**DIGITAL IMAGE PROCESSING COURSE - 2021.FALL   
PRACTICE LABS**

**LAB 04. MORPHOLOGY IMAGE PROCESSING**

**Requirements**

1. Follow the instructions with the help from your instructor.
2. Finish all the exercises (given at the end of this document) in class and do the homework at home. You can update your solutions after class and re-submit all your work in the total submission together with the homework.
3. Grading  
   Total score = 50%\* In-class submission + 50% \* (Total submission) + Homework
4. Plagiarism check

If any 2 of the students have the same output images, then all will get zero for the corresponding exercises.

**INTRODUCTION**

In this Lab, you will learn how to

* You will learn Simple thresholding, Adaptive thresholding, Otsu’s thresholding etc.
* You will learn these functions : cv2.threshold, cv2.adaptiveThreshold etc.
* We will learn different morphological operations like Erosion, Dilation, Opening, Closing etc.
* We will see different functions like : cv2.erode(), cv2.dilate(), cv2.morphologyEx() etc.
* Contour in OpenCV

**INSTRUCTIONS**

1. **Image thresholding**

[**https://opencv24-python-tutorials.readthedocs.io/en/latest/py\_tutorials/py\_imgproc/py\_thresholding/py\_thresholding.html**](https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html)

**Adaptive Thresholding**

Using a global threshold value may not be good choice where image has different lighting conditions in different areas. So, in that case, we may want to use adaptive thresholding. It uses the algorithm that calculates the threshold for a small regions of the image so that we can get different thresholds for different regions of the same image and it gives us better results for images with varying light conditions.

dst = cv.**AdaptiveThreshold**(src, maxValue, adaptive\_method=CV\_ADAPTIVE\_THRESH\_MEAN\_C, thresholdType=CV\_THRESH\_BINARY, blockSize=3, param1=5)

where:

1. src - Source 8-bit single-channel image.
2. dst - Destination image of the same size and the same type as src.
3. maxValue - Non-zero value assigned to the pixels for which the condition is satisfied.
4. adaptiveMethod - Adaptive thresholding algorithm to use, ADAPTIVE\_THRESH\_MEAN\_C (threshold value is the mean of neighbourhood area) or ADAPTIVE\_THRESH\_GAUSSIAN\_C (threshold value is the weighted sum of neighbourhood values where weights are a gaussian window). adaptiveMethod decides how thresholding value is calculated.
5. thresholdType - Thresholding type that must be either THRESH\_BINARY or THRESH\_BINARY\_INV .
6. blockSize - size of a pixel neighborhood that is used to calculate a threshold value for the pixel: 3, 5, 7, and so on.
7. C - Constant subtracted from the mean or weighted mean. Normally, it is positive but may be zero or negative as well. It is just a constant which is subtracted from the mean or weighted mean calculated.

**Example:**

import cv2

img = cv2.imread('grayimage.png',0)

th1 = cv2.**adaptiveThreshold**(img,255,cv2.ADAPTIVE\_THRESH\_MEAN\_C,\

cv2.THRESH\_BINARY)

th2 = cv2.**adaptiveThreshold**(img,255,cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C,\

cv2.THRESH\_BINARY)

**Otsu Threshold (Separating Bimodal Distributions)**

Otsu binarization automatically calculates a threshold value from image histogram for a bimodal image. It uses **cv2.threshold()** function with an extra flag, **cv2.THRESH\_OTSU**. For threshold value, simply pass zero. Then the algorithm finds the optimal threshold value and returns us as the second output, **retVal**. If Otsu thresholding is not used, the **retVal** remains same as the threshold value we used.

**Example:**

import cv2

img = cv2.imread('EinStein.jpg',0)

# global thresholding

ret1,th1 = cv2.**threshold**(img,127,255,cv2.THRESH\_BINARY)

# **Otsu's** thresholding

ret2,th2 = cv2.**threshold**(img,0,255,cv2.THRESH\_BINARY+**cv2.THRESH\_OTSU**)

1. **Morphology Operations**

## **Structuring Element**

We manually created the structuring elements in the previous examples with help of Numpy. It has a rectangular shape. But in some cases, you may need elliptical/circular shaped kernels. So for this purpose, we use OpenCV **cv2.getStructuringElement()**.

