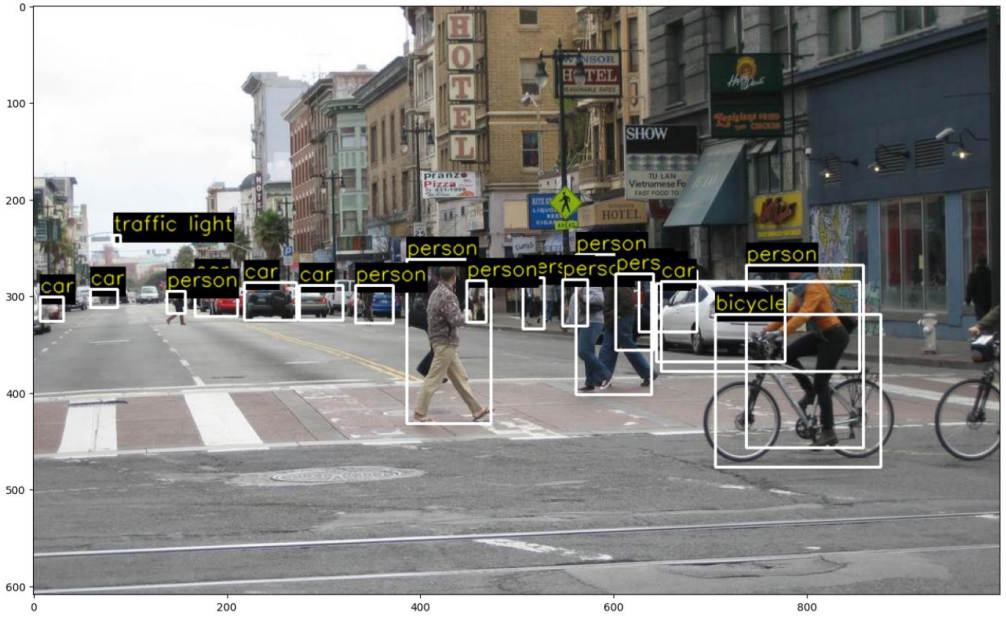
Mechatronics

Day 7-8
Computer Vision
with OpenCV Python

What is OpenCV?

Open-source Computer Vision

1. Image and VDO Processing
2. Object and Face Detections
3. Motion Analysis and Object Tracking
4. Image Segmentation
5. Image Recognition
6. Machine Vision


HOTEL
SHOW
traffic light
person
car
bicycle
0
100
200
300
400
500
600
0
200
400
600
800

Necessary packages for OpenCV

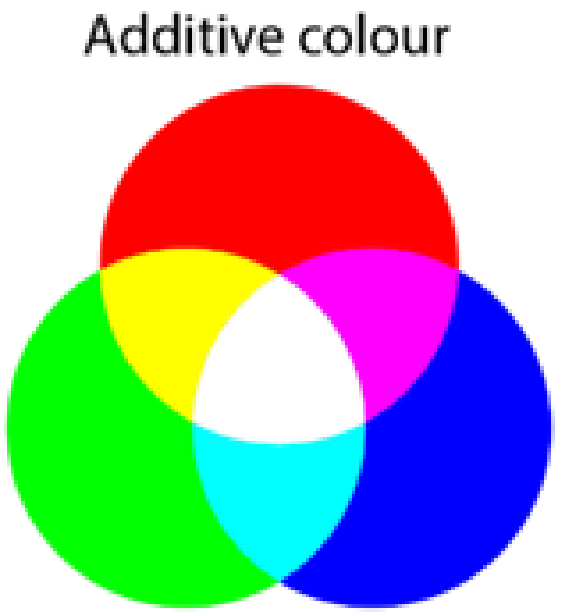
```
python --version
pip list
pip install opencv-python
pip install matplotlib
```

```
(.venv) D:\Desktop\Python_test>pip list
Package         Version
--------------- --------
contourpy       1.2.0
cycler          0.12.1
fonttools       4.47.0
kiwisolver      1.4.5
matplotlib      3.8.2
numpy           1.26.2
opencv-python   4.8.1.78
packaging       23.2
Pillow          10.1.0
pip             23.3.2
pyparsing       3.1.1
python-dateutil 2.8.2
six             1.16.0
```

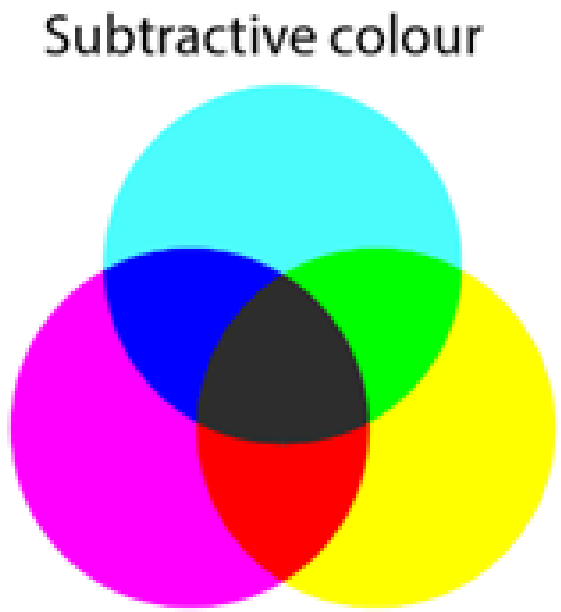
Import OpenCV

```
import cv2
import numpy as np

# Check OpenCV version
cv2.__version__
```

Color Systems



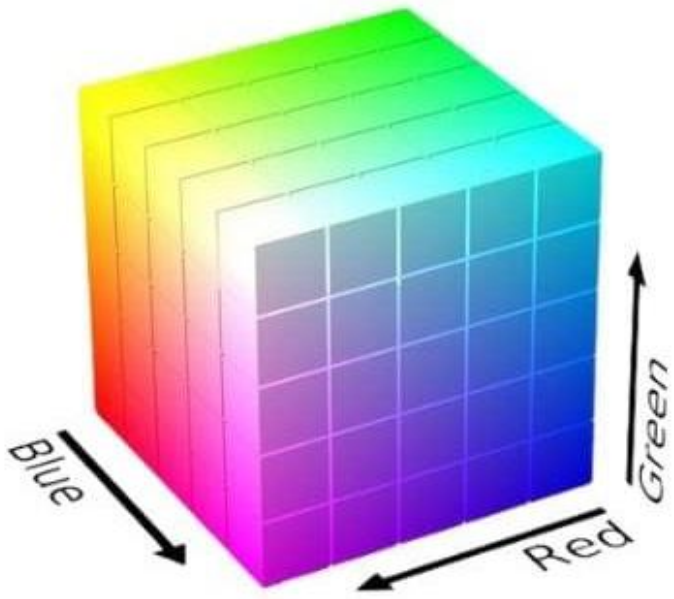
Red Green Blue (RGB)



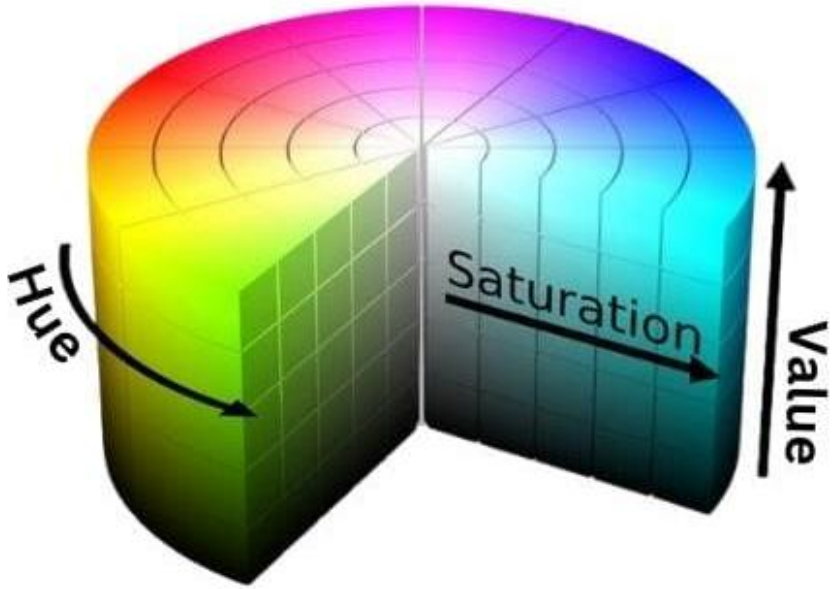
Cyan Magenta Yellow (CMY)
CMYK (K=Black)

Color Systems

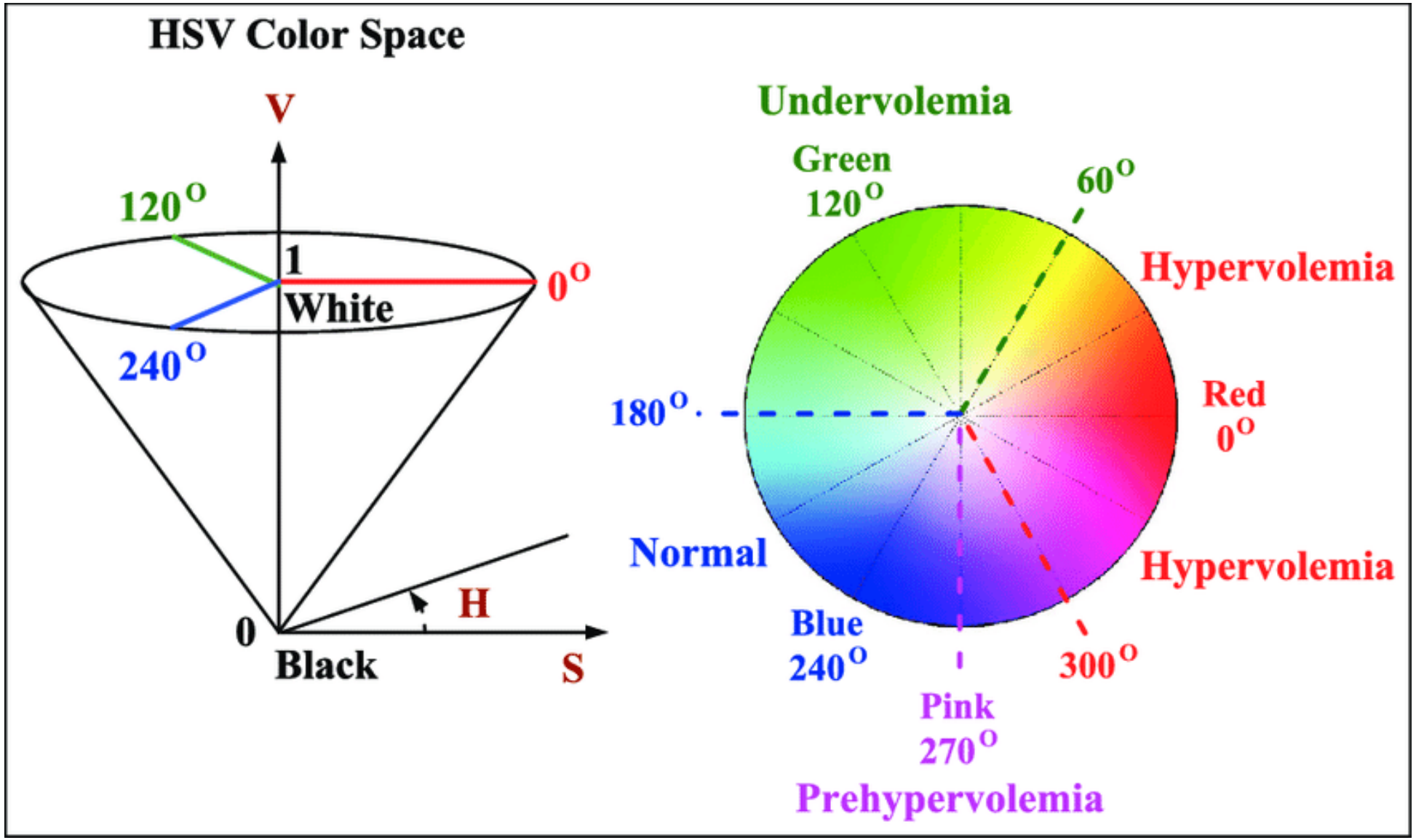
RGB



HSV



Hue-Saturation-Value



https://www.researchgate.net/figure/HSV-color-space-and-RGB-color-transformation_fig4_312678134

Pixel

Pixel = Picture + Element

- It is the smallest basic unit of an image.
- 1 pixel has only 1 color value.
- 2-dimensional pixel is image data (Picture or Image)

Image types

Binary Image

There are only 2 pixel value

0 = black

1 = white

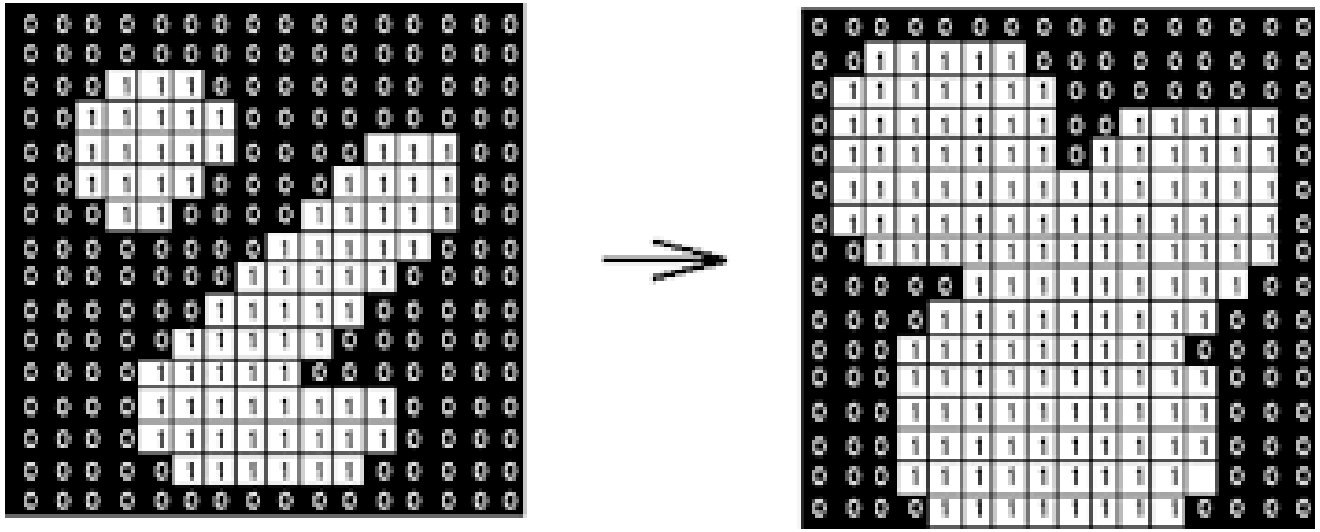
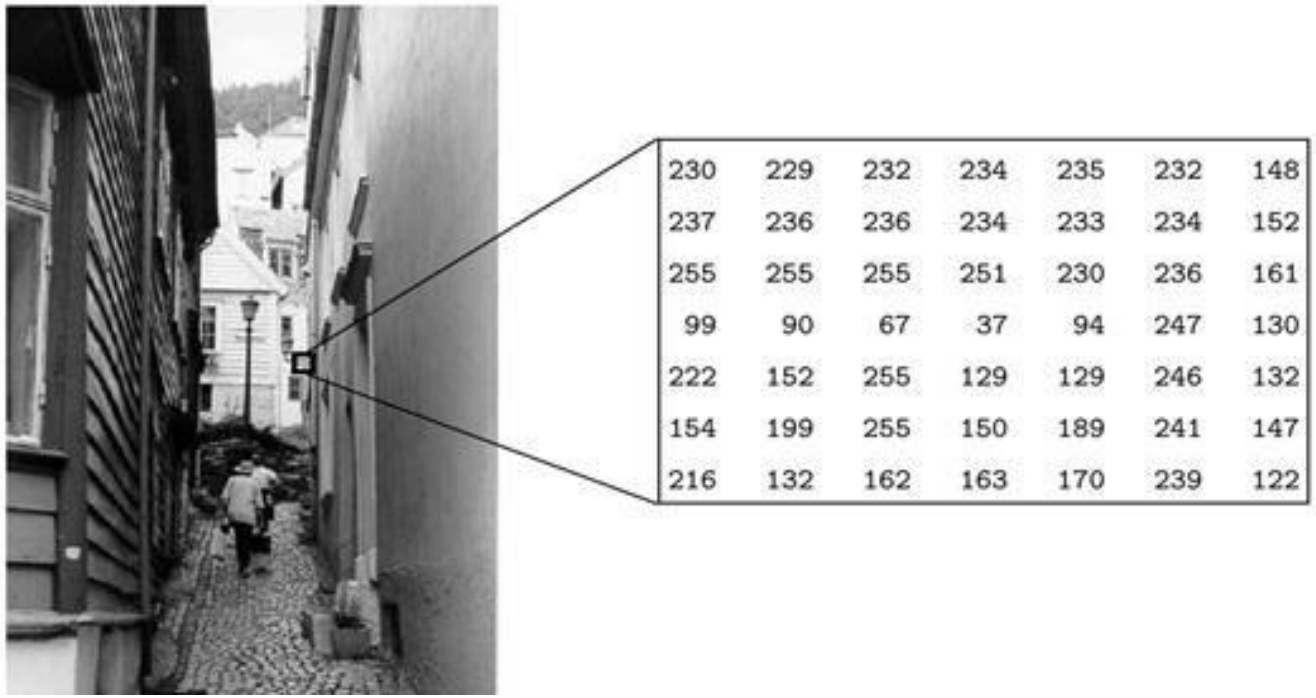


Image types

Grayscale Image

- Pixel value between 0 and 255
- 8-bit (2^8 = 256)

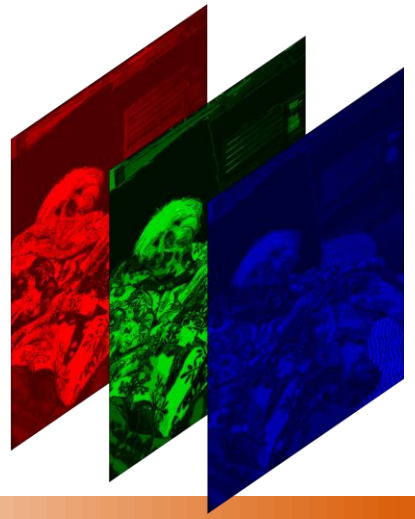


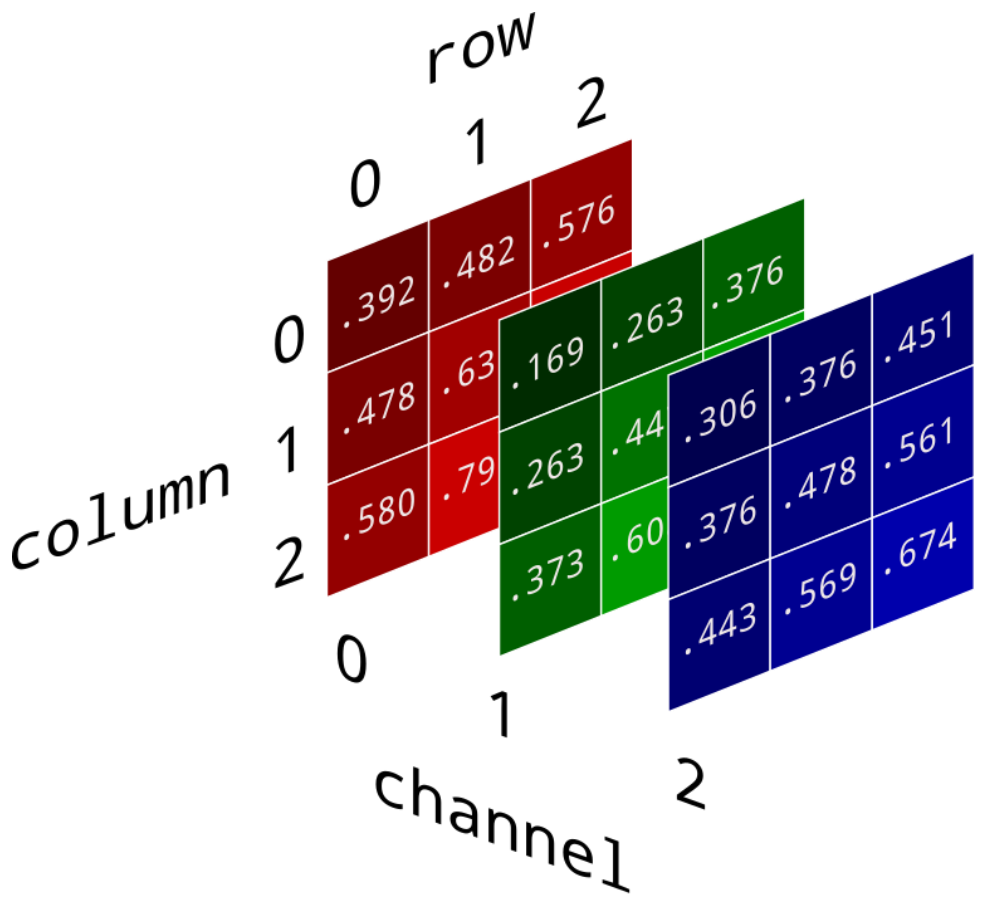
(https://www.researchgate.net/figure/Grayscale-Image-8-bit_fig3_312586031)

Image types

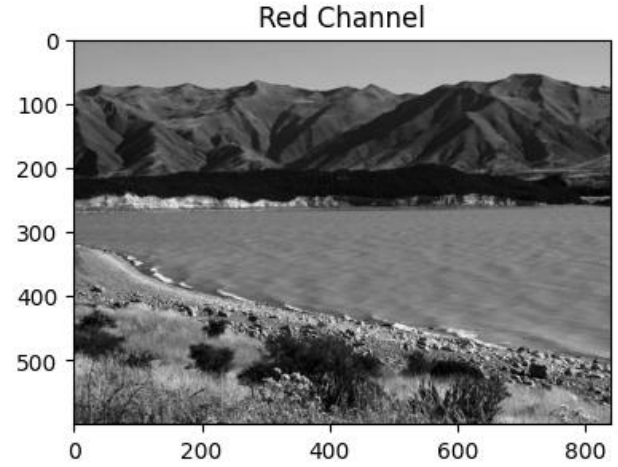
Color Image (RGB Image)

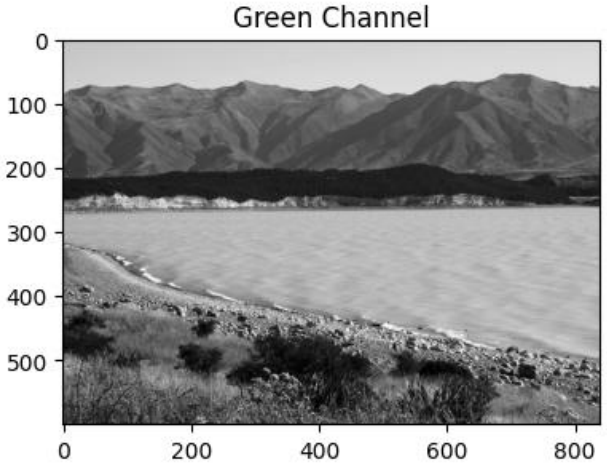
- Pixel value between 0 and 255
- There are 3 Channels : Red, Green, Blue
- 24-bit (2^24 = 16,777,216)

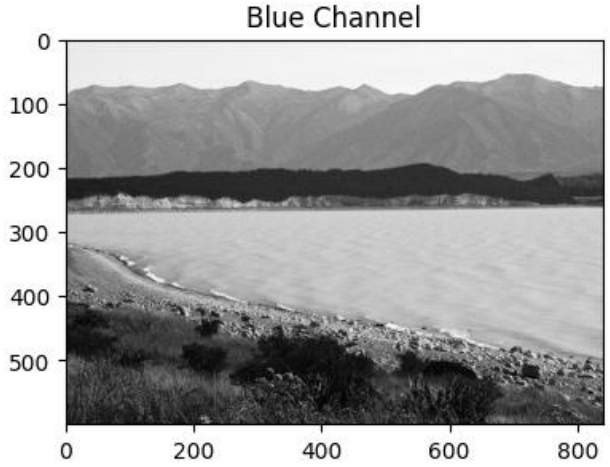


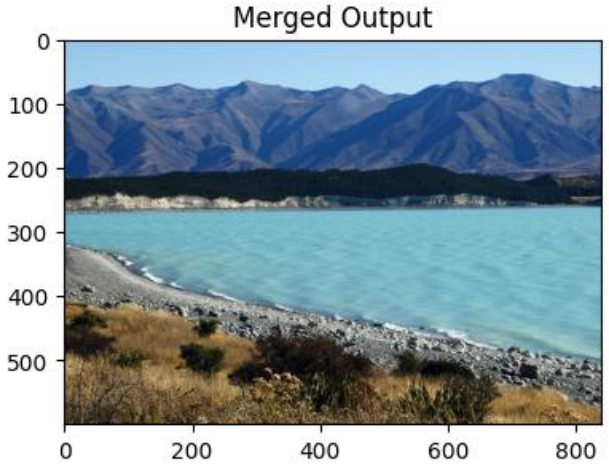


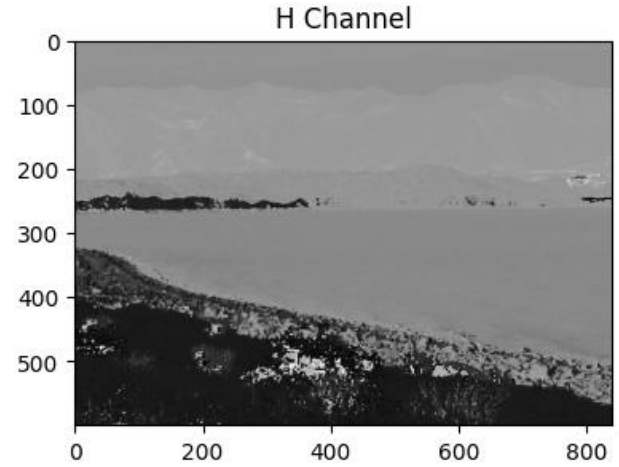
https://e2eml.school/convert_rgb_to_grayscale

