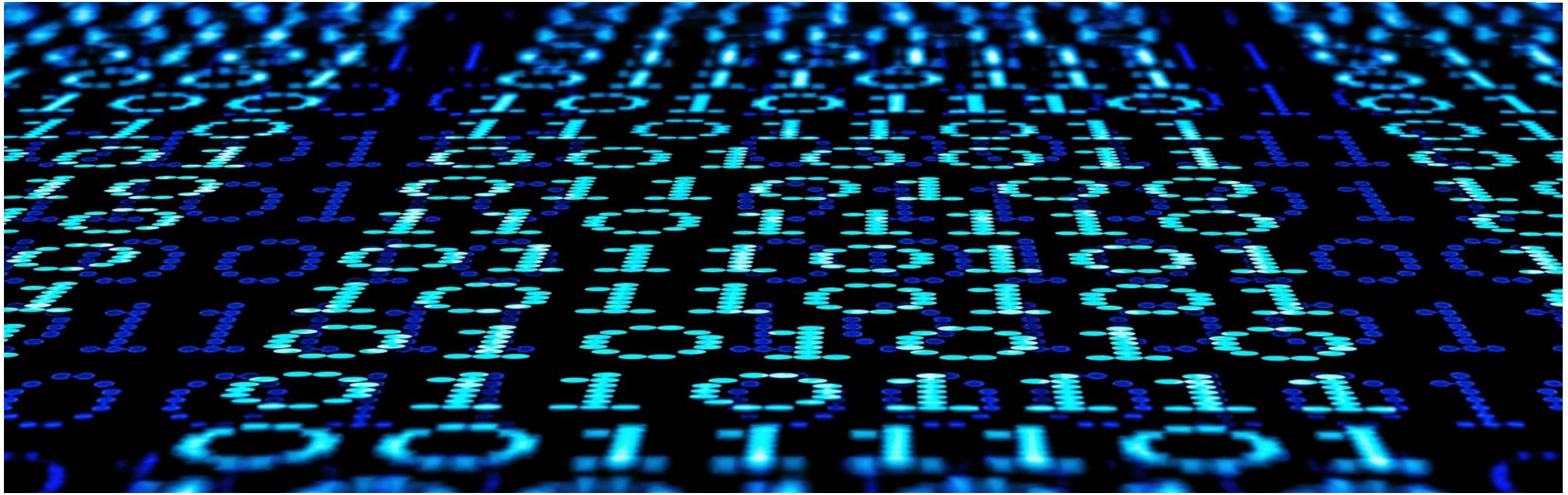
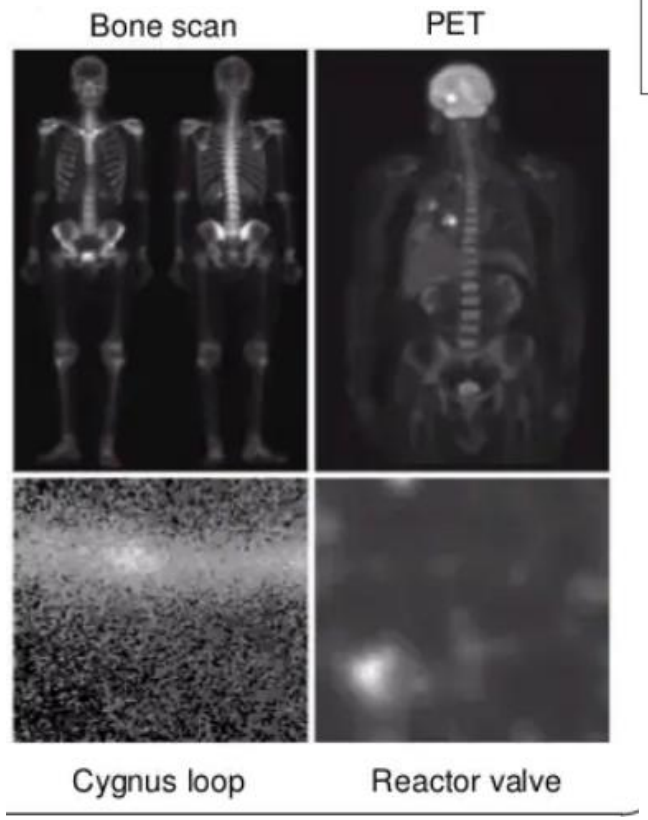


