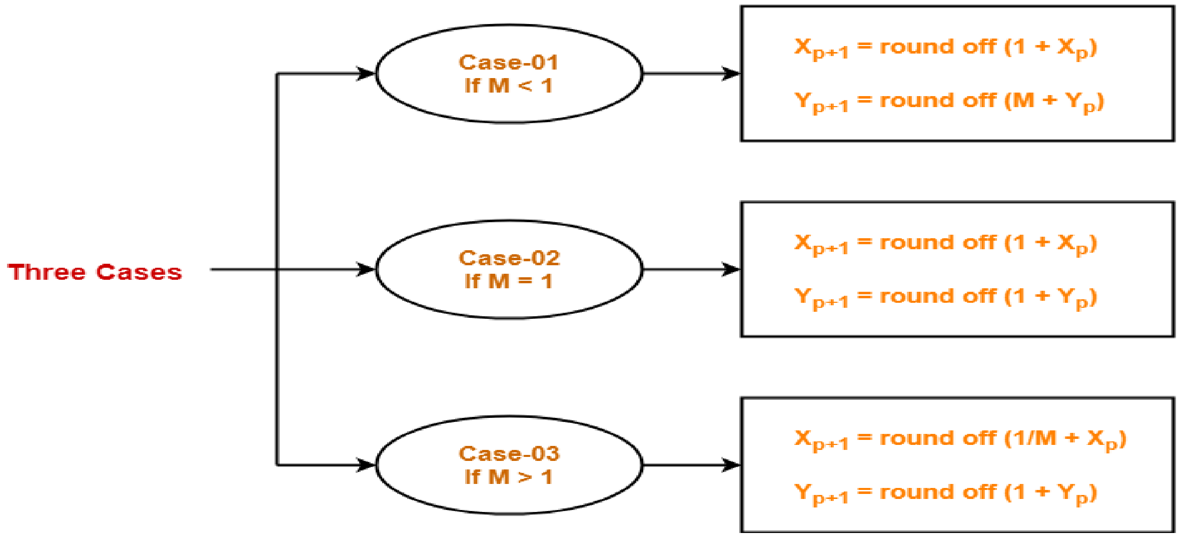
- Write theory and algorithms for both methods

Digital Differential Analyzer is the simple line generation algorithm which is explained step by step here.

Pseudocode for DDA:

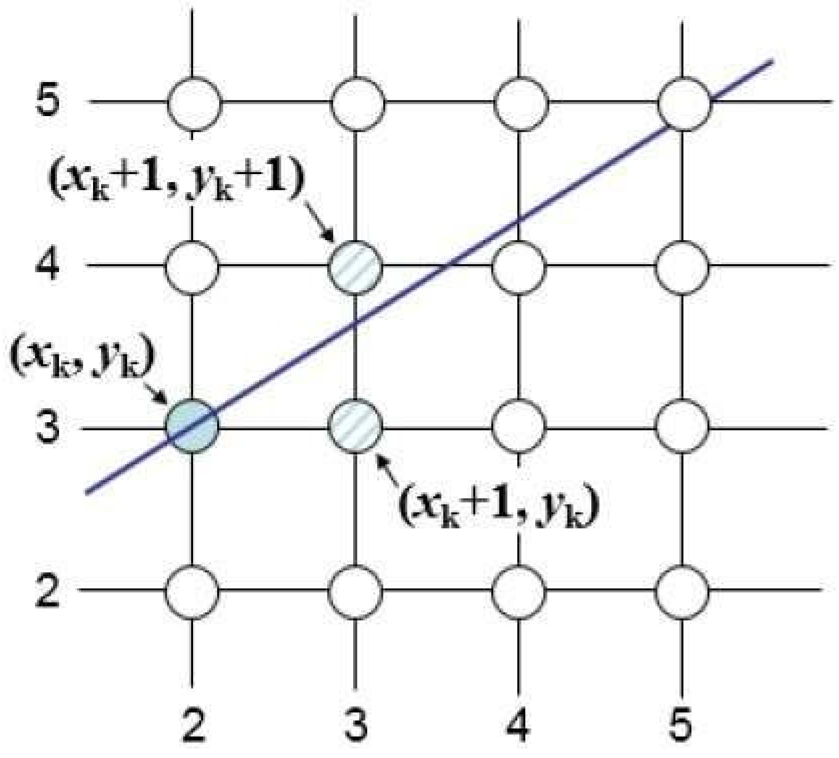
```
Step 1 − Get the input of two endpoints
(X0,Y0) and ((X1,Y1).
Step 2 − Calculate the difference between two endpoints.
dx = X1 - X0
dy = Y1 - Y0
Step 3 − Based on the calculated difference in step-2, you need to identify the number of steps to put pixel. If dx > dy, then you need more steps in x coordinate; otherwise in y coordinate.
if (absolute(dx) > absolute(dy))
  Steps = absolute(dx);
else
  Steps = absolute(dy);
Step 4 − Calculate the increment in x coordinate and y coordinate.
Xincrement = dx / (float) steps;
Yincrement = dy / (float) steps;
Step 5 − Put the pixel by successfully incrementing x and y coordinates accordingly and complete the drawing of the line.
for(int v=0; v < Steps; v++)
{
  x = x + Xincrement;
  y = y + Yincrement;
  putpixel(Round(x), Round(y));
}
```

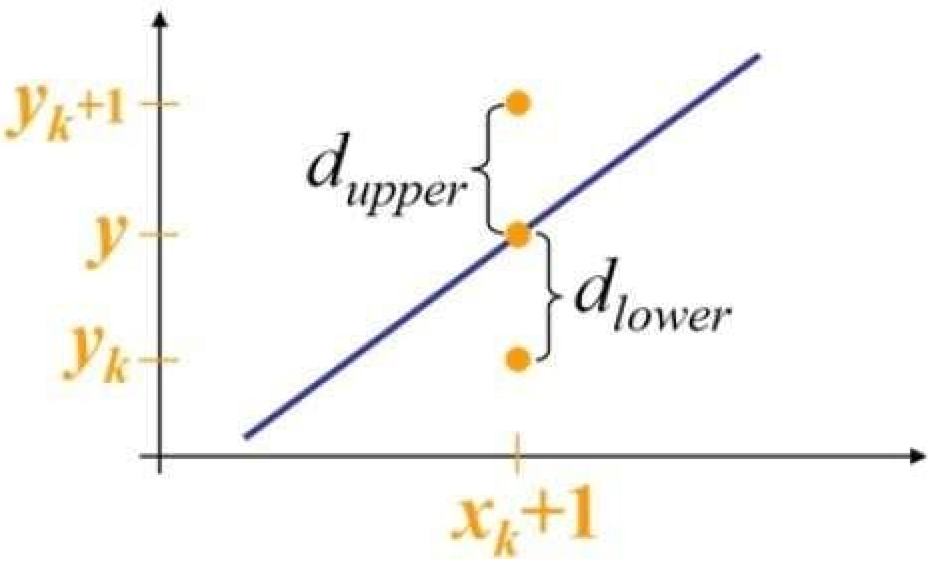
Cases for DDA Algorithm:


Three Cases
Case-01
If M < 1
X_{p+1} = round off (1 + X_p)
Y_{p+1} = round off (M + Y_p)
Case-02
If M = 1
X_{p+1} = round off (1 + X_p)
Y_{p+1} = round off (1 + Y_p)
Case-03
If M > 1
X_{p+1} = round off (1/M + X_p)
Y_{p+1} = round off (1 + Y_p)

Bresenham's Line Generation

The Bresenham algorithm is another incremental scan conversion algorithm. The big advantage of this algorithm is that it uses only integer calculations. Moving across the x axis in unit intervals and at each step choose between two different y coordinates.