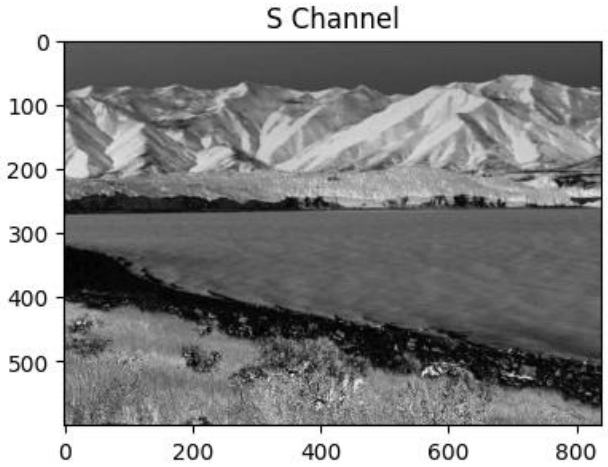

Red Channel
0
100
200
300
400
500
0
200
400
600
800

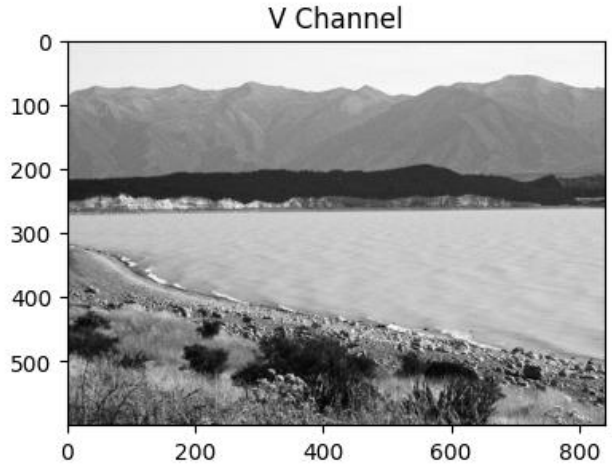

Green Channel
0
100
200
300
400
500
0
200
400
600
800


Blue Channel
0
100
200
300
400
500
0
200
400
600
800


Merged Output
0
100
200
300
400
500
0
200
400
600
800


H Channel
0
100
200
300
400
500
0
200
400
600
800


S Channel
0
100
200
300
400
500
0
200
400
600
800


V Channel
0
100
200
300
400
500
0
200
400
600
800

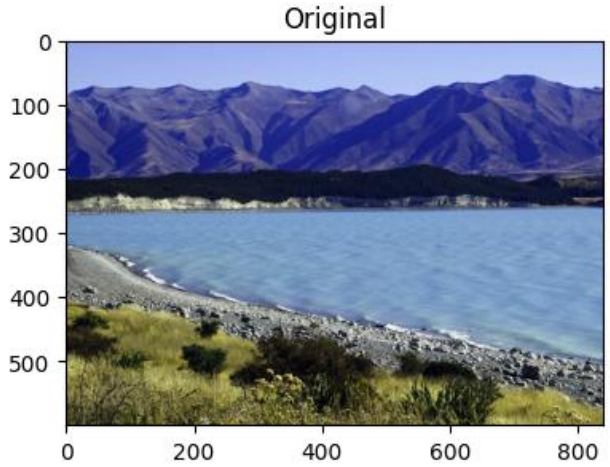
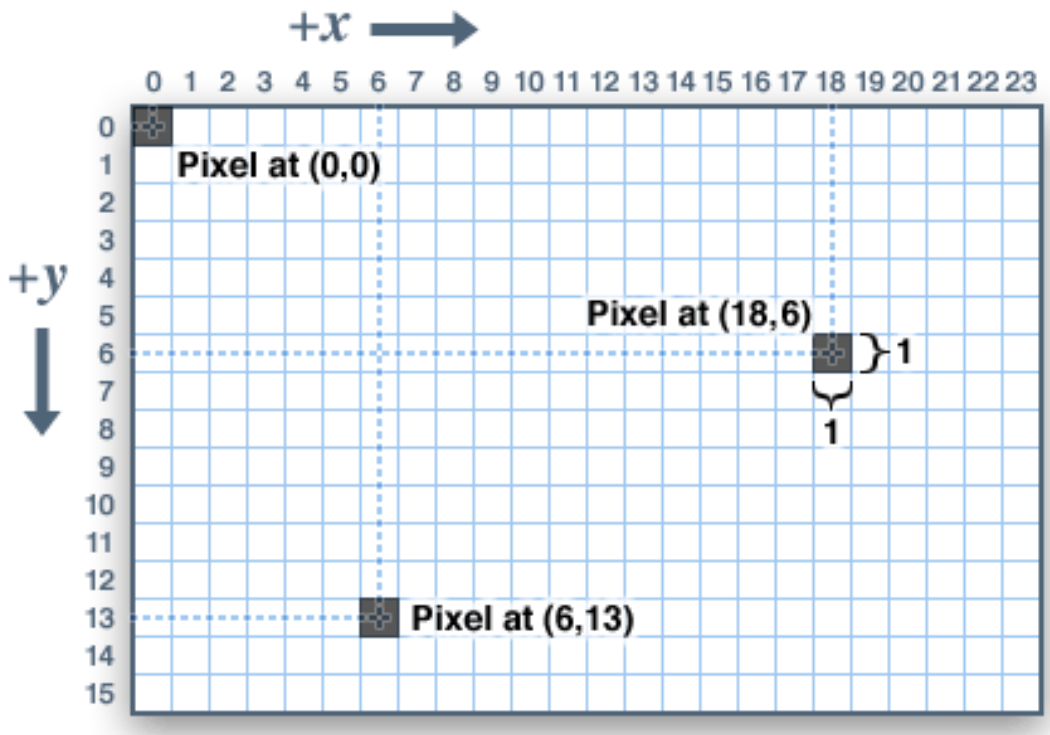
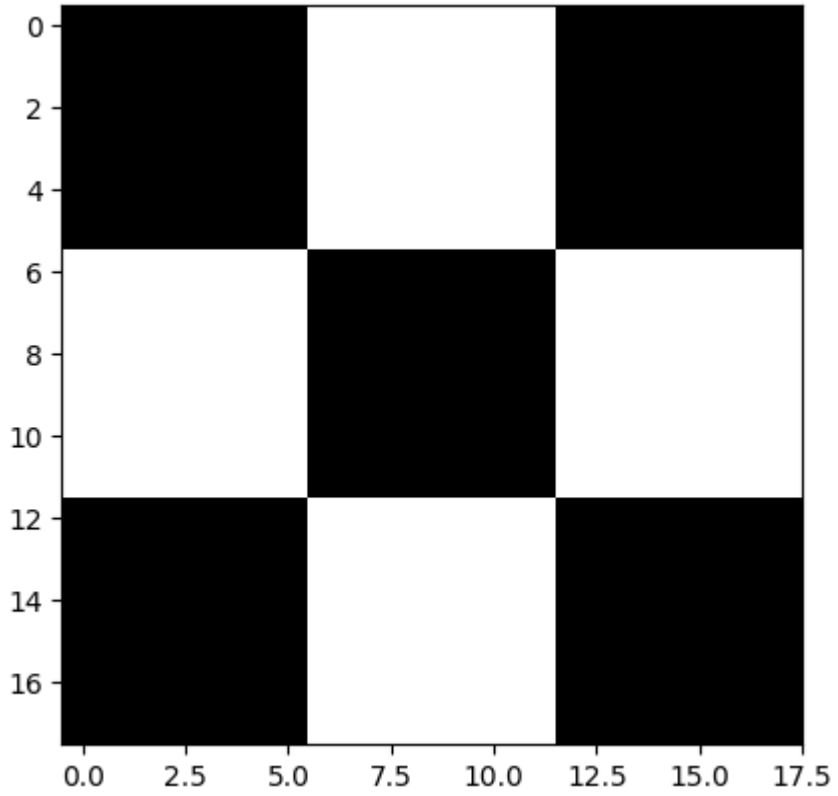

Original
0
100
200
300
400
500
0
200
400
600
800

Image Coordinate





1. Read Image

```python
# Read Image
img = cv2.imread("cat.jpg")

# Show image with numpy array
print(img)
```

```
[[[251 252 255]
  [251 252 255]
  [251 252 255]
  ...
  [251 252 255]
  [251 252 255]
  [251 252 255]]

 [[251 252 255]
  [251 252 255]
  [251 252 255]
  ...
  [251 252 255]
  [251 252 255]
  [251 252 255]]

 [[251 252 255]
  [251 252 255]
  [251 252 255]
  ...
  [251 252 255]
  [251 252 255]
```

1. Read Image

```
# Image Size (Height, Width, Channels)
print(img.shape)

# Image Dimension
print(img.ndim)

# Data Type
print(type(img.ndim)) # integer (int)
```

```
(480, 640, 3)
3
<class 'int'>
```

2.1 Show Image

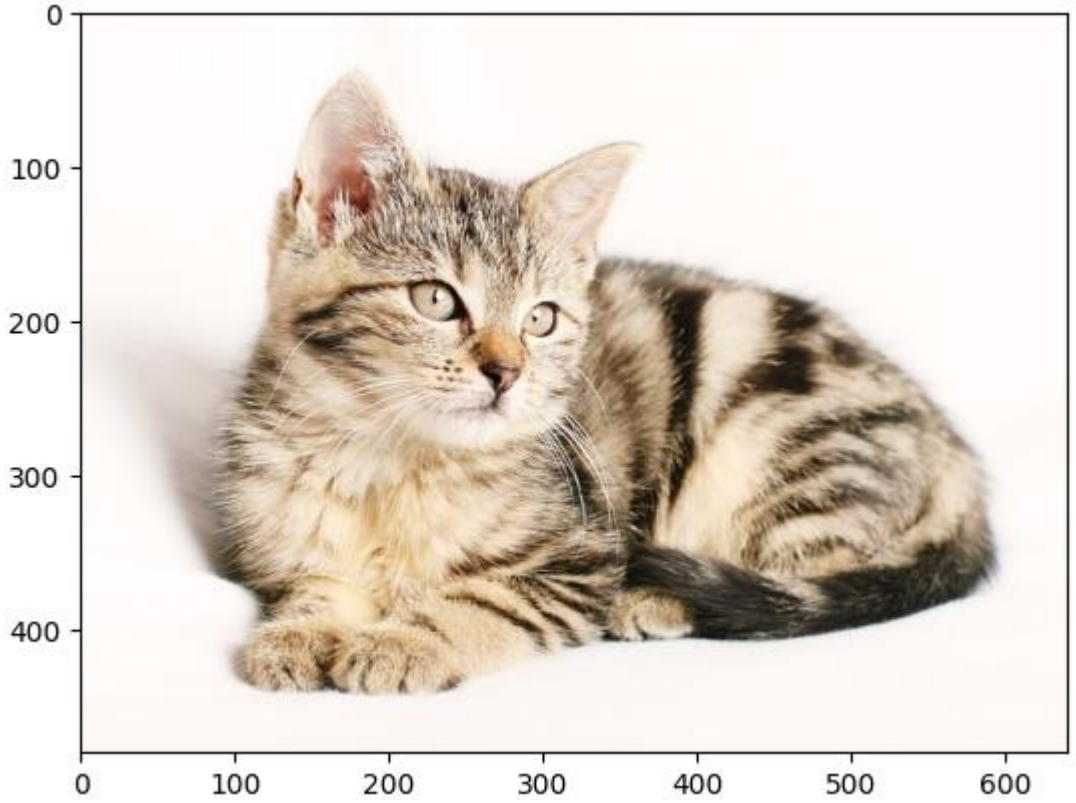
```
# Show Image
cv2.imshow("Show Image",img)

# pause the video at each frame until you press a key to proceed to the next frame.
cv2.waitKey(0)
cv2.destroyAllWindows()

# Show image in 5s (5000 ms) then close
#  cv2.waitKey(delay = 5000)
# cv2.destroyAllWindows()
```

2.1 Show Image

```
import cv2

img = cv2.imread("cat.jpg")

cv2.imshow("Show Cat Image",img)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

2.2 Read & Show Image (with Matplotlib)

```
import matplotlib.pyplot as plt

img = plt.imread('cat.jpg’)

plt.imshow(img)
plt.show()
```

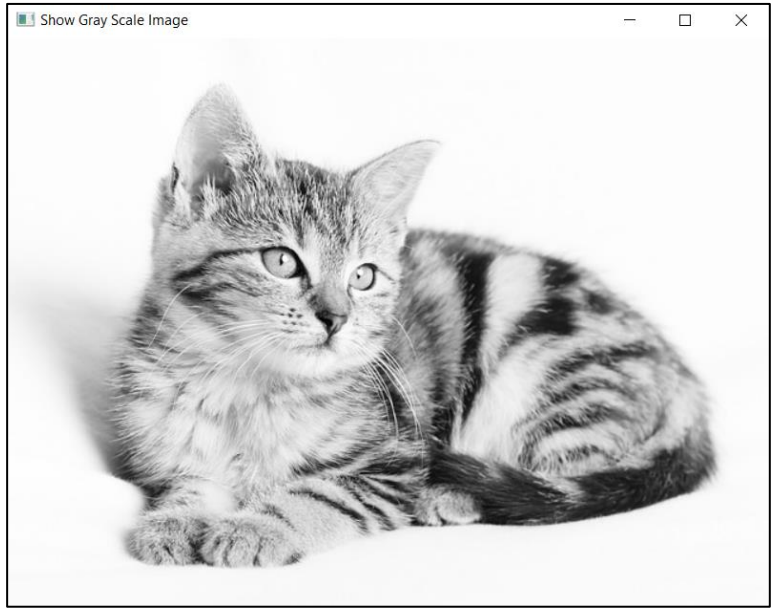


3. Resize Image

```
img_resize = cv2.resize(img,(400,200))
# Resize to w*h = 400*200 pixel
cv2.imshow("Show Resize Image",img_resize)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

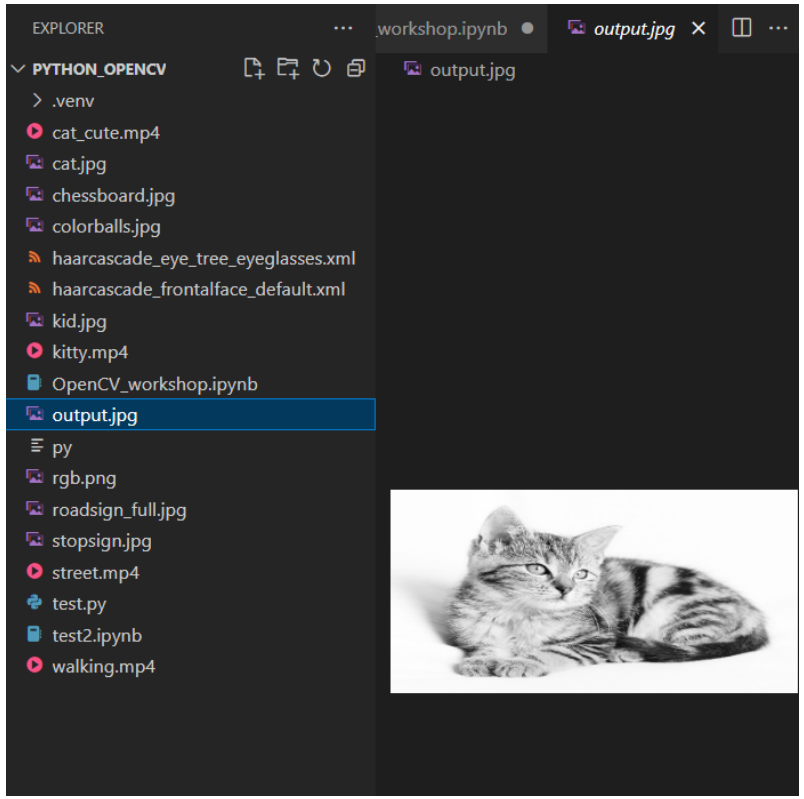
4. Convert color to Grayscale

```
img = cv2.imread("cat.jpg", 0)
# 0 = Gray Scale, 1 = Color

cv2.imshow("Show Gray Scale Image",img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



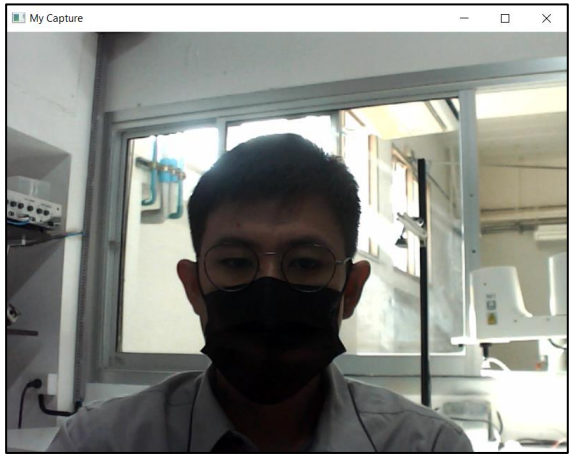
5. Export Image

```python
import cv2
import numpy as np
img = cv2.imread("cat.jpg", 0 )
img_resize = cv2.resize(img,(400,400))
cv2.imshow("Show Resize Image",img_resize)

# Export to Gray Scale – get new file in folder
cv2.imwrite("output_new.jpg", img_resize)

cv2.waitKey(0)
cv2.destroyAllWindows()
```



6. Access Webcam

```
import cv2

cap = cv2.VideoCapture(0) # if you have more camera, please change it to 1,2, ..
while (True):
    # Boolean (Check) if the camera is open.
    # Receive images from the camera continuously - frame-frame
    check, frame = cap.read()
    frame2 = cv2.flip(frame,1)        # flip image
     cv2.imshow("My Capture", frame2)
    # type ‘x’ from keyboard for close
    if cv2.waitKey(1) & 0xFF == ord("x"):
        break
cap.release()
cv2.destroyAllWindows()
```



7. Open Video

```
import cv2

cap = cv2.VideoCapture("shibuya.mp4")
while (cap.isOpened()):
    check , frame = cap.read()

    if check == True :
        cv2.imshow("Output", frame)
        if cv2.waitKey(1) & 0xFF == ord("e"):
            break
    else :
        break

cap.release()
cv2.destroyAllWindows()
```

https://docs.opencv.org/3.4/dd/d43/tutorial_py_video_display.html

8. Convert video to Grayscale

```
import cv2

cap = cv2.VideoCapture("street.mp4")
while (cap.isOpened()):
    check , frame = cap.read()
    if check == True :
# Convert RGB (BRG) to Gray scale
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        cv2.imshow("Output Gray color", gray)
        if cv2.waitKey(1) & 0xFF == ord("e"):
            break
    else :
        break
cap.release()
cv2.destroyAllWindows()
```

https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html

9. Video recorder

```
cv2.VideoWriter('file name', format, frame rate, size (w, h))
```

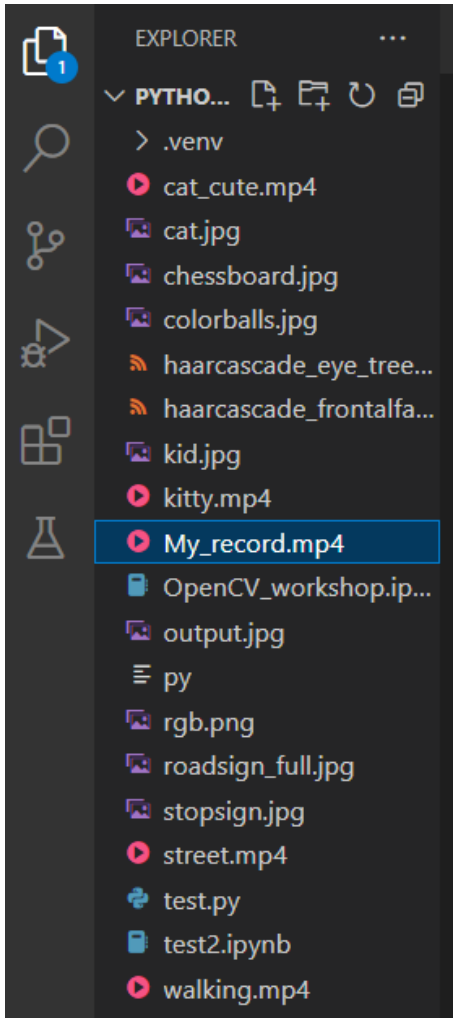
- FourCC code is the file format obtained from the compression: .avi, MJPG (.mp4), DIVX (.avi), and X264 (.mkv).
- XVID is more preferable. MJPG results in high size video. X264 gives very small size video.

https://docs.opencv.org/3.4/dd/d9e/classcv_1_1VideoWriter.html

9. Video recorder

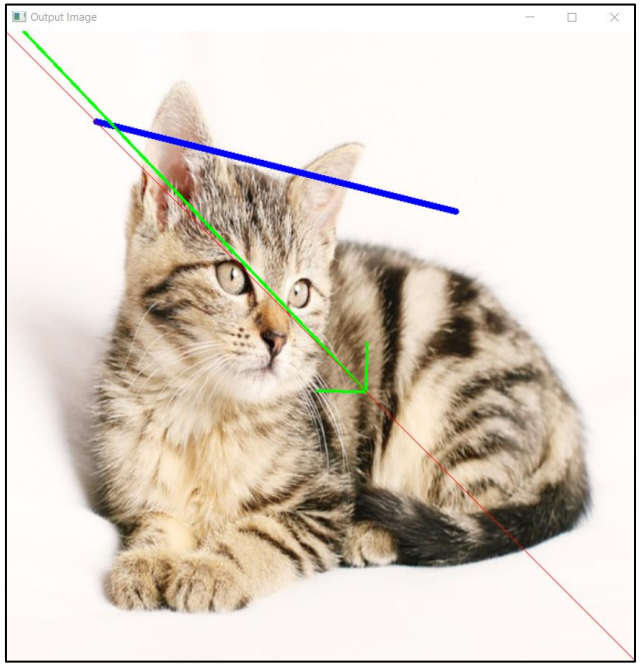
```python
import cv2
cap = cv2.VideoCapture(0)
fourcc = cv2.VideoWriter_fourcc(*'MJPG')
result = cv2.VideoWriter('My_record.mp4', fourcc, 20.0, (640,  480))
while (cap.isOpened()):
    check, frame = cap.read()
    if check == True :
        cv2.imshow("Record VDO",frame)
        result.write(frame)
        if cv2.waitKey(1) & 0xFF == ord("e"):
            break
result.release()
cap.release()
cv2.destroyAllWindows()
```

Frame Rate: 20.0 and VDO size: 640*480



10. Draw: Line and Arrow line

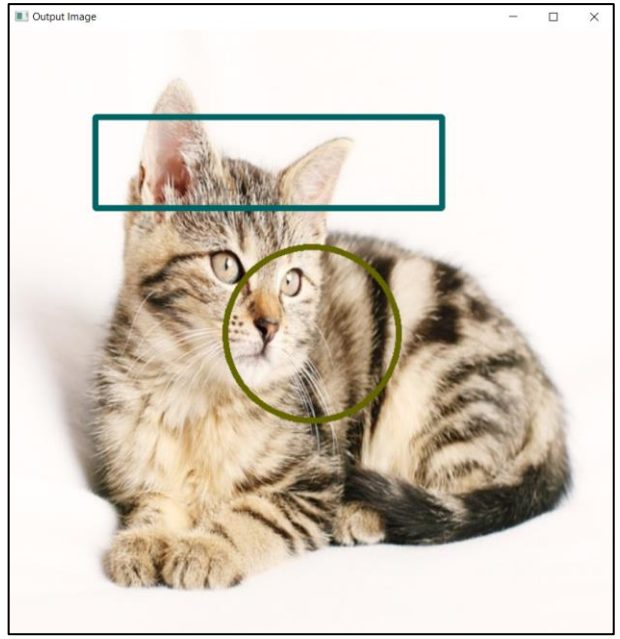
```
import cv2

img = cv2.imread("cat.jpg")
img_resize = cv2.resize(img,(700,700))
# draw line
# cv2.line(image, start(x,y), end (x,y), color (B,G,R), line width)
cv2.line(img_resize, (100,100), (500,200), (255,0,0), 5 )
# draw arrow Line
cv2.arrowedLine(img_resize, (20,0), (400,400), (0,255,0), 2 )
cv2.arrowedLine(img_resize, (0,0), (700,700), (0,0,255), 1 )

cv2.imshow("Output Image",img_resize)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



https://docs.opencv.org/3.4/dc/da5/tutorial_py_drawing_functions.html

11. Draw: Rectangle and Circle

```
import cv2

img = cv2.imread("cat.jpg")
img_resize = cv2.resize(img,(700,700))

# rectangle(image, upper left (x,y), lower right (x,y), color (B,G,R), line width)
cv2.rectangle(img_resize, (100,100), (500,205), (100,100,0), 5 )

# circle(image, center (x,y), radius, color (B,R,G), line width)
cv2.circle(img_resize, (350,350), 100, (0,100,100), 5 )

cv2.imshow("Output Image",img_resize)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



https://docs.opencv.org/3.4/dc/da5/tutorial_py_drawing_functions.html

12. Put Text in Image

```
import cv2

img = cv2.imread("cat.jpg")
img_resize = cv2.resize(img,(700,700))

# putText(Image, “Text”, Start (x,y), Font, Font size, Color (B,G,R), line types)
cv2.putText(img_resize, "Cat", (350, 350), cv2.FONT_HERSHEY_SIMPLEX, 1.5, (0,0,255),
            cv2.LINE_4)

cv2.imshow("Output Image",img_resize)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- Fonts : HERSHEY_SIMPLEX, HERSHEY_PLAIN, FONT_ITALIC etc.
- Line Types : LINE_4, LINE_8, LINE_AA etc

https://docs.opencv.org/3.4/d0/de1/group__core.html#ga0f9314ea6e35f99bb23f29567fc16e11

13.1 Put Text in VDO


```
import cv2
import datetime # Import datetime Library

cap = cv2.VideoCapture("shibuya.mp4")
while (cap.isOpened()):
    check , frame = cap.read()
    if check == True :
        cv2.putText(frame, "Shibuya", (10,30),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0,0,0), cv2.LINE_4)
        cv2.imshow("Output", frame)
        if cv2.waitKey(1) & 0xFF == ord("e"):
            break
    else :
        break

cap.release()
cv2.destroyAllWindows()
```

