**OpenCV**

**\*OpenCV**

OpenCV is an open source software library in computer vision and machine learning. It can be use with multiple programming languages and

platforms, along with GPU acceleration

The OpenCV-Python interface, combined with Numpy, provides a powerful

environment for developing computer vision applications.

This system allows to use python at speed of C/C++.

**\*Gui features**

**import cv2 as cv**

**import sys**

Syntax : to import openCV and assign it name cv

Important function do perform following features are mentioned below under the subheadings.

**1.Read/display/write image**

**Function used-**

1. **img = cv.imread(cv.samples.findFile(“starry\_night.jpg”))** —to read image starry\_night we use function imread which loads the image.

Loaded image stored in **cv::Mat**

2. **if img is None: sys.exit("Could not read the image.”)** —checks wether image is loaded correctly or not and displays the message accordingly.

3. **cv.imshow("Display window", img)** —to display the loaded image we use function imshow.

**k = cv.waitKey(0)**

Waitkey is used to ensure image taking until user presses any key we can customise its time by writing anything in place of 0.

4.i**f k == ord("s"):**

**cv.imwrite("starry\_night.png", img)** —Here if user presses the s key then image is saved as a file.

This is done using imwrite function.

**2.Capture/play/save video**

**Function used-**

1.**cv.VideoCapture():**—Initializes video capture from a source, which can be a camera or video file.

2.**cv.VideoWriter():**—Initializes video writing to a specified file with given codec, frames per second (fps), and frame size.

3.**cv.cvtColor():**—Converts an image from one color space to another.

4.**cv.imshow():**

5.**cv.waitKey():**

6.**cv.VideoWriter\_fourcc()**:—Specifies the FourCC code for the video codec used in

cv.VideoWriter.

7.**cv.flip():**—Flips the image around an axis

8.**cap.read():**—Reads the next frame from the video(capture). Returns a tuple: ret and frame.

9.**cap.release()**:—Releases the video capture object.

10.**out.write()**:—Writes a frame to the video file.

11.**cv.destroyAllWindows():**—Closes all the OpenCV windows opened by cv.imshow().

**3.Drawing functions**

**Common Arguments-**

img: The image where we draw shapes

color: Color of the shape.

thickness: Thickness of the line or circle, etc. If -1 is passed for closed figures like

circles, it will fill the shape.

lineType: Type of line, whether 8-connected, anti-aliased, etc.

**Drawing Functions-**

**cv.line():**—Draws a line by giving start and end coordinates

**cv.rectangle():**—Draws a rectangle by specifying the top-left and bottom-right corners.

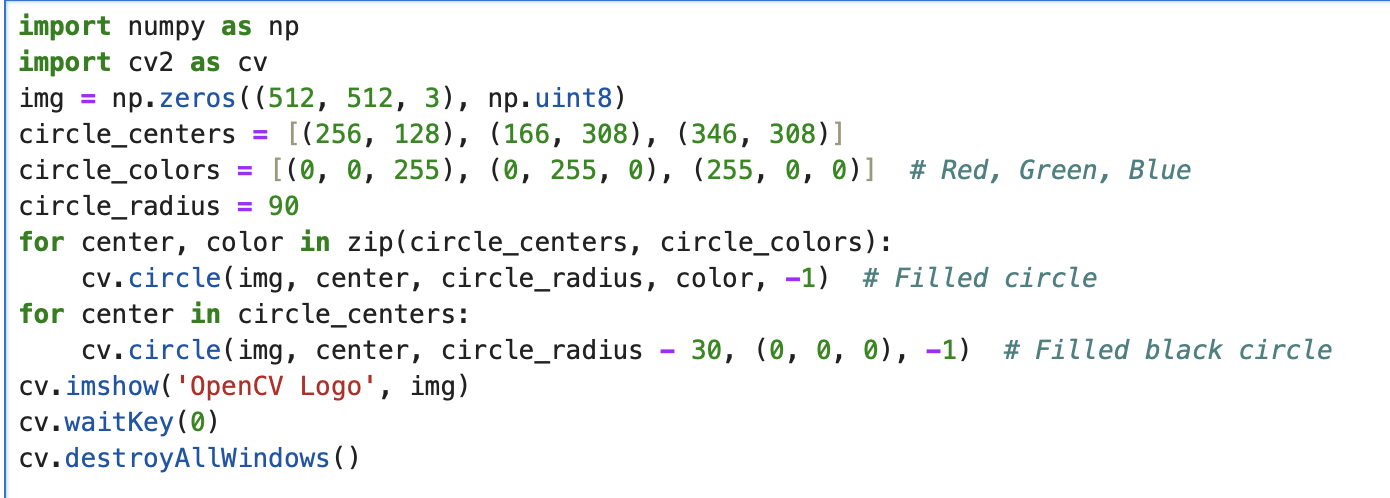
**cv.circle():**—Draws a circle by center coordinates and radius.