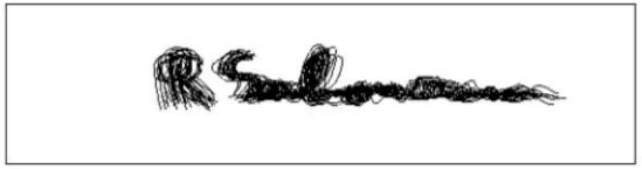
BASIC COMPUTER VISION FOR AUTOMATION ENGINEERING

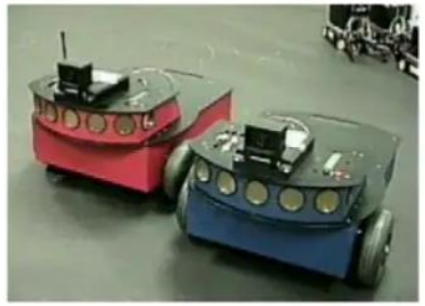
PHUWANAT PHUEAKTHONG

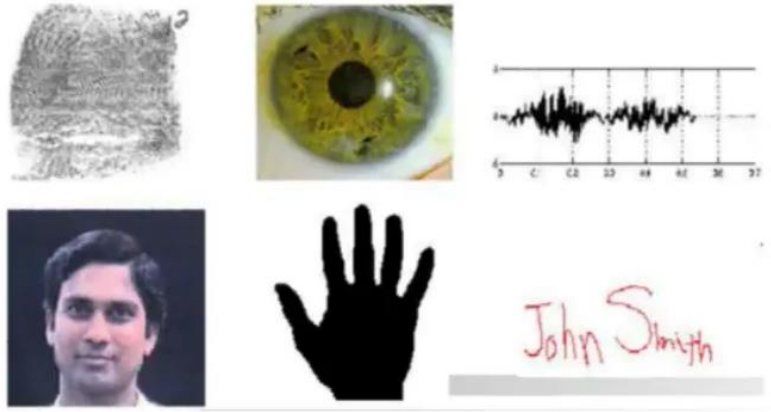
SURANAREE UNIVERSITY OF TECHNOLOGY

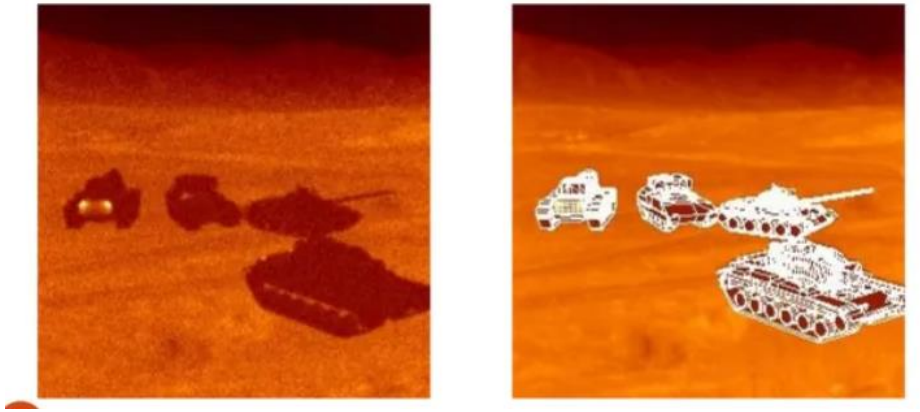
COMPUTER VISION APPLICATION











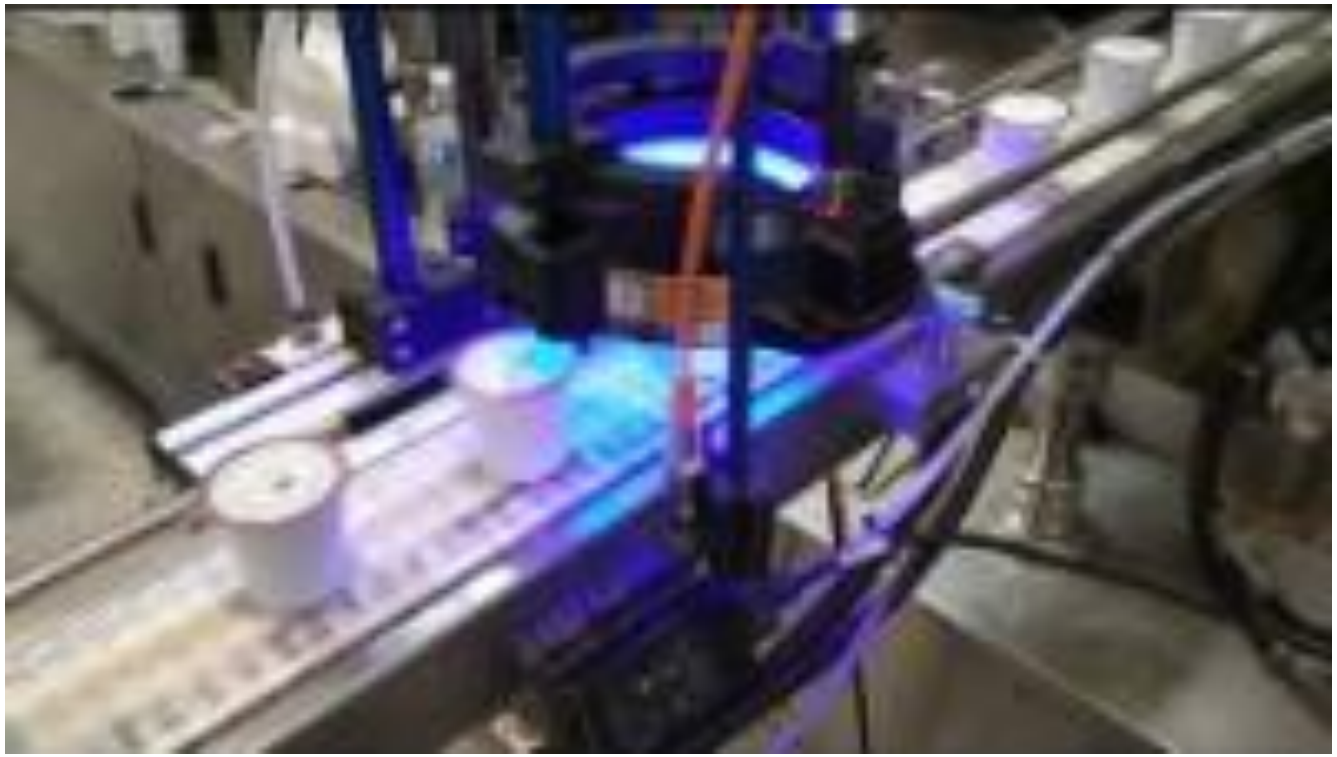
COMPUTER VISION WITH AUTOMATION ENGINEERING