Pseudocode for Bresenham algorithm for slope m < 1 −

```
Step 1 − Input the two end-points of line, storing the left end-point in
(x0,y0) and (x1,y1).
Step 2 − Plot the point (x0,y0)
Step 3 − Calculate the constants dx, dy, 2dy, and 2dy–2dx
and get the first value for the decision parameter as −
P0 =2dy−dx
Step 4 − At each Xk along the line, starting at k = 0, perform the following test −
If pk < 0, the next point to plot is
        (xk+1,yk) and
        Pk+1 = pk+2dy
```

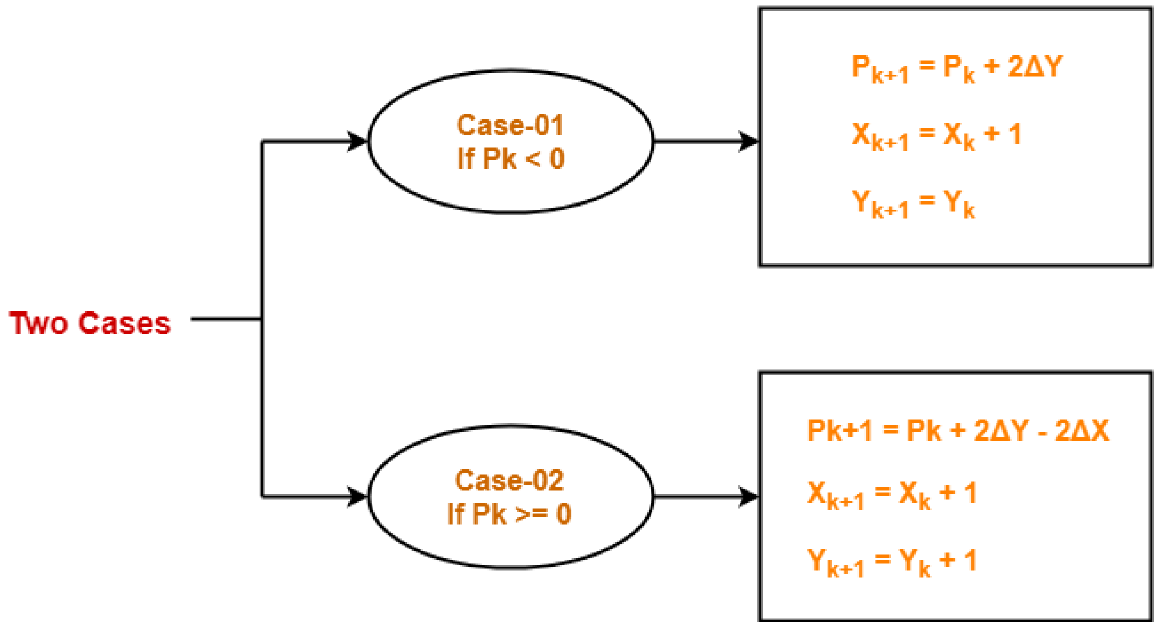
Otherwise,

(xk,yk+1)

pk+1=pk+2dy−2dx

Step 5 − Repeat step 4 dx–1 times.

Cases for Bresenham's Line Drawing Algorithm:



Problem 1.2 : Demonstrate working of Circle Drawing Algorithms (Bresenhams' and Mid-Point)

- Write theory and Algorithms for both methods
- Install PIL library by executing command - pip install pillow in command prompt

Bresenham Circle Drawing Algorithm attempts to generate the points of one octant.

Pseudocode for Bresenham's Circle Generation-

Given-

- Centre point of Circle = (X_0, Y_0)
- Radius of Circle = R

The points generation using Bresenham Circle Drawing Algorithm involves the following steps-

Step-1: Assign the starting point coordinates (X_0, Y_0) as-

- $X_0 = 0$
- $Y_0 = R$

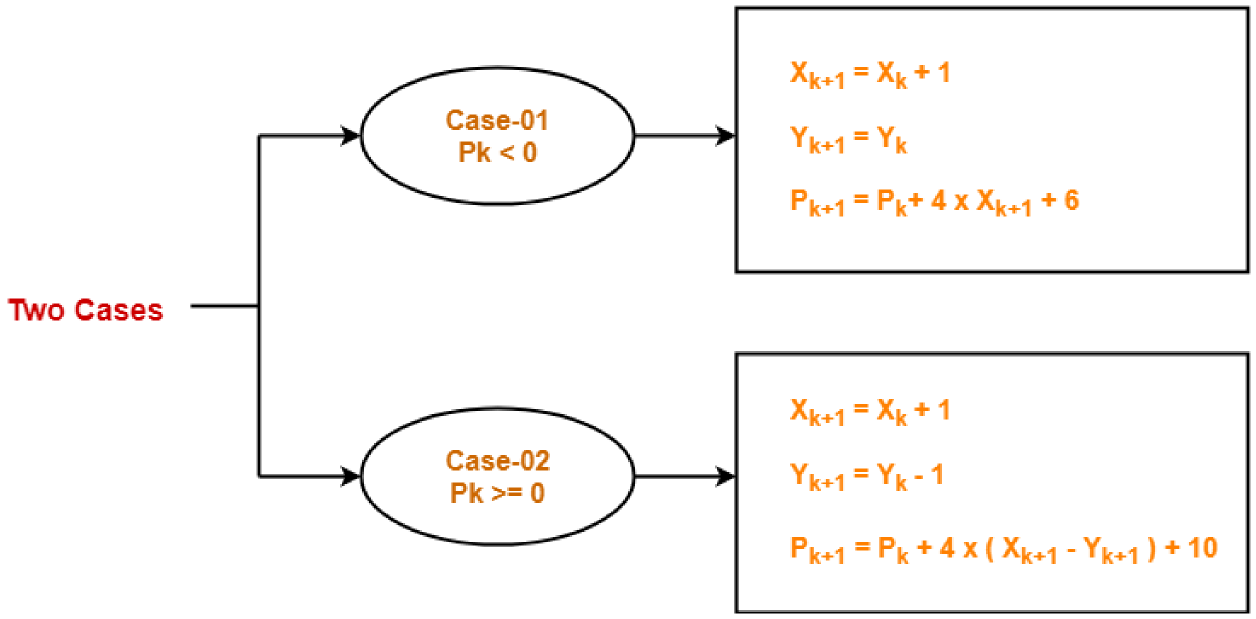
Step-2: Calculate the value of initial decision parameter P_0 as-

$P_0 = 3 – 2 \times R$

Step-3: Suppose the current point is (X_k, Y_k) and the next point is (X_{k+1}, Y_{k+1}).

Find the next point of the first octant depending on the value of decision parameter P_k.

Follow the below two cases-



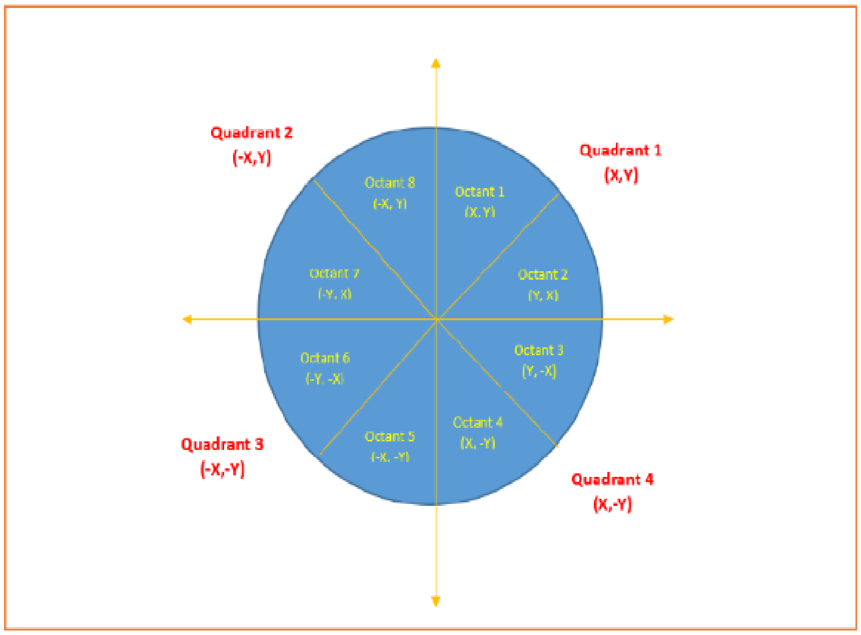
Step-4: If the given centre point (X_0, Y_0) is not (0, 0), then do the following and plot the point-

- $X_{plot} = X_c + X_0$
- $Y_{plot} = Y_c + Y_0$

Here, (X_c, Y_c) denotes the current value of X and Y coordinates.

Step-5: Keep repeating Step-03 and Step-04 until X_{plot} => Y_{plot}.

Step-6: Step-05 generates all the points for one octant. To find the points for the other seven octants, follow the eight symmetry property of the circle. This is depicted by the following figure-



Mid Point Circle Drawing Algorithm attempts to generate the points of one octant.