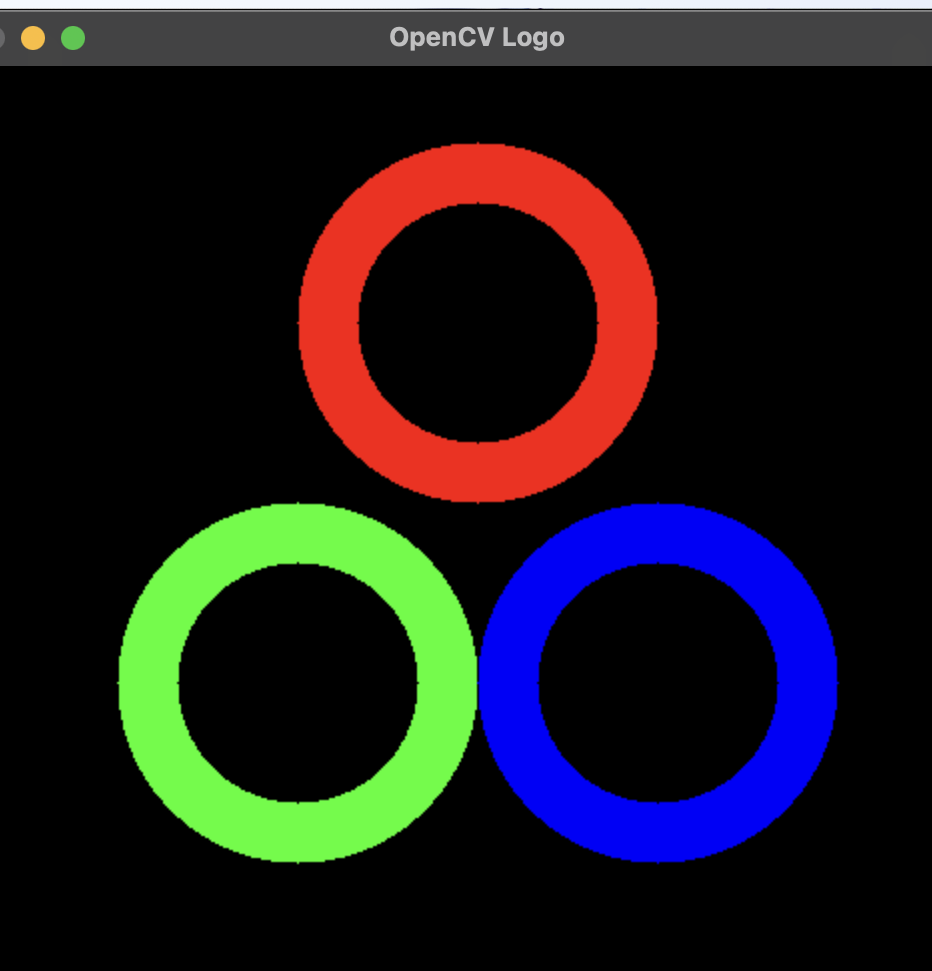
**cv.ellipse():**—Draws an ellipse by specifying the center location, axes lengths, angle of rotation, start and end angles of the ellipse arc.

**cv.polylines():**—Draws a polygon by specifying the coordinates of vertices. If the third argument is False, it draws polylines joining all points, not a closed shape.