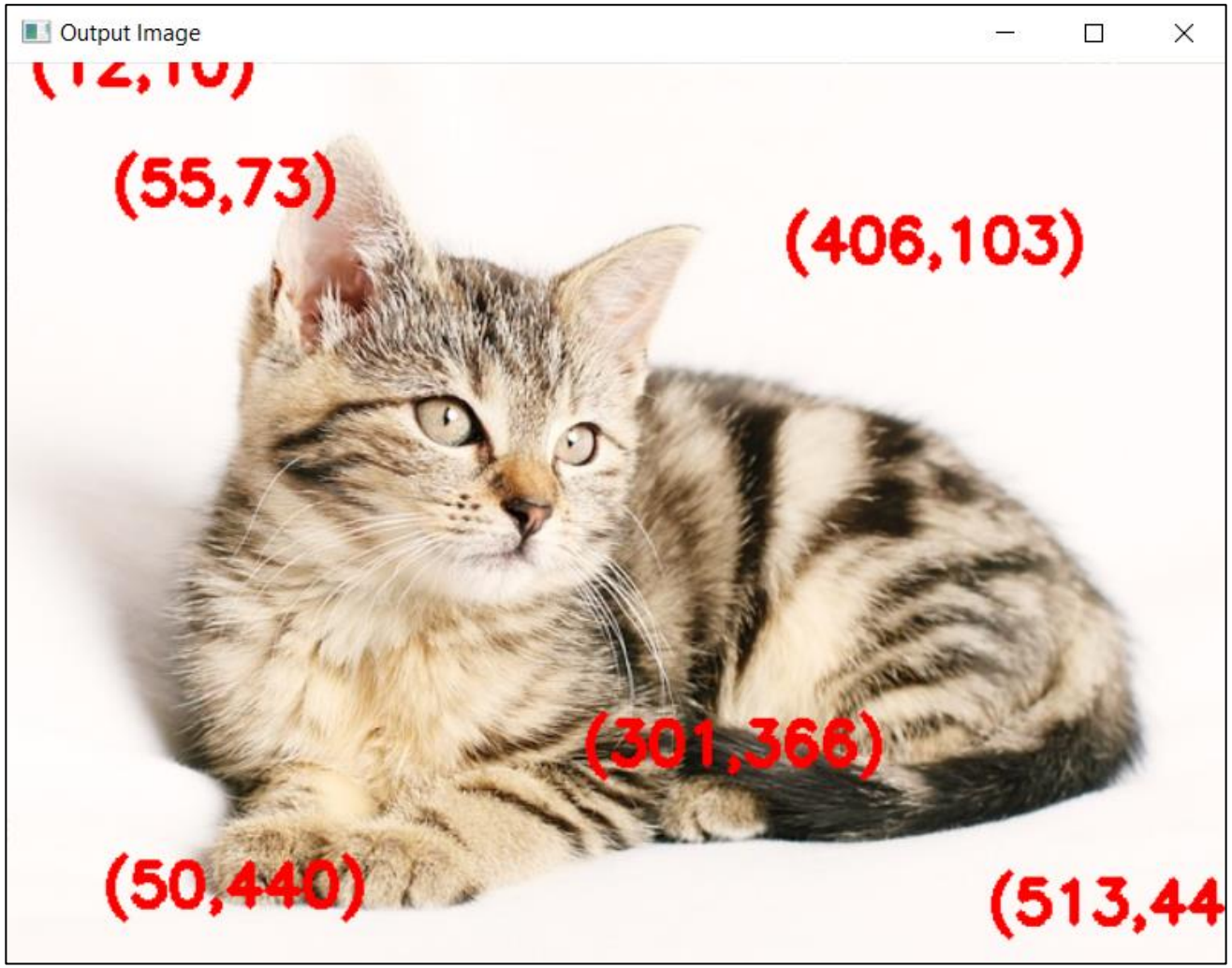
13.2 Put Text in VDO (show current time)

```
import cv2
import datetime
cap = cv2.VideoCapture("shibuya.mp4")
while (cap.isOpened()):
    check , frame = cap.read()
    if check == True :
        dt = str(datetime.datetime.now())
        cv2.putText(frame, "Shibuya", (10,30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,150,255),
                    cv2.LINE_4)
        cv2.putText(frame, dt, (10,60), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0,0,0), cv2.LINE_4)
        cv2.imshow("Output", frame)
        if cv2.waitKey(1) & 0xFF == ord("e"):
            break
    else :
        break
cap.release()
cv2.destroyAllWindows()
```

14. Put Text with Mouse Event

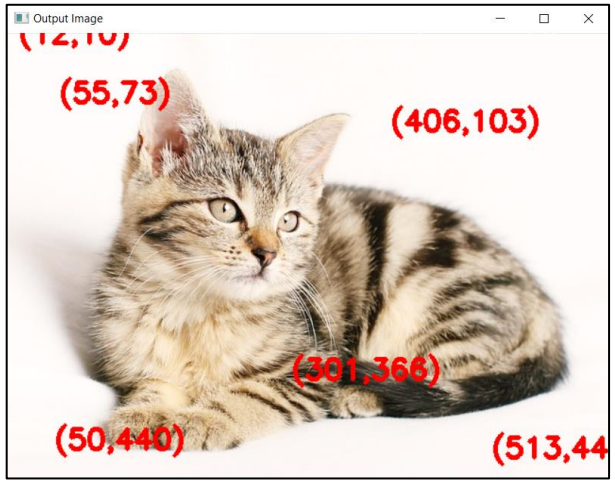
```
import cv2
img = cv2.imread("cat.jpg")

def clickPosition(event,x,y,flags,param):
    if event == cv2.EVENT_LBUTTONDOWN :
        cv2.putText(img,"X",(x,y), cv2.FONT_HERSHEY_SIMPLEX,1,(0,0,255),cv2.LINE_4)
        cv2.imshow("Output Image",img)

cv2.imshow("Output Image",img)

cv2.setMouseCallback("Output Image",clickPosition)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

15. Show coordinate with Mouse Event

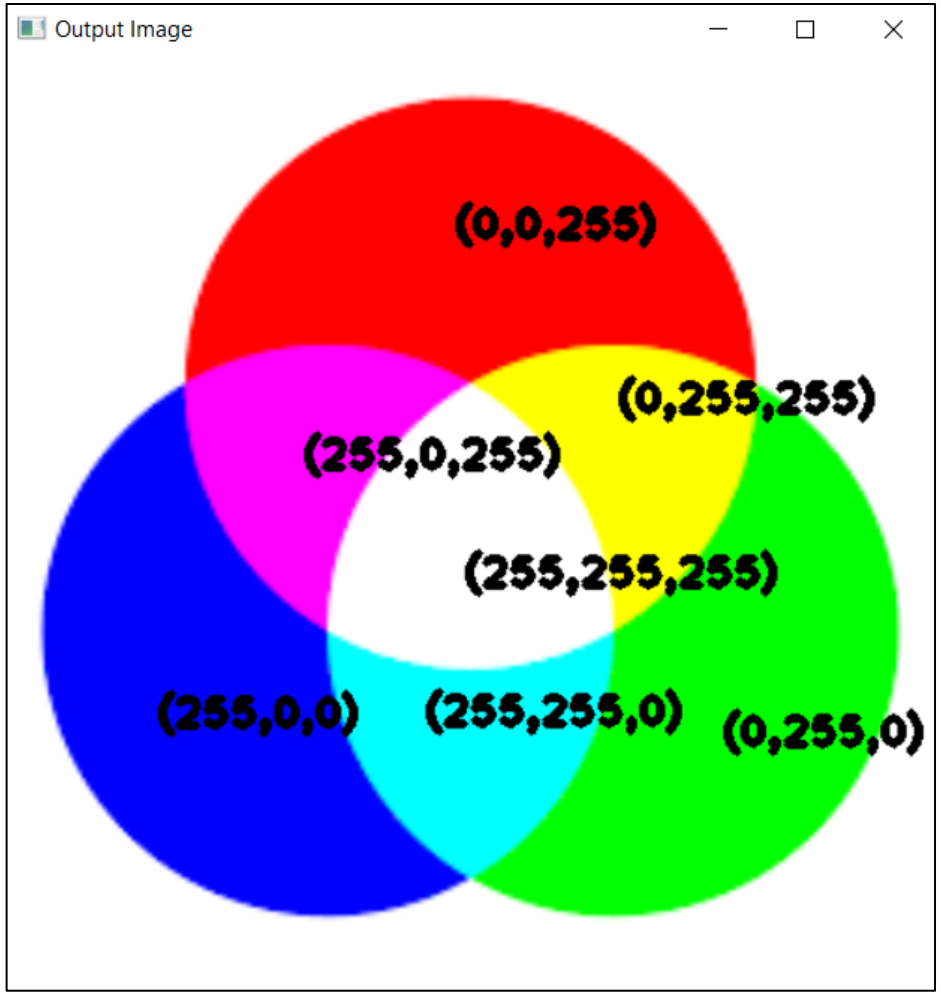


15. Show coordinate with Mouse Event

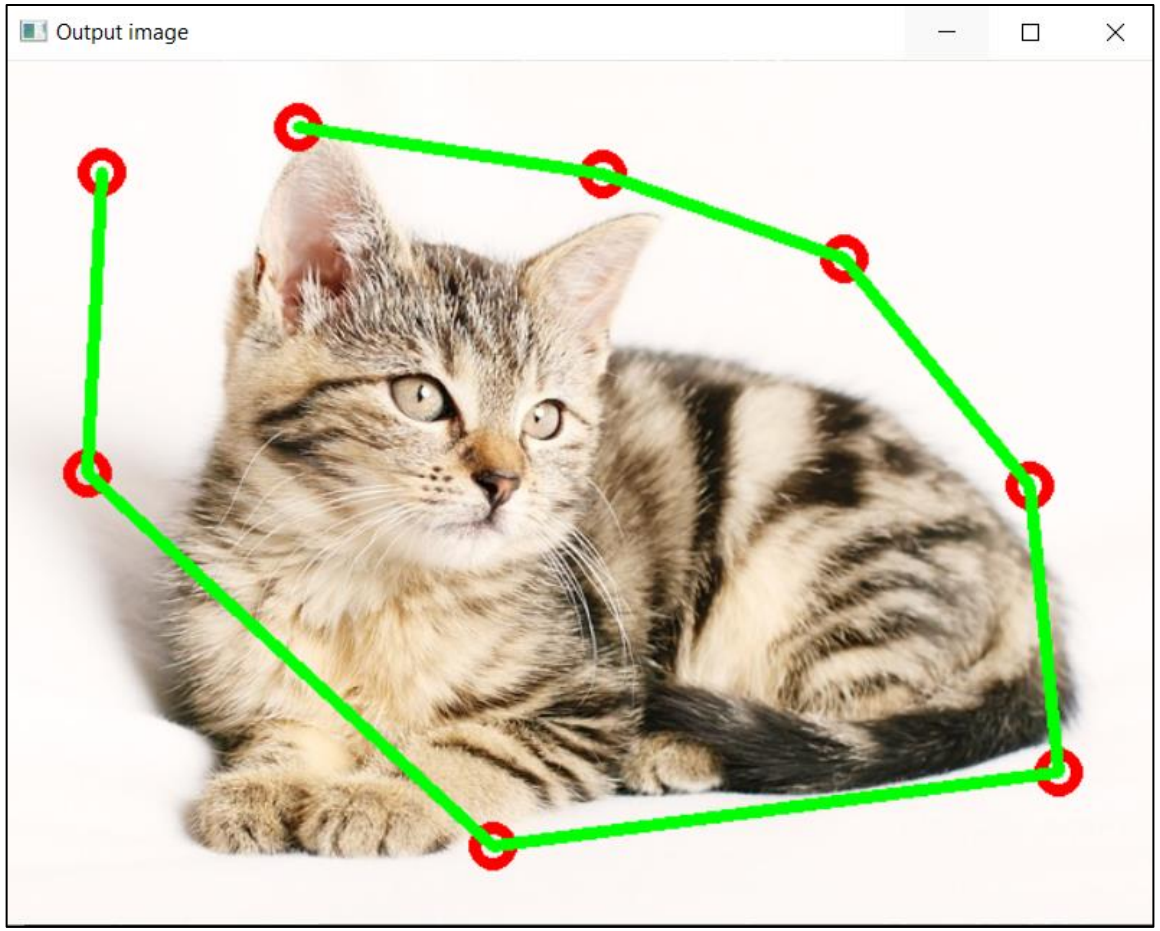
```
import cv2
img = cv2.imread("cat.jpg")

def clickPosition(event,x,y,flags,param):
   # show coordinate (x,y)
    if event == cv2.EVENT_LBUTTONDOWN :
         text = "("+str(x)+","+str(y)+")"
         cv2.putText(img,text,(x,y), cv2.FONT_HERSHEY_SIMPLEX,1,(0,0,255),cv2.LINE_4)
         cv2.imshow("Output Image",img)

cv2.imshow("Output Image",img)

cv2.setMouseCallback("Output Image",clickPosition)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Assignment 1 : Detect color value



16. Draw Connected Points



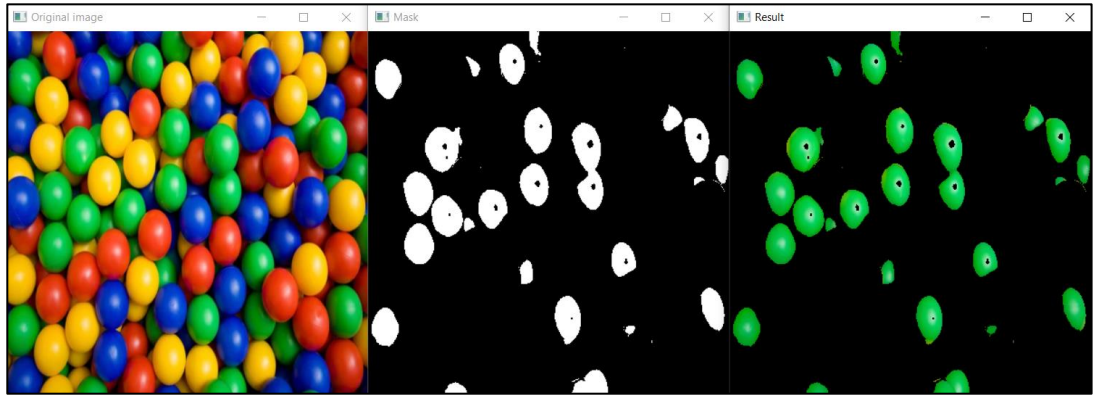
16. Draw Connected Points

```
import cv2
import numpy
# img = numpy.zeros([255,255,0]) # Create black background
img = cv2.imread("cat.jpg")
points =[]

def clickPosition(event,x,y,flags,param):
    if event == cv2.EVENT_LBUTTONDOWN :
    cv2.circle(img,(x,y),10,(0,0,255),5)
        points.append((x,y))
        print(points)
    if len(points) >= 2:
            cv2.line(img,points[-2],points[-1],(0,255,0),5)
   cv2.imshow("Output image", img)

cv2.imshow("Output image", img)
cv2.setMouseCallback("Output image",clickPosition)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

17. Create Mask for detect color

17.1 Create Mask for detect color (Image)

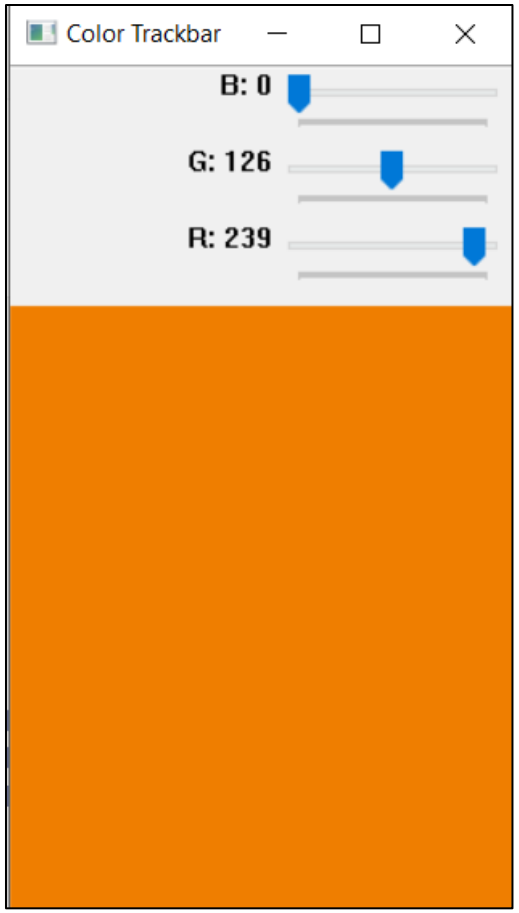
```
import cv2
import numpy

while True :
    img = cv2.imread("colorballs.jpg")
    img = cv2.resize(img,(400,400))

    lower = numpy.array([0,150,0])
    upper = numpy.array([200,255,150])

    mask = cv2.inRange(img, lower, upper)
    result = cv2.bitwise_and(img,img,mask=mask)
    if cv2.waitKey(0) & 0xFF == ord("e") :
        break
    cv2.imshow("Original image",img)
    cv2.imshow("Mask",mask)
    cv2.imshow("Result",result) # Green
cv2.destroyAllWindows()
```

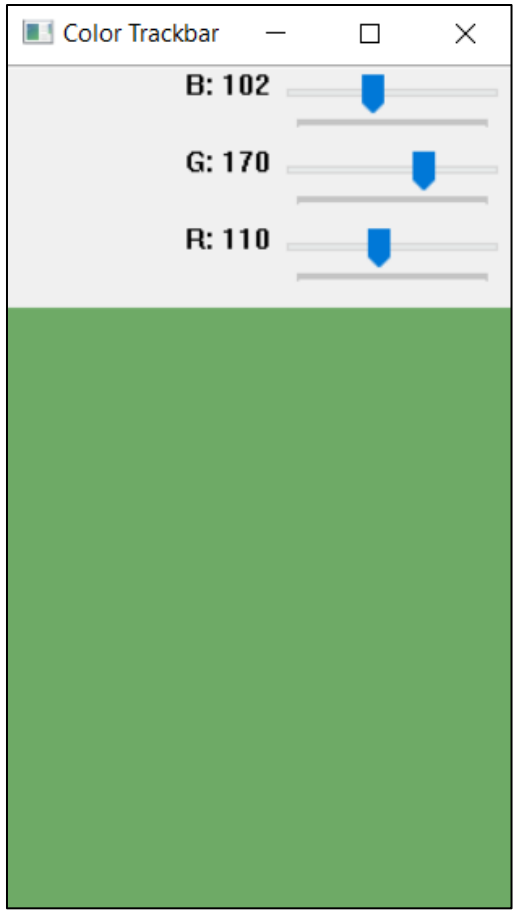


17.2 Create Mask for detect color (Webcam)

```
import cv2
import numpy as np

cap = cv2.VideoCapture(0)
while True:
    check, frame = cap.read()
    # width = int(cap.get(3))
    # height = int(cap.get(4))
    # hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    lower_blue = np.array([0, 150, 0])
    upper_blue = np.array([200, 255, 150])

    # mask = cv2.inRange(hsv, lower_blue, upper_blue)
    mask = cv2.inRange(frame, lower_blue, upper_blue)
    result = cv2.bitwise_and(frame, frame, mask = mask)

    cv2.imshow('frame', result)
    if cv2.waitKey(1) == ord('e'):
        break
cap.release()
cv2.destroyAllWindows()
```

18. GUI : Color Trackbar

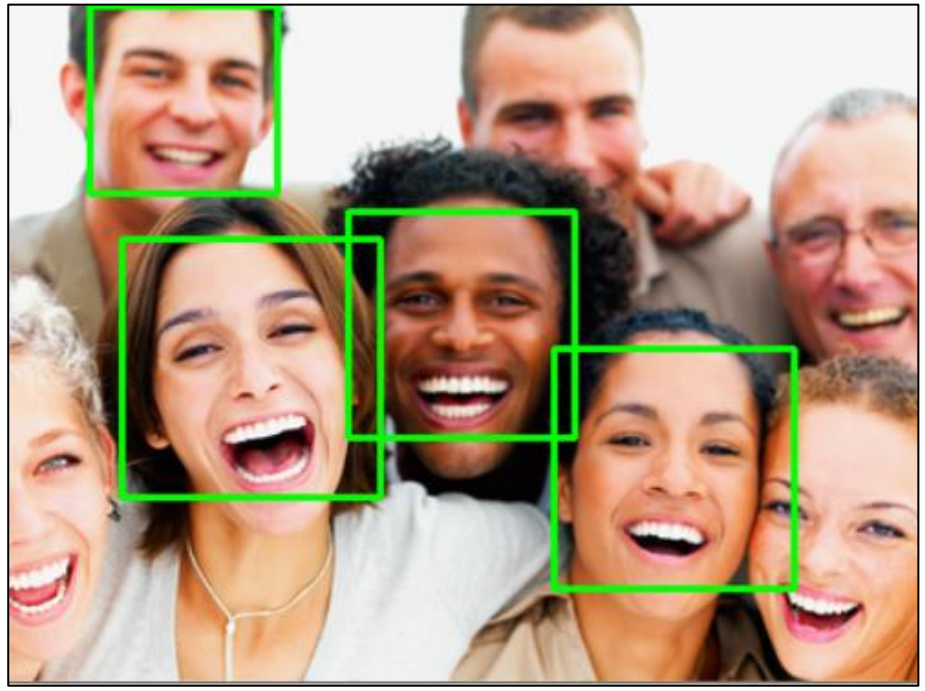




18. GUI : Color Trackbar

```
import cv2
import numpy as np

img = np.zeros((300,250,3), np.uint8)
cv2.namedWindow("Color Trackbar")
def display(value):
    print(value)

cv2.createTrackbar("B", "Color Trackbar", 0,255, display)
cv2.createTrackbar("G", "Color Trackbar", 0,255, display)
cv2.createTrackbar("R", "Color Trackbar", 0,255, display)
while True:
    cv2.imshow("Color Trackbar", img)
    if cv2.waitKey(1) & 0xFF ==ord("e"):
        break
    # Get value from Trackbar
    blue = cv2.getTrackbarPos("B","Color Trackbar")
    green = cv2.getTrackbarPos("G","Color Trackbar")
    red = cv2.getTrackbarPos("R","Color Trackbar")
    # Define value in 3D array
    img[:] = [blue,green,red]
cv2.destroyAllWindows()
```

19. Face and Eye Detection with haarcascades



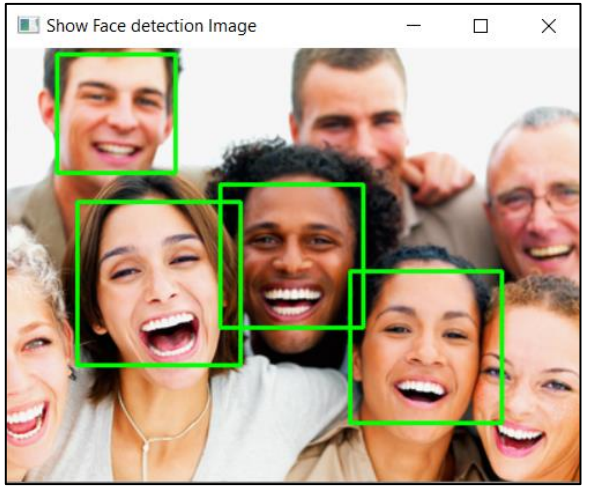
Face Detection

Eye Detection

https://github.com/opencv/opencv/tree/master/data/haarcascades

19.1 Face Detection

```
import cv2
img = cv2.imread("people.jpg")

# Read file for detection - Use xml file for face detection.
face_detection = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
# Convert to grayscale image before use.
gray_img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY) # Can be converted to other color systems such as HSV:
COLOR_BGR2HSV
# Set face detection parameters
scaleFactor = 1.1 # default =1.1 - Scale the image down to 11% before scanning.
minNeighbors = 3
# default =3 - Sets the thresholding value for examining the grayscale image to determine how many objects in a
# given area are closest to each other, i.e. 3, to be considered a human face.

face_detected = face_detection.detectMultiScale(gray_img, scaleFactor, minNeighbors)
# x,y are starting points, w,h are reading points width, heigh
for (x,y,w,h) in face_detected :
    cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),thickness=2)

cv2.imshow("Show Face detection Image",img)
#cv2.imshow("Show Gray Scale Image",gray_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