Pseudocode for Bresenham's Circle Generation-

Given-

- Centre point of Circle = (X_0, Y_0)
- Radius of Circle = R

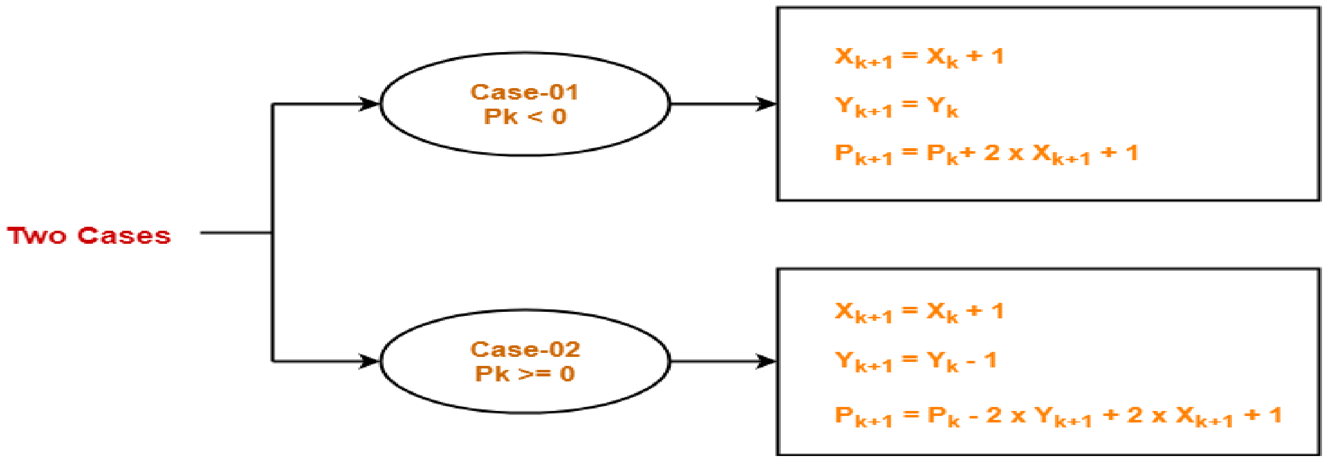
The points generation using Mid Point Circle Drawing Algorithm involves the following steps-

Step-1: Assign the starting point coordinates (X_0, Y_0) as-

- $X_0 = 0$
- $Y_0 = R$

Step-2: Calculate the value of initial decision parameter P_0 as- $P_0 = 1 - R$
Step-3: Suppose the current point is (X_k, Y_k) and the next point is (X_{k+1}, Y_{k+1}).
Find the next point of the first octant depending on the value of decision parameter P_k.

Follow the below two cases-



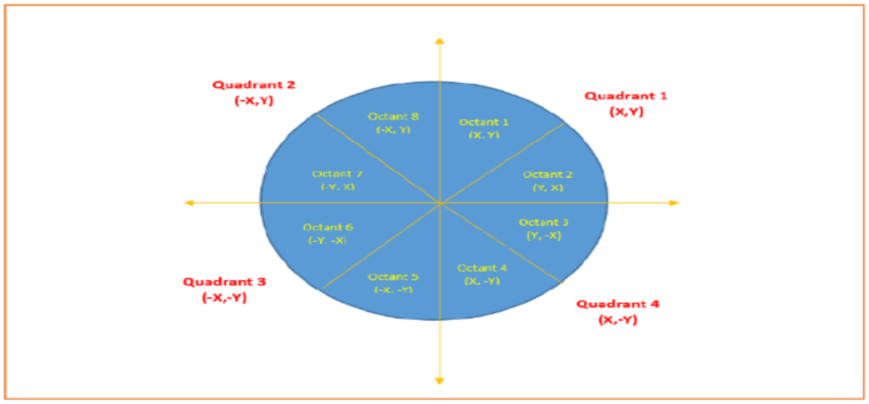
Step-4: If the given centre point (X_0, Y_0) is not (0, 0), then do the following and plot the point-

- $X_{plot} = X_c + X_0$
- $Y_{plot} = Y_c + Y_0$

Here, (X_c, Y_c) denotes the current value of X and Y coordinates.
Step-5: Keep repeating Step-03 and Step-04 until $X_{plot} >= Y_{plot}$.

Step-6: Step-5 generates all the points for one octant.
To find the points for the other seven octants, follow the eight symmetry property of the circle.

This is depicted by the following figure-



Problem 2 : Perform following transformations with all cases on 2D Objects - Triangle and Rectangle : Translation, Rotation, Scaling, Reflection and Shearing

Transformation is a process of modifying and re-positioning the existing graphics.

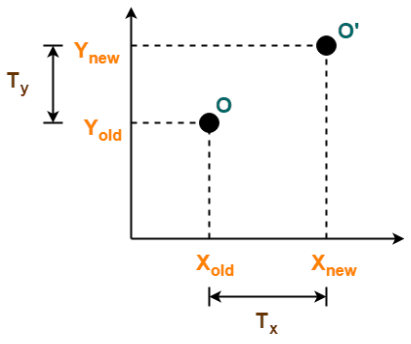
- 2D Transformations take place in a two dimensional plane.
- Transformations are helpful in changing the position, size, orientation, shape etc of the object.

2D Translation is a process of moving an object from one position to another in a two dimensional plane. Let-

- Initial coordinates of the object O = (X_{old}, Y_{old})
- New coordinates of the object O after translation = (X_{new}, Y_{new})
- Translation vector or Shift vector = (T_x, T_y)

Given a Translation vector (T_x, T_y)-

- T_x defines the distance the X_{old} coordinate has to be moved.
- T_y defines the distance the Y_{old} coordinate has to be moved.



2D Translation in Computer Graphics

This translation is achieved by adding the translation coordinates to the old coordinates of the object as-

- $X_{new} = X_{old} + T_x$ (This denotes translation towards X axis)
- $Y_{new} = Y_{old} + T_y$ (This denotes translation towards Y axis)

In Matrix form, the above translation equations may be represented as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix}$$

Translation Matrix

- The homogeneous coordinates representation of (X, Y) is (X, Y, 1).
- Through this representation, all the transformations can be performed using matrix / vector multiplications.

The above translation matrix may be represented as a 3 x 3 matrix as-

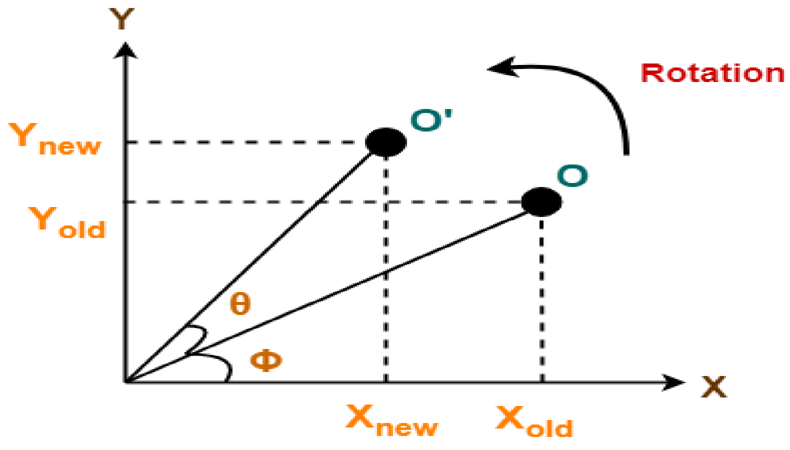
$$\begin{bmatrix} X_{new} \\ Y_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ 1 \end{bmatrix}$$

Translation Matrix

(Homogeneous Coordinates Representation)

2D Rotation is a process of rotating an object with respect to an angle in a two dimensional plane.
Consider a point object O has to be rotated from one angle to another in a 2D plane.
Let-

- Initial coordinates of the object O = (X_{old}, Y_{old})
- Initial angle of the object O with respect to origin = Φ
- Rotation angle = θ
- New coordinates of the object O after rotation = (X_{new}, Y_{new})



2D Rotation in Computer Graphics

This rotation is achieved by using the following rotation equations-

- $X_{new} = X_{old} \times \cos\theta - Y_{old} \times \sin\theta$
- $Y_{new} = X_{old} \times \sin\theta + Y_{old} \times \cos\theta$

In Matrix form, the above rotation equations may be represented as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

Rotation Matrix

For homogeneous coordinates, the above rotation matrix may be represented as a 3 x 3 matrix as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ 1 \end{bmatrix}$$

Rotation Matrix
(Homogeneous Coordinates Representation)

In computer graphics, **scaling** is a process of modifying or altering the size of objects.

- Scaling may be used to increase or reduce the size of objects.
- Scaling subjects the coordinate points of the original object to change.
- Scaling factor determines whether the object size is to be increased or reduced.
- If scaling factor > 1, then the object size is increased.
- If scaling factor < 1, then the object size is reduced.

Consider a point object O has to be scaled in a 2D plane. Let-

- Initial coordinates of the object O = (X_{old}, Y_{old})
- Scaling factor for X-axis = S_x
- Scaling factor for Y-axis = S_y
- New coordinates of the object O after scaling = (X_{new}, Y_{new})

This scaling is achieved by using the following scaling equations-

- $X_{new} = X_{old} \times S_x$
- $Y_{new} = Y_{old} \times S_y$

In Matrix form, the above scaling equations may be represented as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

Scaling Matrix

For homogeneous coordinates, the above scaling matrix may be represented as a 3 x 3 matrix as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ 1 \end{bmatrix}$$

Scaling Matrix
(Homogeneous Coordinates Representation)

2D Reflection in Computer Graphics-

- Reflection is a kind of rotation where the angle of rotation is 180 degree.
- The reflected object is always formed on the other side of the mirror.
- The size of the reflected object is the same as the size of the original object.