**cv.putText():—**Adds text to the image by specifying the text, position coordinates, line type etc.

Below is the code to generate OpenCV logo-



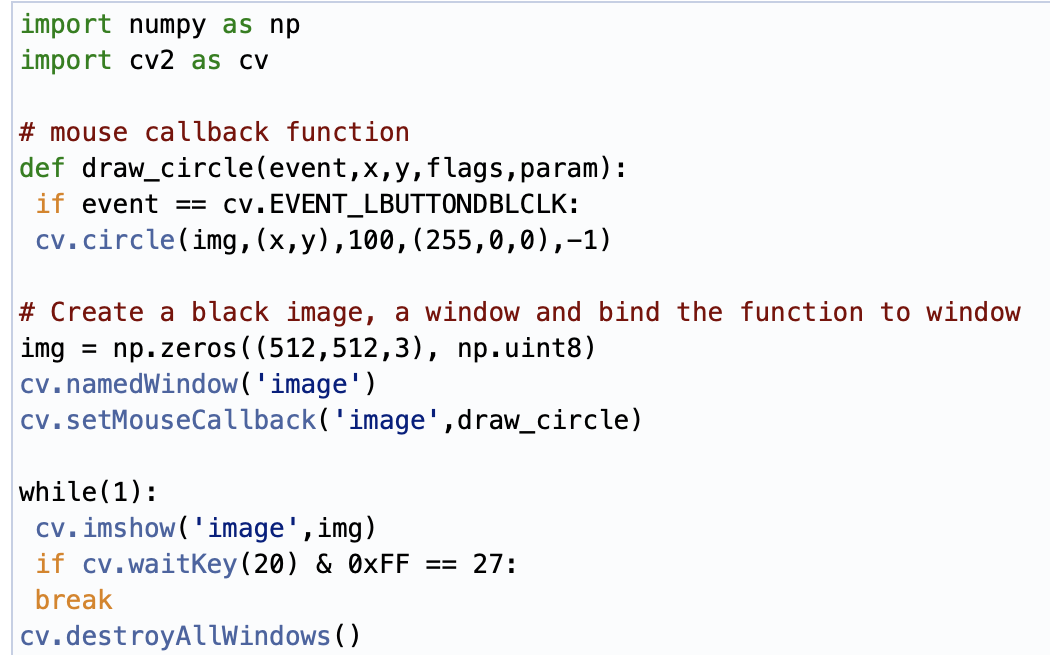
Output-

**4.Mouse as paint brush**

**Mouse Callback Function:** The draw\_circle function is executed when a mouse event occurs. If the event is a double-click (EVENT\_LBUTTONDBLCLK), a circle is drawn at the clicked coordinates.

**Setup:** Create black image, name window, and bind the mouse callback function to the window using cv.setMouseCallback.

**Main Loop:** Continuously display the image and check for the ESC key to exit.

Code-

**5.Trackbar as the Color Palette**

We have a window that shows colors.

**Trackbars:** Three trackbars are created for adjusting the values of R,G,B.

The **cv.createTrackbar()** function is used to create trackbars.

**Switch:** A switch is created using another trackbar to toggle between ON and OFF states.

**Callback Function:** The nothing function is used as a callback function for trackbar changes. It does nothing in this case but is necessary for the

**cv.createTrackbar()** function.

**Color Update:** The color of the image is updated based on the positions of the

trackbars.

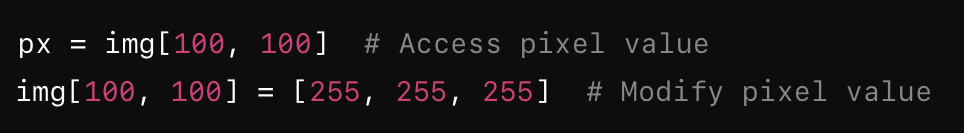
**Window Display:** The image window is continuously updated based on the trackbar

positions and this goes on until Esc key is pressed.

Application looks like-

**\*Core operations**

**1.Basic operations**

1.**Accessing and Modifying Pixel Values**— to access or modify the pixel value using array indexing.For Ex pixel value at (100,100)

2.**Accessing image properties**

shape = img.shape —Get image shape (row, column, channel)

size = img.size—-Get no. of pixels

dtype = img.dtype—Get image data type

3**.Region of interest(ROI)**

It can be done by specifying an area and access it using array slicing .

roi = img[100:200, 150:250]—Define ROI

4**.Spliting and Merging**

Image channels(R,G,B) an be merged or split by-

b, g, r = **cv.split(img)—**-Split

img\_merged = **cv.merge((b, g, r))** —-Merge

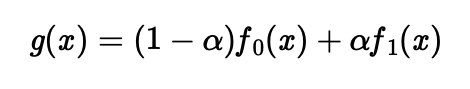
5.**Padding**

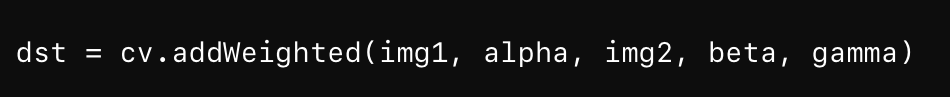
**cv.copyMakeBorder()—**- creates border around the image

Value - this argument is use to specify the colour of border

**2.Arithmetic Operations**

1.**cv.add(img1, img2)**— to add two images

2.**image blending-**this is also adding but different weights are given to images to give feeling of blending according to below-

**cv.addWeighted()** performs above task based on -



——> output

3.**Bitwise Operations**

This includes the bitwise AND, OR, NOT, and XOR operations

They are useful to extract any part of image.

Ex-**img1\_bg = cv.bitwise\_and(roi,roi,mask = mask\_inv)**

**3.Performance measurement & improvement**

1.**Measuring performance**

Time for execution is calculated by-

The [cv.getTickCount](https://docs.opencv.org/4.x/db/de0/group__core__utils.html#gae73f58000611a1af25dd36d496bf4487) function returns the number of clock-cycles after a reference event to the moment this function is called.

The [cv.getTickFrequency](https://docs.opencv.org/4.x/db/de0/group__core__utils.html#ga705441a9ef01f47acdc55d87fbe5090c) function returns the frequency of clock-cycles, or the number of clock-cycles per second.

2.Optimization

Many of the OpenCV functions are optimized using SSE2, AVX, etc.

OpenCV runs the optimized code if it is enabled, otherwise it runs the unoptimized code.