19.2 Eye Detection

```
import cv2

img = cv2.imread("people.jpg")

eye_detection = cv2.CascadeClassifier("haarcascade_eye_tree_eyeglasses.xml")
gray_img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
scaleFactor = 1.1
minNeighbors = 3
eye_detected = eye_detection.detectMultiScale(gray_img, scaleFactor, minNeighbors)
for (x,y,w,h) in eye_detected :
    cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),thickness=2)

cv2.imshow("Show Eyes detection Image",img)
#cv2.imshow("Show Gray Scale Image",gray_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

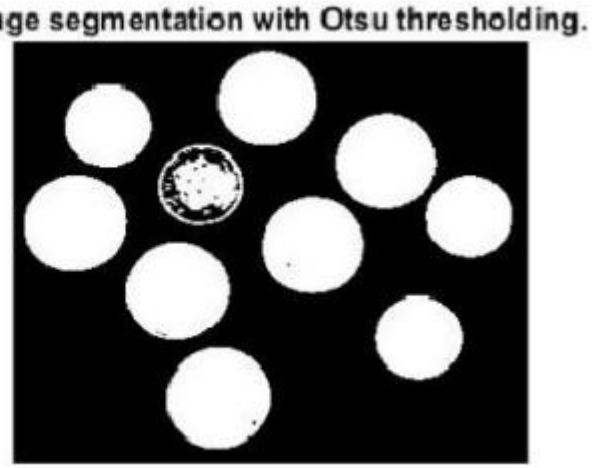
Assignment 2 : Real-time face detection

Build accurate real-time face detection

- Use Webcam
- Can detect many faces
- Draw green rectangle around each face
- Count how many face appear in current frame (if you can)

20. Thresholding

Thresholding is the process of separating the components of an image, or **image segmentation**, by converting a color image to a grayscale image and then converting it to binary image. Then, a **threshold value** is set to enhance the binary image to the desired clarity.



https://www.geeksforgeeks.org/thresholding-based-image-segmentation/

20. Thresholding

- Use GrayScale and Binary Image
- Gray Scale value : 0 (Black) – 255 (White)
- Binary Value : 0 (Black) – 1 (White) or 0 (Black) – 255 (White)

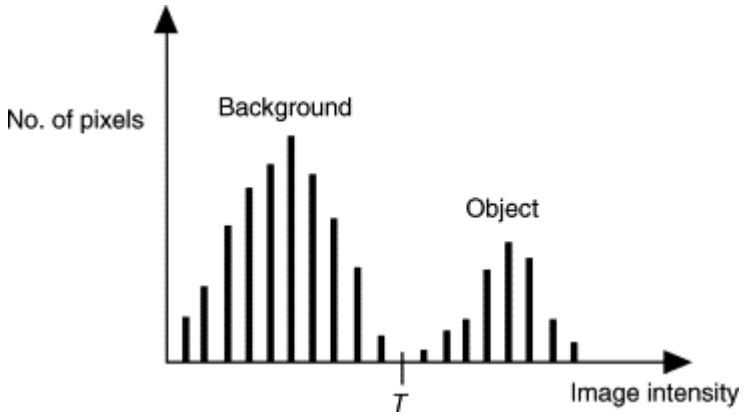
Color Image

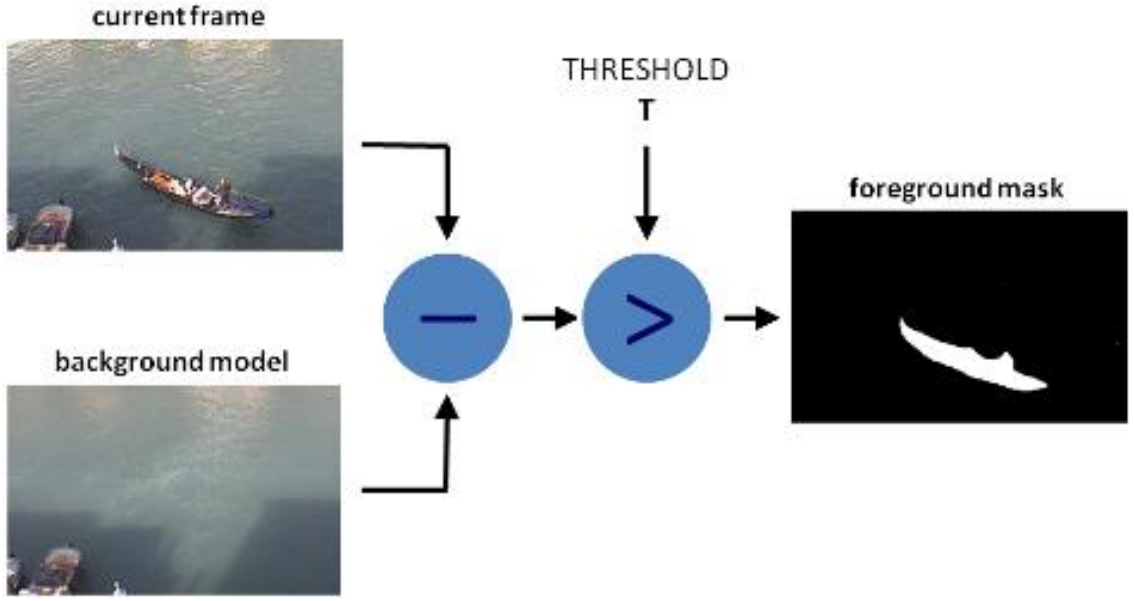
GrayScale

Binary

20.1 Foreground and Background

- Binary Image can separate image to Foreground and Back ground
- Foreground is interested object
- Background is area around object

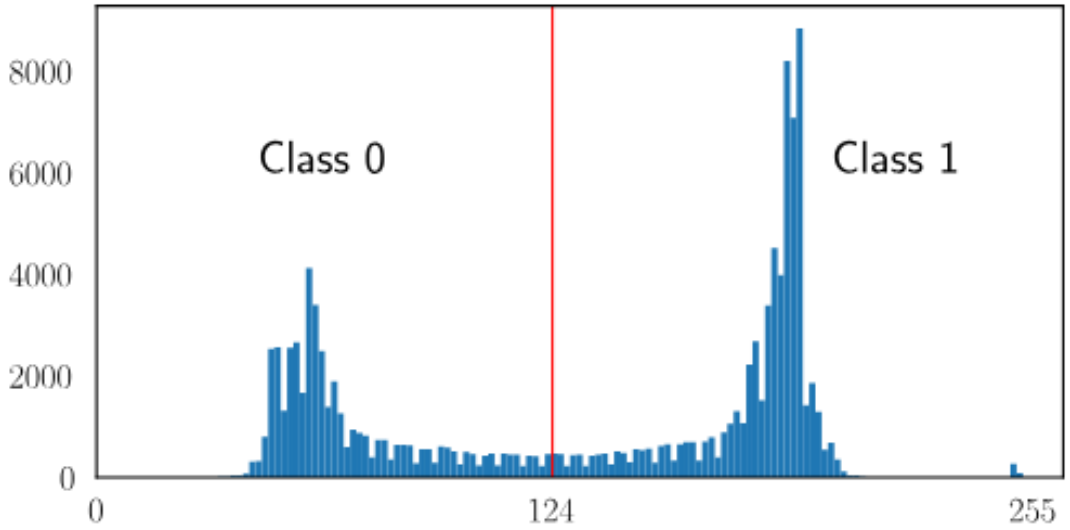




https://docs.opencv.org/4.x/d1/dc5/tutorial_background_subtraction.html

20.2 Thresholding method

- The threshold value is selected by selecting a value to convert the grayscale image to binary.
- A common midpoint used for division is 128, which should be an image with no noise and a uniform background.
- Values greater than the threshold value convert the point to white (1 or 255).
- Values less than the threshold value convert to black (0).



20.3 Thresholding types

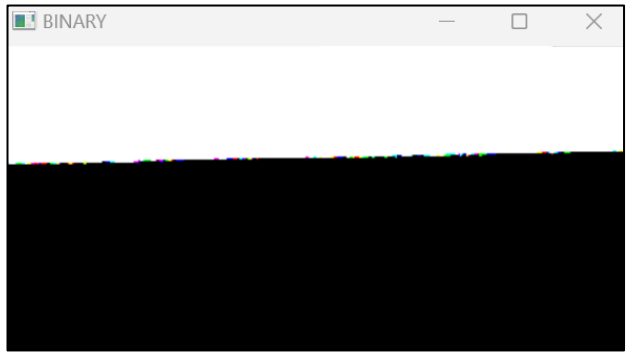
```
cv2.threshold(grayscale image, threshold value, max value, type)
```

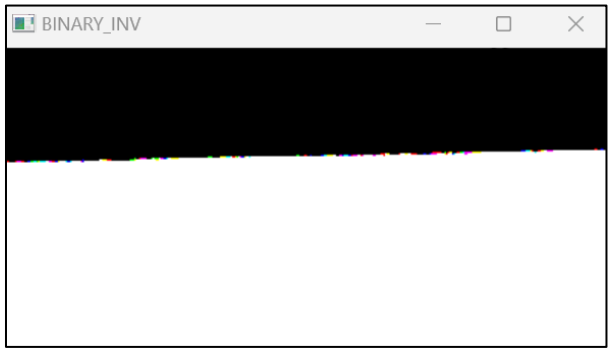
Thresholding type	Meaning
cv2.THRESH_BINARY	If the value is higher than the threshold, it will change to 255, otherwise it will be 0.
cv2.THRESH_BINARY_INV	If the value is higher than the threshold, it will change to 0, otherwise it will be 255.
cv2.THRESH_TRUNC	If the value is higher than the threshold, it will change to the value equal to the threshold value. Otherwise, the value remains the same.
cv2.THRESH_TOZERO	If the value is lower than the threshold, it will change to 0, otherwise the value remains the same.
cv2.THRESH_TOZERO_INV	If the value is higher than the threshold, it will change to 0, otherwise the value remains the same.

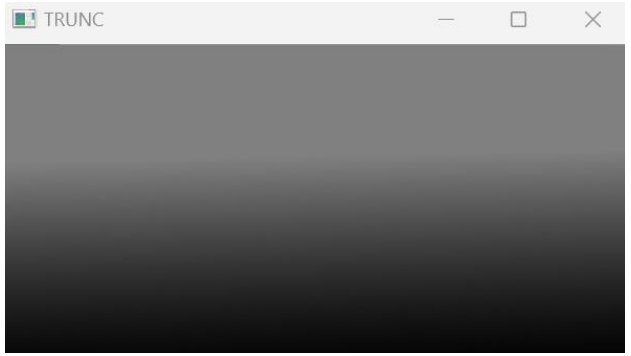
20.4 Thresholding (OpenCV)

```
import cv2
gray_img = cv2.imread("gradient.png")

thresh,th1 = cv2.threshold(gray_img,128,255,cv2.THRESH_BINARY)
thresh,th2 = cv2.threshold(gray_img,128,255,cv2.THRESH_BINARY_INV)
thresh,th3 = cv2.threshold(gray_img,128,255,cv2.THRESH_TRUNC)
thresh,th4 = cv2.threshold(gray_img,128,255,cv2.THRESH_TOZERO)
thresh,th5 = cv2.threshold(gray_img,128,255,cv2.THRESH_TOZERO_INV)

cv2.imshow("Orignal",gray_img)
cv2.imshow("BINARY",th1)
cv2.imshow("BINARY_INV",th2)
cv2.imshow("TRUNC",th3)
cv2.imshow("TOZERO",th4)
cv2.imshow("TOZERO_INV",th5)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

20.4 Thresholding (OpenCV)



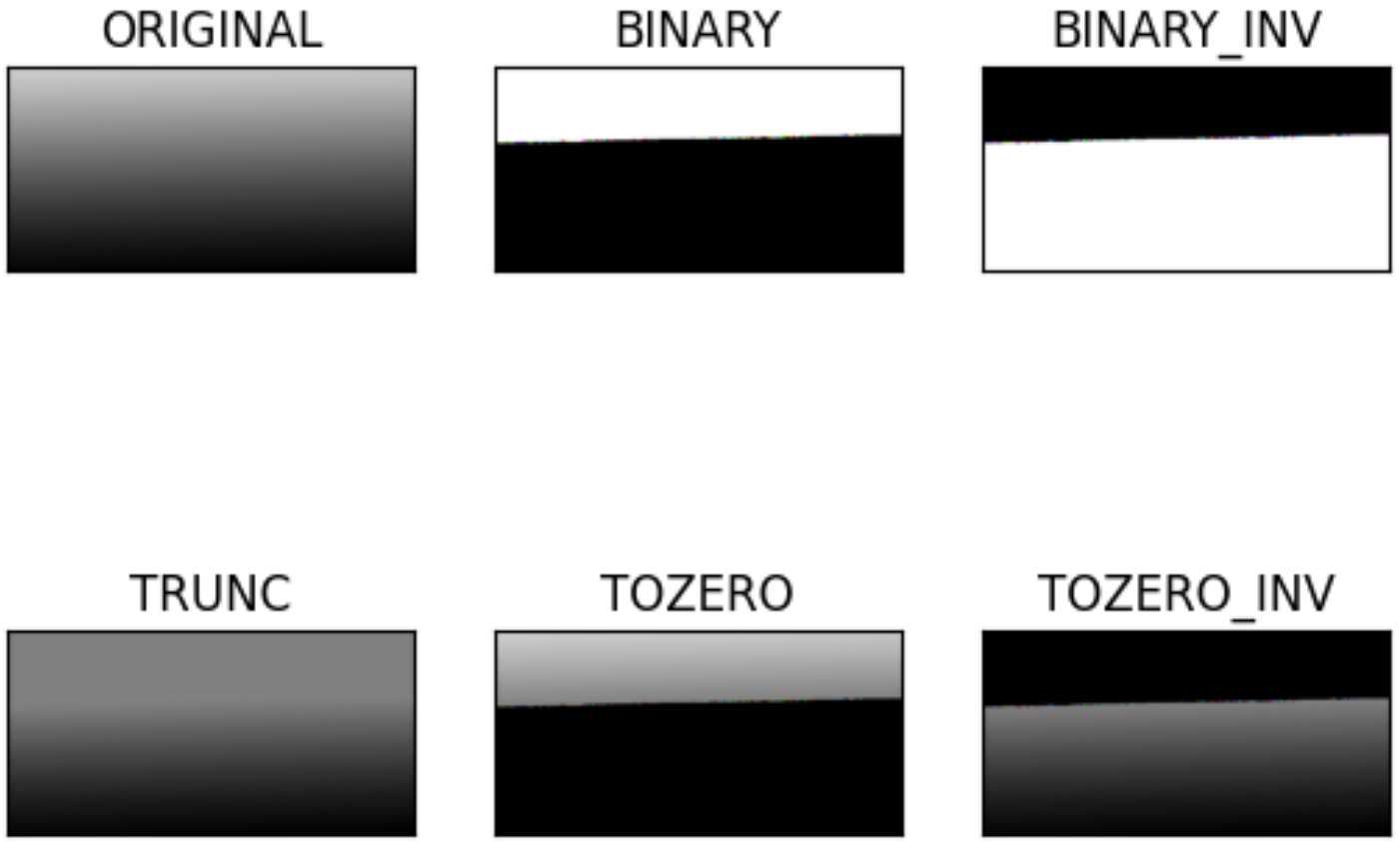




20.5 Thresholding (Matplotlib)

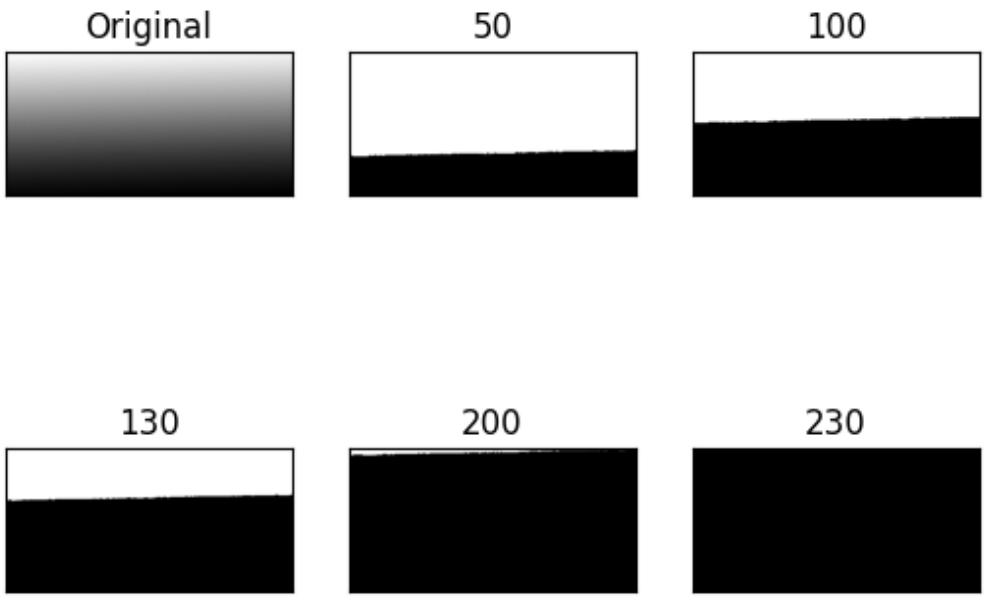
```
import cv2
import matplotlib.pyplot as plt
gray_img = cv2.imread("image/gradient.png")
thresh,th1 = cv2.threshold(gray_img,128,255,cv2.THRESH_BINARY)
thresh,th2 = cv2.threshold(gray_img,128,255,cv2.THRESH_BINARY_INV)
thresh,th3 = cv2.threshold(gray_img,128,255,cv2.THRESH_TRUNC)
thresh,th4 = cv2.threshold(gray_img,128,255,cv2.THRESH_TOZERO)
thresh,th5 = cv2.threshold(gray_img,128,255,cv2.THRESH_TOZERO_INV)
images = [gray_img,th1,th2,th3,th4,th5]
titles = ["ORIGINAL","BINARY","BINARY_INV","TRUNC","TOZERO","TOZERO_INV"]
for i in range(len(images)):
    plt.subplot(2,3,i+1)
    plt.imshow(images[i])
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

20.5 Thresholding (Matplotlib)



20.6 Compare Threshold values (Matplotlib)

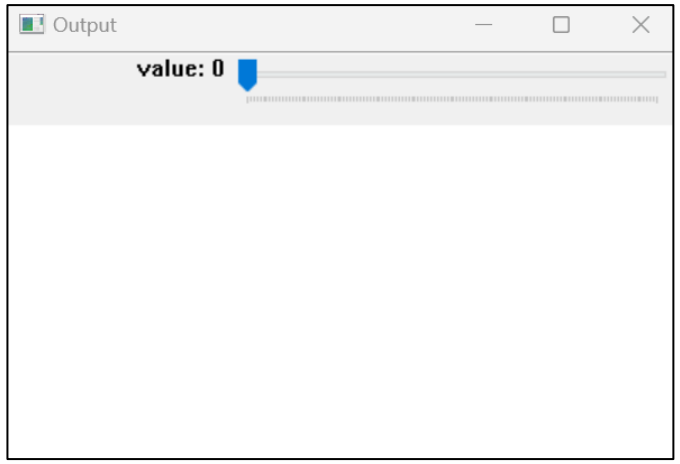
```
import cv2
import matplotlib.pyplot as plt

img = cv2.imread("gradient.png")
gray_img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

thresh_value = [50,100,130,200,230]
plt.subplot(231,xticks=[],yticks=[])
plt.title("Original")
plt.imshow(gray_img,cmap="gray")

for i in range(len(thresh_value)):
    thresh,result = cv2.threshold(gray_img,thresh_value[i],
                                  255,cv2.THRESH_BINARY)
    plt.subplot(232+i)
    plt.title("%d"%thresh_value[i])
    plt.imshow(result,cmap="gray")
    plt.xticks([]),plt.yticks([])

plt.show()
```

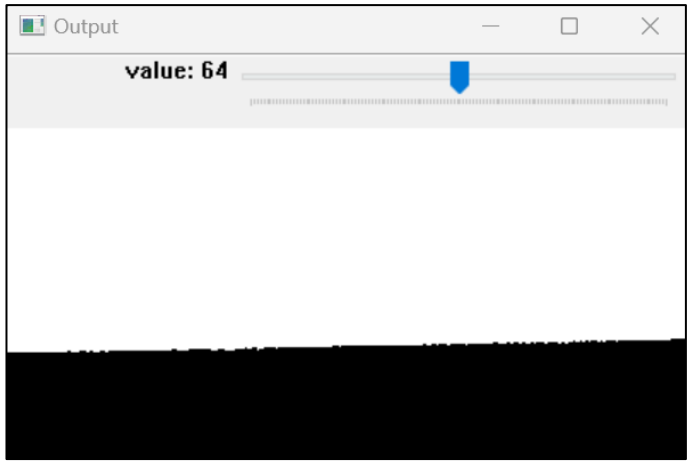


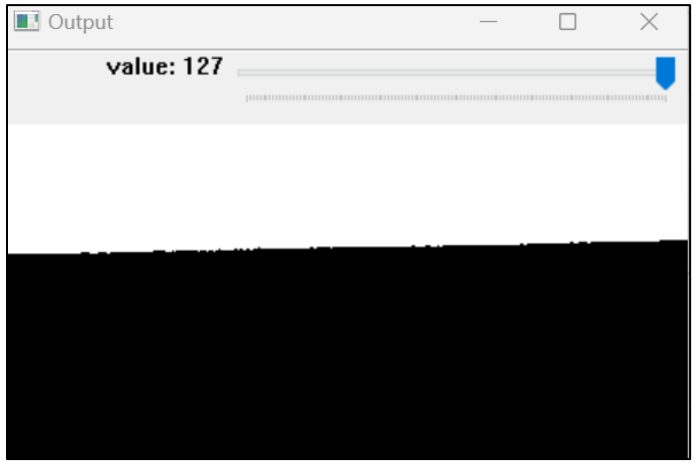
20.7 Using Trackbar for change threshold values

```
import cv2

def display(value):
    pass

cv2.namedWindow("Output")
# Limit the trackbar to a range of 0-127
cv2.createTrackbar("value","Output",0,127,display)

while True :
    gray_img = cv2.imread("gradient.png",0)
    thresh_value = cv2.getTrackbarPos("value","Output")
    thresh, result = cv2.threshold(gray_img,thresh_value,255,cv2.THRESH_BINARY)
    if cv2.waitKey(1) & 0xFF==ord("e"):
        break
    cv2.imshow("Output",result)