Consider a point object O has to be reflected in a 2D plane. Let-

- Initial coordinates of the object O = (X_{old}, Y_{old})
- New coordinates of the reflected object O after reflection = (X_{new}, Y_{new})

Reflection On X-Axis:

This reflection is achieved by using the following reflection equations-

- $X_{new} = X_{old}$
- $Y_{new} = -Y_{old}$

In Matrix form, the above reflection equations may be represented as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

Reflection Matrix

(Reflection Along X Axis)

For homogeneous coordinates, the above reflection matrix may be represented as a 3 x 3 matrix as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ 1 \end{bmatrix}$$

Reflection Matrix

(Reflection Along X Axis)

(Homogeneous Coordinates Representation)

Reflection On Y-Axis:

This reflection is achieved by using the following reflection equations-

- $X_{new} = -X_{old}$
- $Y_{new} = Y_{old}$

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

Reflection Matrix

(Reflection Along Y Axis)

In Matrix form, the above reflection equations may be represented as-
For homogeneous coordinates, the above reflection matrix may be represented as a 3 x 3 matrix as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ 1 \end{bmatrix}$$

Reflection Matrix

(Reflection Along Y Axis)

(Homogeneous Coordinates Representation)

2D Shearing is an ideal technique to change the shape of an existing object in a two dimensional plane.

1. Shearing in X direction
2. Shearing in Y direction

Consider a point object O has to be sheared in a 2D plane. Let-

- Initial coordinates of the object O = (X_{old}, Y_{old})
- Shearing parameter towards X direction = Sh_x
- Shearing parameter towards Y direction = Sh_y
- New coordinates of the object O after shearing = (X_{new}, Y_{new})

Shearing in X Axis-

Shearing in X axis is achieved by using the following shearing equations-

- $X_{new} = X_{old} + Sh_x \times Y_{old}$
- $Y_{new} = Y_{old}$

In Matrix form, the above shearing equations may be represented as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 1 & Sh_x \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

Shearing Matrix

(In X axis)

For homogeneous coordinates, the above shearing matrix may be represented as a 3 x 3 matrix as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ Sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ 1 \end{bmatrix}$$

Shearing Matrix
(In X axis)
(Homogeneous Coordinates Representation)

Shearing in Y Axis-

Shearing in Y axis is achieved by using the following shearing equations-

- $X_{new} = X_{old}$
- $Y_{new} = Y_{old} + Sh_y \times X_{old}$

In Matrix form, the above shearing equations may be represented as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ Sh_y & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \end{bmatrix}$$

Shearing Matrix
(In Y axis)

For homogeneous coordinates, the above shearing matrix may be represented as a 3 x 3 matrix as-

$$\begin{bmatrix} X_{new} \\ Y_{new} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & Sh_y & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_{old} \\ Y_{old} \\ 1 \end{bmatrix}$$

Shearing Matrix
(In Y axis)
(Homogeneous Coordinates Representation)

Practical Examples of Unit - 3, 4 & 5

APL-CV-Lab Assignments - Part 2: Image and Video Processing with OpenCV

3.1 Working with basic operations on image:

- Reading and Writing on images
- Blending of images and geometric shapes on images
- Performing geometric, affine and perspective transformation on images.

3.2 Capturing Video through Webcam or Analysing Static Video (Reading, Writing and Displaying)

4.1 Working with various preprocessing operations on an image:

- Image Thresholding, Gradient and Histograms
- Edge Detection on Image
- Morphological operations on an Image

4.2 Motion detection and estimation using contour-based segmentation.

5.1 Face and Eye Detection from Image and Video

5.2 Object detection and tracking using template matching.

Computer Vision is an interdisciplinary field that deals with how computers can be made to gain a high-level understanding from digital images or videos.

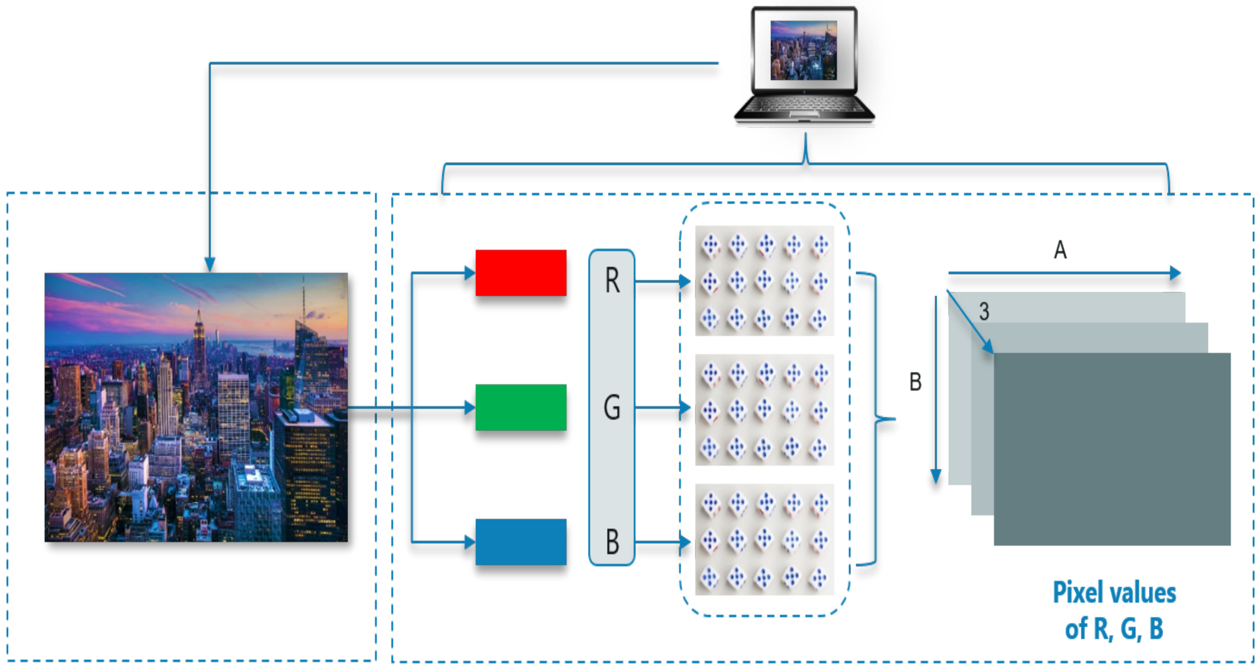
How Does A Computer Read An Image?

The computer reads any image as a range of values between 0 and 255.

For any color image, there are 3 primary channels – Red, green and blue. How it works is pretty simple. A matrix is formed for every primary color and later these matrices combine to provide a Pixel value for the individual R, G, B colors. Each element of the matrices provide data pertaining to the intensity of brightness of the pixel.

As shown, the size of the image here can be calculated as B x A x 3.

Note: For a black-white image, there is only one single channel.



What Is OpenCV?

OpenCV is a Python library which is designed to solve computer vision problems. OpenCV was originally developed in 1999 by Intel but later it was supported by Willow Garage. OpenCV supports a wide variety of programming languages such as C++, Python, Java etc. Support for multiple platforms including Windows, Linux, and MacOS. OpenCV Python is nothing but a wrapper class for the original C++ library to be used with Python. Using this, all of the OpenCV array structures get converted to/from NumPy arrays.

3.1.1 : Basic Operations on an Image

Loading an image:

```
import cv2
# colored Image
img = cv2.imread("WindowsLogo.jpg",1)
# Black and White (gray scale)
img_1= cv2.imread("WindowsLogo.jpg",0)
```

Image Shape/Resolution:

```
import cv2
img = cv2.imread("WindowsLogo.jpg",0)
print(img.shape)
```

Displaying an image:

```
import cv2
img = cv2.imread("WindowsLogo.jpg",0)
cv2.imshow("Image", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Resizing the image:

```
import cv2
img = cv2.imread("WindowsLogo.jpg",0)
cv2.imshow("Image", img)
resized_image = cv2.resize(img, (650,500))
cv2.imshow("Resized Image", resized_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

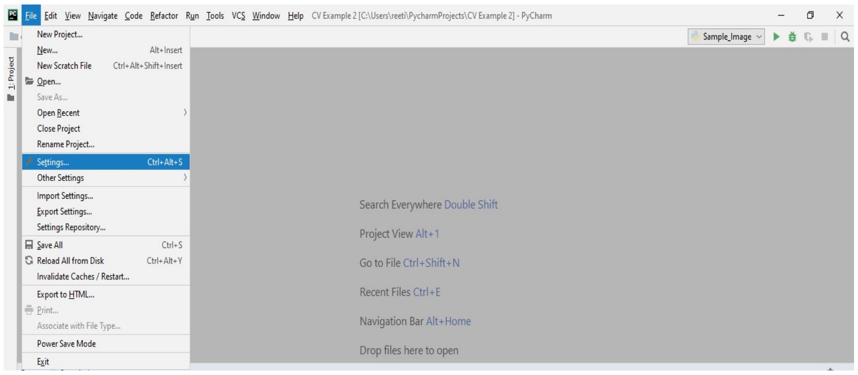
Sample Program : Sample_Image.py in PyCharm