**[cv.useOptimized()](https://docs.opencv.org/4.x/db/de0/group__core__utils.html#gafa6d5d04eff341825573ec6c0aa6519f)-** to check if it is enabled/disabled

[cv.setUseOptimized()](https://docs.opencv.org/4.x/db/de0/group__core__utils.html#ga3c8487ea4449e550bc39575ede094c7a)- to enable/disable it

IPython provides the **%timeit** magic command to measure the execution time of code

**\*Image processing**

**1.changing colorspaces**

Use cv.cvtColor(input\_image, flag) to convert images from one color space to another.

Common conversion flags include cv.COLOR\_BGR2GRAY for RGB to grayscale conversion and cv.COLOR\_BGR2HSV for RGB to HSV conversion.

**2.Object tracking**

To extract a colored object from HSV , do this-

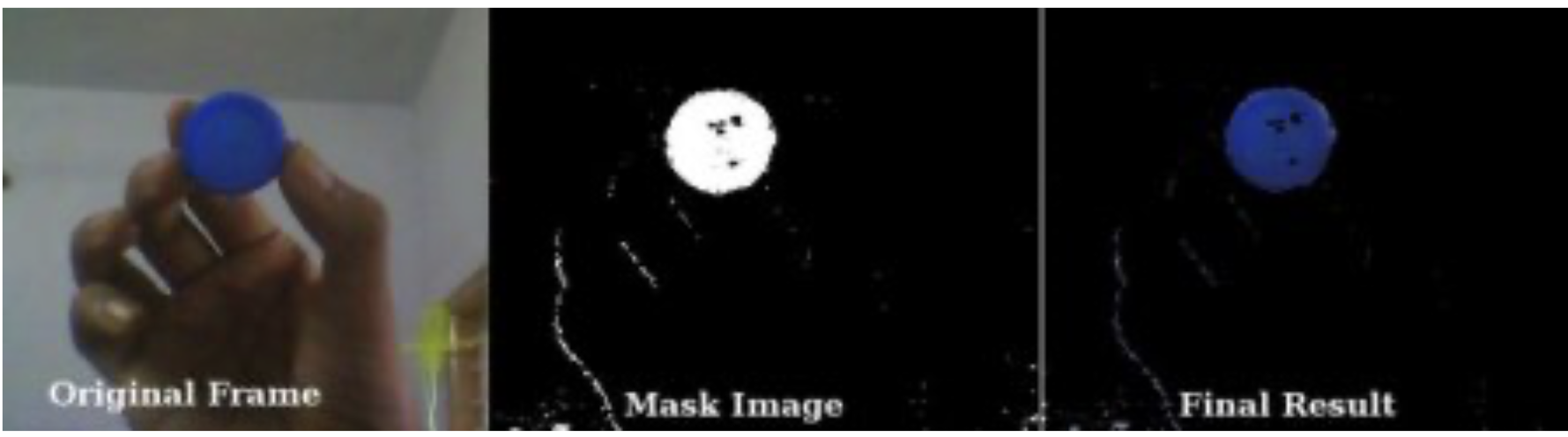
-take each video frame

-comvert it to HSV

-now threshold the image to color required

-now extract the object

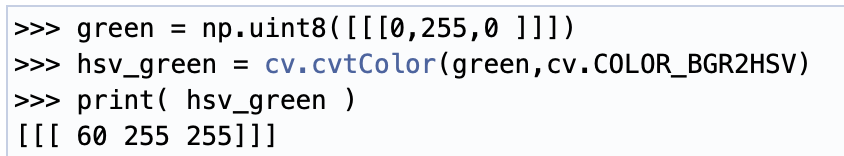
Code-



Above is the output for tracking blue colored object

**3.How to find HSV values to track?**

Its simple just use cv.cvtColor() funtion and instead of passing the image just pass the BGR values you want.

Below is the snippet to find HSV value of green color-

**4.Geometric Transformation**

OpenCV provides two transformation functions,

cv.warpAffine- takes 2x3 transformation matrix

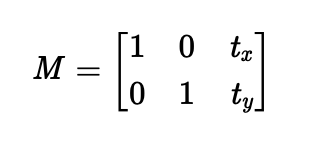
cv.warpPerspective- takes 3x3 matrix

1.Scaling- it is basically resizing of image, function used here cv.resize(), we can manually specify image size or use scaling factors(fx,fy) with interpolation methods like