cv2.destroyAllWindows()
```

20.7 Using Trackbar for change threshold values

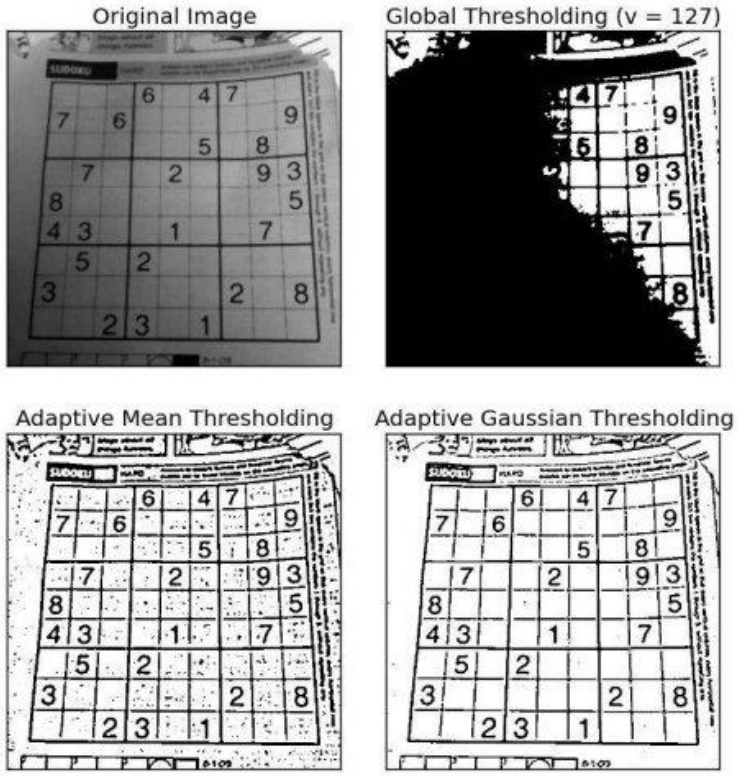






21. Adaptive Thresholding

Used to correct uneven brightness of image segmentation by referring to the neighboring area to set the appropriate threshold value.



https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html

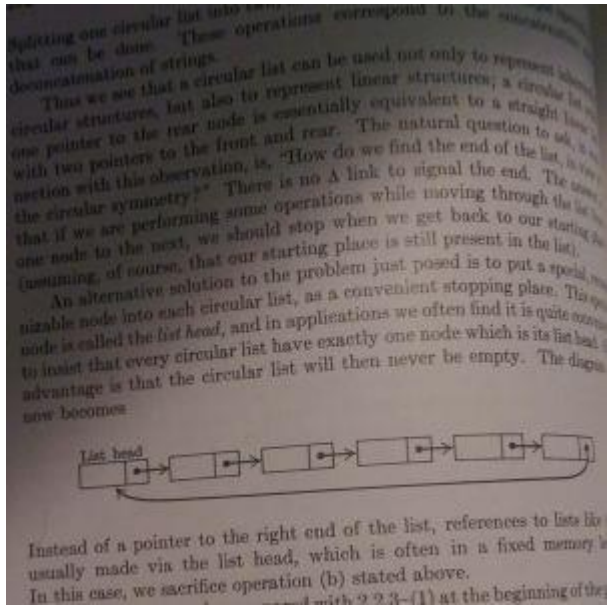
21. Adaptive Thresholding

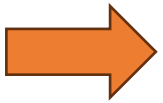
```
cv2.adaptiveThreshold(A, B, C, D, E, F)
```

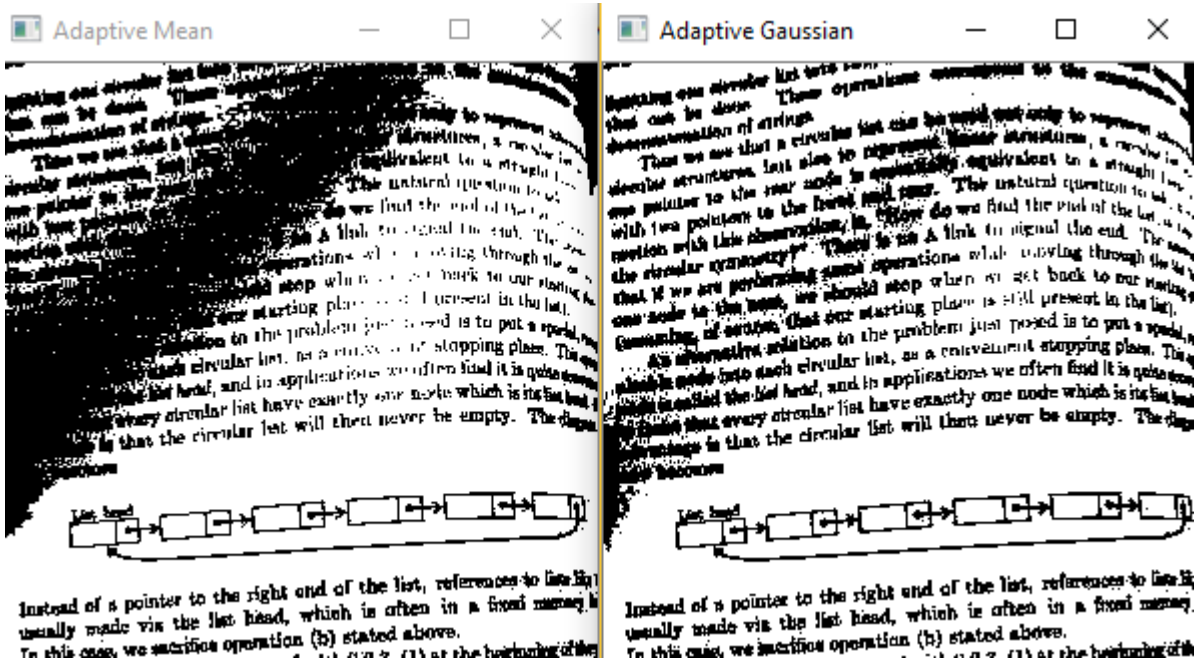
```
A = GrayScale image
B = max value
C = Adaptive method
D = Threshold type
E = Block Size
F = C-value
```

Adaptive method	Meaning
cv2.ADAPTIVE_THRESH_MEAN_C	Calculate the average around the Block size and subtract it from the C value.
cv2.ADAPTIVE_THRESH_GAUSSIAN_C	Calculate the average around the Block size and use the Gaussian function and the C value.

21. Adaptive Thresholding

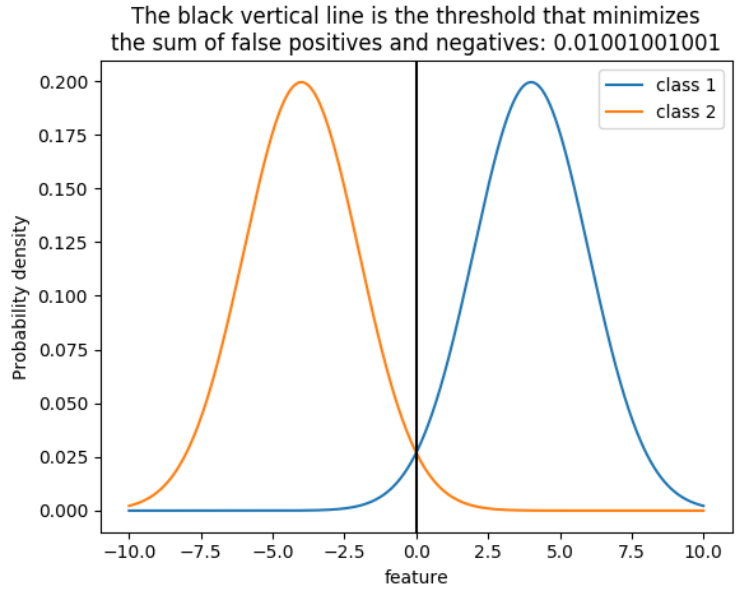


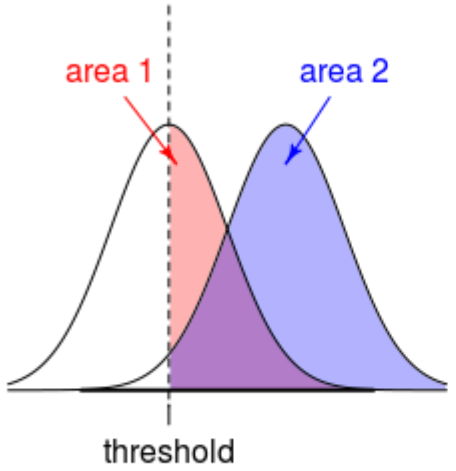


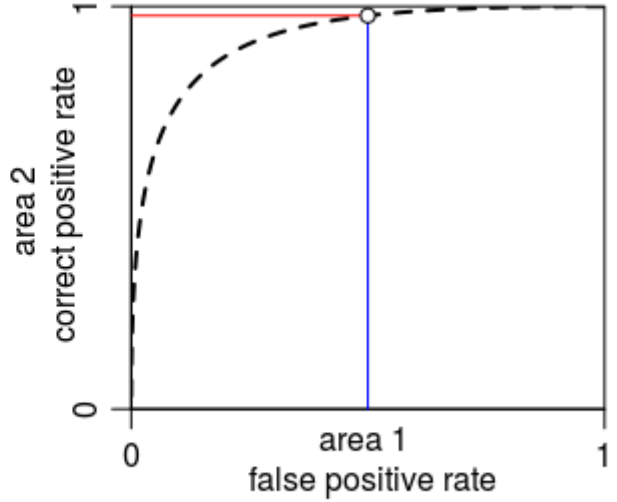


GAUSSIAN distribution (Normal distribution)

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$



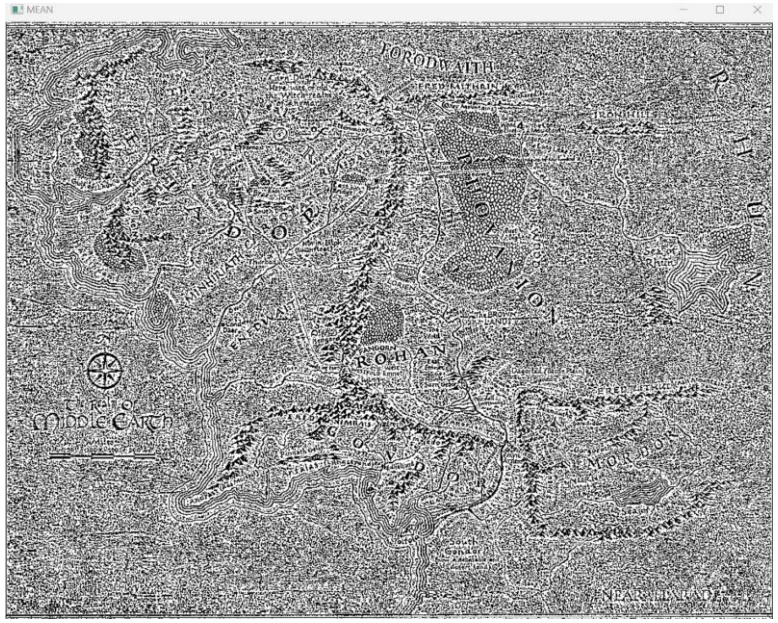


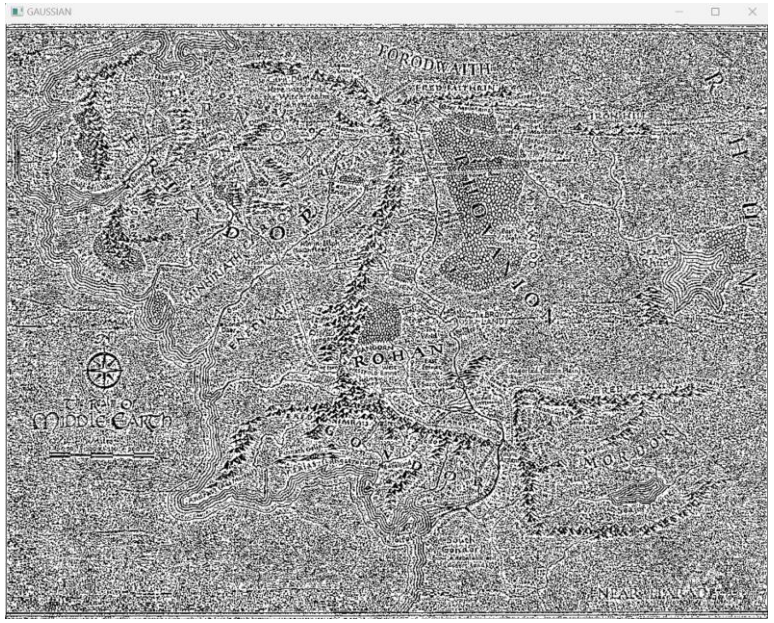


21. Adaptive Thresholding

```
import cv2
img = cv2.imread("map.png",0)

thresh,th1 = cv2.threshold(img,128,255,cv2.THRESH_BINARY)
th2 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_BINARY,3,1)
th3 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY,3,1)

cv2.imshow("THRESH",th1)
cv2.imshow("MEAN",th2)
cv2.imshow("GAUSSIAN",th3)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

21. Adaptive Thresholding





21.1 Compare BlockSize

```
import cv2
import matplotlib.pyplot as plt

img = cv2.imread("map.png",0)

size = [3,5,9,17,33]

plt.subplot(321,xticks=[],yticks=[])
plt.imshow(img,cmap="gray")

for i in range(len(size)):
    result = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY,size[i],1)
    plt.subplot(322+i)
    plt.title("%d"%size[i])
    plt.imshow(result,cmap="gray")
    plt.xticks([]),plt.yticks([])

plt.show()
```

21.1 Compare BlockSize

5

17

3

9

33

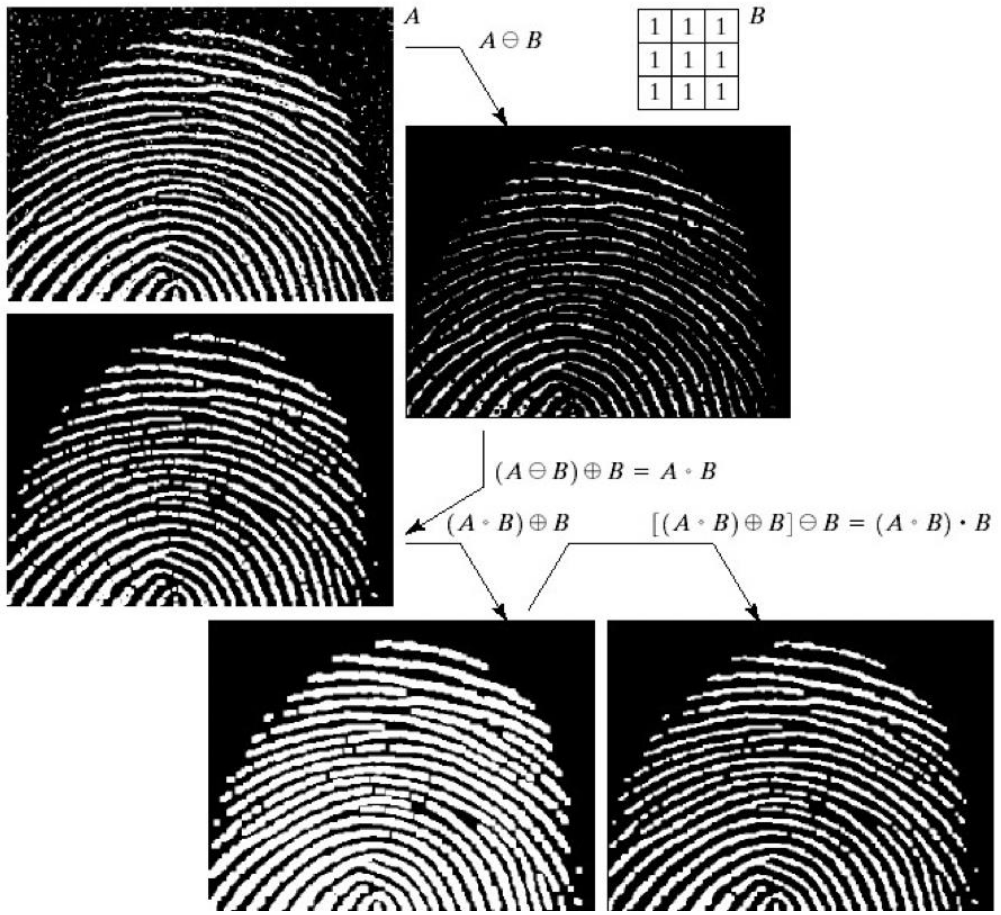
21.2 C-value

A larger C value leads to a higher threshold.

- More pixels being classified as background (black).
- Thinner object boundaries.
- Increased sensitivity to noise.

The optimal value of C depends heavily on the specific image characteristics and the desired outcome.

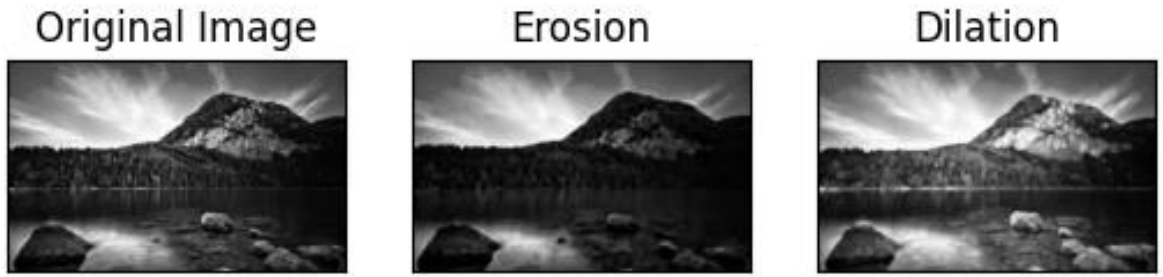
22. Morphological



22. Morphological

It is a technique used to enhance digital images using 4 operators

1. Erosion
2. Dilation
3. Opening
4. Closing



Other techniques:

- Gradient
- Top Hat
- Black Hat

etc.

22. Morphological operators

Original

Erosion

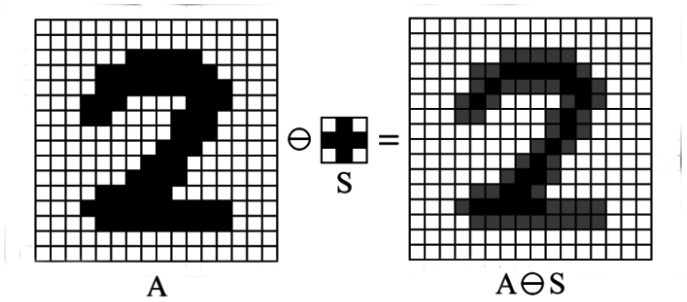
Dilation

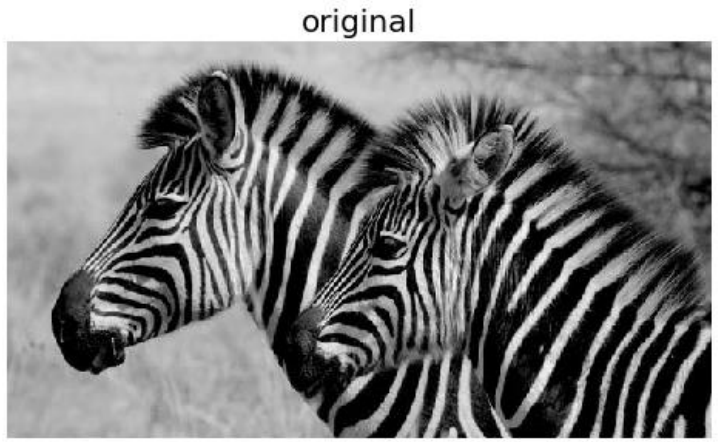
Opening

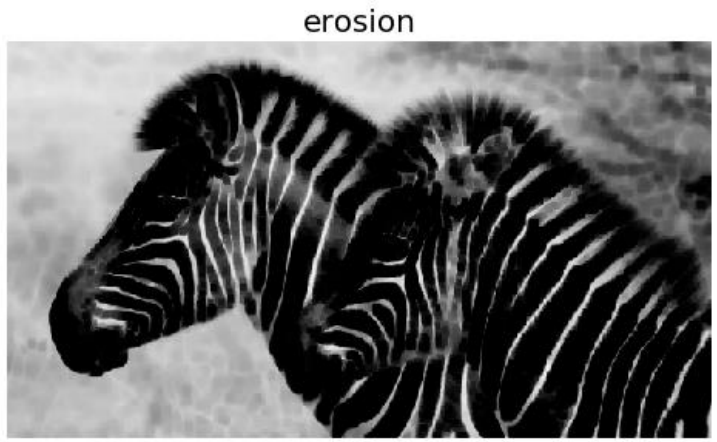
Closing

22.1 Erosion

- Erosion is a fundamental morphological operation that reduces the size of objects in a binary image. It works by removing pixels from the boundaries of objects.
- Use for remove small noise, detach connected objects, and erode boundaries.







22.1 Erosion

BW image

	(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)	(0,8)	(0,9)	(0,10)	(0,11)	(0,12)	(0,13)
(0,0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(1,0)	0	0	0	0	0	0	0	0	0	0	0	1	1	0
(2,0)	0	0	1	1	0	0	0	0	0	0	0	1	1	0
(3,0)	0	0	1	1	1	1	0	0	0	0	0	1	1	0
(4,0)	0	0	1	1	1	1	0	0	0	0	0	1	1	0
(5,0)	1	1	1	1	1	1	1	1	1	1	1	1	1	1
(6,0)	0	0	1	1	0	0	0	0	0	0	0	0	0	0
(7,0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(8,0)	0	0	0	0	0	0	1	1	1	1	1	1	0	0
(9,0)	0	0	0	0	0	0	1	1	1	1	1	1	0	0
(10,0)	0	0	0	0	1	1	1	1	1	1	1	1	0	0
(11,0)	0	1	1	1	1	1	1	1	1	1	1	1	0	0
(12,0)	0	1	1	1	0	0	1	1	1	1	1	1	0	0
(13,0)	0	1	1	1	0	0	1	1	1	1	1	1	0	0
(14,0)	0	0	0	0	0	0	1	1	1	1	1	1	0	0
(15,0)	0	0	0	0	0	0	1	1	0	1	1	0	0	0
(16,0)	0	0	0	0	0	0	1	1	0	1	1	0	0	0

X

kernel

	(0,0)	(0,1)	(0,2)	
(0,0)	1	1	1	(0,2)
(1,0)	1	1	1	(1,2)
(2,0)	1	1	1	(2,2)
	(2,0)	(2,1)	(2,2)	

→

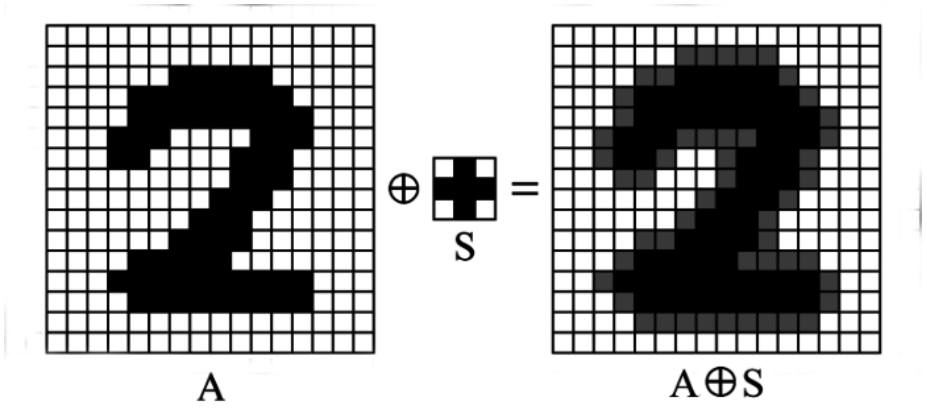
eroded BW image

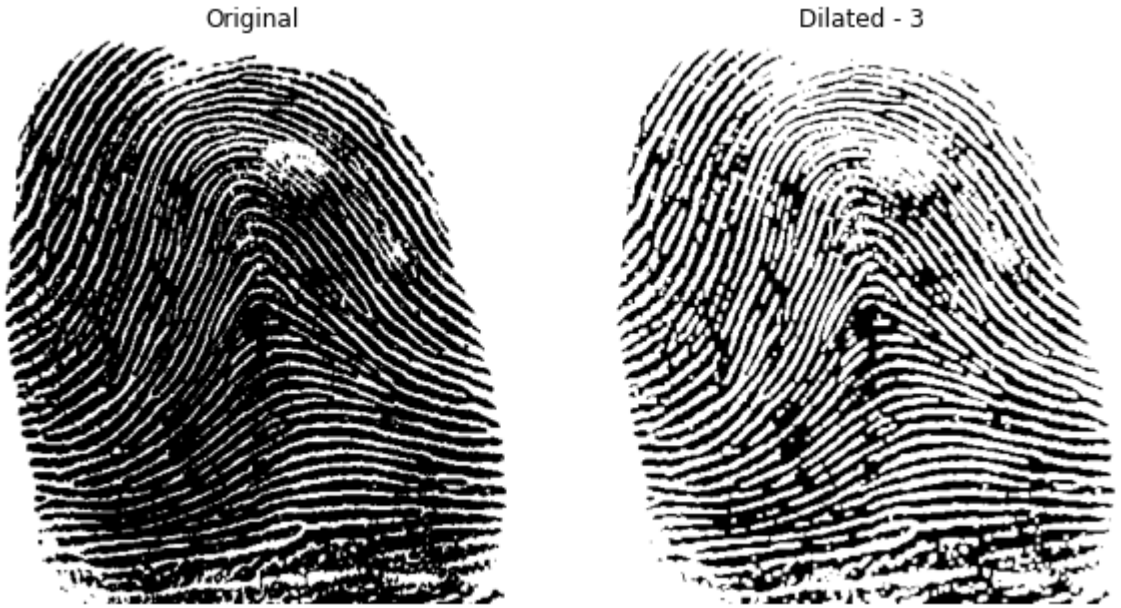
	(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)	(0,8)	(0,9)	(0,10)	(0,11)	(0,12)	(0,13)
(0,0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(1,0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(2,0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(3,0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(4,0)	0	0	0	1	1	0	0	0	0	0	0	0	0	0
(5,0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(6,0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(7,0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(8,0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(9,0)	0	0	0	0	0	0	0	1	1	1	1	0	0	0
(10,0)	0	0	0	0	0	0	0	1	1	1	1	0	0	0
(11,0)	0	0	0	0	0	0	0	1	1	1	1	0	0	0
(12,0)	0	0	1	0	0	0	0	1	1	1	1	0	0	0
(13,0)	0	0	0	0	0	0	0	1	1	1	1	0	0	0
(14,0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(15,0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(16,0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0

https://medium.com/@sasasulakshi/opencv-morphological-dilation-and-erosion-fab65c29efb3

22.2 Dilation

- Dilation is the opposite of erosion and is used to increase the size of objects in an image.
- Use for join adjacent objects, fill small holes, and enhance features.





22.2 Erosion

BW image

	(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)	(0,8)	(0,9)	(0,10)	(0,11)	(0,12)	(0,13)
(0,0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(1,0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(2,0)	0	0	1	1	0	0	0	0	0	0	0	0	0	0
(3,0)	0	0	1	1	1	1	0	0	0	0	0	0	0	0
(4,0)	0	0	1	1	1	1	0	0	0	0	0	0	0	0
(5,0)	0	0	1	1	0	0	0	0	0	0	0	0	0	0
(6,0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(7,0)	0	0	0	0	0	0	1	1	1	1	1	1	0	0
(8,0)	0	0	0	0	0	0	1	1	1	1	1	1	0	0
(9,0)	0	0	0	0	1	1	1	1	1	1	1	1	0	0
(10,0)	0	0	0	0	1	1	1	1	1	1	1	1	0	0
(11,0)	0	0	0	0	0	0	1	1	1	1	1	1	0	0
(12,0)	0	0	0	0	0	0	1	1	1	1	1	1	0	0
(13,0)	0	0	0	0	0	0	1	1	1	1	1	1	0	0
(14,0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(15,0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(16,0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0

x

kernel

	(0,0)	(0,1)	(0,2)	
(0,0)	1	1	1	(0,2)
(1,0)	1	1	1	(1,2)
(2,0)	1	1	1	(2,2)
	(2,0)	(2,1)	(2,2)	

→

dilated BW image

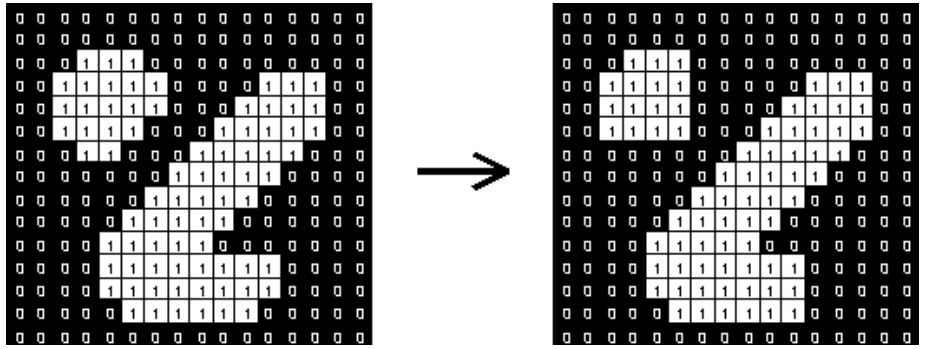
	(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)	(0,8)	(0,9)	(0,10)	(0,11)	(0,12)	(0,13)
(0,0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(1,0)	0	1	1	1	1	0	0	0	0	0	0	0	0	0
(2,0)	0	1	1	1	1	1	1	0	0	0	0	0	0	0
(3,0)	0	1	1	1	1	1	1	0	0	0	0	0	0	0
(4,0)	0	1	1	1	1	1	1	0	0	0	0	0	0	0
(5,0)	0	1	1	1	1	1	1	0	0	0	0	0	0	0
(6,0)	0	1	1	1	1	1	1	1	1	1	1	1	1	0
(7,0)	0	0	0	0	0	1	1	1	1	1	1	1	1	0
(8,0)	0	0	0	1	1	1	1	1	1	1	1	1	1	0
(9,0)	0	0	0	1	1	1	1	1	1	1	1	1	1	0
(10,0)	0	0	0	1	1	1	1	1	1	1	1	1	1	0
(11,0)	0	0	0	1	1	1	1	1	1	1	1	1	1	0
(12,0)	0	0	0	0	0	1	1	1	1	1	1	1	1	0
(13,0)	0	0	0	0	0	1	1	1	1	1	1	1	1	0
(14,0)	0	0	0	0	0	1	1	1	1	1	1	1	1	0
(15,0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(16,0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0

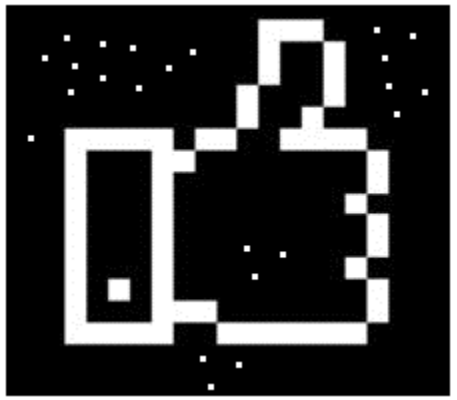
https://medium.com/@sasasulakshi/opencv-morphological-dilation-and-erosion-fab65c29efb3