Steps:

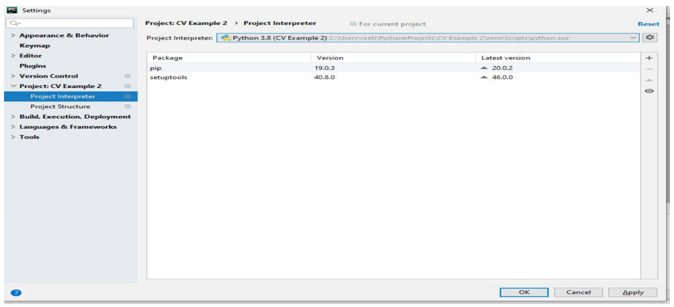
1. Open PyCharm IDE - **Before installing Pycharm, kindly download and install Python from https://www.python.org/downloads/ - Python 3.8.2. While configuring Python, give path **(Create New Folder as Python in C Drive) : C:\Python\Python38\ for installing python.exe.*** **If not configured, download it from https://www.jetbrains.com/pycharm/download/#section=windows - select Community and download. Pycharm files get installed in **C:\Program Files\JetBrains\PyCharm Community Edition 2019.3.3, after successful installation open PyCharm IDE.**

   Steps for configuring PyCharm:

   1. Open PyCharm IDE and select File Option - New - New Project - Name of Project (In Example - CV2 Example Project) and then again Select File-> Settings:



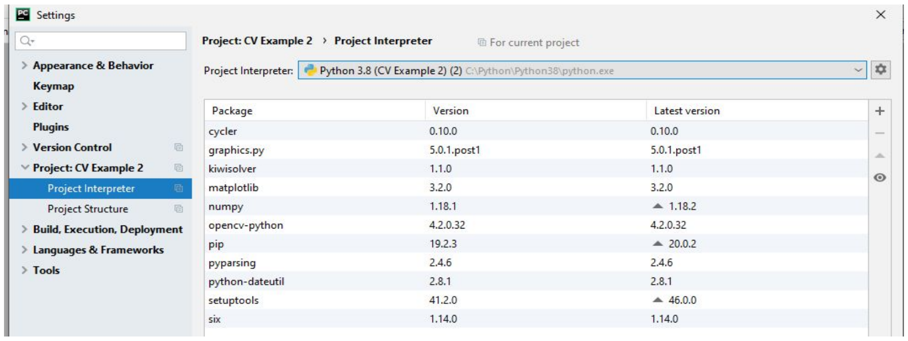
   2. Open Settings - > Select Project : CV Example 2 - > Project Interpreter and Project Structure - > Select Project Interpreter -> Default Interpreter is displayed:



Project Interpreter: Python 3.8 (CV Example 2) C:\Users\reeti\PycharmProjects\CV Example 2\venv\Scripts\python.exe

Package	Version	Latest version
pip	19.0.3	▲ 20.0.2
setuptools	40.8.0	▲ 46.0.0

   3. Change Project Interpreter Path to: *C:\Python\Python38\ for installing python.exe.*



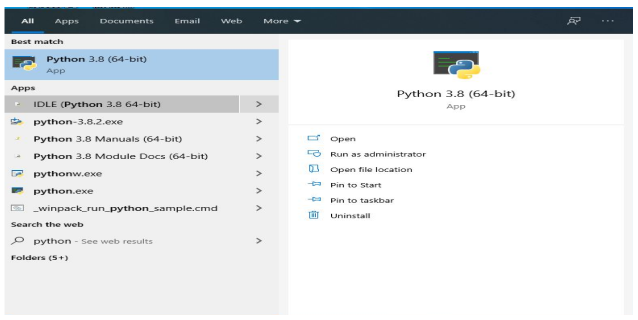
Project Interpreter: Python 3.8 (CV Example 2) (2) C:\Python\Python38\python.exe

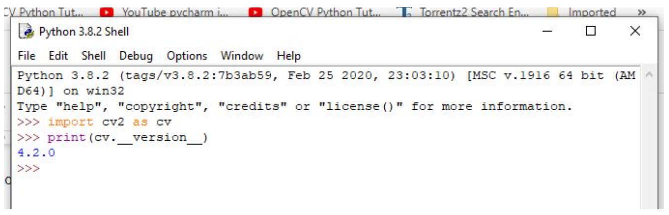
Package	Version	Latest version
cycler	0.10.0	0.10.0
graphics.py	5.0.1.post1	5.0.1.post1
kiwisolver	1.1.0	1.1.0
matplotlib	3.2.0	3.2.0
numpy	1.18.1	▲ 1.18.2
opencv-python	4.2.0.32	4.2.0.32
pip	19.2.3	▲ 20.0.2
pyparsing	2.4.6	2.4.6
python-dateutil	2.8.1	2.8.1
setuptools	41.2.0	▲ 46.0.0
six	1.14.0	1.14.0

***You can see there are many packages installed with this interpreter. Kindly install following packages through Command Prompt (Windows + R -> cmd) using pip command:

- Pip install opencv-python
- Pip install numpy
- Pip install matplotlib
- Open Python and type the following codes in Python terminal.

```
>>> import cv2 as cv
>>> print( cv.__version__ )#cv.two underscore version then two underscore
```





```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 23:03:10) [MSC v.1916 64 bit (AM
D64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import cv2 as cv
>>> print(cv.__version__)
4.2.0
>>>
```

3.1.2 : Arithmetic and Geometric operations on images

Blending of Images (Arithmetic Operations):

Blending of images is the addition of two images as addition of two number matrices.

Syntax: cv2.add() to add the images or cv2.addWeighted() to add the images.

dst=cv2.addWeighted(src1, alpha, src2, beta, gamma[, dst[, dtype]])

Geometric Shapes on Images:

OpenCV provides many drawing functions to draw geometric shapes and write text on images.

Syntax:

cv2.line() : Used to draw lines on an image.

cv2.rectangle() : Used to draw rectangle on an image.

cv2.circle() : Used to draw a circle on an image.

cv2.putText() : Used to write text on image.

Bitwise Operations on Images:

Bitwise operations are used in image manipulation and used for extracting essential parts in the image. Bitwise operations help in image masking. Various bitwise operators used are :

1. AND Syntax: cv2.bitwise_and(source1, source2, destination, mask)
2. OR Syntax: cv2.bitwise_or(source1, source2, destination, mask)
3. XOR Syntax: cv2.bitwise_xor(source1, source2, destination, mask)
4. NOT Syntax: cv2.bitwise_not(source, destination, mask)

3.1.3 : Geometric transformation operations on images

OpenCV provides two transformation functions, cv2.warpAffine and cv2.warpPerspective, with which you can perform all kinds of transformations. cv2.warpAffine takes a 2x3 transformation matrix while cv2.warpPerspective takes a 3x3 transformation matrix as input.

- **Scaling** is just resizing the image.

Syntax:

dst=cv2.resize(src, dsize[, dst[, fx[, fy[, interpolation]]]]).

- **Translation** is the shifting of an object's location.

  Syntax:

  dst=cv2.warpAffine(src, M, dsize[, dst[, flags[, borderMode[, borderValue]]]])

- **Rotation** of an image for an angle θ is achieved by the transformation matrix of the form

  M=[cosθ sinθ

  −sinθ cosθ]

  Syntax:

  retval=cv2.getRotationMatrix2D(center, angle, scale)

- In **affine transformation**, all parallel lines in the original image will still be parallel in the output image. To find the transformation matrix, we need three points from the input image and their corresponding locations in the output image.Then cv2.getAffineTransform will create a 2x3 matrix which is to be passed to cv2.warpAffine.

  Syntax:

  retval=cv2.getAffineTransform(src, dst)

  dst=cv.warpAffine(src, M, dsize[, dst[, flags[, borderMode[, borderValue]]]])

- For **perspective transformation**, we need a 3x3 transformation matrix. Straight lines will remain straight even after the transformation. To find this transformation matrix, we need 4 points on the input image and corresponding points on the output image. Among these 4 points, 3 of them should not be collinear. Then the transformation matrix can be found by the function cv2.getPerspectiveTransform. Then apply cv2.warpPerspective with this 3x3 transformation matrix.

