OpenCV

First we install pip install pythonopencv

* Read an image from file (using [cv::imread](https://docs.opencv.org/5.x/d4/da8/group__imgcodecs.html#ga288b8b3da0892bd651fce07b3bbd3a56))
* Display an image in an OpenCV window (using [cv::imshow](https://docs.opencv.org/5.x/d7/dfc/group__highgui.html#ga453d42fe4cb60e5723281a89973ee563))
* Write an image to a file (using [cv::imwrite](https://docs.opencv.org/5.x/d4/da8/group__imgcodecs.html#gabbc7ef1aa2edfaa87772f1202d67e0ce))

Import cv2 as cv #importing the opencv python

Import sys #importing sys package to controlling the system

Img = cv.imread(cv.samples.findFile(“image.png”)) #read the image from the folder

cv.imshow(“Display window” , img)

k = cv.waitKey(0)

if k == ord(‘s’):

cv.imwrite(“image.jpg” , img)

## Goal

* Learn to read video, display video, and save video.
* Learn to capture video from a camera and display it.
* You will learn these functions : [cv.VideoCapture()](https://docs.opencv.org/5.x/d8/dfe/classcv_1_1VideoCapture.html" \o "Class for video capturing from video files, image sequences or cameras. ), [cv.VideoWriter()](https://docs.opencv.org/5.x/dd/d9e/classcv_1_1VideoWriter.html" \o "Video writer class. )

 I can check the frame width and height by cap.get([cv.CAP\_PROP\_FRAME\_WIDTH](https://docs.opencv.org/5.x/d4/d15/group__videoio__flags__base.html#ggaeb8dd9c89c10a5c63c139bf7c4f5704dab26d2ba37086662261148e9fe93eecad)) and cap.get([cv.CAP\_PROP\_FRAME\_HEIGHT](https://docs.opencv.org/5.x/d4/d15/group__videoio__flags__base.html#ggaeb8dd9c89c10a5c63c139bf7c4f5704dad8b57083fd9bd58e0f94e68a54b42b7e)). It gives me 640x480 by default. But I want to modify it to 320x240. Just use ret = cap.set([cv.CAP\_PROP\_FRAME\_WIDTH](https://docs.opencv.org/5.x/d4/d15/group__videoio__flags__base.html#ggaeb8dd9c89c10a5c63c139bf7c4f5704dab26d2ba37086662261148e9fe93eecad),320) and ret = cap.set([cv.CAP\_PROP\_FRAME\_HEIGHT](https://docs.opencv.org/5.x/d4/d15/group__videoio__flags__base.html#ggaeb8dd9c89c10a5c63c139bf7c4f5704dad8b57083fd9bd58e0f94e68a54b42b7e),240).

ret = cap.set([cv.CAP\_PROP\_FRAME\_WIDTH](https://docs.opencv.org/5.x/d4/d15/group__videoio__flags__base.html#ggaeb8dd9c89c10a5c63c139bf7c4f5704dab26d2ba37086662261148e9fe93eecad),320)

ret = cap.set([cv.CAP\_PROP\_FRAME\_HEIGHT](https://docs.opencv.org/5.x/d4/d15/group__videoio__flags__base.html#ggaeb8dd9c89c10a5c63c139bf7c4f5704dad8b57083fd9bd58e0f94e68a54b42b7e),240).

**Note**

Make sure a proper version of ffmpeg or gstreamer is installed. Sometimes it is a headache to work with video capture, mostly due to wrong installation of ffmpeg/gstreamer.

[FourCC](https://en.wikipedia.org/wiki/FourCC) is a 4-byte code used to specify the video codec. The list of available codes can be found in [fourcc.org](http://www.fourcc.org/codecs.php). It is platform dependent. The following codecs work fine for me.

* In Fedora: DIVX, XVID, MJPG, X264, WMV1, WMV2. (XVID is more preferable. MJPG results in high size video. X264 gives very small size video)
* In Windows: DIVX (More to be tested and added)
* In OSX: MJPG (.mp4), DIVX (.avi), X264 (.mkv).

FourCC code is passed as `

cv.VideoWriter\_fourcc('M','J','P','G')

or cv.VideoWriter\_fourcc(\*'MJPG')` for MJPG.

* You will learn these functions : [cv.line()](https://docs.opencv.org/5.x/d6/d6e/group__imgproc__draw.html#ga7078a9fae8c7e7d13d24dac2520ae4a2), [cv.circle()](https://docs.opencv.org/5.x/d6/d6e/group__imgproc__draw.html#gaf10604b069374903dbd0f0488cb43670) , [cv.rectangle()](https://docs.opencv.org/5.x/d6/d6e/group__imgproc__draw.html#ga07d2f74cadcf8e305e810ce8eed13bc9), [cv.ellipse()](https://docs.opencv.org/5.x/d6/d6e/group__imgproc__draw.html#ga28b2267d35786f5f890ca167236cbc69), [cv.putText()](https://docs.opencv.org/5.x/d6/d6e/group__imgproc__draw.html#ga5126f47f883d730f633d74f07456c576) etc.