Computer vision with automation machine

Quality control : https://www.youtube.com/watch?v=pNpmFYgEtRk

COMPUTER VISION WITH AUTOMATION ENGINEERING

Computer vision with automation machine



https://www.youtube.com/watch?v=poLXa7aerqM

COMPUTER VISION WITH AUTOMATION ENGINEERING

Computer vision with robot arm

https://www.youtube.com/watch?v=gjfhlysedFE

COMPUTER VISION WITH AUTOMATION ENGINEERING

Computer vision with robot arm

https://www.youtube.com/watch?v=hthdcTOLkyE

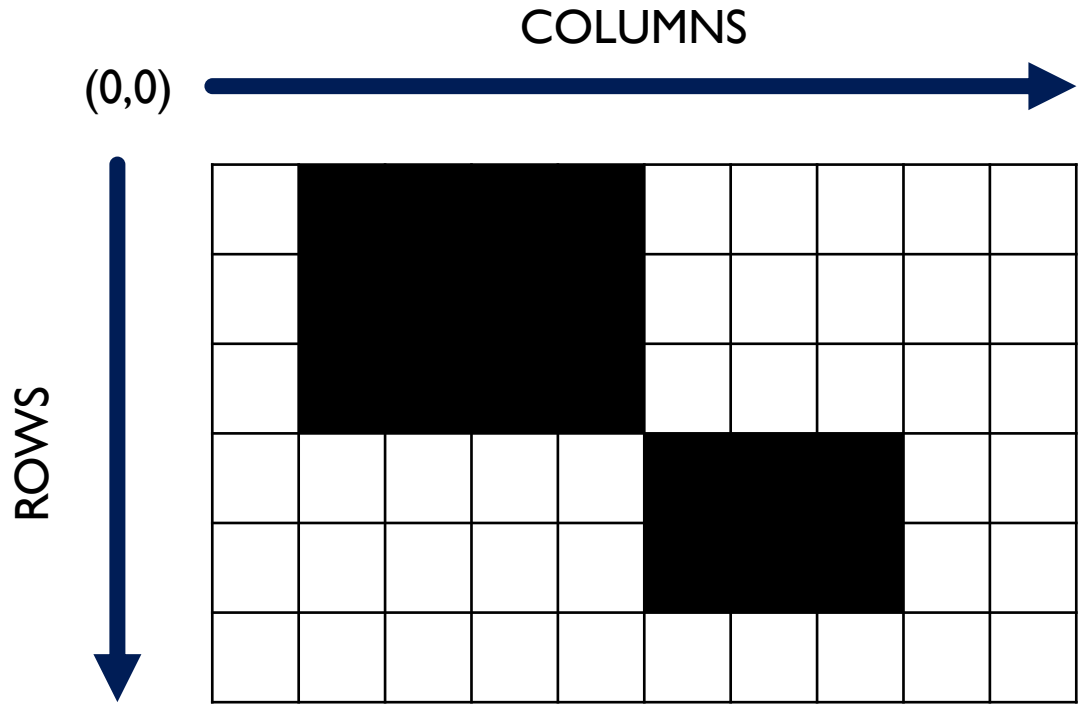
COMPUTER VISION WITH AUTOMATION ENGINEERING