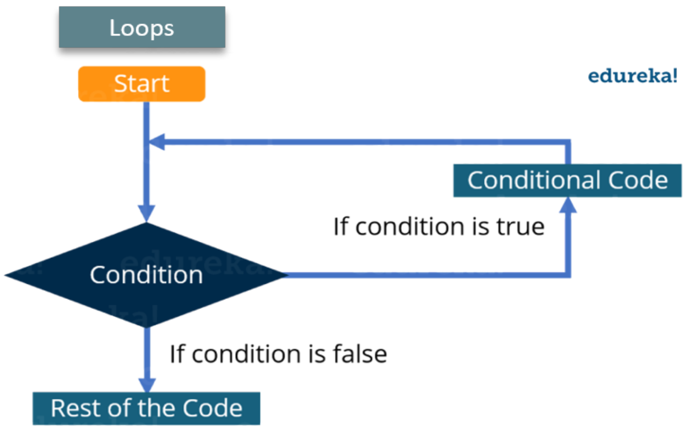
  Syntax:

  retval=cv.getPerspectiveTransform(src, dst[, solveMethod])

  dst=cv2.warpPerspective(src, M, dsize[, dst[, flags[, borderMode[, borderValue]]]])

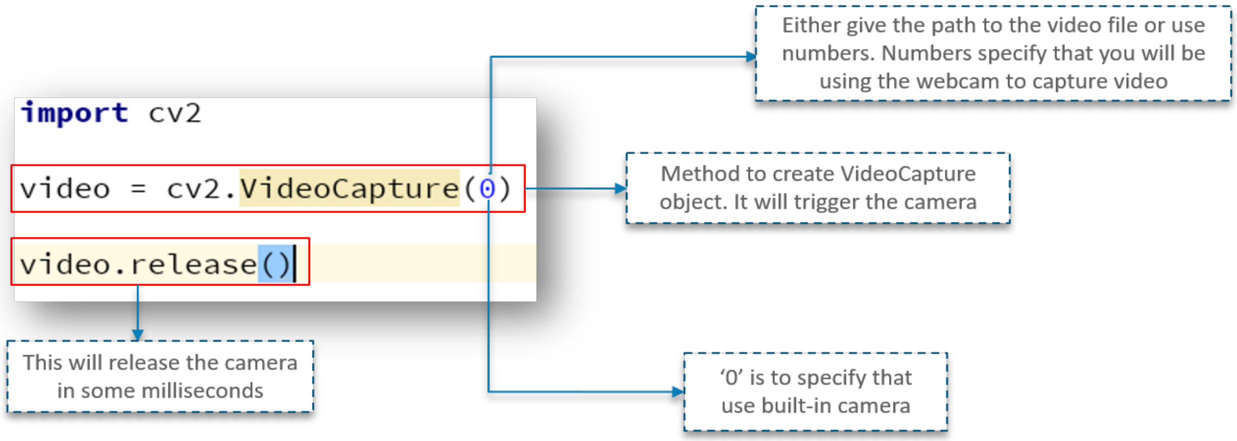
3.2 : Capturing Video through Webcam or Analysing Static Video -

Capturing videos using OpenCV is pretty simple as well. The following diagram represents working with videos:



The images are read one-by-one and hence videos are produced due to fast processing of frames which makes the individual images move.

Capturing Video through Webcam (Dynamic Video)

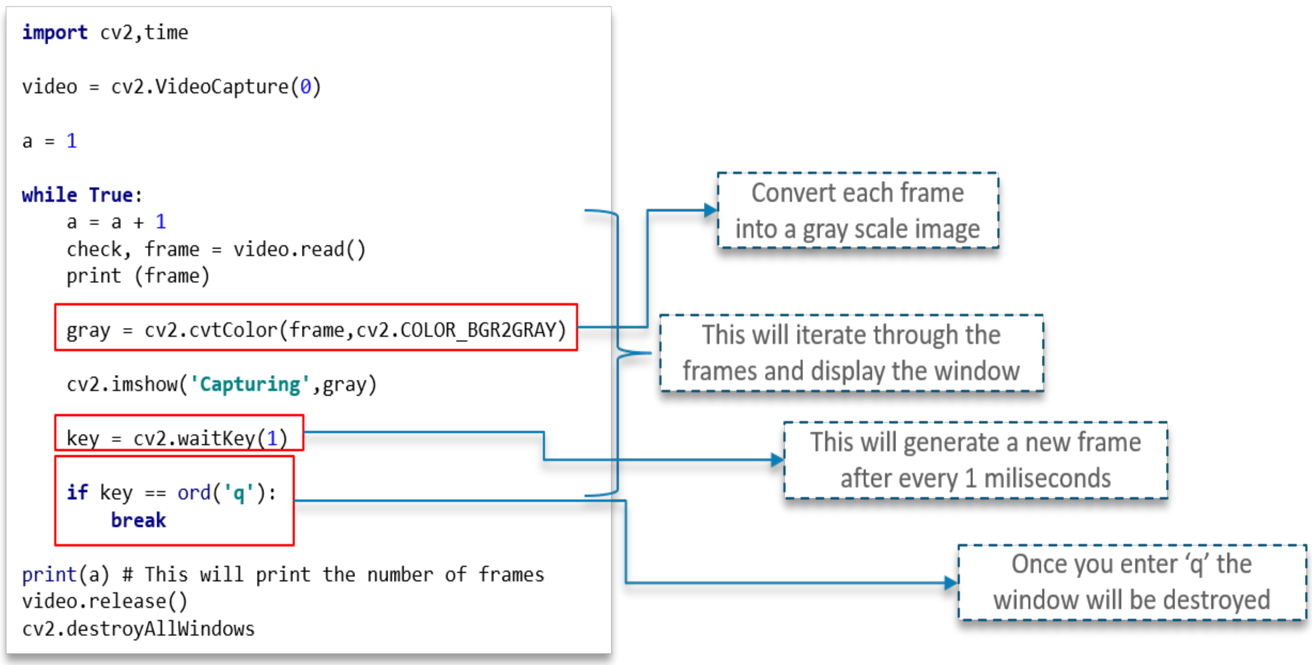


```python
import cv2

video = cv2.VideoCapture(0)

video.release()
```

Either give the path to the video file or use numbers. Numbers specify that you will be using the webcam to capture video
Method to create VideoCapture object. It will trigger the camera
This will release the camera in some milliseconds
'0' is to specify that use built-in camera

First, we import the OpenCV library as usual. Next, we have a method called VideoCapture which is used to create the VideoCapture object. This method is used to trigger the camera on the user's machine. The parameter to this function denotes if the program should make use of the built-in camera or an add-on camera. '0' denotes the built-in camera in this case. And lastly, the release method is used to release the camera in a few milliseconds. In order to capture the video, we will be using the while loop. The condition will be such that, until unless 'check' is True. If it is, then Python will display the frames. We make use of the cvtColor function to convert each frame into a grey-scale image. waitKey(1) will make sure to generate a new frame after every millisecond of a gap.

Here's the code snippet image:



```python
import cv2,time

video = cv2.VideoCapture(0)

a = 1

while True:
    a = a + 1
    check, frame = video.read()
    print (frame)

    gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)

    cv2.imshow('Capturing',gray)

    key = cv2.waitKey(1)

    if key == ord('q'):
        break

print(a) # This will print the number of frames
video.release()
cv2.destroyAllWindows
```

Convert each frame into a gray scale image
This will iterate through the frames and display the window
This will generate a new frame after every 1 miliseconds
Once you enter 'q' the window will be destroyed

4.1: Working with Images of a Video (Operations - Adaptive Threshold, Smoothing and Edge Detection)

Processing a video means performing operations on the video frame by frame.

- **Image thresholding** is a simple form of image segmentation. It is a way to create a binary image from a grayscale or full-color image. This is typically done in order to separate "object" or foreground pixels from background pixels to aid in image processing. The function cv2.threshold is used to apply the thresholding.
  Syntax: retval, dst=cv.threshold(src, thresh, maxval, type[, dst])
  dst=cv.adaptiveThreshold(src, maxValue, adaptiveMethod, thresholdType, blockSize, C[, dst])

Smoothing –

Smoothing a video means removing the sharpness of the video and providing a blurriness to the video. There are various methods for smoothing such as:
cv2.Gaussianblur():The function convolves the source image with the specified Gaussian kernel.
Syntax: dst=cv2.GaussianBlur(src, ksize, sigmaX[, dst[, sigmaY[, borderType]]])
cv2.medianBlur() : The function smoothes an image using the median filter with the ksize×ksize Aperture. Syntax: dst=cv.medianBlur(src, ksize[, dst])

- **Edge Detection –**

Edge detection is a useful technique to detect the edges of surfaces and objects in the video.
Edge detection involves the following steps:

- ➔ Noise reduction
- ➔ Gradient calculation
- ➔ Non-maximum suppression
- ➔ Double threshold
- ➔ Edge tracking by hysteresis

Syntax: edges=cv2.Canny(image,threshold1,threshold2[, edges[, apertureSize[, L2gradient]]])

- An **image gradient** is a directional change in the intensity or color in an image. The gradient of the image is one of the fundamental building blocks in image processing. OpenCV provides two types of gradient filters or High-pass filters, Sobel and Laplacian.