## Code

In all the above functions, you will see some common arguments as given below:

* img : The image where you want to draw the shapes
* color : Color of the shape. for BGR, pass it as a tuple, eg: (255,0,0) for blue. For grayscale, just pass the scalar value.
* thickness : Thickness of the line or circle etc. If **-1** is passed for closed figures like circles, it will fill the shape. default thickness = 1
* lineType : Type of line, whether 8-connected, anti-aliased line etc. By default, it is 8-connected. [cv.LINE\_AA](https://docs.opencv.org/5.x/d6/d6e/group__imgproc__draw.html" \l "ggaf076ef45de481ac96e0ab3dc2c29a777a85fdabe5335c9e6656563dfd7c94fb4f" \o "antialiased line ) gives anti-aliased line which looks great for curves.

import numpy as np

import cv2 as cv

# Create a black image

img = np.zeros((512,512,3), np.uint8)

# Draw a diagonal blue line with thickness of 5 px

[cv.line](https://docs.opencv.org/5.x/d6/d6e/group__imgproc__draw.html#ga7078a9fae8c7e7d13d24dac2520ae4a2)(img,(0,0),(511,511),(255,0,0),5)

### Adding Text to Images:

To put texts in images, you need specify following things.

* Text data that you want to write
* Position coordinates of where you want put it (i.e. bottom-left corner where data starts).
* Font type (Check [cv.putText()](https://docs.opencv.org/5.x/d6/d6e/group__imgproc__draw.html" \l "ga5126f47f883d730f633d74f07456c576" \o "Draws a text string. ) docs for supported fonts)
* Font Scale (specifies the size of font)
* regular things like color, thickness, lineType etc. For better look, lineType = [cv.LINE\_AA](https://docs.opencv.org/5.x/d6/d6e/group__imgproc__draw.html" \l "ggaf076ef45de481ac96e0ab3dc2c29a777a85fdabe5335c9e6656563dfd7c94fb4f" \o "antialiased line ) is recommended.

font = cv.FONT\_HERSHEY\_COMPLEX

cv.putText(img , "Hello madan" , (250 , 200) , font, 1 , (255 , 164 , 200) , 2 , 3)

## Goal

* Learn to handle mouse events in OpenCV
* You will learn these functions : [cv.setMouseCallback()](https://docs.opencv.org/5.x/d7/dfc/group__highgui.html" \l "ga89e7806b0a616f6f1d502bd8c183ad3e" \o "Sets mouse handler for the specified window. )

## Goal

* Learn to bind trackbar to OpenCV windows
* You will learn these functions : [cv.getTrackbarPos()](https://docs.opencv.org/5.x/d7/dfc/group__highgui.html" \l "ga122632e9e91b9ec06943472c55d9cda8" \o "Returns the trackbar position. ), [cv.createTrackbar()](https://docs.opencv.org/5.x/d7/dfc/group__highgui.html" \l "gaf78d2155d30b728fc413803745b67a9b" \o "Creates a trackbar and attaches it to the specified window. ) etc.

For [cv.createTrackbar()](https://docs.opencv.org/5.x/d7/dfc/group__highgui.html" \l "gaf78d2155d30b728fc413803745b67a9b" \o "Creates a trackbar and attaches it to the specified window. ) function,

first argument is the trackbar name,

second one is the window name to which it is attached,

third argument is the default value,

fourth one is the maximum value and

fifth one is the callback function which is executed every time trackbar value changes.

The callback function always has a default argument which is the trackbar position. In our case, function does nothing, so we simply pass.

**Goal**

Learn to:

* Access pixel values and modify them
* Access image properties
* Set a Region of Interest (ROI)
* Split and merge images

## Making Borders for Images (Padding)

If you want to create a border around an image, something like a photo frame, you can use [cv.copyMakeBorder()](https://docs.opencv.org/5.x/d2/de8/group__core__array.html" \l "ga2ac1049c2c3dd25c2b41bffe17658a36" \o "Forms a border around an image. ). But it has more applications for convolution operation, zero padding etc. This function takes following arguments:

* **src** - input image
* **top**, **bottom**, **left**, **right** - border width in number of pixels in corresponding directions
* **borderType** - Flag defining what kind of border to be added. It can be following types:
  + [cv.BORDER\_CONSTANT](https://docs.opencv.org/5.x/d2/de8/group__core__array.html#gga209f2f4869e304c82d07739337eae7c5aed2e4346047e265c8c5a6d0276dcd838) - Adds a constant colored border. The value should be given as next argument.
  + [cv.BORDER\_REFLECT](https://docs.opencv.org/5.x/d2/de8/group__core__array.html#gga209f2f4869e304c82d07739337eae7c5a815c8a89b7cb206dcba14d11b7560f4b) - Border will be mirror reflection of the border elements, like this : fedcba|abcdefgh|hgfedcb
  + [cv.BORDER\_REFLECT\_101](https://docs.opencv.org/5.x/d2/de8/group__core__array.html#gga209f2f4869e304c82d07739337eae7c5ab3c5a6143d8120b95005fa7105a10bb4) or [cv.BORDER\_DEFAULT](https://docs.opencv.org/5.x/d2/de8/group__core__array.html" \l "gga209f2f4869e304c82d07739337eae7c5afe14c13a4ea8b8e3b3ef399013dbae01" \o "same as BORDER_REFLECT_101 ) - Same as above, but with a slight change, like this : gfedcb|abcdefgh|gfedcba
  + [cv.BORDER\_REPLICATE](https://docs.opencv.org/5.x/d2/de8/group__core__array.html#gga209f2f4869e304c82d07739337eae7c5aa1de4cff95e3377d6d0cbe7569bd4e9f) - Last element is replicated throughout, like this: aaaaaa|abcdefgh|hhhhhhh
  + [cv.BORDER\_WRAP](https://docs.opencv.org/5.x/d2/de8/group__core__array.html#gga209f2f4869e304c82d07739337eae7c5a697c1b011884a7c2bdc0e5caf7955661) - Can't explain, it will look like this : cdefgh|abcdefgh|abcdefg
* **value** - Color of border if border type is [cv.BORDER\_CONSTANT](https://docs.opencv.org/5.x/d2/de8/group__core__array.html" \l "gga209f2f4869e304c82d07739337eae7c5aed2e4346047e265c8c5a6d0276dcd838" \o "iiiiii|abcdefgh|iiiiiii with some specified i )