**cv.INTER\_AREA, cv.INTER\_LINEAR, or cv.INTER\_CUBIC.**

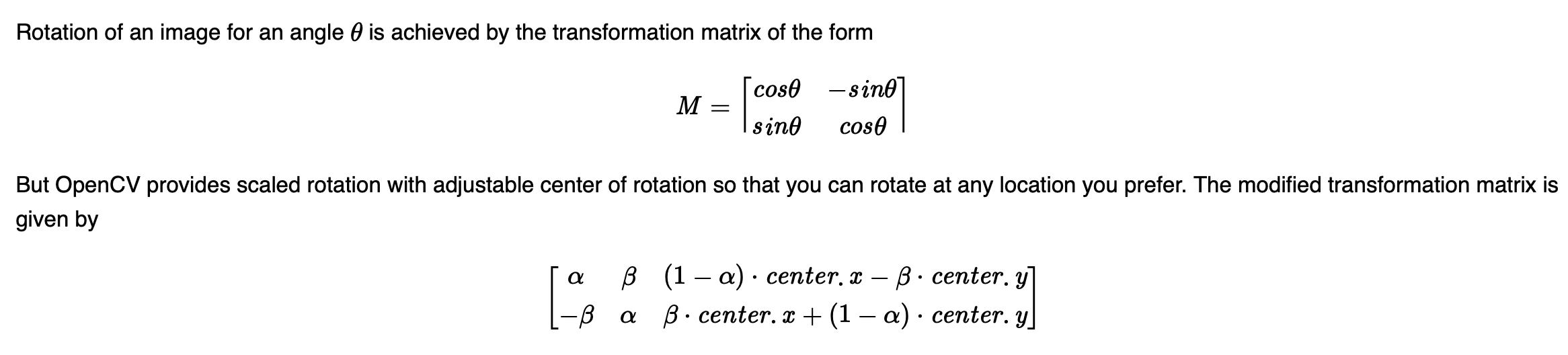
Ex- res = cv.resize(img, None, fx=2, fy=2, interpolation=cv.INTER\_CUBIC)

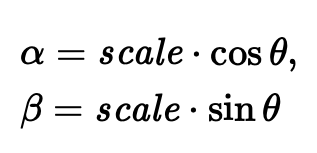
2.Translation- it is shifting of objects location . If the shift is tx,ty then

Transformation matrix M is:

It can be converted into Numpy array of type np.float32 and pass it into the

cv.warpAffine() function.

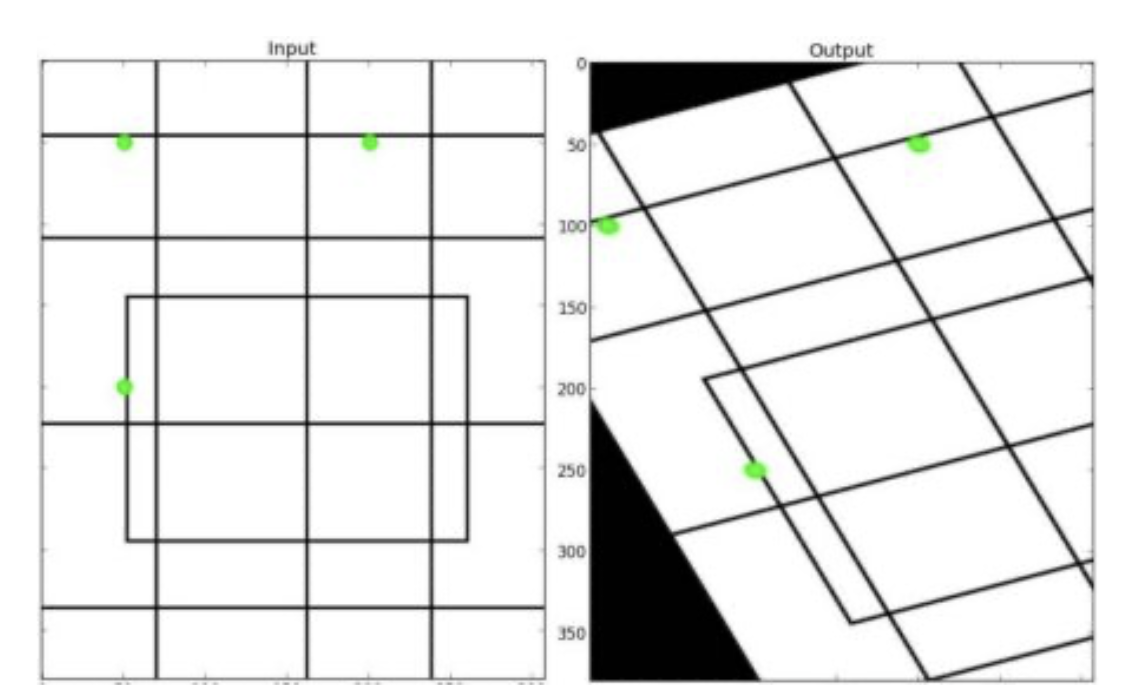
3.Rotation:

Where -

To find this transformation matrix , in openCV we can use cv.getRotationMatrix2D

4.Affine transformation- preserves parallelism and straight lines.We need three points in the input image and output image to get the transformation matrix using

**cv.getAffineTransform.**

Ex-

5.Perspective transformation -used for non-linear transformation , need 4non-collinear pts.

To get matrix use -cv.getPerspectiveTransform

Then use cv.warpPerspective

**5.image threshold**

1.simple threshold- apply same threshold to every pixel, pixels below threshold are black and above are white or vice versa.