Computer vision with robot arm

https://youtu.be/7NGfkXuIYCE

PIXELS

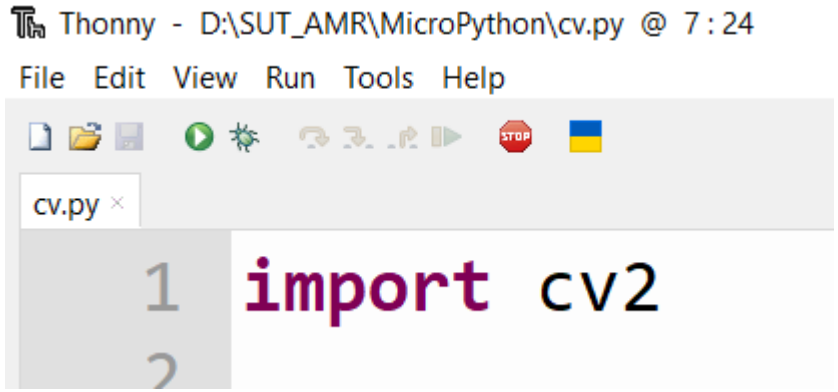
- Pixels are the raw building blocks of an image. There is no finer granularity than the pixel.
- An image with 500 x 300 resolution has 150,000 pixels



INSTALL RELATED LIBRARY

- Install OpenCV with command :
- pip install opencv-python
- pip install opencv-contrib-python
- pip install matplotlib

Test with code



No error = Work

READ IMAGE FROM OPENCV

- Write a code to read image to openCV

```
import cv2

img = cv2.imread("D:\AutonomousFLightEngineer\OpenCV\m1.jpg")

cv2.imshow("Meow", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Path of image

Image show function

Wait for key pressing function

Destroy all OpenCV windows

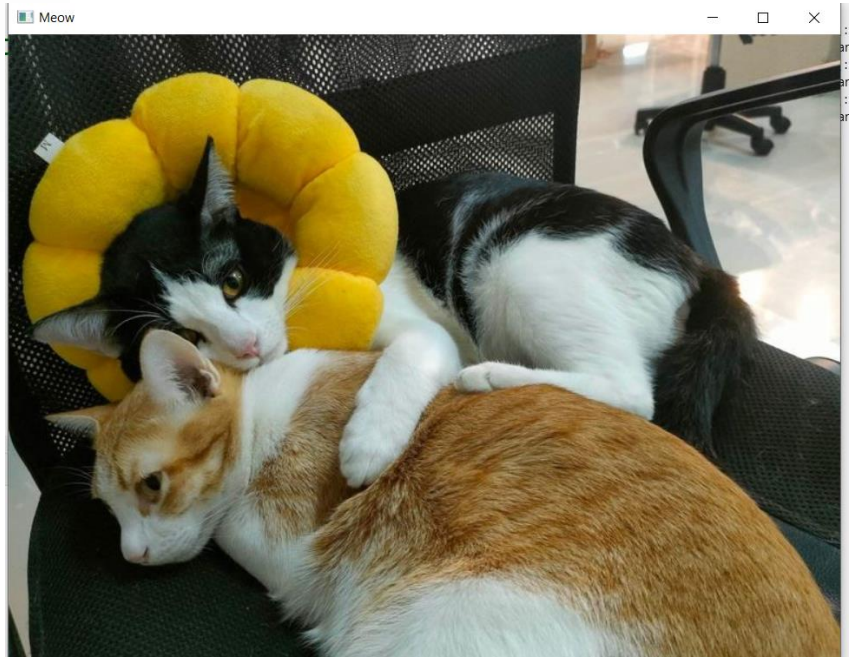
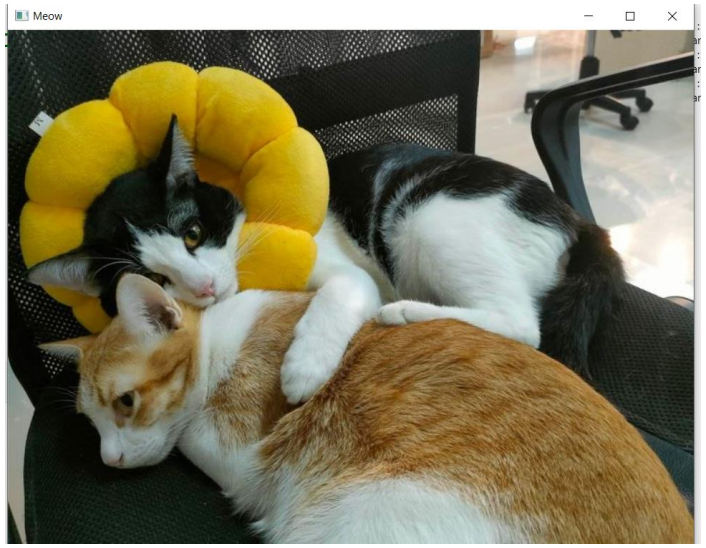
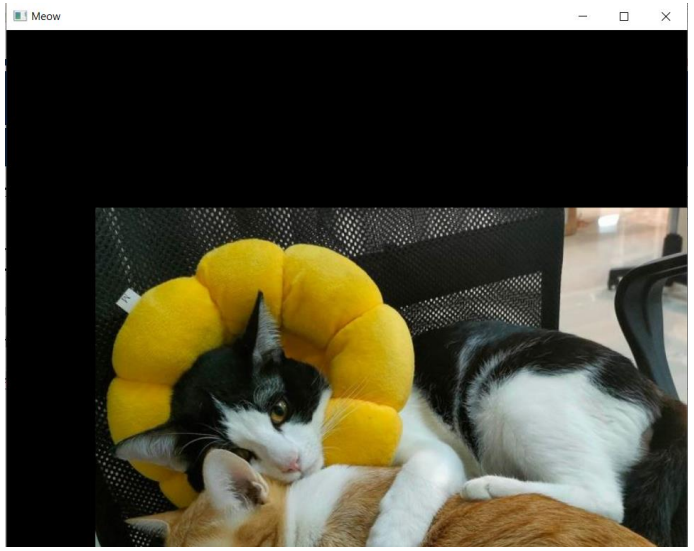


IMAGE TRANSFORMATION : TRANSLATION