## Image Blending

This is also image addition, but different weights are given to images in order to give a feeling of blending or transparency. Images are added as per the equation below:

g(x)=(1−α)f0(x)+αf1(x)

By varying α from 0→1, you can perform a cool transition between one image to another.

Here I took two images to blend together. The first image is given a weight of 0.7 and the second image is given 0.3. [cv.addWeighted()](https://docs.opencv.org/5.x/d2/de8/group__core__array.html" \l "gafafb2513349db3bcff51f54ee5592a19" \o "Calculates the weighted sum of two arrays. ) applies the following equation to the image:

dst=α⋅img1+β⋅img2+γ

**Performance Measurement and Improvement Techniques**

## Goal

In image processing, since you are dealing with a large number of operations per second, it is mandatory that your code is not only providing the correct solution, but that it is also providing it in the fastest manner. So in this chapter, you will learn:

* To measure the performance of your code.
* Some tips to improve the performance of your code.
* You will see these functions: [cv.getTickCount](https://docs.opencv.org/5.x/db/de0/group__core__utils.html" \l "gae73f58000611a1af25dd36d496bf4487" \o "Returns the number of ticks. ), [cv.getTickFrequency](https://docs.opencv.org/5.x/db/de0/group__core__utils.html" \l "ga705441a9ef01f47acdc55d87fbe5090c" \o "Returns the number of ticks per second. ), etc.
* The [cv.getTickCount](https://docs.opencv.org/5.x/db/de0/group__core__utils.html" \l "gae73f58000611a1af25dd36d496bf4487" \o "Returns the number of ticks. ) function returns the number of clock-cycles after a reference event (like the moment the machine was switched ON) to the moment this function is called. So if you call it before and after the function execution, you get the number of clock-cycles used to execute a function.
* The [cv.getTickFrequency](https://docs.opencv.org/5.x/db/de0/group__core__utils.html" \l "ga705441a9ef01f47acdc55d87fbe5090c" \o "Returns the number of ticks per second. ) function returns the frequency of clock-cycles, or the number of clock-cycles per second. So to find the time of execution in seconds,

## Default Optimization in OpenCV

Many of the OpenCV functions are optimized using SSE2, AVX, etc. It contains the unoptimized code also. So if our system support these features, we should exploit them (almost all modern day processors support them). It is enabled by default while compiling. So OpenCV runs the optimized code if it is enabled, otherwise it runs the unoptimized code. You can use [cv.useOptimized()](https://docs.opencv.org/5.x/db/de0/group__core__utils.html" \l "gafa6d5d04eff341825573ec6c0aa6519f" \o "Returns the status of optimized code usage. ) to check if it is enabled/disabled and [cv.setUseOptimized()](https://docs.opencv.org/5.x/db/de0/group__core__utils.html" \l "ga3c8487ea4449e550bc39575ede094c7a" \o "Enables or disables the optimized code. ) to enable/disable it.

In [35]: %timeit z = [cv.countNonZero](https://docs.opencv.org/5.x/d2/de8/group__core__array.html#gaa4b89393263bb4d604e0fe5986723914)(img)

100000 loops, best of 3: 15.8 us per loop

In [36]: %timeit z = np.count\_nonzero(img)

1000 loops, best of 3: 370 us per loop

See, the OpenCV function is nearly 25x faster than the Numpy function.

There are several techniques and coding methods to exploit maximum performance of Python and Numpy. Only relevant ones are noted here and links are given to important sources. The main thing to be noted here is, first try to implement the algorithm in a simple manner. Once it is working, profile it, find the bottlenecks, and optimize them.

1. Avoid using loops in Python as much as possible, especially double/triple loops etc. They are inherently slow.
2. Vectorize the algorithm/code to the maximum extent possible, because Numpy and OpenCV are optimized for vector operations.
3. Exploit the cache coherence.
4. Never make copies of an array unless it is necessary. Try to use views instead. Array copying is a costly operation.

f your code is still slow after doing all of these operations, or if the use of large loops is inevitable, use additional libraries like Cython to make it faster.

**Changing Colorspaces**

## Goal

* In this tutorial, you will learn how to convert images from one color-space to another, like BGR ↔ Gray, BGR ↔ HSV, etc.
* In addition to that, we will create an application to extract a colored object in a video
* You will learn the following functions: [cv.cvtColor()](https://docs.opencv.org/5.x/d8/d01/group__imgproc__color__conversions.html" \l "ga397ae87e1288a81d2363b61574eb8cab" \o "Converts an image from one color space to another. ), [cv.inRange()](https://docs.opencv.org/5.x/d2/de8/group__core__array.html" \l "ga48af0ab51e36436c5d04340e036ce981" \o "Checks if array elements lie between the elements of two other arrays. ), etc.