Source: [https://opencv24-python-tutorials.readthedocs.io/en/latest/py\_tutorials/py\_imgproc/py\_morphological\_ops/py\_morphological\_ops.html](https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html#exercises)

# Rectangular Kernel

>>> cv2.**getStructuringElement**(cv2.**MORPH\_RECT**,(5,5))

array([[1, 1, 1, 1, 1],

[1, 1, 1, 1, 1],

[1, 1, 1, 1, 1],

[1, 1, 1, 1, 1],

[1, 1, 1, 1, 1]], dtype=uint8)

# Elliptical Kernel

>>> cv2.**getStructuringElement**(cv2.**MORPH\_ELLIPSE**,(5,5))

array([[0, 0, 1, 0, 0],

[1, 1, 1, 1, 1],

[1, 1, 1, 1, 1],

[1, 1, 1, 1, 1],

[0, 0, 1, 0, 0]], dtype=uint8)

# Cross-shaped Kernel

>>> cv2.**getStructuringElement**(cv2.**MORPH\_CROSS**,(5,5))

array([[0, 0, 1, 0, 0],

[0, 0, 1, 0, 0],

[1, 1, 1, 1, 1],

[0, 0, 1, 0, 0],

[0, 0, 1, 0, 0]], dtype=uint8)

## **Erosion**

The basic idea of erosion is just like soil erosion only, it erodes away the boundaries of foreground object (Always try to keep foreground in white). So what does it do? The kernel slides through the image (as in 2D convolution). A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero). So what happends is that, all the pixels near boundary will be discarded depending upon the size of kernel. So the thickness or size of the foreground object decreases or simply white region decreases in the image. It is useful for removing small white noises (as we have seen in colorspace chapter), detach two connected objects etc.

Here, as an example, I would use a 5x5 kernel with full of ones. Let’s see it how it works:

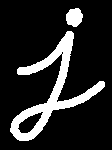
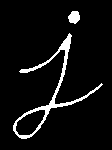
import cv2

import numpy as np

img = cv2.imread('j.png',0)

kernel = np.**ones**((5,5),np.uint8)

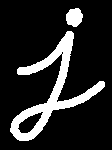
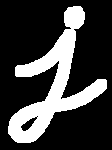
erosion = cv2.**erode**(img,kernel,iterations = 1)

## **Dilation**

It is just opposite of erosion. Here, a pixel element is ‘1’ if atleast one pixel under the kernel is ‘1’. So it increases the white region in the image or size of foreground object increases. Normally, in cases like noise removal, erosion is followed by dilation. Because, erosion removes white noises, but it also shrinks our object. So we dilate it. Since noise is gone, they won’t come back, but our object area increases. It is also useful in joining broken parts of an object.

dilation = cv2.**dilate**(img,kernel,iterations = 1)

## **Opening**

Opening is just another name of **erosion followed by dilation**. It is useful in removing noise, as we explained above. Here we use the function, **cv2.morphologyEx()**

opening = cv2.**morphologyEx**(img, cv2.**MORPH\_OPEN**, kernel)



## **Closing**

Closing is reverse of Opening, **Dilation followed by Erosion**. It is useful in closing small holes inside the foreground objects, or small black points on the object.

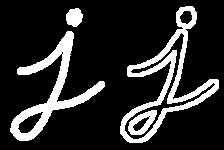
closing = cv2.morphologyEx(img, cv2.MORPH\_CLOSE, kernel)



## **Morphological Gradient**

It is the difference between dilation and erosion of an image. The result will look like the outline of the object.

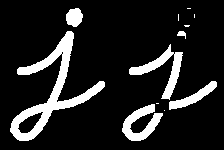
gradient = cv2.**morphologyEx**(img, cv2.**MORPH\_GRADIENT**, kernel)



## **Top Hat**

It is the difference between input image and Opening of the image. Below example is done for a 9x9 kernel.

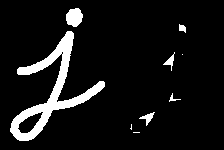
tophat = cv2.**morphologyEx**(img, cv2.**MORPH\_TOPHAT**, kernel)