```
M = np.float32([[1,0,25],[0,1,50]])
output = cv2.warpAffine(src ,M , dsize)
```

- src = input image
- M = transformation matrix [x y pixel]
- dsize = ouput size (columns, rows)





$$M = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$$

```
rows,cols,_ = img.shape

M = np.float32([[1,0,100],
                [0,1,200]])
trans = cv2.warpAffine(img, M,(cols,rows))
```

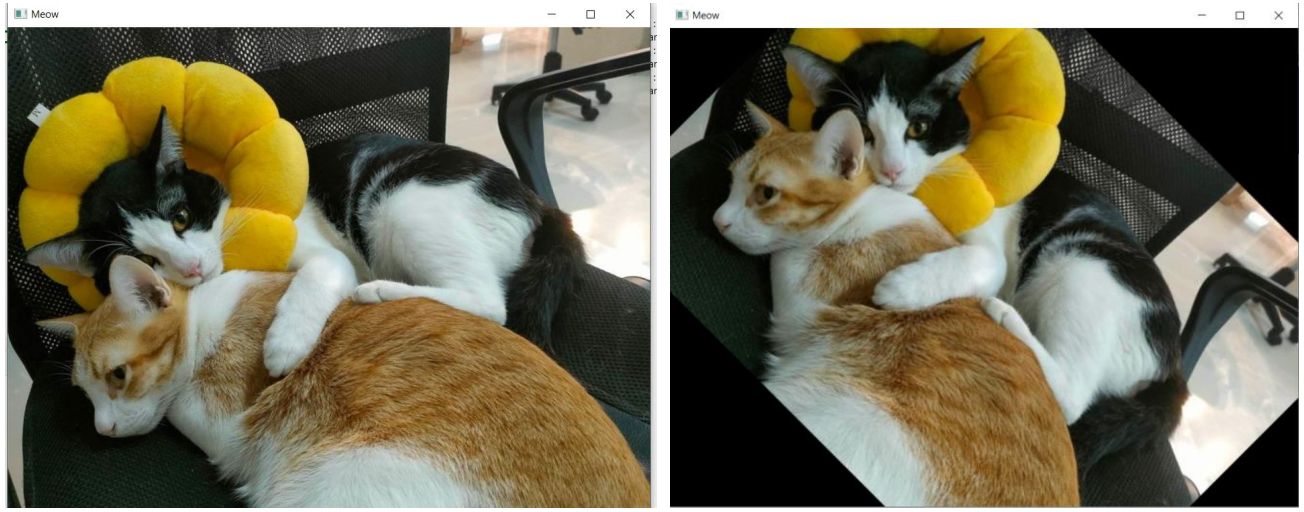
IMAGE TRANSFORMATION : ROTATION

Center = (w/2,h/2)

M = cv2.getRotationMatrix2D(center, deg, scale)

output = cv2.warpAffine(src ,M , dsize)

- src = input image
- m = transformation matrix
- dsize = ouput size (columns, rows)
- deg = rotation angle
- Scale = scaling factor

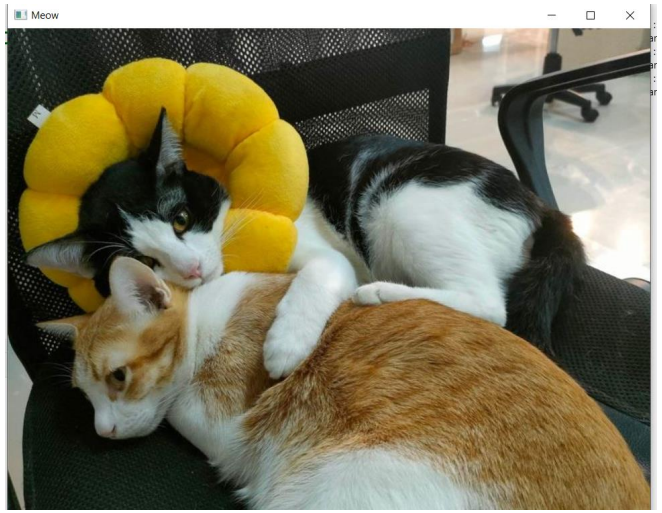


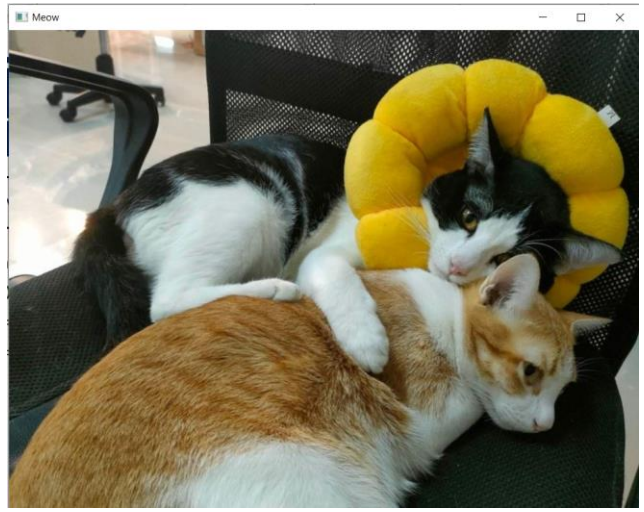