Now you take [H-10, 100,100] and [H+10, 255, 255] as the lower bound and upper bound respectively.

**Geometric Transformations of Images**

## Goals

* Learn to apply different geometric transformations to images, like translation, rotation, affine transformation etc.
* You will see these functions: [cv.getPerspectiveTransform](https://docs.opencv.org/5.x/da/d54/group__imgproc__transform.html" \l "ga20f62aa3235d869c9956436c870893ae" \o "Calculates a perspective transform from four pairs of the corresponding points. )

**Transformations**

OpenCV provides two transformation functions, [cv.warpAffine](https://docs.opencv.org/5.x/da/d54/group__imgproc__transform.html" \l "ga0203d9ee5fcd28d40dbc4a1ea4451983" \o "Applies an affine transformation to an image. ) and [cv.warpPerspective](https://docs.opencv.org/5.x/da/d54/group__imgproc__transform.html" \l "gaf73673a7e8e18ec6963e3774e6a94b87" \o "Applies a perspective transformation to an image. ), with which you can perform all kinds of transformations. [cv.warpAffine](https://docs.opencv.org/5.x/da/d54/group__imgproc__transform.html" \l "ga0203d9ee5fcd28d40dbc4a1ea4451983" \o "Applies an affine transformation to an image. ) takes a 2x3 transformation matrix while [cv.warpPerspective](https://docs.opencv.org/5.x/da/d54/group__imgproc__transform.html" \l "gaf73673a7e8e18ec6963e3774e6a94b87" \o "Applies a perspective transformation to an image. ) takes a 3x3 transformation matrix as input.

### Scaling

Scaling is just resizing of the image. OpenCV comes with a function [cv.resize()](https://docs.opencv.org/5.x/da/d54/group__imgproc__transform.html" \l "ga47a974309e9102f5f08231edc7e7529d" \o "Resizes an image. ) for this purpose. The size of the image can be specified manually, or you can specify the scaling factor. Different interpolation methods are used. Preferable interpolation methods are [cv.INTER\_AREA](https://docs.opencv.org/5.x/da/d54/group__imgproc__transform.html" \l "gga5bb5a1fea74ea38e1a5445ca803ff121acf959dca2480cc694ca016b81b442ceb) for shrinking and [cv.INTER\_CUBIC](https://docs.opencv.org/5.x/da/d54/group__imgproc__transform.html" \l "gga5bb5a1fea74ea38e1a5445ca803ff121a55e404e7fa9684af79fe9827f36a5dc1) (slow) & [cv.INTER\_LINEAR](https://docs.opencv.org/5.x/da/d54/group__imgproc__transform.html" \l "gga5bb5a1fea74ea38e1a5445ca803ff121ac97d8e4880d8b5d509e96825c7522deb) for zooming.

### Translation

Translation is the shifting of an object's location. If you know the shift in the (x,y) direction and let it be (tx,ty), you can create the transformation matrix M as follows:

M=[1 0 0 1 t x ty]

You can take make it into a Numpy array of type np.float32 and pass it into the [cv.warpAffine()](https://docs.opencv.org/5.x/da/d54/group__imgproc__transform.html" \l "ga0203d9ee5fcd28d40dbc4a1ea4451983" \o "Applies an affine transformation to an image. ) function.

### Rotation

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

M=[cosθsinθ−sinθcosθ]

But OpenCV provides scaled rotation with adjustable center of rotation so that you can rotate at any location you prefer. The modified transformation matrix is given by

[α−ββα(1−α)⋅center.x−β⋅center.yβ⋅center.x+(1−α)⋅center.y]

where:

α=scale⋅cosθ,β=scale⋅sinθ

To find this transformation matrix, OpenCV provides a function, [cv.getRotationMatrix2D](https://docs.opencv.org/5.x/da/d54/group__imgproc__transform.html#gafbbc470ce83812914a70abfb604f4326)

**Smoothing Images**

**Goals**

Learn to:

* Blur images with various low pass filters
* Apply custom-made filters to images (2D convolution)

## 2D Convolution ( Image Filtering )

As in one-dimensional signals, images also can be filtered with various low-pass filters (LPF), high-pass filters (HPF), etc. LPF helps in removing noise, blurring images, etc. HPF filters help in finding edges in images.

**Image Blurring (Image Smoothing)**

Image blurring is achieved by convolving the image with a low-pass filter kernel. It is useful for removing noise. It actually removes high frequency content (eg: noise, edges) from the image. So edges are blurred a little bit in this operation (there are also blurring techniques which don't blur the edges). OpenCV provides four main types of blurring techniques.

**1. Averaging**  [cv.blur()](https://docs.opencv.org/5.x/d4/d86/group__imgproc__filter.html" \l "ga8c45db9afe636703801b0b2e440fce37" \o "Blurs an image using the normalized box filter. ) or [cv.boxFilter()](https://docs.opencv.org/5.x/d4/d86/group__imgproc__filter.html" \l "gad533230ebf2d42509547d514f7d3fbc3" \o "Blurs an image using the box filter. ).