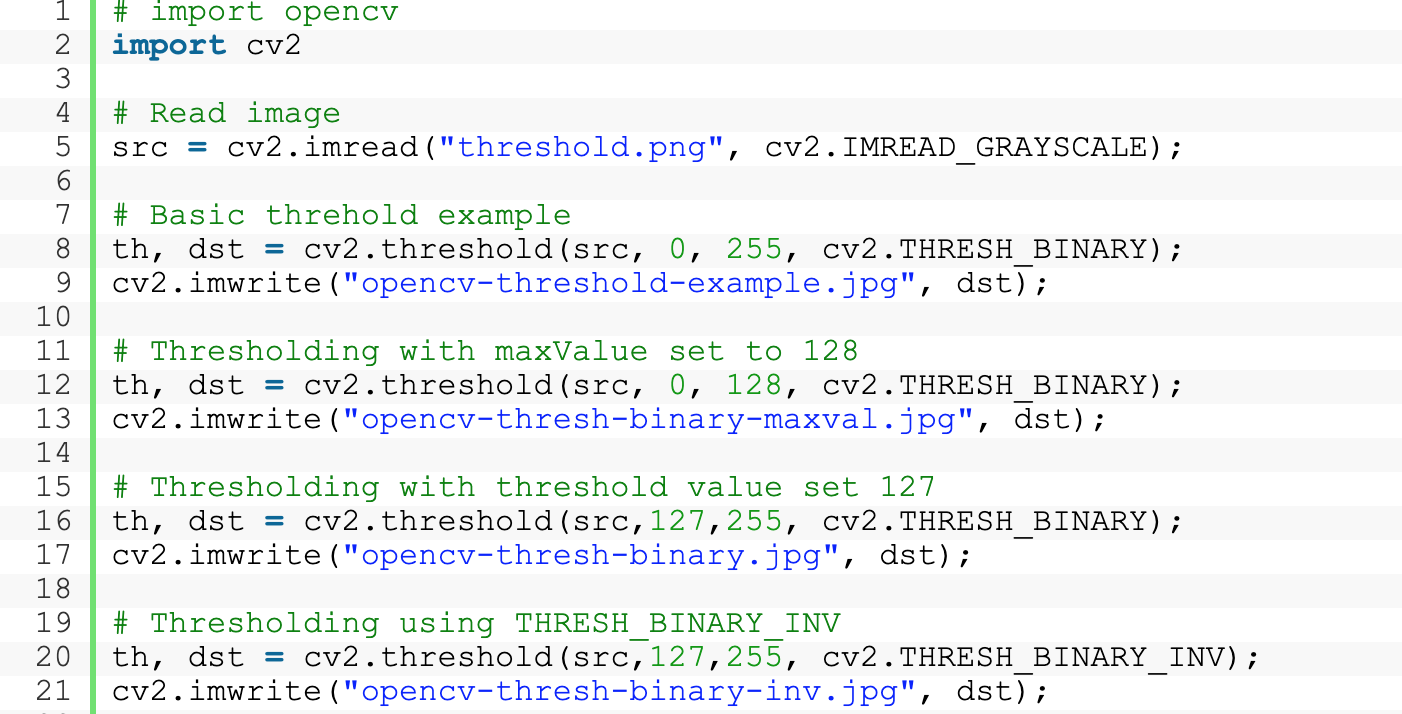
## **Black Hat**

It is the difference between the closing of the input image and input image.

blackhat = cv2.**morphologyEx**(img, cv2.**MORPH\_BLACKHAT**, kernel)



## 

****Source: <https://learnopencv.com/opencv-threshold-python-cpp/>

## Contour Detection using OpenCV

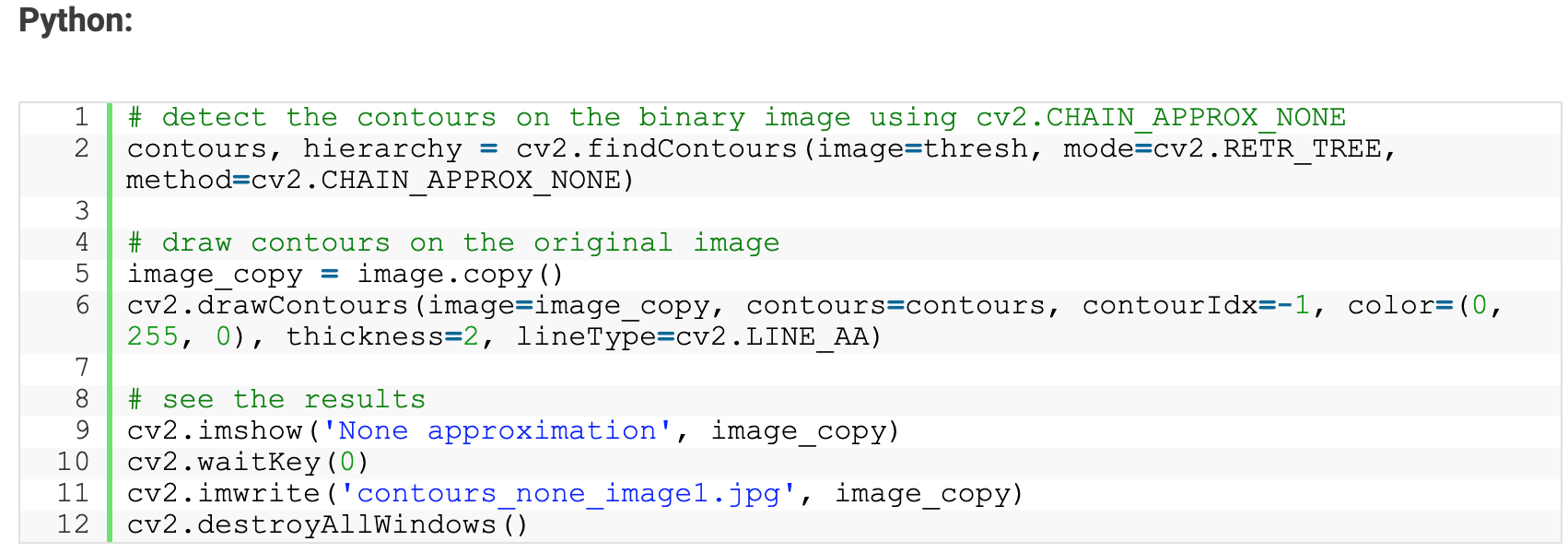
Using contour detection, we can detect the borders of objects, and localize them easily in an image. It is often the first step for many interesting applications, such as image-foreground extraction, simple-image segmentation, detection and recognition.