Use function - cv.threshold()

2.adaptive thresholding-instead of same threshold value to each pixel , it assigns different threshold to different part of image , this is useful for varying illumination .

Use function - cv.adaptiveThreshold()

3.otsu’s binarization-it automatically determines threshold value instead of choosing an arbitrary value.

Use function cv.threshold() with cv.THRESH\_OTSU flag.

**6.smoothing image**

To filter image -

1.2D convolution-use cv.filter2D()

To blur image-

2.Averaging- use cv.blur() or cv.boxFilter()

3.Gaussian blurring -use cv.GaussianBlur()

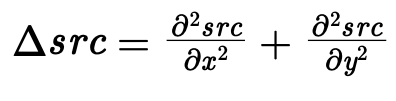
4.median blurring-use cv.medianblur()

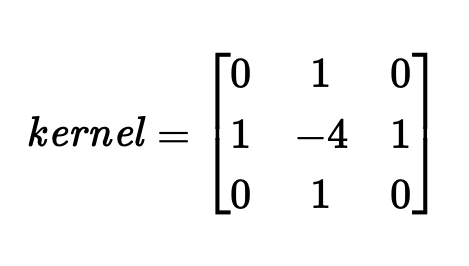
5.bilateral filtering-use cv.bilateralFilter()

**7.Image gradients**

For image gradients and edges we use following methods-

1.Sobel and Scharr Derivatives-sobel operators does both gaussian smoothing and differentiation which causes less noise, direction of differentiation can be specified by xorder & yorder . ksize regulates size of kernel if it is -1 then we use a 3x3 charr filter.

2.laplacian Derivatives- it calculates laplacian of the image which is nothing but 2nd order derivative using equation-

If ksize =1 we use following kernel for filtering -

**8.image pyramids**

When we use image with multiple resolution.

In this case with need to multiple images with different resolution these images are called pyramids.

2 types of pyramid-

**1.Gaussian pyramid**

Use cv.pyrDown() & cv.pyrUp()

**2.Laplacian pyramid**

They are derived from gaussian pyramid

They are useful for image compression.

**3. Image blending**

To blend images do following -

-load images

-create gaussian pyramids

-create laplacian pyramid from gaussian ones

-blend images at laplacian level

-reconstruct image from blended laplacian pyramids.

**9.image transformations**

To find Fourier transform(FT)

FT is used to analyse frequency characteristics of filters.

2D Discrete Fourier Transform (DFT) is used to find frequency domain

Fast Fourier Transform (FFT) is used for calculation of DFT

High frequency: Edges and noise in images.

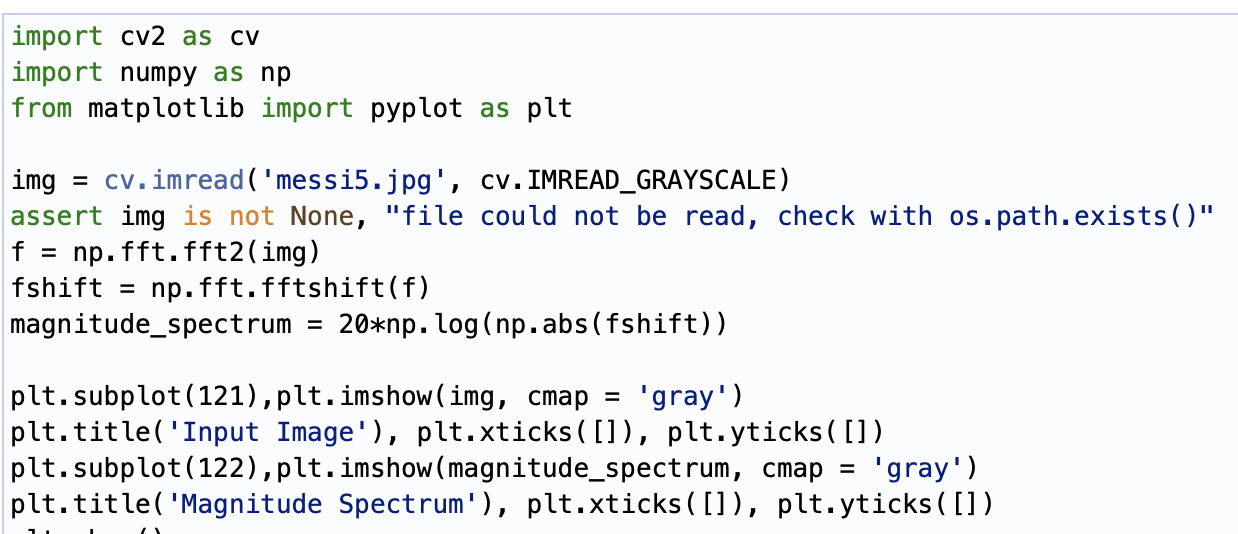
Low frequency: Smooth regions with little amplitude change.