**2. Gaussian Blurring  [cv.GaussianBlur()](https://docs.opencv.org/5.x/d4/d86/group__imgproc__filter.html" \l "gaabe8c836e97159a9193fb0b11ac52cf1" \o "Blurs an image using a Gaussian filter. ).**

**3. Median Blurring** [**cv.medianBlur()**](https://docs.opencv.org/5.x/d4/d86/group__imgproc__filter.html#ga564869aa33e58769b4469101aac458f9)

**4. Bilateral Filtering** [**cv.bilateralFilter()**](https://docs.opencv.org/5.x/d4/d86/group__imgproc__filter.html#ga9d7064d478c95d60003cf839430737ed)

**Morphological Transformations**

## Goal

In this chapter,

* We will learn different morphological operations like Erosion, Dilation, Opening, Closing etc.
* We will see different functions like : [cv.erode()](https://docs.opencv.org/5.x/d4/d86/group__imgproc__filter.html" \l "gaeb1e0c1033e3f6b891a25d0511362aeb" \o "Erodes an image by using a specific structuring element. ), [cv.dilate()](https://docs.opencv.org/5.x/d4/d86/group__imgproc__filter.html" \l "ga4ff0f3318642c4f469d0e11f242f3b6c" \o "Dilates an image by using a specific structuring element. ), [cv.morphologyEx()](https://docs.opencv.org/5.x/d4/d86/group__imgproc__filter.html" \l "ga67493776e3ad1a3df63883829375201f" \o "Performs advanced morphological transformations. ) etc.

**Image Gradients**

## Goal

In this chapter, we will learn to:

* Find Image gradients, edges etc
* We will see following functions : [cv.Sobel()](https://docs.opencv.org/5.x/d4/d86/group__imgproc__filter.html" \l "gacea54f142e81b6758cb6f375ce782c8d" \o "Calculates the first, second, third, or mixed image derivatives using an extended Sobel operator...), [cv.Scharr()](https://docs.opencv.org/5.x/d4/d86/group__imgproc__filter.html" \l "gaa13106761eedf14798f37aa2d60404c9" \o "Calculates the first x- or y- image derivative using Scharr operator. ), [cv.Laplacian()](https://docs.opencv.org/5.x/d4/d86/group__imgproc__filter.html" \l "gad78703e4c8fe703d479c1860d76429e6" \o "Calculates the Laplacian of an image. ) etc

**Image Pyramids**

## Goal

In this chapter,

* We will learn about Image Pyramids
* We will use Image pyramids to create a new fruit, "Orapple"
* We will see these functions: [cv.pyrUp()](https://docs.opencv.org/5.x/d4/d86/group__imgproc__filter.html" \l "gada75b59bdaaca411ed6fee10085eb784" \o "Upsamples an image and then blurs it. ), [cv.pyrDown()](https://docs.opencv.org/5.x/d4/d86/group__imgproc__filter.html" \l "gaf9bba239dfca11654cb7f50f889fc2ff" \o "Blurs an image and downsamples it. )

**Contours : Getting Started**

**Next Tutorial:** [Contour Features](https://docs.opencv.org/5.x/dd/d49/tutorial_py_contour_features.html)

## Goal

* Understand what contours are.
* Learn to find contours, draw contours etc
* You will see these functions : [cv.findContours()](https://docs.opencv.org/5.x/d3/dc0/group__imgproc__shape.html" \l "gadf1ad6a0b82947fa1fe3c3d497f260e0" \o "Finds contours in a binary image. ), [cv.drawContours()](https://docs.opencv.org/5.x/d6/d6e/group__imgproc__draw.html" \l "ga746c0625f1781f1ffc9056259103edbc" \o "Draws contours outlines or filled contours. )

there are three arguments in [cv.findContours()](https://docs.opencv.org/5.x/d3/dc0/group__imgproc__shape.html" \l "gadf1ad6a0b82947fa1fe3c3d497f260e0" \o "Finds contours in a binary image. ) function,

first one is source image,

second is contour retrieval mode,

third is contour approximation method.

**Contour Features**

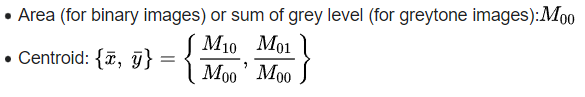
## Goal

In this article, we will learn

* To find the different features of contours, like area, perimeter, centroid, bounding box etc
* You will see plenty of functions related to contours.

## 1. Moments

Image moments help you to calculate some features like center of mass of the object, area of the object etc. Check out the wikipedia page on [Image Moments](https://en.wikipedia.org/wiki/Image_moment)



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

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

## 2. Contour Area

Contour area is given by the function [cv.contourArea()](https://docs.opencv.org/5.x/d3/dc0/group__imgproc__shape.html" \l "ga2c759ed9f497d4a618048a2f56dc97f1" \o "Calculates a contour area. ) or from moments, **M['m00']**.

## 3. Contour Perimeter

It is also called arc length. It can be found out using [cv.arcLength()](https://docs.opencv.org/5.x/d3/dc0/group__imgproc__shape.html" \l "ga8d26483c636be6b35c3ec6335798a47c" \o "Calculates a contour perimeter or a curve length. ) function. Second argument specify whether shape is a closed contour (if passed True), or just a curve.

## 7. Bounding Rectangle

There are two types of bounding rectangles.