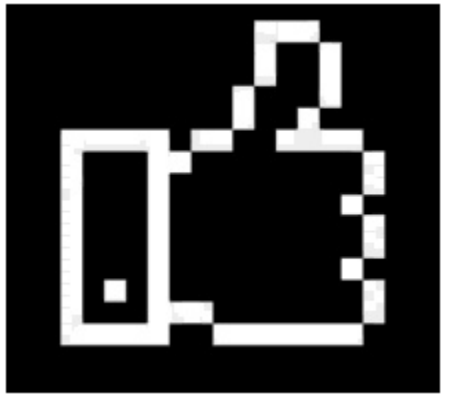
22.3 Opening

- Opening is a compound operation that involves erosion followed by dilation.
- Use for remove small objects or noise from the image while preserving the shape and size of larger objects.

$$A \circ B = (A \ominus B) \oplus B$$



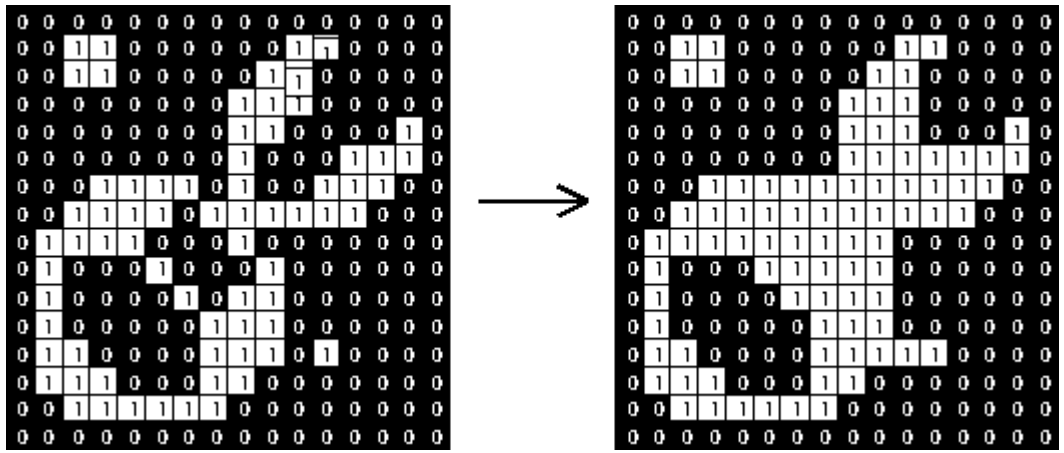


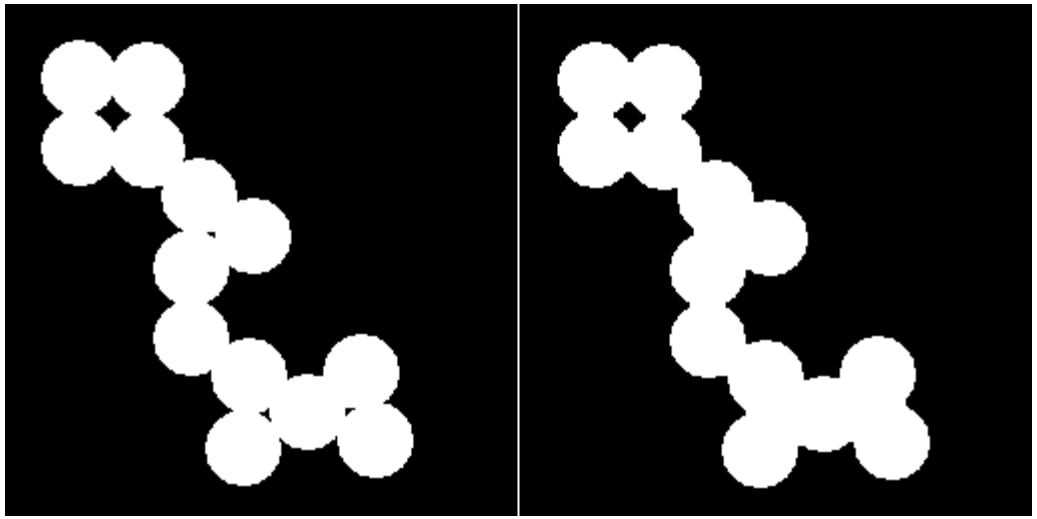


22.4 Closing

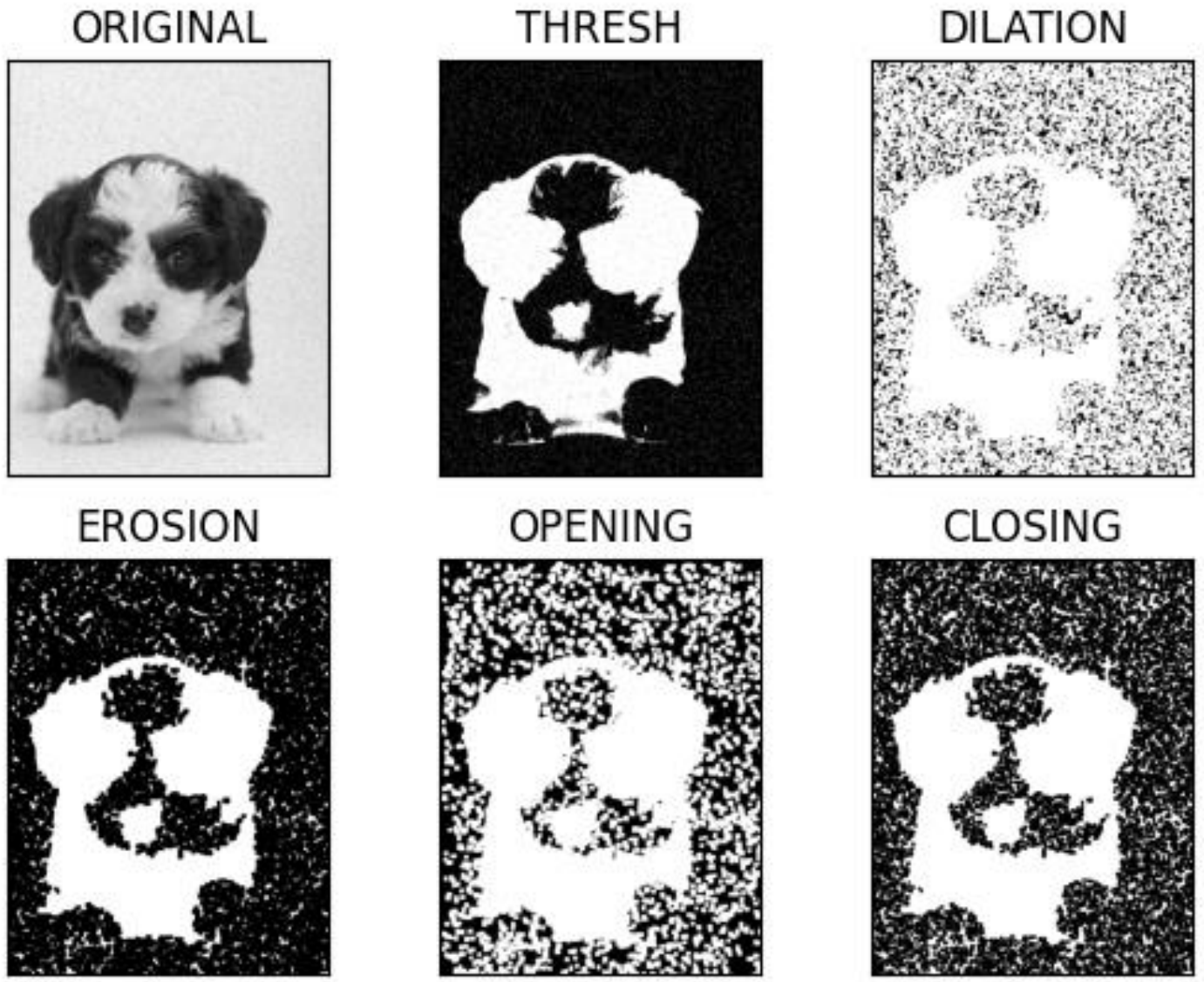
- Closing is another compound operation that consists of dilation followed by erosion.
- Use for fill small holes and gaps in objects while preserving their overall shape.

$$A \bullet B = (A \oplus B) \ominus B$$



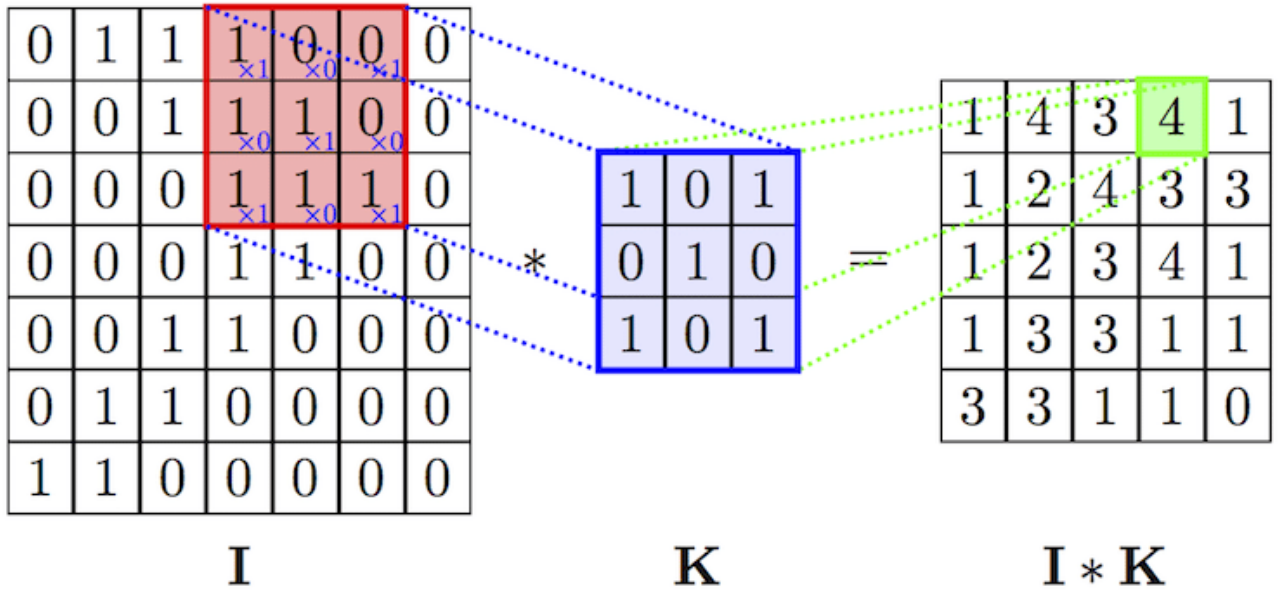


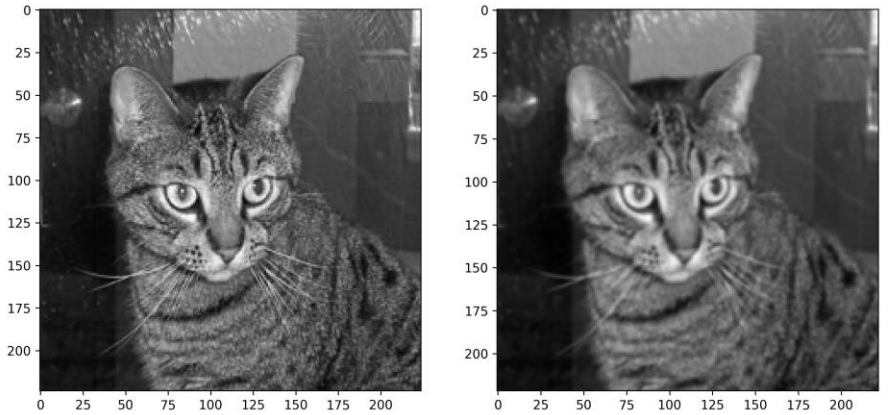
```python
import cv2
import matplotlib.pyplot as plt
import numpy as np

img = cv2.imread("dog-noise.jpg",0)
thresh , result = cv2.threshold(img,170,255,cv2.THRESH_BINARY_INV)
kernel = np.ones((2,2),np.uint8)
# Dilation
dilation = cv2.dilate(result,kernel,iterations=5)
# Erosion
erosion = cv2.erode(dilation,kernel,iterations=7)
# Opening
opening = cv2.morphologyEx(dilation,cv2.MORPH_OPEN,kernel,iterations=7)
# Closing
closing = cv2.morphologyEx(result,cv2.MORPH_CLOSE,kernel,iterations=5)

titles = ["ORIGINAL","THRESH","DILATION","EROSION","OPENING","CLOSING"]
images = [img,result,dilation,erosion,opening,closing]
for i in range(len(images)):
    plt.subplot(2,3,i+1)
    plt.imshow(images[i],cmap="gray")
    plt.title(titles[i])
    plt.xticks([])
    plt.yticks([])

plt.show()
```


ORIGINAL
THRESH
DILATION
EROSION
OPENING
CLOSING

23. Convolution

It is a method of improving the quality of an image by removing noise from the image using filters or “kernels”.

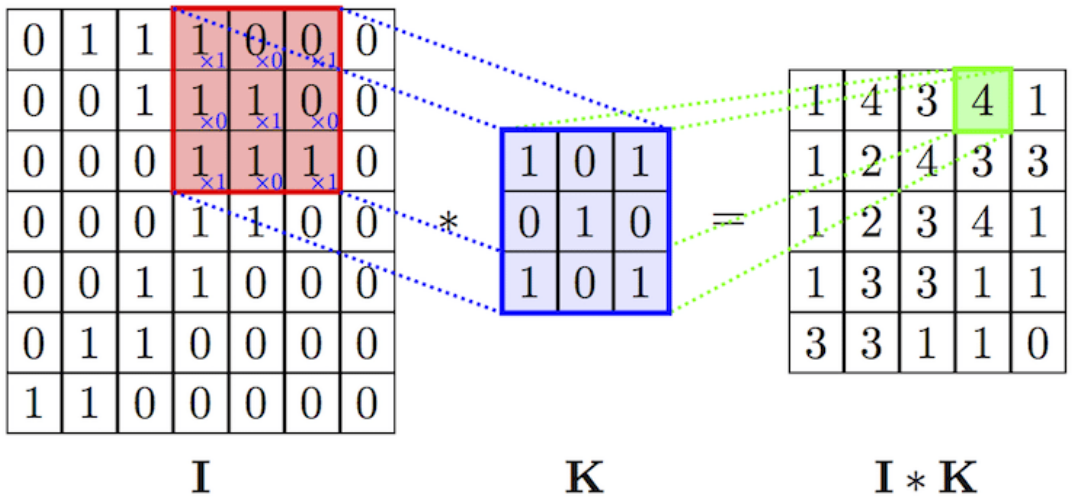




https://www.researchgate.net/figure/An-example-of-convolution-operation-in-2D-2_fig3_324165524

23.1 Filter2D

```
cv2.filter2D(Array image, ddepth = '-1' refer to original image, kernel size (w x h))
```



Kernel size: 3x3

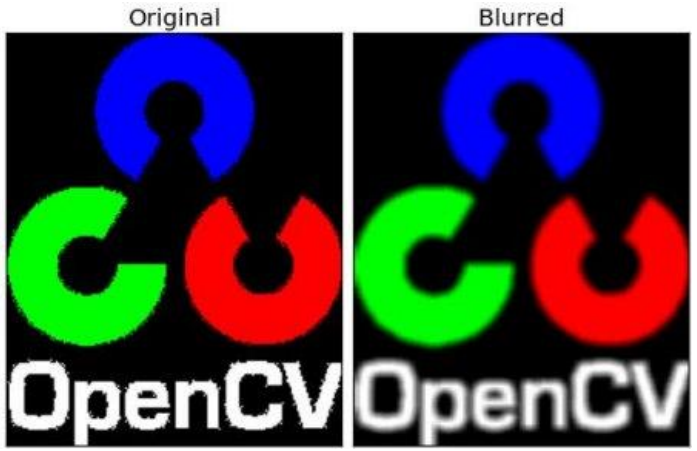
```
np.ones((3,3),np.float32)/9
```

23.2 Mean filtering (Blur)

It 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.

```
cv2.blur(Array image, kernel size (w x h))
```

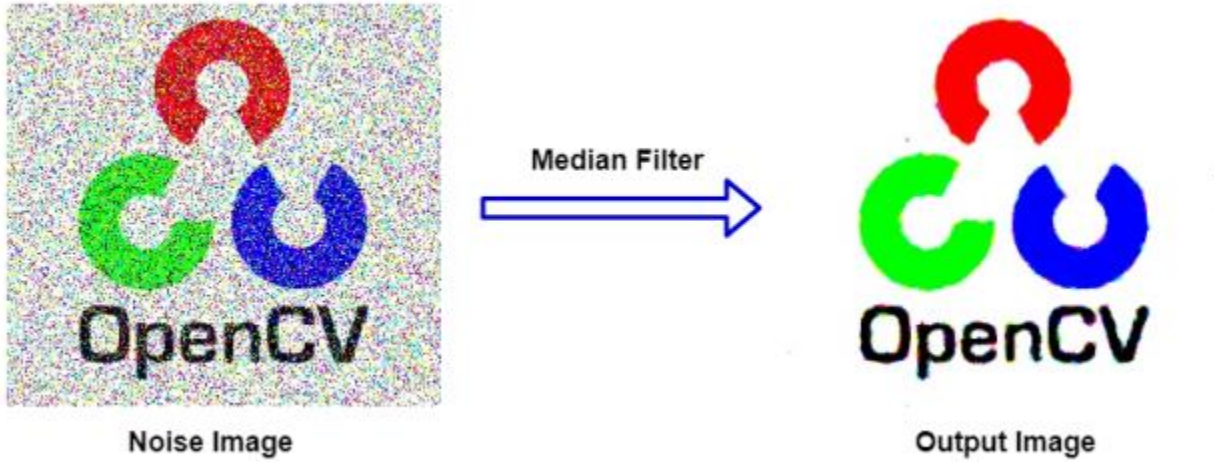
$$K = \frac{1}{\texttt{ksize.width*ksize.height}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & 1 & 1 & \cdots & 1 & 1 \\ \cdots & & & & & \\ 1 & 1 & 1 & \cdots & 1 & 1 \end{bmatrix}$$



23.3 Median filtering

Used to eliminate random noise. It can reduce edge blur well without reducing image sharpness. It works well in cases where there are scattered white or black dots throughout the image.

```
cv2.medianBlur(Array image, kernel size (w x h))
```



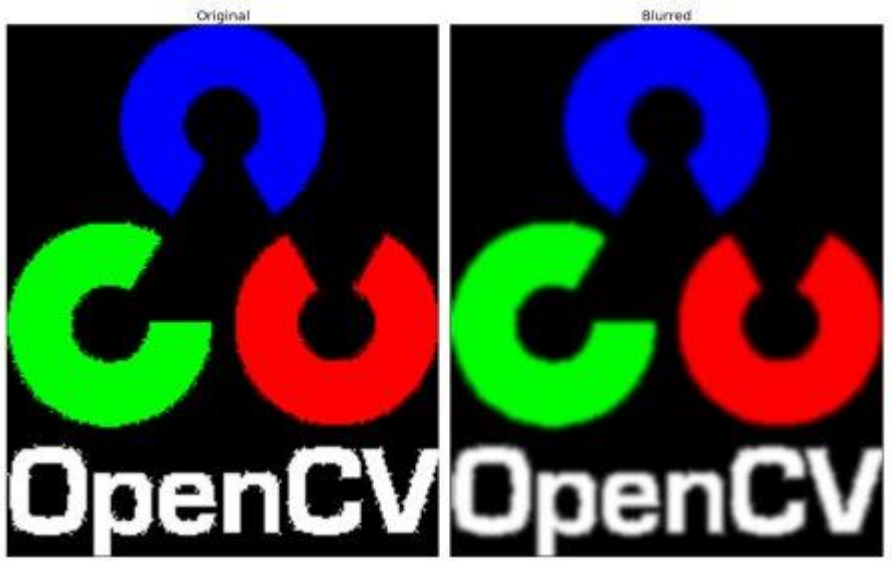
Noise Image

Output Image

23.4 Gaussian filter (Gaussian blur)

- Used to reduce noise and reduce image detail. Filters with equal values are used as the average of all values.
- Gaussian blurring is highly effective in removing Gaussian noise from an image.
- Must specify the standard deviation in the X (sigma X) and Y (sigma Y) directions.
  if only sigma X is specified, sigma Y is taken as the same as sigmaX. If both are given as zeros

```
cv2.GaussianBlur(Array image, kernel size (w x h), Sigma)
```



```
# Convolution Filter with Filter2D
import cv2
import numpy as np
import matplotlib.pyplot as plt

# original image
img = cv2.imread("dog-noise.jpg")
# filter2d
filter2d = cv2.filter2D(img,-1,np.ones((5,5),np.float32)/25)
# blur
mean = cv2.blur(img,(5,5))
# median
mblur=cv2.medianBlur(img,5)
# Gaussian
gblur = cv2.GaussianBlur(img,(5,5),1)
titles = ["ORIGINAL","FILTER2D","MEAN","MEDIAN BLUR","GAUSSIAN BLUR"]
images = [img,filter2d,mean,mblur,gblur]

for  i in range(len(images)):
    plt.subplot(2,3,i+1)
    plt.imshow(images[i])
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])

plt.show()
```

ORIGINAL

FILTER2D

MEAN

MEDIAN BLUR

GAUSSIAN BLUR

24. Feature Detectors

OpenCV provides several feature detectors like Key point detection, corner detection, edge detection etc.

Commonly uses Edges Detectors:

- Canny method
- Sobel method
- Laplacian method

Commonly uses Key point Detectors:

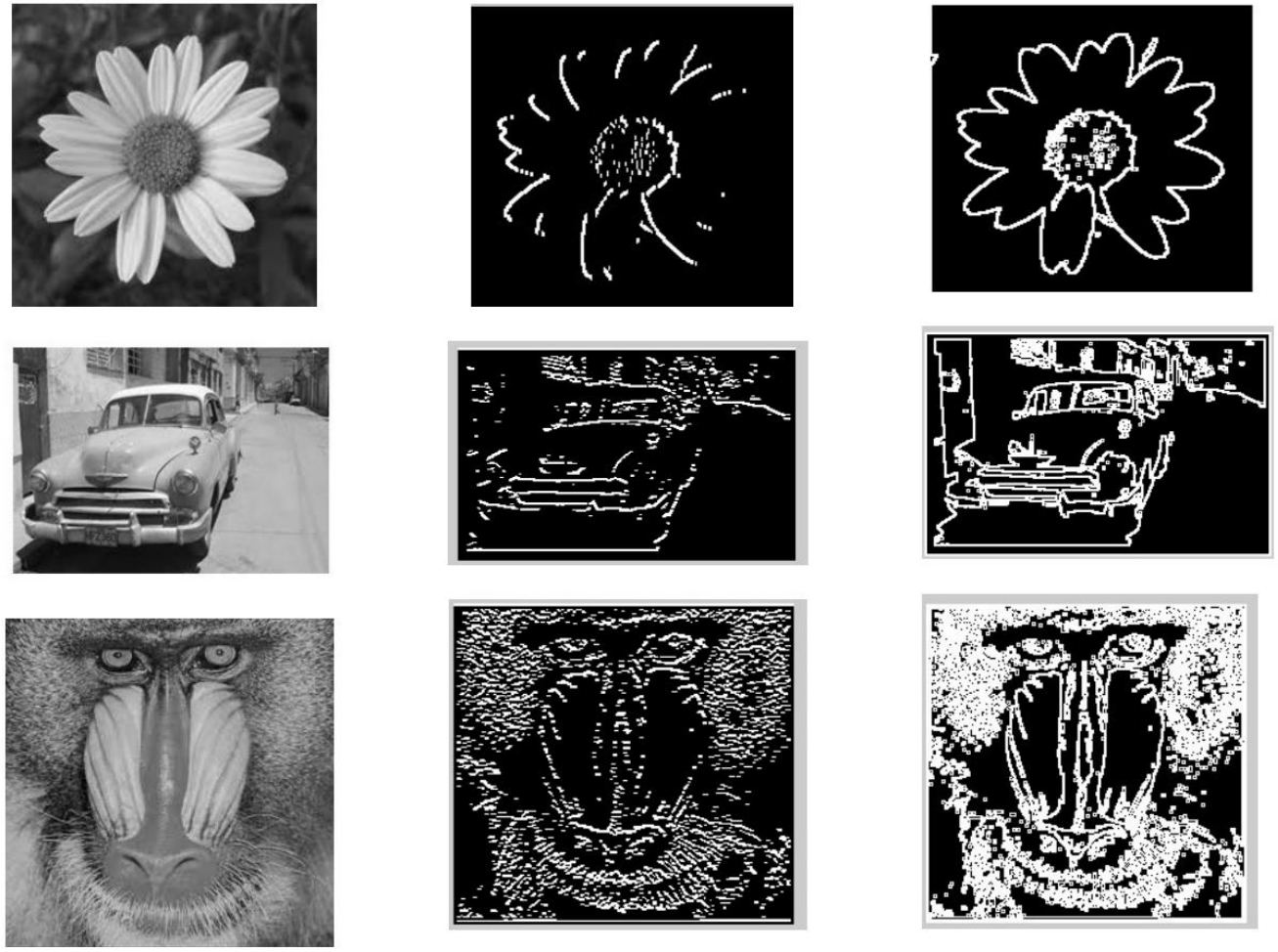
- SIFT (Scale-Invariant Feature Transform)
- ORB (Oriented FAST and Rotated BRIEF)

Commonly uses Corner Detectors:

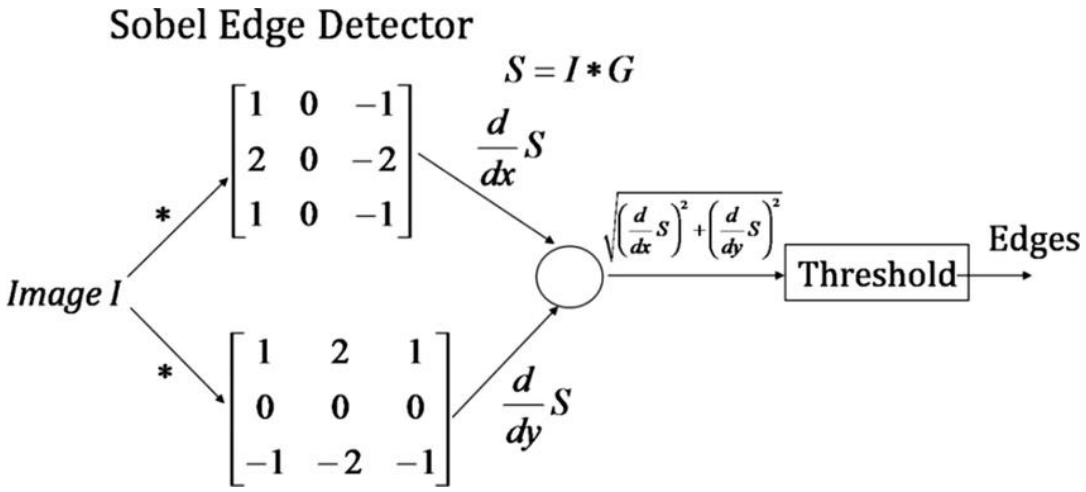
- Harris Corner Detector
- Shi-Tomasi Corner Detector

24.1 Edge Detection

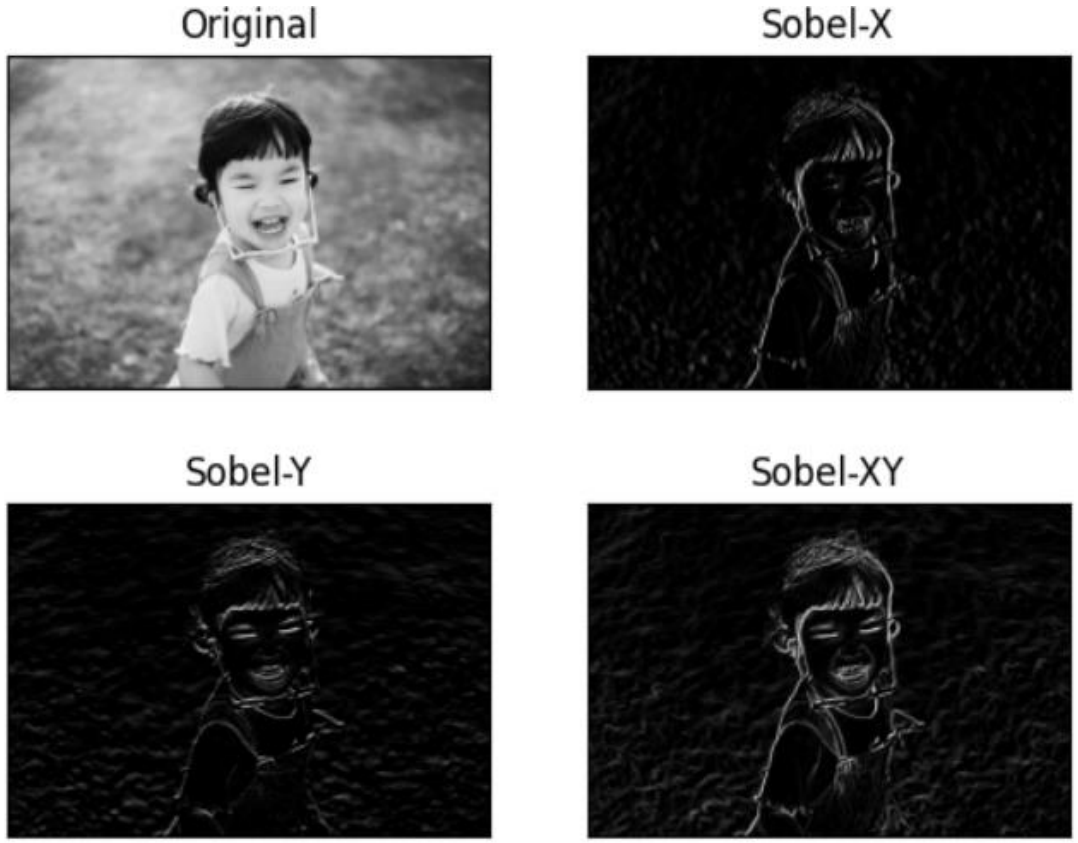
- Sobel method
- Laplacian method
- Canny method



24.1.1 Sobel Method



https://www.researchgate.net/figure/Sobel-edge-detection-methodology_fig3_372858011



https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html

24.1.1 Sobel Method

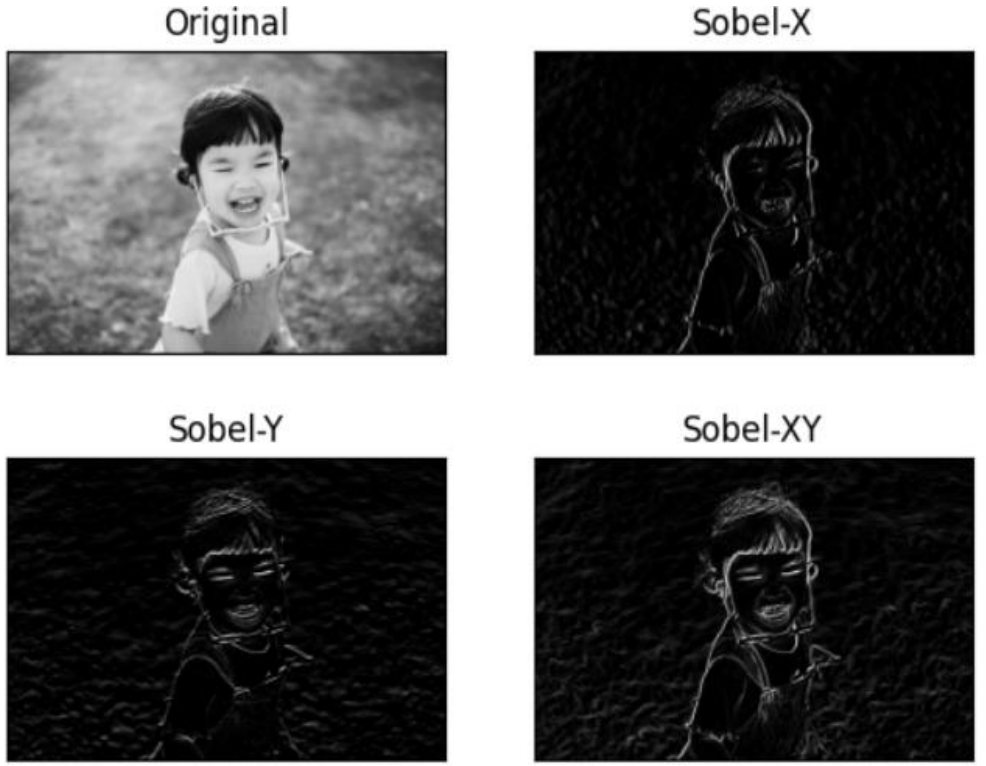
```
# cv2.Sobel(array image, parameter type in array = -1 (from original image),
# dx = filter size -x, dy = filter size -y)

import cv2
import matplotlib.pyplot as plt

img = cv2.imread("kid.jpg",0) #gray scale


SobelX = cv2.Sobel(img, -1, 1, 0) # find Edge in X axis
SobelY = cv2.Sobel(img, -1, 0, 1) # find Edge in Y axis


cv2.imshow("Output", img)
cv2.imshow("Sobel-X", SobelX)
cv2.imshow("Sobel-Y", SobelY)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

24.1.1 Sobel Method

```
import cv2
import matplotlib.pyplot as plt

img = cv2.imread("kid.jpg",0) #gray scale


SobelX = cv2.Sobel(img, -1, 1, 0) # find Edge in X axis
SobelY = cv2.Sobel(img, -1, 0, 1) # find Edge in Y axis
# Sum Sobel X + Y
SobelXY = cv2.bitwise_or(SobelX,SobelY)

cv2.imshow("Output", img)
cv2.imshow("Sobel-XY", SobelXY)
cv2.imshow("Sobel-X", SobelX)
cv2.imshow("Sobel-Y", SobelY)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

24.1.1 Sobel Method

```
import cv2
import matplotlib.pyplot as plt

img = cv2.imread("kid.jpg",0) #gray scale


SobelX = cv2.Sobel(img, -1, 1, 0) # find Edge in X axis
SobelY = cv2.Sobel(img, -1, 0, 1) # find Edge in Y axis
# Sum Sobel X + Y
SobelXY = cv2.bitwise_or(SobelX,SobelY)

images = [img,SobelX,SobelY,SobelXY]
title = ["Original", "Sobel-X", "Sobel-Y", "Sobel-XY"]

for i in range(len(images)):
    plt.subplot(2,2,i+1) # plot: 2 rows , 2 columns
    # plt.imshow(images[i])
    plt.imshow(images[i], cmap = "gray") # Show in grayscale
    plt.title(title[i])
    plt.xticks([]), plt.yticks([])

plt.show()
```



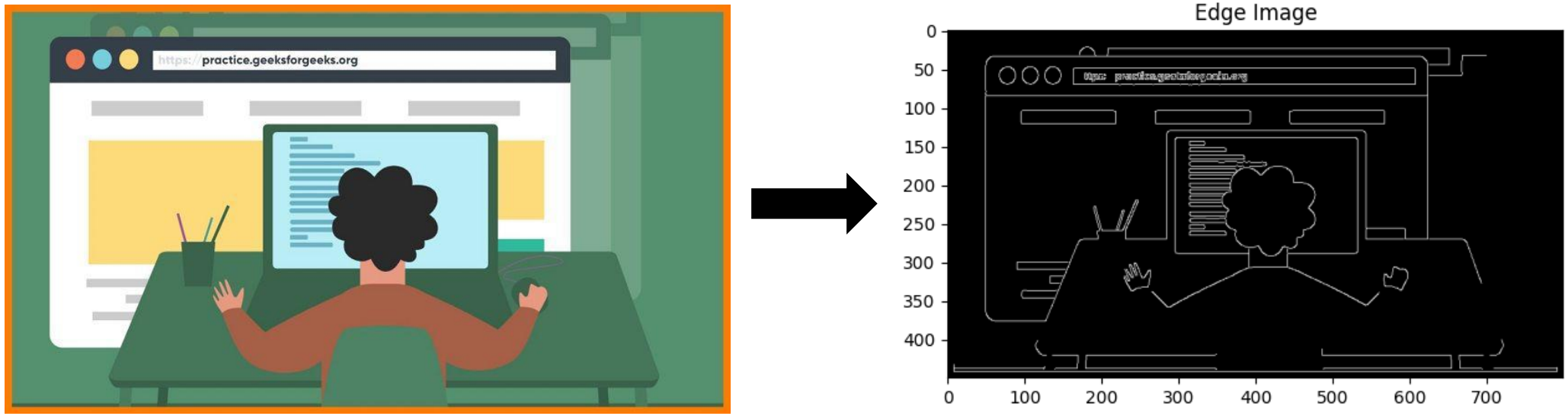
24.1.2 Laplacian Method

```
# Laplacian method - Find image edges and filter the image to improve the quality simultaneously.

import cv2
import matplotlib.pyplot as plt

img = cv2.imread("kid.jpg",0) #gray scale


lap = cv2.Laplacian(img, -1)


cv2.imshow("Original img", img)
cv2.imshow("Laplacian img ", lap)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

https://docs.opencv.org/3.4/d5/db5/tutorial_laplace_operator.html

24.1.3 Canny Method

The Canny edge detection algorithm smooths the image to reduce noise, calculates the gradient to find edge strength and direction, applies non-maximum suppression to thin edges, and uses hysteresis for final edge tracking, resulting in a black and white image with edges in white. This makes it highly effective for object detection and image segmentation.

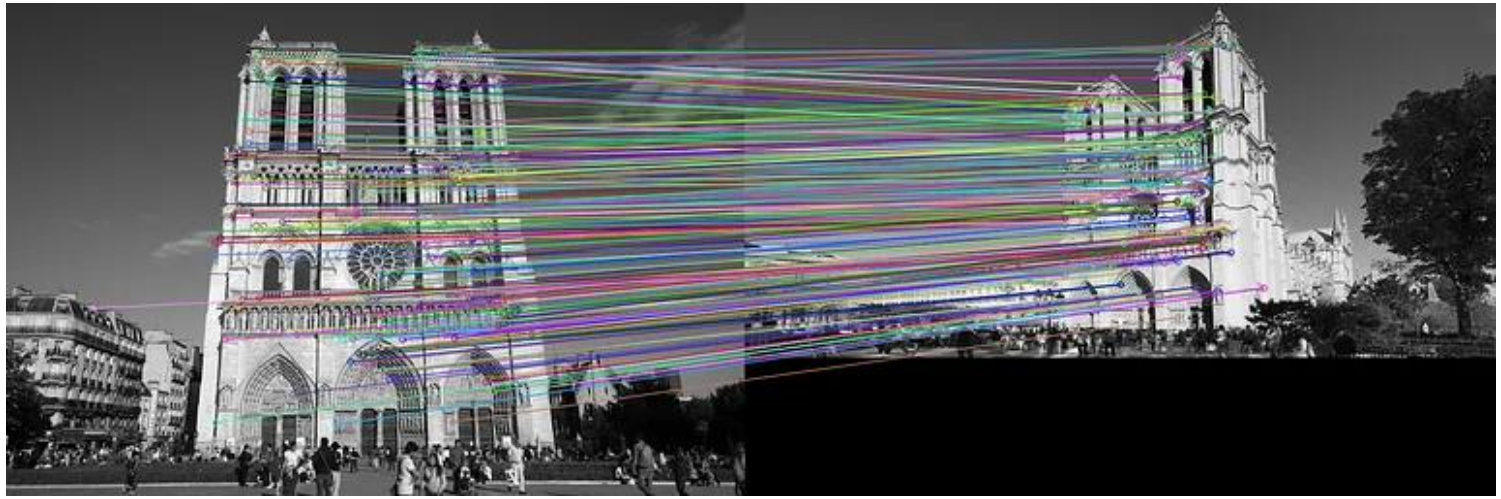


24.1.3 Canny Method

```
# Canny method
# Finds the edges of the image by setting 2 threshold values (0-255) that indicate different intensity.
# Not suitable for images with a lot of noise.

import cv2
import matplotlib.pyplot as plt

img = cv2.imread("kid.jpg",0) #gray scale


canny = cv2.Canny(img, 50, 200)


cv2.imshow("Original img", img)
cv2.imshow("Canny img ", canny)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html

24.2. Key point Detectors



24.2.1 Scale-Invariant Feature Transform (SIFT)

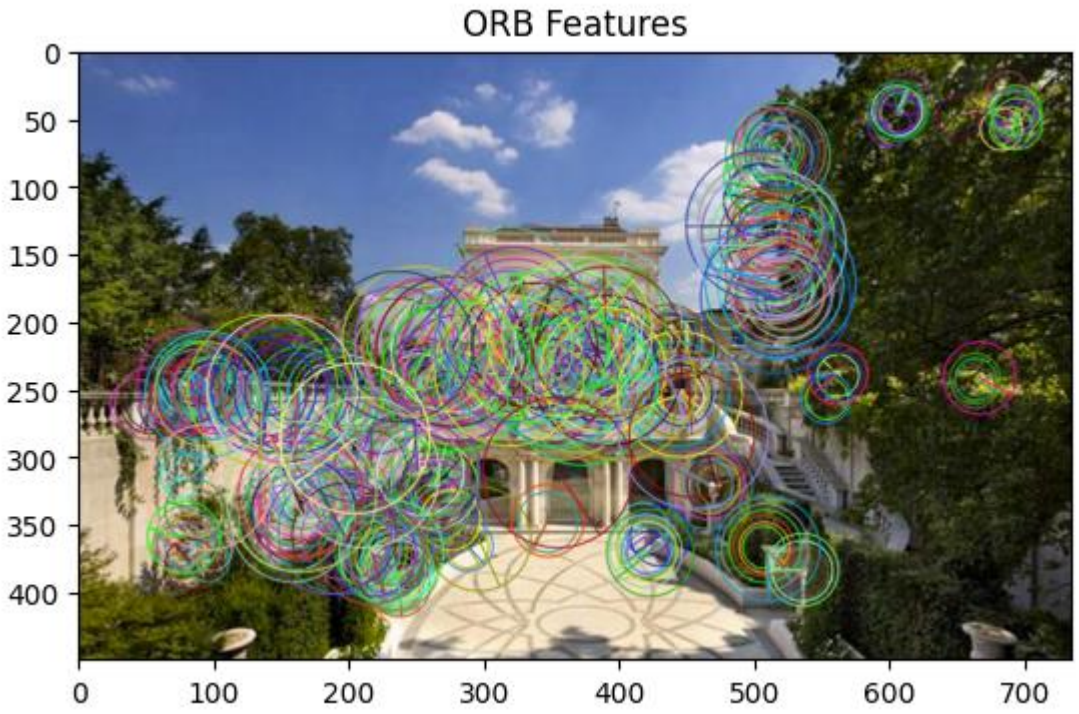
SIFT can find specific points in an image by looking at their size, orientation, and local details.

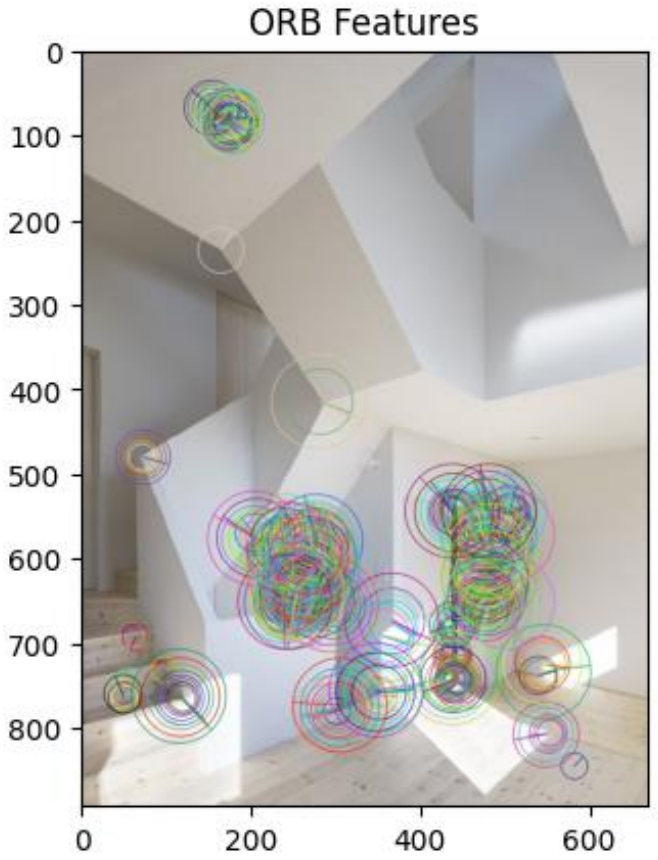
24.2.1 Scale-Invariant Feature Transform (SIFT)

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

image = cv2.imread('house.jpg')
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

sift = cv2.SIFT_create()
keypoints, descriptors = sift.detectAndCompute(gray_image, None)
image_with_sift = cv2.drawKeypoints(image, keypoints, None)
plt.imshow(cv2.cvtColor(image_with_sift, cv2.COLOR_BGR2RGB))
plt.title('SIFT Features')
plt.show()
```

24.2.2 Oriented FAST and Rotated BRIEF (ORB)

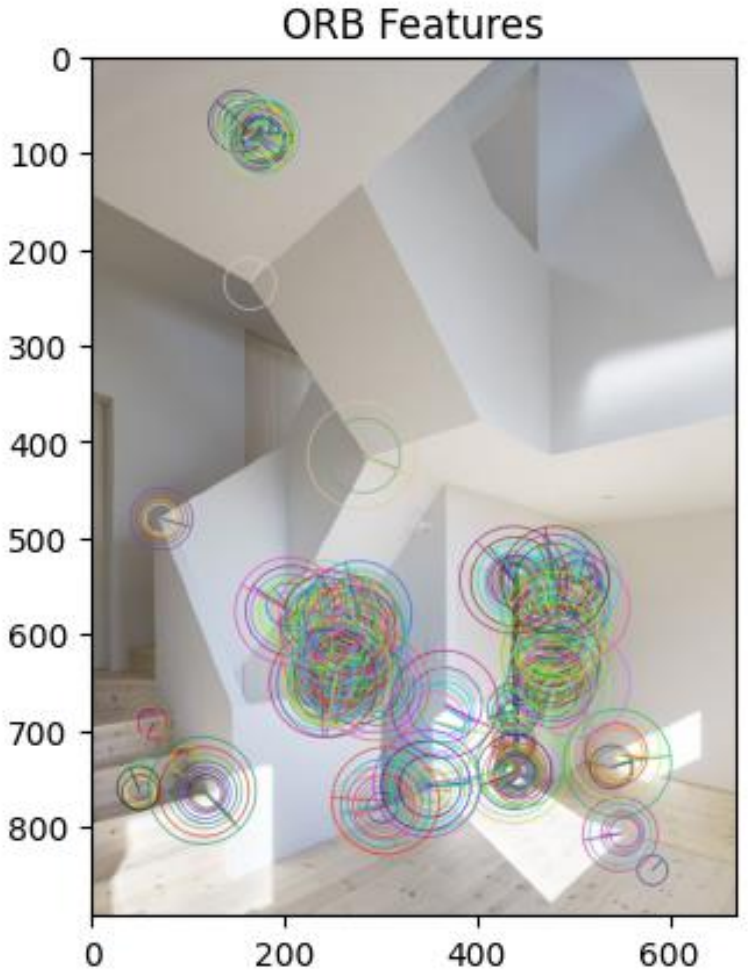
ORB is a faster alternative to SIFT, suitable for real-time applications.





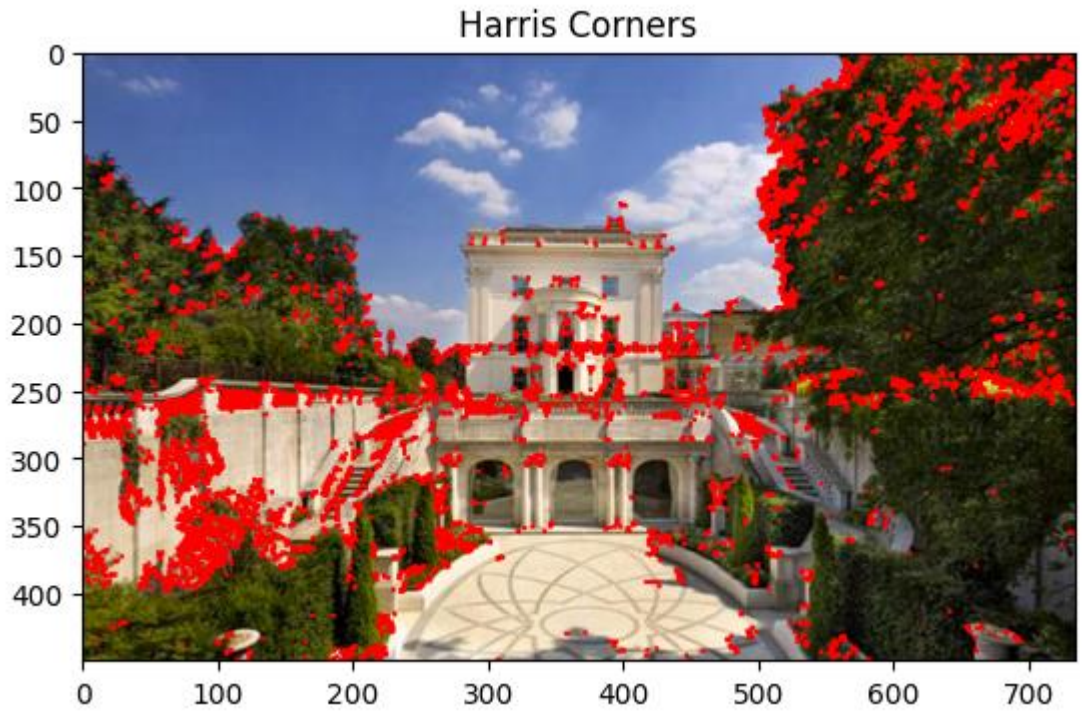
24.2.2 Oriented FAST and Rotated BRIEF (ORB)

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image = cv2.imread('house.jpg')
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

orb = cv2.ORB_create()
keypoints, descriptors = orb.detectAndCompute(gray_image, None)
image_with_orb = cv2.drawKeypoints(image, keypoints, None,
                 flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
plt.imshow(cv2.cvtColor(image_with_orb, cv2.COLOR_BGR2RGB))
plt.title('ORB Features')
plt.show()
```



24.3.1 Corner Detectors : Harris Corner Detector

This algorithm detects corners by measuring the change in image intensity when the image is shifted in different directions.