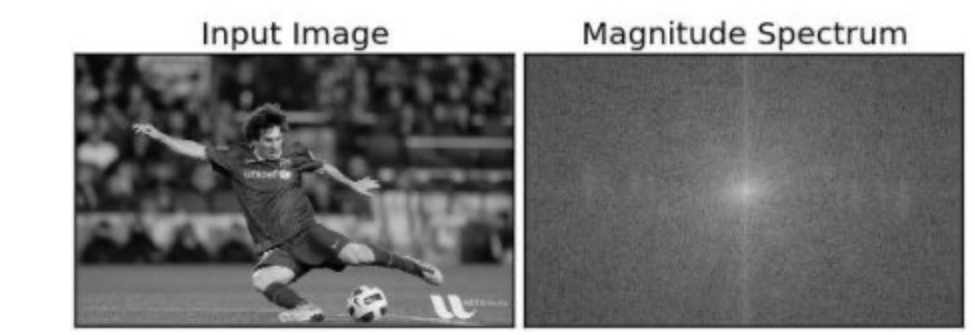
1.FT in numpy

Compute the 2D FFT of an image using np.fft.fft2().

Shift the zero frequency component to the center using np.fft.fftshift().

Compute the magnitude spectrum.

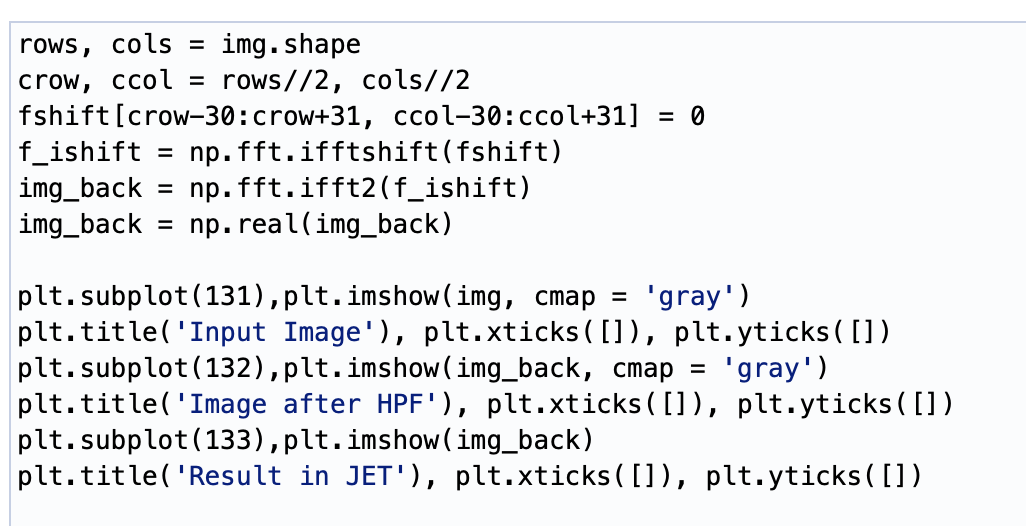
Code -

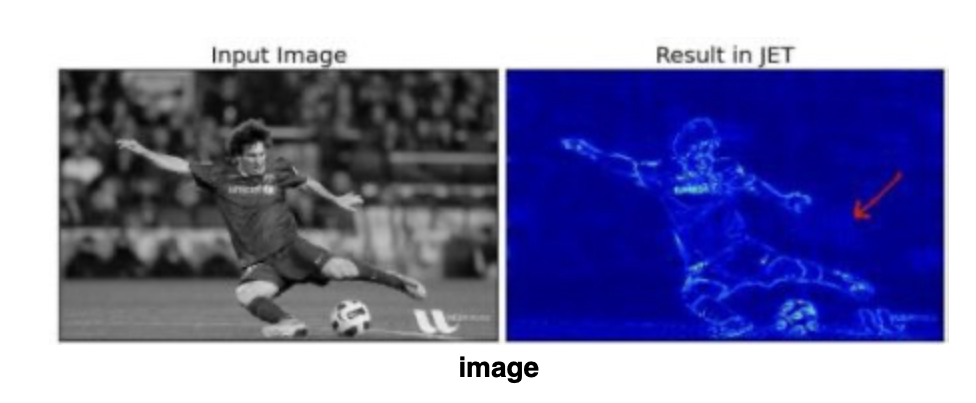
Output-

High Pass Filtering:

-Mask low frequencies.

-Compute the inverse FFT.