### 7.a. Straight Bounding Rectangle

It is a straight rectangle, it doesn't consider the rotation of the object. So area of the bounding rectangle won't be minimum. It is found by the function [cv.boundingRect()](https://docs.opencv.org/5.x/d3/dc0/group__imgproc__shape.html" \l "ga103fcbda2f540f3ef1c042d6a9b35ac7" \o "Calculates the up-right bounding rectangle of a point set or non-zero pixels of gray-scale image...).

### 7.b. Rotated Rectangle

Here, bounding rectangle is drawn with minimum area, so it considers the rotation also. The function used is [cv.minAreaRect()](https://docs.opencv.org/5.x/d3/dc0/group__imgproc__shape.html" \l "ga3d476a3417130ae5154aea421ca7ead9" \o "Finds a rotated rectangle of the minimum area enclosing the input 2D point set. ). It returns a Box2D structure which contains following details - ( center (x,y), (width, height), angle of rotation ). But to draw this rectangle, we need 4 corners of the rectangle. It is obtained by the function [cv.boxPoints()](https://docs.opencv.org/5.x/d3/dc0/group__imgproc__shape.html" \l "gaf78d467e024b4d7936cf9397185d2f5c" \o "Finds the four vertices of a rotated rect. Useful to draw the rotated rectangle. )

**8. Minimum Enclosing Circle**

Next we find the circumcircle of an object using the function [cv.minEnclosingCircle()](https://docs.opencv.org/5.x/d3/dc0/group__imgproc__shape.html" \l "ga8ce13c24081bbc7151e9326f412190f1" \o "Finds a circle of the minimum area enclosing a 2D point set. ). It is a circle which completely covers the object with minimum area.

**9. Fitting an Ellipse**

Next one is to fit an ellipse to an object. It returns the rotated rectangle in which the ellipse is inscribed.

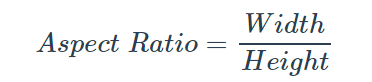
**10. Fitting a Line**

Similarly we can fit a line to a set of points. Below image contains a set of white points. We can approximate a straight line to it.

**Contour Properties**

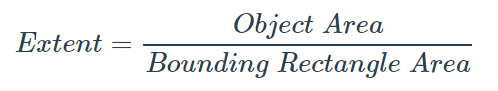
## 1. Aspect Ratio

It is the ratio of width to height of bounding rect of the object.



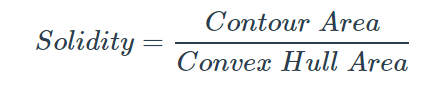
## 2. Extent

Extent is the ratio of contour area to bounding rectangle area.



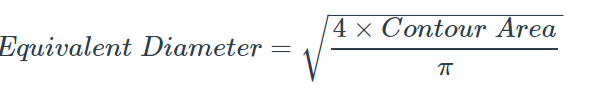
**3. Solidity**

Solidity is the ratio of contour area to its convex hull area.



## 4. Equivalent Diameter

Equivalent Diameter is the diameter of the circle whose area is same as the contour area.



## 5. Orientation

Orientation is the angle at which object is directed. Following method also gives the Major Axis and Minor Axis lengths.

(x,y),(MA,ma),angle = [cv.fitEllipse](https://docs.opencv.org/5.x/d3/dc0/group__imgproc__shape.html#gaf259efaad93098103d6c27b9e4900ffa)(cnt)

## 6. Mask and Pixel Points

In some cases, we may need all the points which comprises that object

## 7. Maximum Value, Minimum Value and their locations

min\_val, max\_val, min\_loc, max\_loc = [cv.minMaxLoc](https://docs.opencv.org/5.x/d2/de8/group__core__array.html#ga8873b86a29c5af51cafdcee82f8150a7)(imgray,mask = mask)

## 8. Mean Color or Mean Intensity

Here, we can find the average color of an object. Or it can be average intensity of the object in grayscale mode. We again use the same mask to do it.

mean\_val = [cv.mean](https://docs.opencv.org/5.x/d2/de8/group__core__array.html#ga191389f8a0e58180bb13a727782cd461)(im,mask = mask)

## 9. Extreme Points

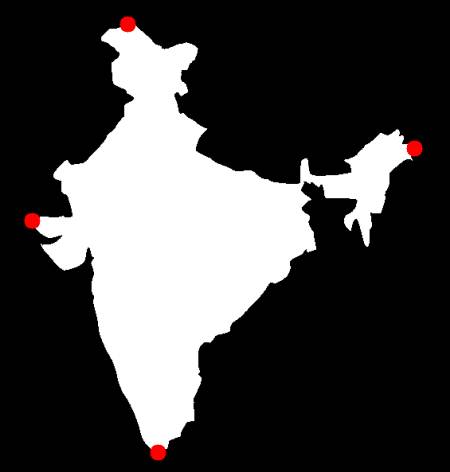
Extreme Points means topmost, bottommost, rightmost and leftmost points of the object.

leftmost = tuple(cnt[cnt[:,:,0].argmin()][0])

rightmost = tuple(cnt[cnt[:,:,0].argmax()][0])

topmost = tuple(cnt[cnt[:,:,1].argmin()][0])

bottommost = tuple(cnt[cnt[:,:,1].argmax()][0])



**1. Convexity Defects**

We saw what is convex hull in second chapter about contours. Any deviation of the object from this hull can be considered as convexity defect.