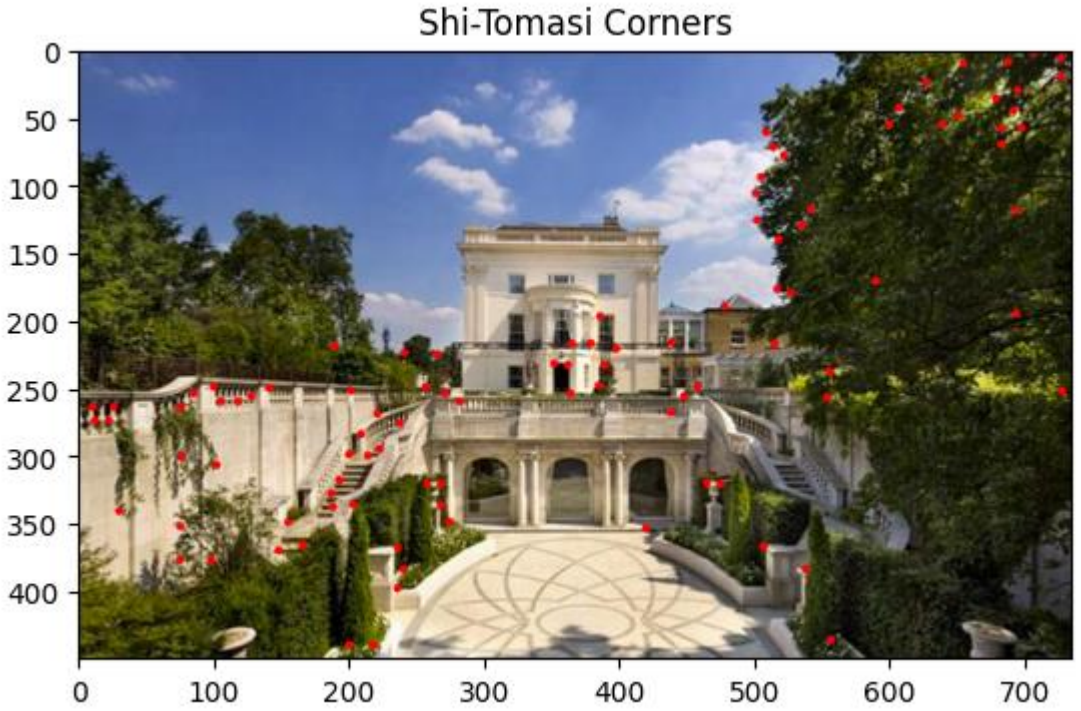


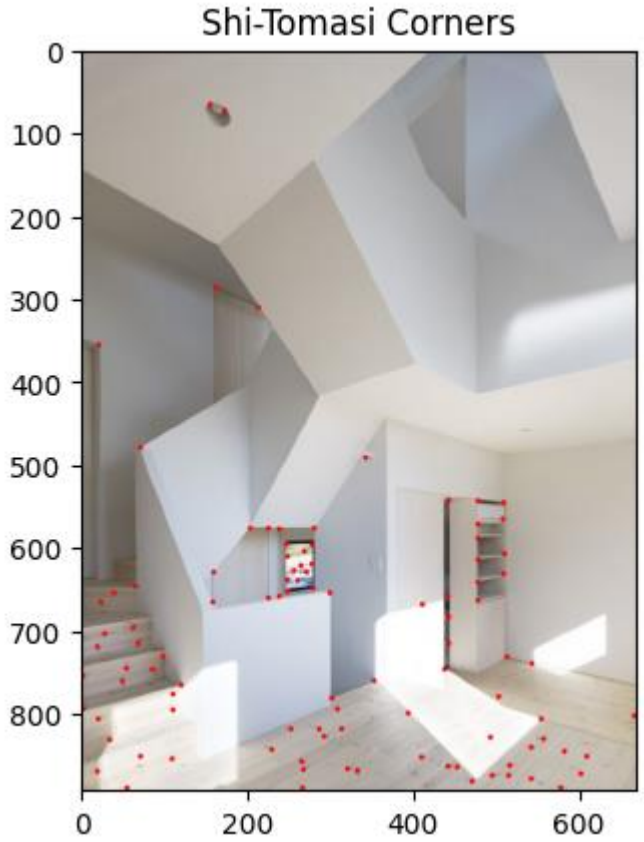
24.3.1 Corner Detectors : Harris Corner Detector

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

image = cv2.imread('house.jpg')
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

dst = cv2.cornerHarris(gray_image, 2, 3, 0.04)
dst = cv2.dilate(dst, None)
thresh = 0.01 * dst.max()

corner_image = image.copy()
corner_image[dst > thresh] = [0, 0, 255]

plt.imshow(cv2.cvtColor(corner_image, cv2.COLOR_BGR2RGB))
plt.title('Harris Corners')
plt.show()
```

24.3.2 Corner Detectors: Shi–Tomasi Corner Detector

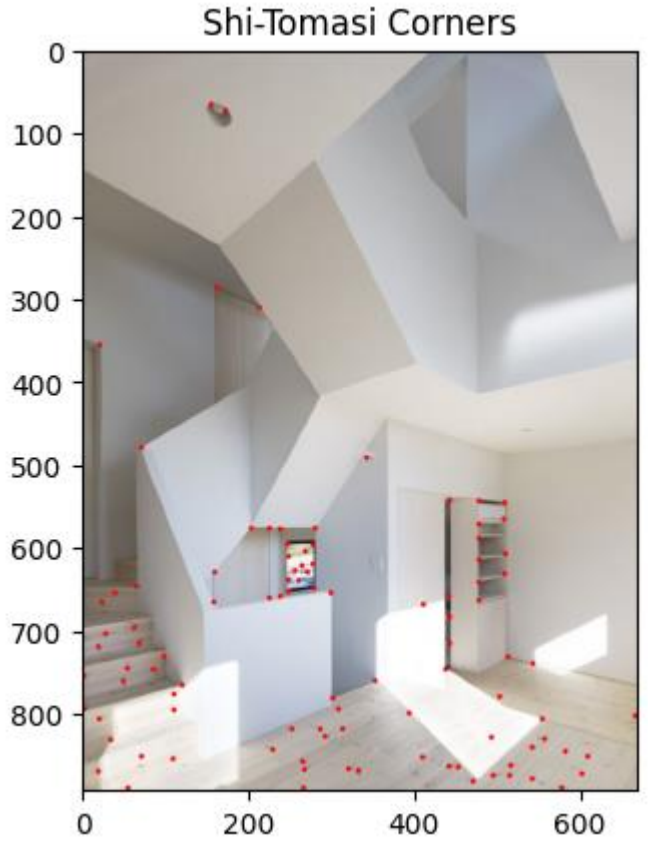
This is a variation of the Harris Corner Detector that is more efficient.





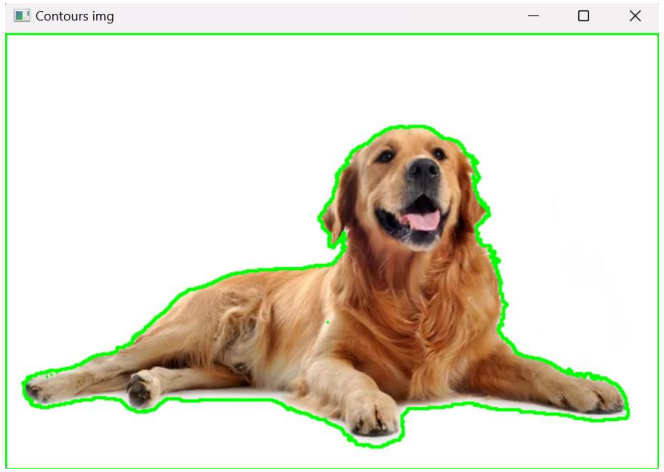
24.3.2 Corner Detectors: Shi–Tomasi Corner Detector

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

image = cv2.imread('castle.png')
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)


corners = cv2.goodFeaturesToTrack(gray_image, 100, 0.01, 10)
corners = np.intp(corners)
for i in corners:
    x, y = i.ravel()
    cv2.circle(image, (x, y), 3, (0, 0, 255), -1)

plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title('Shi-Tomasi Corners')
plt.show()
```



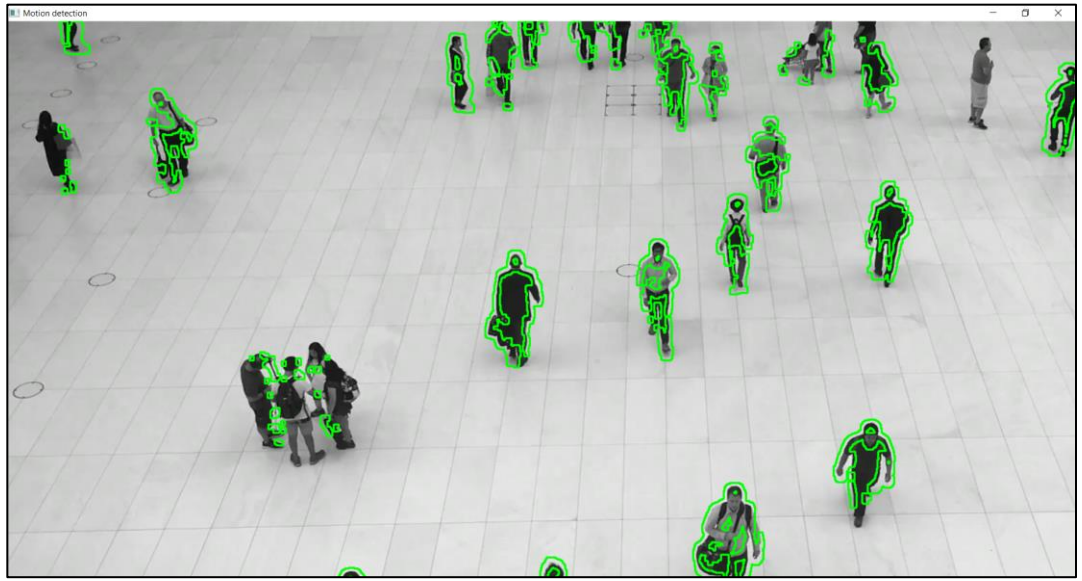
25. Contour

```
import cv2
import matplotlib.pyplot as plt

img = cv2.imread("golden.jpg")
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY) # Gray scale

# Set threshold value to create binary image - separate area into black and white
# You may need to experiment with threshold value or use other image enhancement techniques before using the image.
thresh, result = cv2.threshold(gray, 200,255, cv2.THRESH_BINARY)         # 246,255 is better

# Find Image Contour
contours, hierarchy = cv2.findContours(result, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)

print(contours) # Show array of contours
print(len(contours)) # Count points of contour

# Draw contour
cv2.drawContours(img, contours, -1, (0,255,0), 2) # draw green line, width = 2

# cv2.imshow("Binary img", result)
cv2.imshow("Contours img", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html#ga17ed9f5d79ae97bd4c7cf18403e1689a

26. Motion Detection



```python
import cv2

cap = cv2.VideoCapture("walking.mp4")
while (cap.isOpened()):
    check , frame = cap.read()
    if check == True :
        gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY) # convert to gray scale
        thresh, result = cv2.threshold(gray, 215,255, cv2.THRESH_BINARY)
        contours, hierarchy = cv2.findContours(result, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
        cv2.drawContours(frame, contours, -1, (0,255,0), 2)
        cv2.imshow("Output", frame)
        if cv2.waitKey(1) & 0xFF == ord("e"):
            break
    else :
        break

cap.release()
cv2.destroyAllWindows()

# It was found that people were not detected.
```

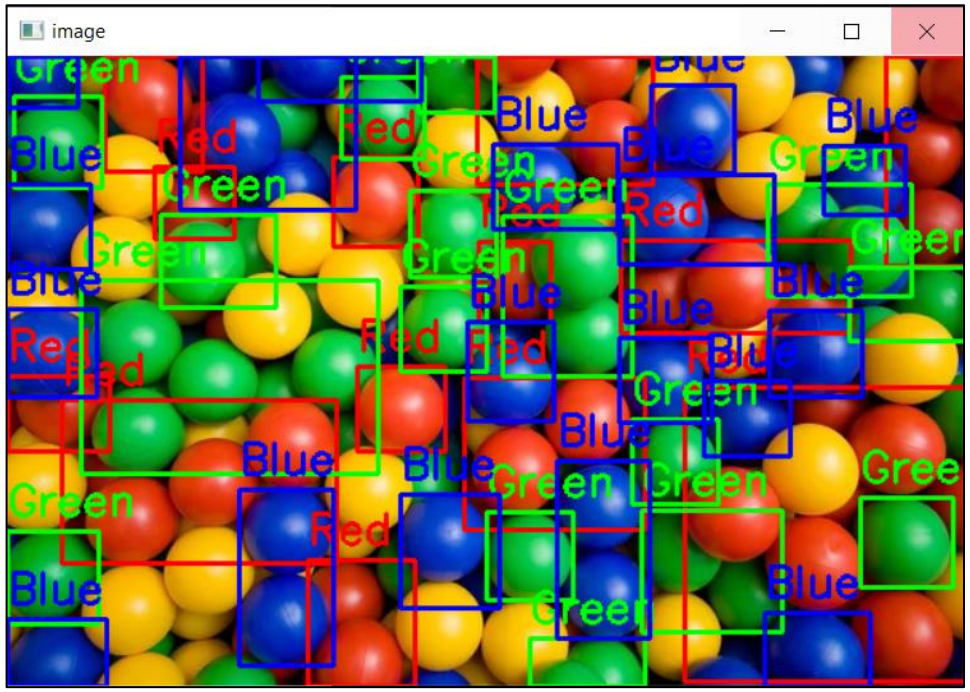
```
import cv2

cap = cv2.VideoCapture("walking.mp4")

# Read 2 frames to check for movement in any area from the difference between frames.
check , frame1 = cap.read() # Mark contour
check , frame2 = cap.read()

while (cap.isOpened()):
    if check == True :
        motiondiff = cv2.absdiff(frame1,frame2)
        gray = cv2.cvtColor(motiondiff,cv2.COLOR_BGR2GRAY) # convert to gray scale
        blur = cv2.GaussianBlur(gray, (5,5), 0) # Blur to reduce noise
        thresh, result = cv2.threshold(blur, 15, 255, cv2.THRESH_BINARY) # Convert to Binary
        dilation = cv2.dilate(result,None,iterations = 3) # Enlarge the human image area

  contours, hierarchy = cv2.findContours(dilation, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
        cv2.drawContours(frame1, contours, -1, (0,255,0), 2)
        cv2.imshow("Motion detection", frame1) # Draw contour
        frame1 = frame2
        check, frame2 = cap.read() # Display contour lines to the next frame.
  if cv2.waitKey(1) & 0xFF == ord("e"):
            break
    else :
        break
cap.release()
cv2.destroyAllWindows()
```

27. Motion Tracker

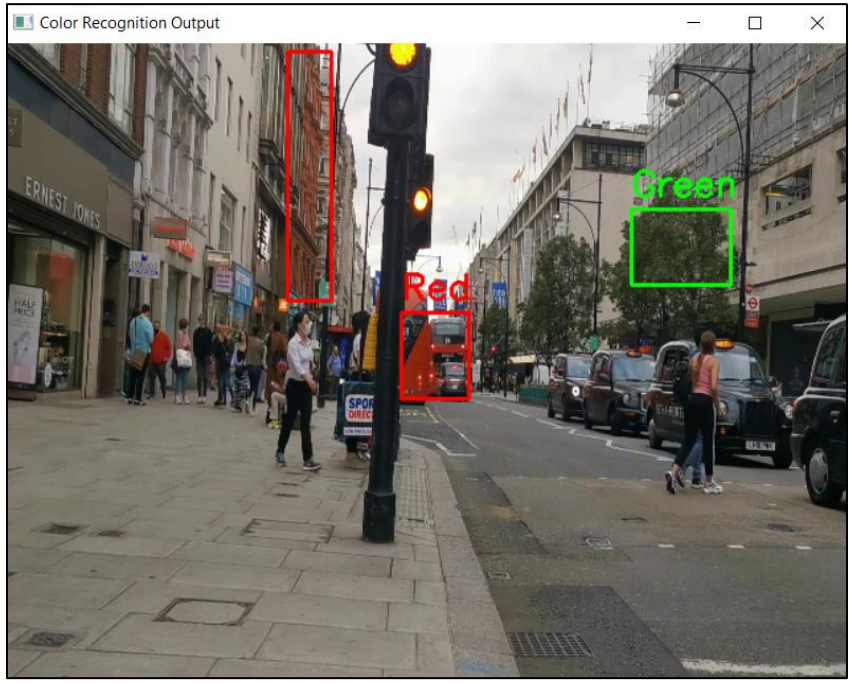
```
import cv2
cap = cv2.VideoCapture("walking.mp4")

check , frame1 = cap.read()
check , frame2 = cap.read()

while (cap.isOpened()):
    if check == True :
        motiondiff = cv2.absdiff(frame1,frame2)
        gray = cv2.cvtColor(motiondiff,cv2.COLOR_BGR2GRAY)
        blur = cv2.GaussianBlur(gray, (5,5), 0)
        thresh, result = cv2.threshold(blur, 15, 255, cv2.THRESH_BINARY)
        dilation = cv2.dilate(result,None,iterations = 3)
        contours, hierarchy = cv2.findContours(dilation, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
        # Draw regtangle on moving human
        for contour in contours:
            # read coordinate in contour
            (x,y,w,h) = cv2.boundingRect(contour)
          cv2.rectangle(frame1, (x,y), (x+w, y+h), (0,255,0), 2)
       cv2.imshow("Motion detection", frame1)
       frame1 = frame2
        check, frame2 = cap.read()
        if cv2.waitKey(1) & 0xFF == ord("e"):
            break
     else :
        break
cap.release()
cv2.destroyAllWindows()
```

```
# Draw a green square box on the moved coordinates. Set the size of the square box.
import cv2
cap = cv2.VideoCapture("walking.mp4")
check , frame1 = cap.read()
check , frame2 = cap.read()

while (cap.isOpened()):
    if check == True :
        motiondiff = cv2.absdiff(frame1,frame2)
        gray = cv2.cvtColor(motiondiff,cv2.COLOR_BGR2GRAY)
        blur = cv2.GaussianBlur(gray, (5,5), 0)
        thresh, result = cv2.threshold(blur, 15, 255, cv2.THRESH_BINARY)
        dilation = cv2.dilate(result,None,iterations = 3)
        contours, hierarchy = cv2.findContours(dilation, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
        for contour in contours:
            (x,y,w,h) = cv2.boundingRect(contour)
            # Define Rectangle size
            if cv2.contourArea(contour) < 1500:
                continue
            cv2.rectangle(frame1, (x,y), (x+w, y+h), (0,255,0), 2)
            cv2.imshow("Motion detection", frame1)
            frame1 = frame2
            check, frame2 = cap.read()
            if cv2.waitKey(1) & 0xFF == ord("e"):
                break
    else :
        break
cap.release()
cv2.destroyAllWindows()
```

28. Color Detection



Image



VDO / Webcam

28.1 Image Color Detection

```
import cv2
import numpy as np
img = cv2.imread("colorballs.jpg")
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

# define range of red color in HSV
lower_red = np.array([0, 50, 50])
upper_red = np.array([10, 255, 255])
# define range of green color in HSV
lower_green = np.array([40, 20, 50])
upper_green = np.array([90, 255, 255])
# define range of blue color in HSV
lower_blue = np.array([100, 50, 50])
upper_blue = np.array([130, 255, 255])
# create a mask for red color
mask_red = cv2.inRange(hsv, lower_red, upper_red)
# create a mask for green color
mask_green = cv2.inRange(hsv, lower_green, upper_green)
# create a mask for blue color
mask_blue = cv2.inRange(hsv, lower_blue, upper_blue)
# find contours in the red mask
contours_red, _ = cv2.findContours(mask_red, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
# find contours in the green mask
contours_green, _ = cv2.findContours(mask_green, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
# find contours in the blue mask
contours_blue, _ = cv2.findContours(mask_blue, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
```

```
# Continue …
# loop through the red contours and draw a rectangle around them
for cnt in contours_red:
    contour_area = cv2.contourArea(cnt)
    if contour_area > 1000:
        x, y, w, h = cv2.boundingRect(cnt)
        cv2.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), 2)
        cv2.putText(img, 'Red', (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 2)
# loop through the green contours and draw a rectangle around them
for cnt in contours_green:
    contour_area = cv2.contourArea(cnt)
    if contour_area > 1000:
        x, y, w, h = cv2.boundingRect(cnt)
        cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
        cv2.putText(img, 'Green', (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)
# loop through the blue contours and draw a rectangle around them
for cnt in contours_blue:
    contour_area = cv2.contourArea(cnt)
    if contour_area > 1000:
        x, y, w, h = cv2.boundingRect(cnt)
        cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)
        cv2.putText(img, 'Blue', (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 0, 0), 2)

# Display final output for multiple color detection opencv python
cv2.imshow('image', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

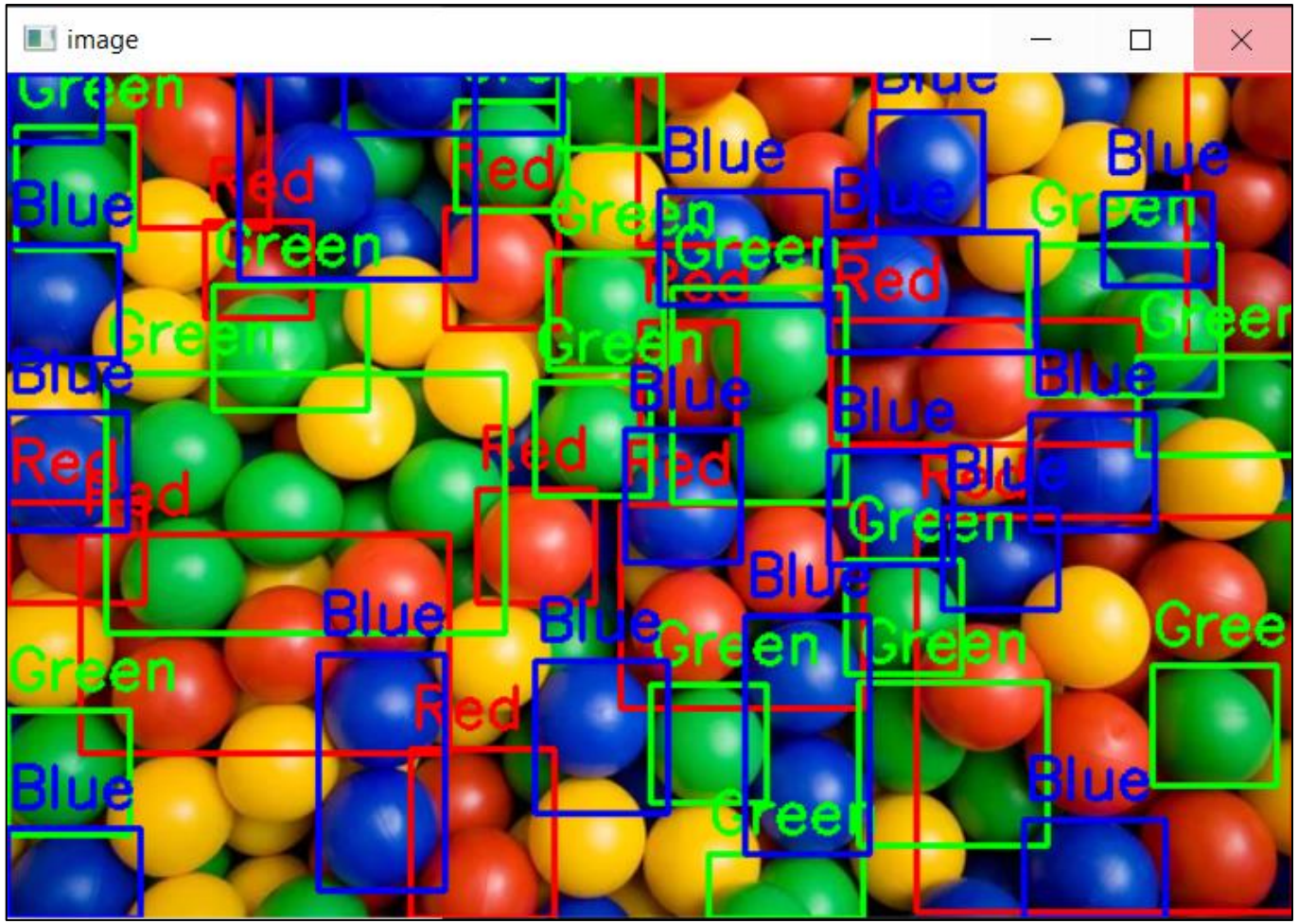


image
Blue
Green
Red

28.2 VDO Color Detection

```
import cv2
import numpy as np

cap = cv2.VideoCapture('street.mp4')
while True:
    check, img = cap.read()
    img_bcp = img.copy()
    img = cv2.resize(img, (640, 480))
    # Make a copy to draw contour outline
    input_image_cpy = img.copy()
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    # define range of red color in HSV
    lower_red = np.array([0, 50, 50])
    upper_red = np.array([10, 255, 255])
    # define range of green color in HSV
    lower_green = np.array([40, 20, 50])
    upper_green = np.array([90, 255, 255])
    # define range of blue color in HSV
    lower_blue = np.array([100, 50, 50])
    upper_blue = np.array([130, 255, 255])
    # create a mask for red color
    mask_red = cv2.inRange(hsv, lower_red, upper_red)
    # create a mask for green color
    mask_green = cv2.inRange(hsv, lower_green, upper_green)
    # create a mask for blue color
    mask_blue = cv2.inRange(hsv, lower_blue, upper_blue)
```

```
# Continue …
    # find contours in the red, green and blue masks
    contours_red, _ = cv2.findContours(mask_red, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    contours_green, _ = cv2.findContours(mask_green, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    contours_blue, _ = cv2.findContours(mask_blue, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    # loop through the red, green and blue contours and draw a rectangle around them
    for cnt in contours_red:
        contour_area = cv2.contourArea(cnt)
        if contour_area > 1000:
            x, y, w, h = cv2.boundingRect(cnt)
            cv2.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), 2)
            cv2.putText(img, 'Red', (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 2)
    for cnt in contours_green:
        contour_area = cv2.contourArea(cnt)
        if contour_area > 1000:
            x, y, w, h = cv2.boundingRect(cnt)
            cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
            cv2.putText(img, 'Green', (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)
    for cnt in contours_blue:
        contour_area = cv2.contourArea(cnt)
        if contour_area > 1000:
            x, y, w, h = cv2.boundingRect(cnt)
            cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)
            cv2.putText(img, 'Blue', (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 0, 0), 2)
    cv2.imshow('Color Recognition Output', img)
    if cv2.waitKey(1) & 0xFF == ord('e'):
        break
cap.release()
cv2.destroyAllWindows()
```


Color Recognition Output
Green
Red


Color Recognition Output
Green
Red

Assignment 3 : Real-time Color detection

Build accurate real-time multi-colors object detection

- Detect objects in real-time using a webcam.
- Detect objects by distinguishing them by color, at least 3 colors.
- Draw a frame around the object, specifying the color of the frame to match the color of the object, and write a message naming the color.
- Can count the number of objects that appear in each color.


SUT
SURANAREE
UNIVERSITY OF
TECHNOLOGY
INSTITUTE OF
ENGINEERING
Mecha tronics
Modern Automotive