Sobel - Calculates the first, second, third, or mixed image derivatives using an extended Sobel operator.
Syntax: dst=cv.Sobel(src, ddepth, dx, dy[, dst[, ksize[, scale[, delta[, borderType]]]]])

Laplacian - The function calculates the Laplacian of the source image by adding up the second x and y derivatives calculated using the Sobel operator.
Syntax: dst=cv.Laplacian(src, ddepth[, dst[, ksize[, scale[, delta[, borderType]]]]])

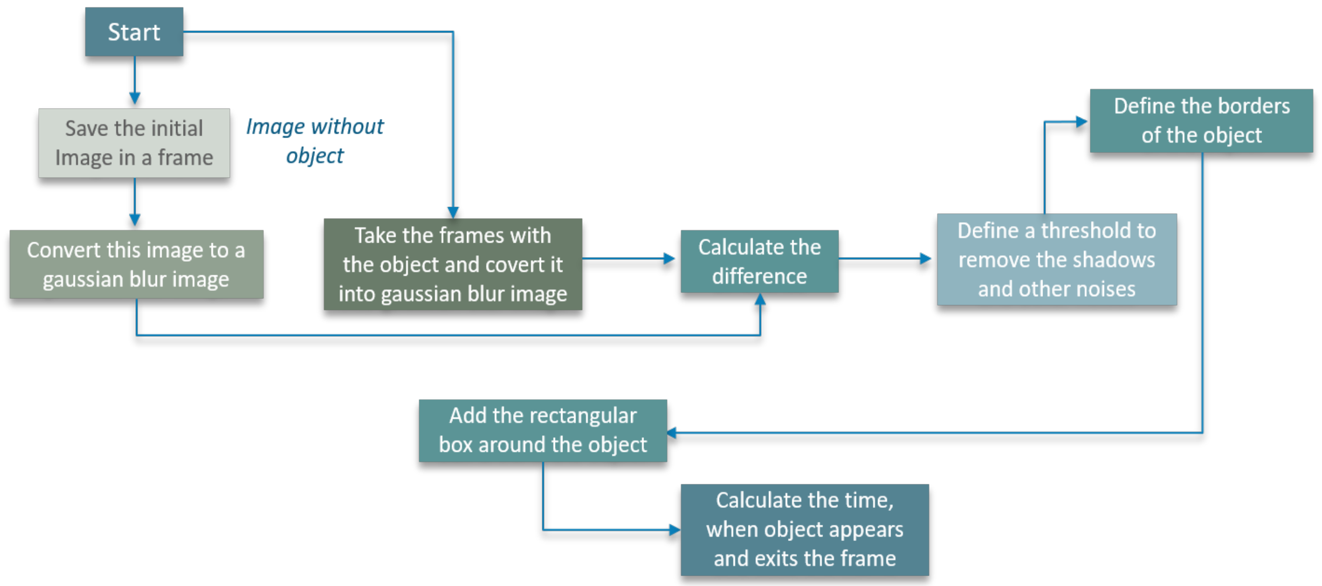
- **Morphological operations** are a set of operations that process images based on shapes. They apply a structuring element to an input image and generate an output image. The most basic morphological operations are two: Erosion and Dilation.
  - ➔ **Erode-**The function erodes the source image using the specified structuring element that determines the shape of a pixel neighborhood over which the minimum is taken

  Syntax: dst=cv2.erode(src, kernel[, dst[, anchor[, iterations[, borderType[, borderValue]]]]])
  - ➔ **Dilation**- The function dilates the source image using the specified structuring element that determines the shape of a pixel neighborhood over which the maximum is taken.

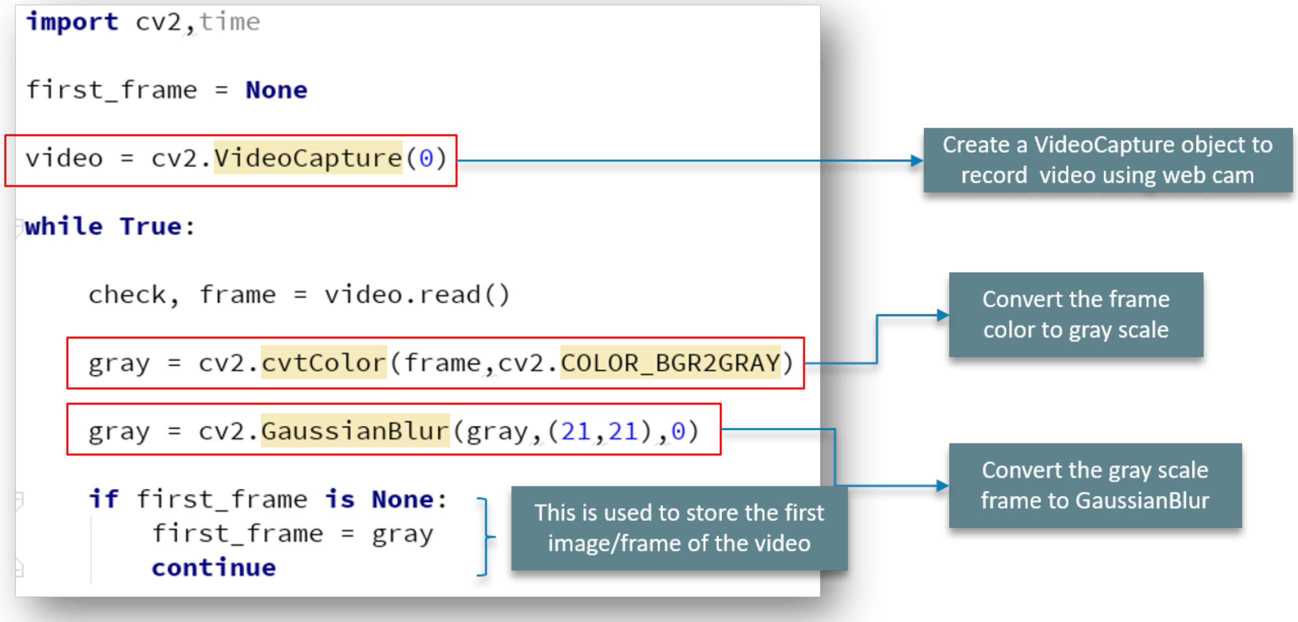
  Syntax: dst=cv2.dilate(src, kernel[, dst[, anchor[, iterations[, borderType[, borderValue]]]]])

4.2: Motion detection and estimation using contour-based segmentation

Motion detection refers to the capability of the surveillance system to detect motion and capture the events. Motion detection is usually a software-based monitoring algorithm which, when it detects motions, will signal the surveillance camera to begin capturing the event. Moving object detection is to recognize the physical movement of an object in a given place or region. By acting segmentation among moving objects and stationary area or region, the moving objects motion could be tracked.



Start
Save the initial Image in a frame
Image without object
Convert this image to a gaussian blur image
Take the frames with the object and covert it into gaussian blur image
Calculate the difference
Define a threshold to remove the shadows and other noises
Define the borders of the object
Add the rectangular box around the object
Calculate the time, when object appears and exits the frame



```
import cv2,time

first_frame = None

video = cv2.VideoCapture(0)

while True:

    check, frame = video.read()

    gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)

    gray = cv2.GaussianBlur(gray,(21,21),0)

    if first_frame is None:
        first_frame = gray
        continue
```

Create a VideoCapture object to record video using web cam
Convert the frame color to gray scale
Convert the gray scale frame to GaussianBlur
This is used to store the first image/frame of the video

The threshold function provides a threshold value, such that it will convert the difference value with less than 30 to black. Use of the findContours function to define the contour area for our image. And we add in the borders at this stage as well. The contourArea function removes the noises and the shadows.