```
rows,cols,_ = img.shape
center = (cols/2, rows/2)

M = cv2.getRotationMatrix2D(center, -45.0, 1.0)
rot = cv2.warpAffine(img, M,(cols,rows))
```

IMAGE TRANSFORMATION : FLIPPING

flipped = cv2.flip(src, code)

- src = image source
- code : 0 = vertically flipping
  1 = Horizontally flipping
  -1 = Both axis flipping





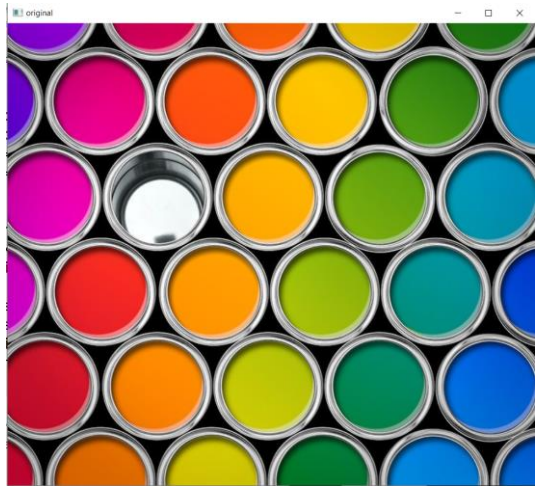

```
flipped = cv2.flip(img, 1)
```

RESIZE

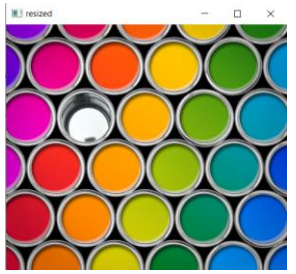
output = cv2.resize(src, dsize, fx = 1, fy = 1, interpolation= cv2.INTER_LINEAR)

- src = source image
- dsize = (w,h)
- fx = horizontal scale factor
- fy = vertical scale factor

```
import cv2
import numpy as np
img = cv2.imread("D:\AutonomousFLightEngineer\OpenCV\color.jpg")
resized = cv2.resize(img, None, fx = 0.5, fy = 0.5,
                     interpolation= cv2.INTER_LINEAR)

cv2.imshow("resized", resized)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Original