OpenCV makes it really easy to find and draw contours in images. It provides two simple functions:

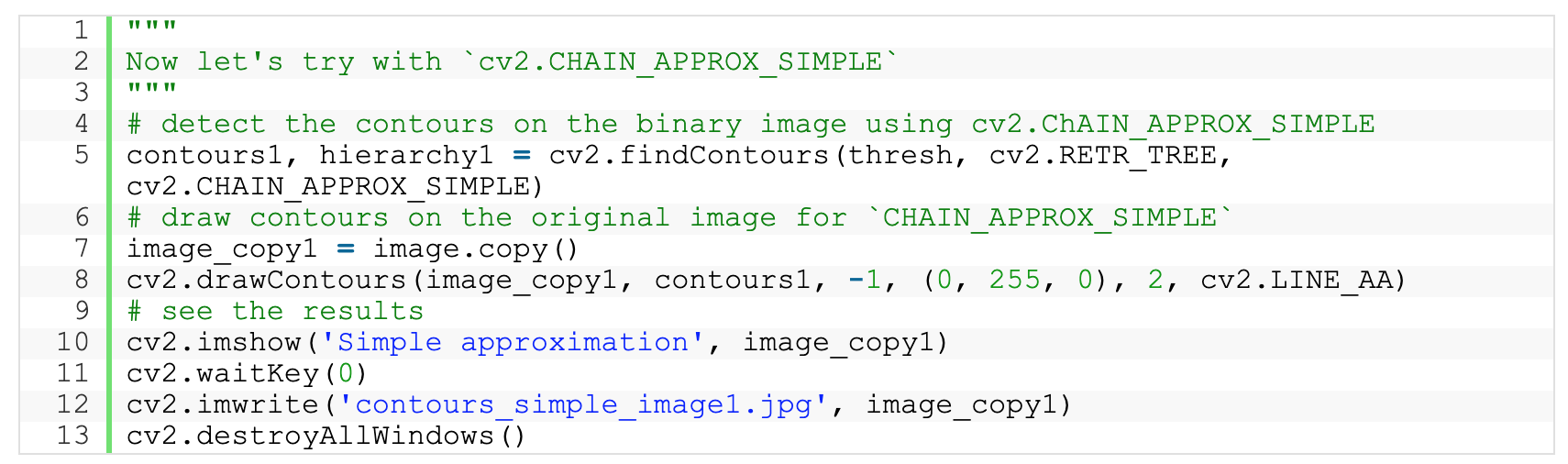
* findContours()
* drawContours()

For more information: <https://learnopencv.com/contour-detection-using-opencv-python-c/>

Find contours with CHAIN\_APPROX\_NONE



Find contours with CHAIN\_APPROX\_SIMPLE



The only difference here is that we specify the method for findContours() as CHAIN\_APPROX\_SIMPLE instead of CHAIN\_APPROX\_NONE.