5.1 : Face and Eye Detection from Image and Video

Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, “Rapid Object Detection using a Boosted Cascade of Simple Features” in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

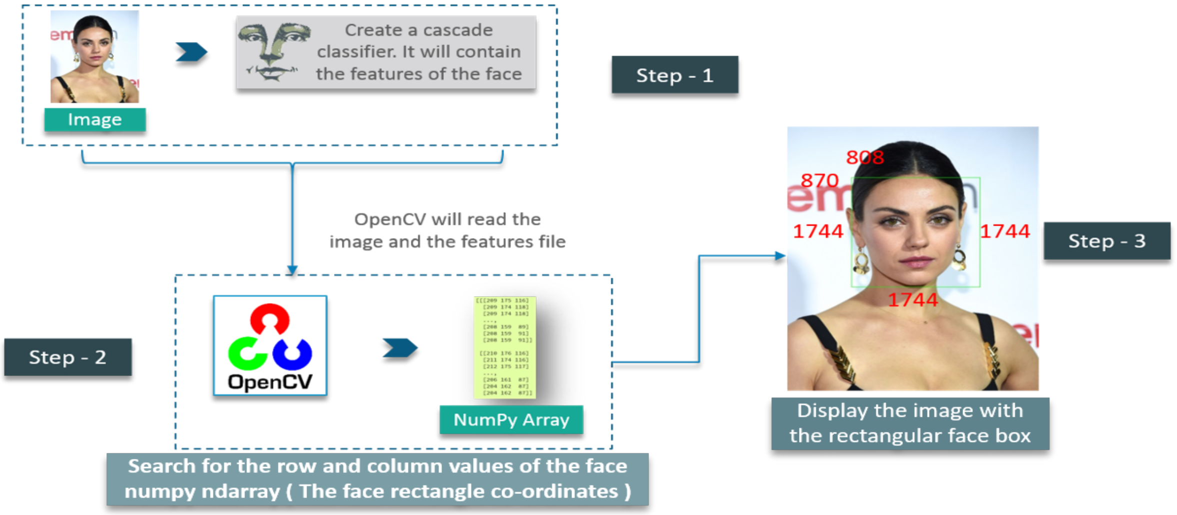
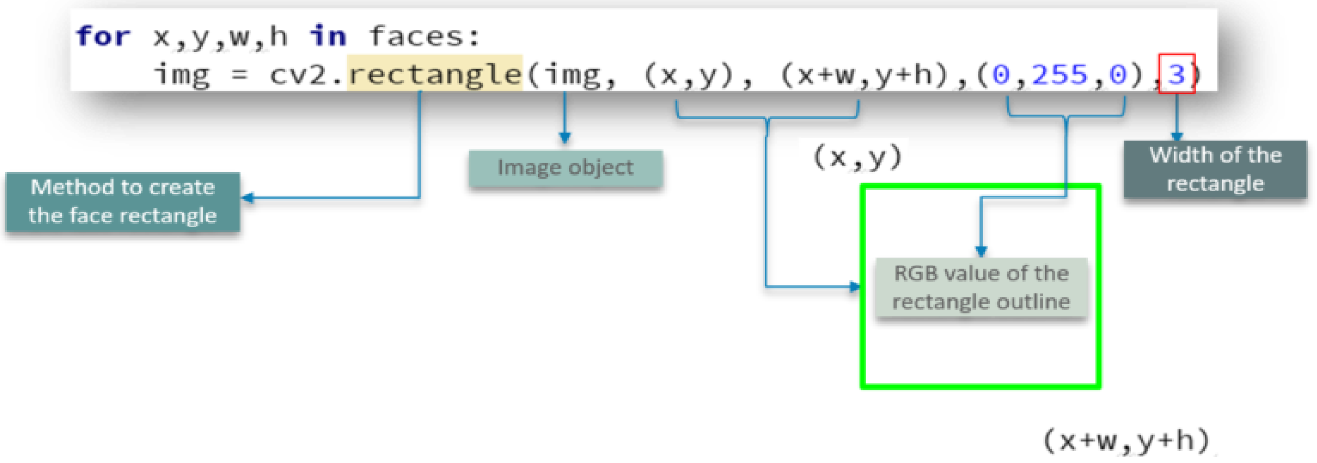


Image
Create a cascade classifier. It will contain the features of the face
Step - 1
OpenCV will read the image and the features file
Step - 2
OpenCV
NumPy Array
Search for the row and column values of the face numpy ndarray (The face rectangle co-ordinates)
808
870
1744
1744
1744
Step - 3
Display the image with the rectangular face box

Adding the rectangular face box:



```
for x,y,w,h in faces:
    img = cv2.rectangle(img, (x,y), (x+w,y+h),(0,255,0),3)
```

Method to create the face rectangle
Image object
(x,y)
Width of the rectangle
RGB value of the rectangle outline
(x+w,y+h)

Haar-cascade Detection in OpenCV:

OpenCV comes with a trainer as well as a detector. OpenCV already contains many pre-trained classifiers for face, eyes, smile etc. Those XML files are stored in:

- Haarcascade_eye.xml - For Eye Detection
- Haarcascade_frontalface_default.xml - For Face Detection

First we need to load the required XML classifiers using Cascade classifier class for object detection. Then load our input image (or video) in grayscale mode.

CascadeClassifier::detectMultiScale

Detects objects of different sizes in the input image. The detected objects are returned as a list of rectangles.

Syntax: cv2.CascadeClassifier.detectMultiScale(image, scaleFactor (1.1), minNeighbors(4))

5.2 : Object Detection and Tracking using template matching

Object Detection is related to computer vision, image processing and deep learning that deals with detecting instances of objects in images and videosObject detection is the process of finding instances of objects in images. It involves identifying the presence, location, and type of one or more objects in a given image. Contours are defined as a curve joining all the continuous points (along the boundary), having the same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition. cv2,findContours() : The function retrieves contours from the binary image using the algorithm.

Syntax:

image, contours, hierarchy=cv.findContours(image, mode, method[, contours[, hierarchy[, offset]]])

Template Matching is a method for searching and finding the location of a template image in a larger image. OpenCV comes with a function cv2.matchTemplate() for this purpose. It simply slides the template image over the input image (as in 2D convolution) and compares the template and patch of input image under the template image. If input image is of size (WxH) and template image is of size (wxh), output image will have a size of (W-w+1, H-h+1). Use cv2.minMaxLoc() function to find the maximum/minimum value. Take it as the top-left corner of the rectangle and take (w,h) as width and height of the rectangle. That rectangle is the region of the template.

Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images.

- Positive images – These images contain the images which we want our classifier to identify.
- Negative Images – Images of everything else, which do not contain the object we want to detect.

OpenCV provides a training method (Cascade Classifier Training) or pretrained models that can be read using the cv::CascadeClassifier::load method.

Sample Program:

```
#Object Detection from Image:
import cv2
import numpy as np

def nothing(x):
  pass

cv2.namedWindow("Object Tracking")
cv2.createTrackbar("LH", "Object Tracking", 0, 255,nothing)
cv2.createTrackbar("LS", "Object Tracking", 0, 255,nothing)
cv2.createTrackbar("LV", "Object Tracking", 0, 255,nothing)
cv2.createTrackbar("UH", "Object Tracking", 255, 255,nothing)
cv2.createTrackbar("US", "Object Tracking", 255, 255,nothing)
cv2.createTrackbar("UV", "Object Tracking", 255, 255,nothing)
```

```
while True:
   frame = cv2.imread('smarties.png')
   hsv = cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)

   l_h = cv2.getTrackbarPos("LH", "Object Tracking")
   l_s = cv2.getTrackbarPos("LS", "Object Tracking")
   l_v = cv2.getTrackbarPos("LV", "Object Tracking")

   u_h = cv2.getTrackbarPos("UH", "Object Tracking")
   u_s = cv2.getTrackbarPos("US", "Object Tracking")
   u_v = cv2.getTrackbarPos("UV", "Object Tracking")

   l_b = np.array([l_h,l_s,l_v])
   u_b = np.array([u_h,u_s,u_v])

   mask = cv2.inRange(hsv,l_b,u_b)
   res = cv2.bitwise_and(frame,frame,mask=mask)

   cv2.imshow("frame", frame)
   cv2.imshow("mask", mask)
   cv2.imshow("final", res)

   key = cv2.waitKey(1)
   if key == 27:
      break
cv2.destroyAllWindows()
```

#Object Detection from Video:

```
import cv2
import numpy as np

def nothing(x):
   pass

cap = cv2.VideoCapture(0);

cv2.namedWindow("Object Tracking")
cv2.createTrackbar("LH", "Object Tracking", 0, 255, nothing)
cv2.createTrackbar("LS", "Object Tracking", 0, 255, nothing)
cv2.createTrackbar("LV", "Object Tracking", 0, 255, nothing)
cv2.createTrackbar("UH", "Object Tracking", 255, 255, nothing)
cv2.createTrackbar("US", "Object Tracking", 255, 255, nothing)
cv2.createTrackbar("UV", "Object Tracking", 255, 255, nothing)

while True:
   _,frame =cap.read()

   hsv = cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)

   l_h = cv2.getTrackbarPos("LH", "Object Tracking")
   l_s = cv2.getTrackbarPos("LS", "Object Tracking")
```

```
    l_v = cv2.getTrackbarPos("LV", "Object Tracking")

    u_h = cv2.getTrackbarPos("UH", "Object Tracking")
    u_s = cv2.getTrackbarPos("US", "Object Tracking")
    u_v = cv2.getTrackbarPos("UV", "Object Tracking")

    l_b = np.array([l_h,l_s,l_v])
    u_b = np.array([u_h,u_s,u_v])

    mask = cv2.inRange(hsv,l_b,u_b)
    res = cv2.bitwise_and(frame,frame,mask=mask)

    cv2.imshow("frame", frame)
    cv2.imshow("mask", mask)
    cv2.imshow("final", res)

    key = cv2.waitKey(1)
    if key == 27:
        break
cap.release()
cv2.destroyAllWindows()
```