Resized

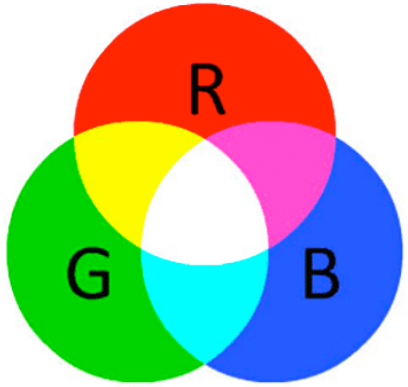
COLOR SPACE

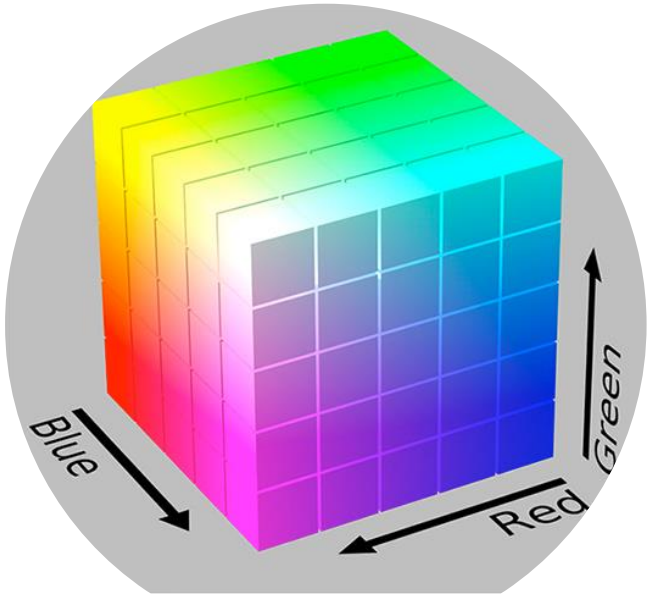
Like different coordinate system in mathematics, an image can store it color in different forms.

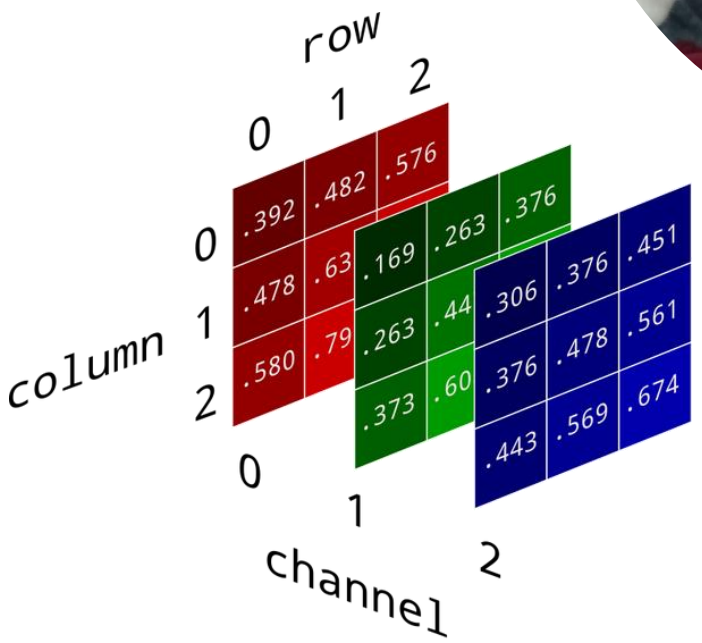
- Grayscale
- RGB (Red Green Blue)
- CMYK (Cyan Magenta Yellow Key)
- HSV (Hue, Saturation, Value)
- Others

RGB COLOR

- Each pixel is a combination of three values, each representing a color in red, green, and blue channels
- RGB ←→ BGR