The CHAIN\_APPROX\_SIMPLE algorithm compresses horizontal, vertical, and diagonal segments along the contour and leaves only their end points. This means that any of the points along the straight paths will be dismissed, and we will be left with only the end points. For example, consider a contour, along a rectangle. All the contour points, except the four corner points will be dismissed. This method is faster than the CHAIN\_APPROX\_NONE because the algorithm does not store all the points, uses less memory, and therefore, takes less time to execute.

## Contour Features

To find the different features of contours, like area, perimeter, centroid, bounding box etc

You will see plenty of functions related to contours.

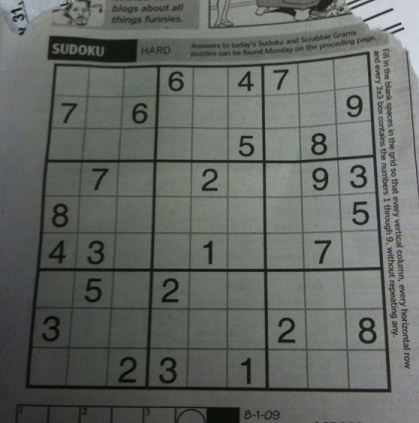
Reference:

<https://docs.opencv.org/4.5.3/dd/d49/tutorial_py_contour_features.html>

**EXERCISE**

**Ex4.1. Image binarization**

Using thresholding techniques to convert the grayscale image into a binary image. Choose the most good-looking one.

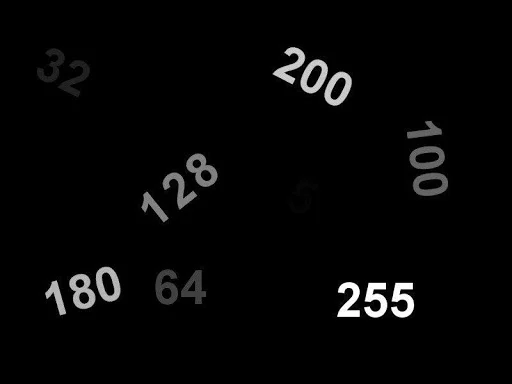


Download the image via this [link](http://aishack.in/static/img/tut/sudoku-original.jpg)

***Hint***: Try with simple or adaptive or Otsu’s thresholding method.

**Ex4.2. Number bounding-box**

Given an input image as follows (numbers are pixel intensities). Using morphological operations and contour to draw rectangles surrounding each number (bounding boxes).



Download: <https://learnopencv.com/wp-content/uploads/2021/06/input_image.jpg>

***Hint***:

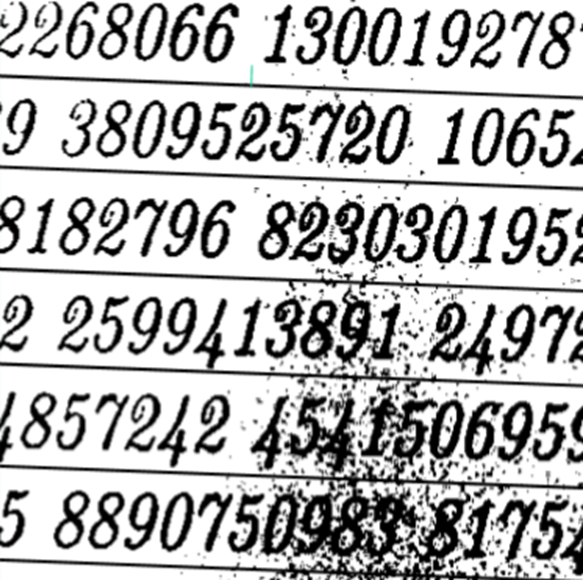
(1) Convert to binary image (mask of numbers) using thresholding techniques

(2) Using dilation/closing to connect digits of each numbers (choose a suitable kernel in size)

(3) Find contours and bounding boxes (using methods in Contour features: boundingRect)

**Ex4.3. Digit bounding-box**

Using morphological operations and contour to draw rectangles surrounding each digit (bounding boxes).



***Hint***:

(1) Convert to binary image (mask of numbers) using thresholding techniques

(2) Using opening and closing to remove noise and connect broken digits

(3) Using dilation to connect digits of each numbers (choose a suitable kernel in size)

(4) Find contours and bounding boxes (using methods in Contour features)

**HOMEWORK 4**

1. Improve the result of each exercise 4.2-4.3
2. Explore some “Hand detection and finger counting” applications
3. <https://madhavmishra121.medium.com/?p=5b594704eb08>
4. <https://github.com/kostasthanos/Hand-Detection-and-Finger-Counting>