OpenCV comes with a ready-made function to find this, [cv.convexityDefects()](https://docs.opencv.org/5.x/d3/dc0/group__imgproc__shape.html" \l "gada4437098113fd8683c932e0567f47ba" \o "Finds the convexity defects of a contour. ).

**Histograms - 1 : Find, Plot, Analyze !!!**

## Goal

Learn to

* Find histograms, using both OpenCV and Numpy functions
* Plot histograms, using OpenCV and Matplotlib functions
* You will see these functions : [cv.calcHist()](https://docs.opencv.org/5.x/d6/dc7/group__imgproc__hist.html" \l "ga4b2b5fd75503ff9e6844cc4dcdaed35d" \o "Calculates a histogram of a set of arrays. ), **np.histogram()** etc.

### 1. Histogram Calculation in OpenCV

So now we use [cv.calcHist()](https://docs.opencv.org/5.x/d6/dc7/group__imgproc__hist.html" \l "ga4b2b5fd75503ff9e6844cc4dcdaed35d" \o "Calculates a histogram of a set of arrays. ) function to find the histogram. Let's familiarize with the function and its parameters :

cv.calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]])

1. images : it is the source image of type uint8 or float32. it should be given in square brackets, ie, "[img]".
2. channels : it is also given in square brackets. It is the index of channel for which we calculate histogram. For example, if input is grayscale image, its value is [0]. For color image, you can pass [0], [1] or [2] to calculate histogram of blue, green or red channel respectively.
3. mask : mask image. To find histogram of full image, it is given as "None". But if you want to find histogram of particular region of image, you have to create a mask image for that and give it as mask. (I will show an example later.)
4. histSize : this represents our BIN count. Need to be given in square brackets. For full scale, we pass [256].
5. ranges : this is our RANGE. Normally, it is [0,256].

## Histograms Equalization in OpenCV

OpenCV has a function to do this, [cv.equalizeHist()](https://docs.opencv.org/5.x/d6/dc7/group__imgproc__hist.html" \l "ga7e54091f0c937d49bf84152a16f76d6e" \o "Equalizes the histogram of a grayscale image. ). Its input is just grayscale image and output is our histogram equalized image.

## CLAHE (Contrast Limited Adaptive Histogram Equalization)

**adaptive histogram equalization** is used. In this, image is divided into small blocks called "tiles" (tileSize is 8x8 by default in OpenCV). Then each of these blocks are histogram equalized as usual. So in a small area, histogram would confine to a small region (unless there is noise). If noise is there, it will be amplified. To avoid this, **contrast limiting** is applied. If any histogram bin is above the specified contrast limit (by default 40 in OpenCV), those pixels are clipped and distributed uniformly to other bins before applying histogram equalization. After equalization, to remove artifacts in tile borders, bilinear interpolation is applied.

**Template Matching**

## Goals

In this chapter, you will learn

* To find objects in an image using Template Matching
* You will see these functions : [cv.matchTemplate()](https://docs.opencv.org/5.x/df/dfb/group__imgproc__object.html" \l "ga586ebfb0a7fb604b35a23d85391329be" \o "Compares a template against overlapped image regions. ), [cv.minMaxLoc()](https://docs.opencv.org/5.x/d2/de8/group__core__array.html" \l "gab473bf2eb6d14ff97e89b355dac20707" \o "Finds the global minimum and maximum in an array. )

**Hough Line Transform**

## Goal

In this chapter,

* We will understand the concept of the Hough Transform.
* We will see how to use it to detect lines in an image.
* We will see the following functions: [cv.HoughLines()](https://docs.opencv.org/5.x/dd/d1a/group__imgproc__feature.html" \l "ga46b4e588934f6c8dfd509cc6e0e4545a" \o "Finds lines in a binary image using the standard Hough transform. ), [cv.HoughLinesP()](https://docs.opencv.org/5.x/dd/d1a/group__imgproc__feature.html" \l "ga8618180a5948286384e3b7ca02f6feeb" \o "Finds line segments in a binary image using the probabilistic Hough transform. )

 First parameter, Input image should be a binary image, so apply threshold or use canny edge detection before applying hough transform.

Second and third parameters are ρ and θ accuracies respectively.

Fourth argument is the threshold,

**Hough Circle Transform**

## Goal

In this chapter,

* We will learn to use Hough Transform to find circles in an image.
* We will see these functions: [cv.HoughCircles()](https://docs.opencv.org/5.x/dd/d1a/group__imgproc__feature.html" \l "ga47849c3be0d0406ad3ca45db65a25d2d" \o "Finds circles in a grayscale image using the Hough transform. )

**Interactive Foreground Extraction using GrabCut Algorithm**

**Goal**

In this chapter

* We will see GrabCut algorithm to extract foreground in images
* We will create an interactive application for this.

**Harris Corner Detection**

## Goal

In this chapter,

* We will understand the concepts behind Harris Corner Detection.
* We will see the following functions: [cv.cornerHarris()](https://docs.opencv.org/5.x/dd/d1a/group__imgproc__feature.html" \l "gac1fc3598018010880e370e2f709b4345" \o "Harris corner detector. ), [cv.cornerSubPix()](https://docs.opencv.org/5.x/dd/d1a/group__imgproc__feature.html" \l "ga354e0d7c86d0d9da75de9b9701a9a87e" \o "Refines the corner locations. )

## Theory

In the last chapter, we saw that corners are regions in the image with large variation in intensity in all the directions. One early attempt to find these corners was done by **Chris Harris & Mike Stephens** in their paper **A Combined Corner and Edge Detector** in 1988, so now it is called the Harris Corner Detector. He took this simple idea to a mathematical form

**Introduction to SIFT (Scale-Invariant Feature Transform)**

**Goal**

In this chapter,

* We will learn about the concepts of SIFT algorithm
* We will learn to find SIFT Keypoints and Descriptors.