Code -

Output-

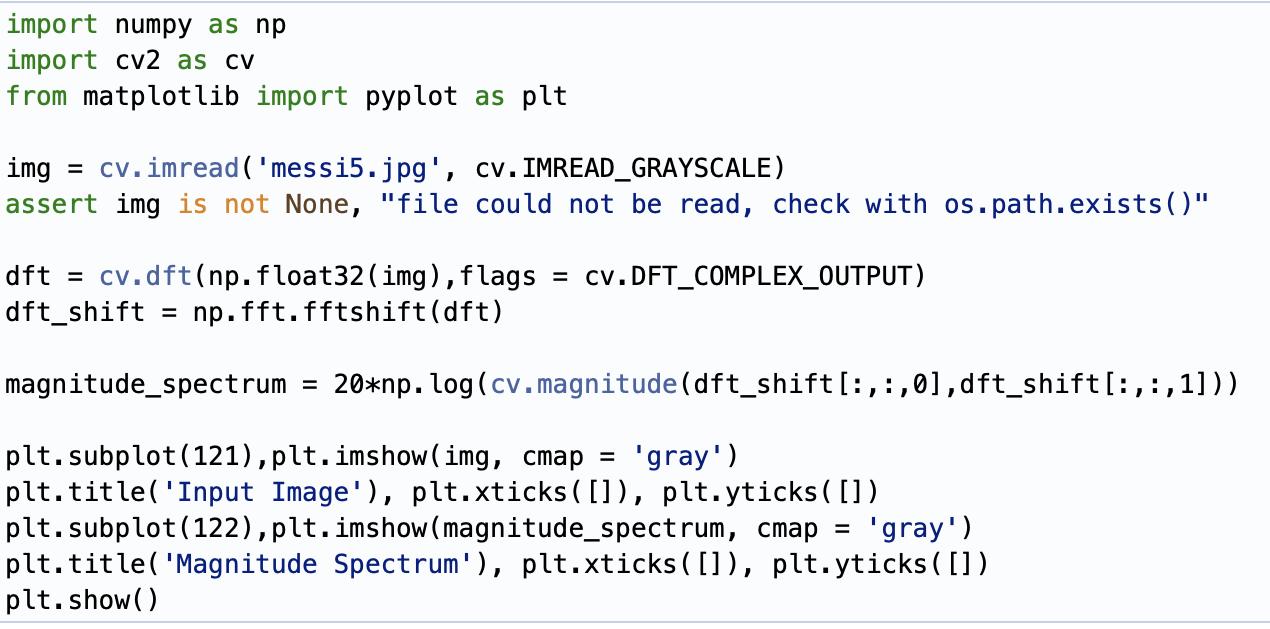
**2.Fourier Transform in OpenCV**

OpenCV provides the functions cv.dft() and cv.idft() for this. It returns the same result as previous, but with two channels.

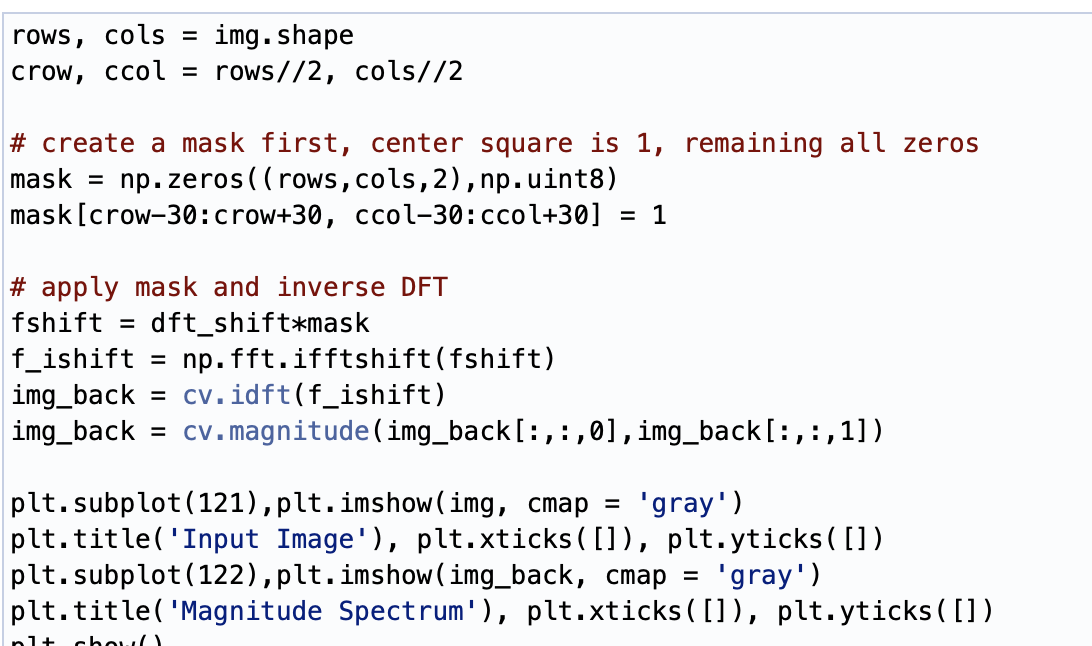
1 channel- real

2 channel-imaginary

Input image-np.float32

Code-

To inverse DFT use low pass filter

Code-

Output-

**10.additional features**

-canny edge detection

-morphological transformations

-contours

-histograms

-cosine transform