## SIFT in OpenCV

Now let's see SIFT functionalities available in OpenCV. Note that these were previously only available in [the opencv contrib repo](https://github.com/opencv/opencv_contrib), but the patent expired in the year 2020. So they are now included in the main repo.

 To combine this exposures it is useful to know your camera’s response function and there are algorithms to estimate it. After the HDR image has been blended it has to be converted back to 8-bit to view it on usual displays. This process is called tonemapping.

* **Estimate camera response**

calibrate = [cv.createCalibrateDebevec](https://docs.opencv.org/5.x/d6/df5/group__photo__hdr.html#ga7fed9707ad5f2cc0e633888867109f90)()

response = calibrate.process(images, times)

* **Make HDR image**

merge\_debevec = [cv.createMergeDebevec](https://docs.opencv.org/5.x/d6/df5/group__photo__hdr.html#gaa8eab36bc764abb2a225db7c945f87f9)()

hdr = merge\_debevec.process(images, times, response)

* **Tonemap HDR image**

tonemap = [cv.createTonemap](https://docs.opencv.org/5.x/d6/df5/group__photo__hdr.html#gabcbd653140b93a1fa87ccce94548cd0d)(2.2)

ldr = tonemap.process(hdr)

* **Perform exposure fusion**

merge\_mertens = [cv.createMergeMertens](https://docs.opencv.org/5.x/d6/df5/group__photo__hdr.html#ga79d59aa3cb3a7c664e59a4b5acc1ccb6)()

fusion = merge\_mertens.process(images)

**How to Use Background Subtraction Methods**

## Goals

In this tutorial you will learn how to:

1. Read data from videos or image sequences by using [cv::VideoCapture](https://docs.opencv.org/5.x/d8/dfe/classcv_1_1VideoCapture.html) ;
2. Create and update the background model by using [cv::BackgroundSubtractor](https://docs.opencv.org/5.x/d7/df6/classcv_1_1BackgroundSubtractor.html) class;
3. Get and show the foreground mask by using [cv::imshow](https://docs.opencv.org/5.x/d7/dfc/group__highgui.html#ga453d42fe4cb60e5723281a89973ee563) ;

To support this tutorial, several official OpenCV applications will be used: [opencv\_createsamples](https://github.com/opencv/opencv/tree/5.x/apps/createsamples), [opencv\_annotation](https://github.com/opencv/opencv/tree/5.x/apps/annotation), [opencv\_traincascade](https://github.com/opencv/opencv/tree/5.x/apps/traincascade) and [opencv\_visualisation](https://github.com/opencv/opencv/tree/5.x/apps/visualisation).

**Positive Samples**

Positive samples are created by the opencv\_createsamples application. They are used by the boosting process to define what the model should actually look for when trying to find your objects of interest. The application supports two ways of generating a positive sample dataset.

1. You can generate a bunch of positives from a single positive object image.
2. You can supply all the positives yourself and only use the tool to cut them out, resize them and put them in the opencv needed binary format.

* Please note that you need more than a single positive samples before you give it to the mentioned application, because it only applies perspective transformation.
* If you want a robust model, take samples that cover the wide range of varieties that can occur within your object class. For example in the case of faces you should consider different races and age groups, emotions and perhaps beard styles. This also applies when using the second approach.

the operation of the SVM algorithm is based on finding the hyperplane that gives the largest minimum distance to the training examples. Twice, this distance receives the important name of **margin** within SVM's theory. Therefore, the optimal separating hyperplane *maximizes* the margin of the training data.

## Image Denoising in OpenCV

OpenCV provides four variations of this technique.

1. [cv.fastNlMeansDenoising()](https://docs.opencv.org/5.x/d1/d79/group__photo__denoise.html#ga4c6b0031f56ea3f98f768881279ffe93) - works with a single grayscale images
2. [cv.fastNlMeansDenoisingColored()](https://docs.opencv.org/5.x/d1/d79/group__photo__denoise.html#ga03aa4189fc3e31dafd638d90de335617) - works with a color image.
3. [cv.fastNlMeansDenoisingMulti()](https://docs.opencv.org/5.x/d1/d79/group__photo__denoise.html#gaf4421bf068c4d632ea7f0aa38e0bf172) - works with image sequence captured in short period of time (grayscale images)
4. [cv.fastNlMeansDenoisingColoredMulti()](https://docs.opencv.org/5.x/d1/d79/group__photo__denoise.html#gaa501e71f52fb2dc17ff8ca5e7d2d3619) - same as above, but for color images.

Common arguments are:

* h : parameter deciding filter strength. Higher h value removes noise better, but removes details of image also. (10 is ok)
* hForColorComponents : same as h, but for color images only. (normally same as h)
* templateWindowSize : should be odd. (recommended 7)
* searchWindowSize : should be odd. (recommended 21)