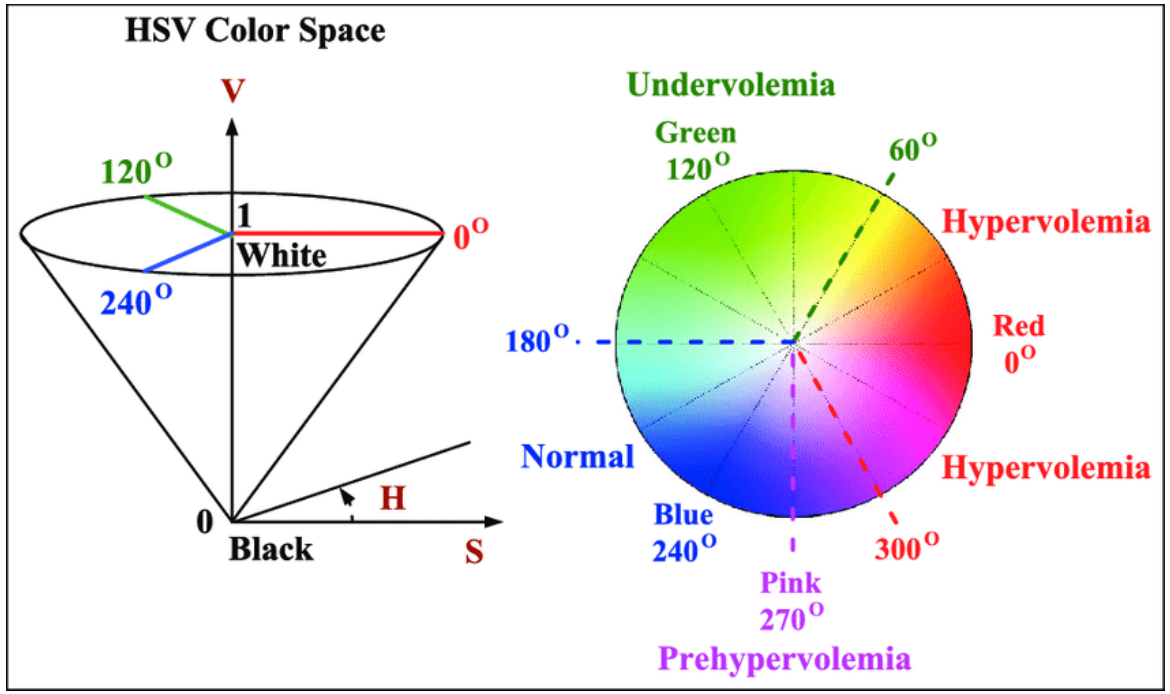






https://www.quora.com/How-do-I-get-RGB-color-matrices-of-an-image-using-MATLAB

HSV COLOR

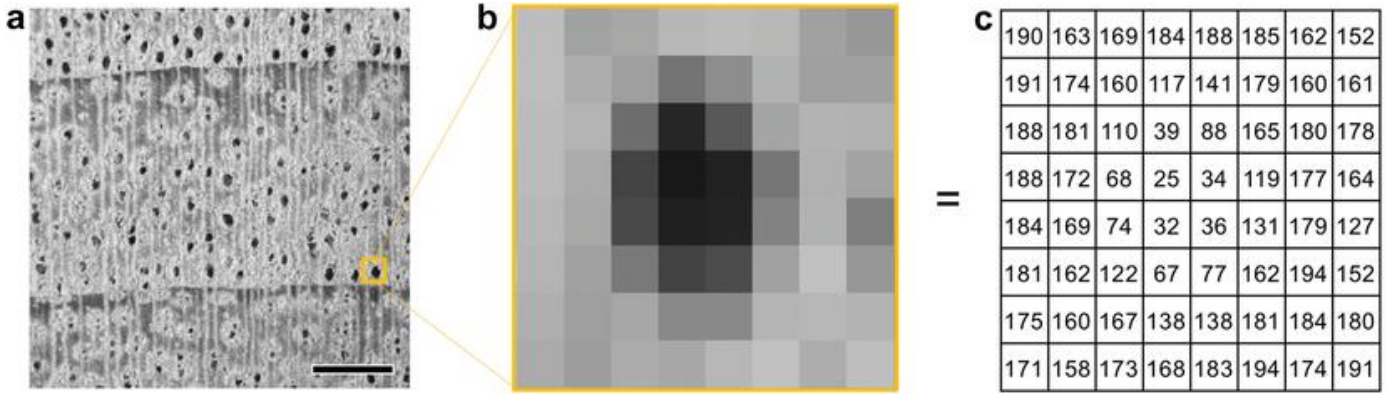


https://www.researchgate.net/figure/HSV-color-space-and-RGB-color-transformation_fig4_312678134

- A cylindrical coordinate system where we project RGB values onto a cylinder.
- There is no clear intuition to how the color progresses in the RGB space
- HSV unit is (0-180 degrees,0-255,0-255) = Hue, Saturation and Value numberss

GRAYSCALE

- One of the simplest color space for computer vision
- A grayscale image store a single value in range 0 (black)-255(white)



(H. Sung-Wook, 2021)

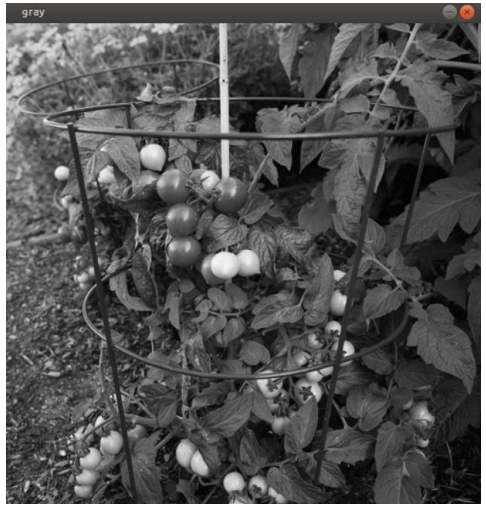
COLOR SPACE CONVERSION

- cv2.cvtColor(image, cv2.COLOR_BGR2HSV) : Convert RGB color to HSV
- cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) : Convert RGB color to HSV

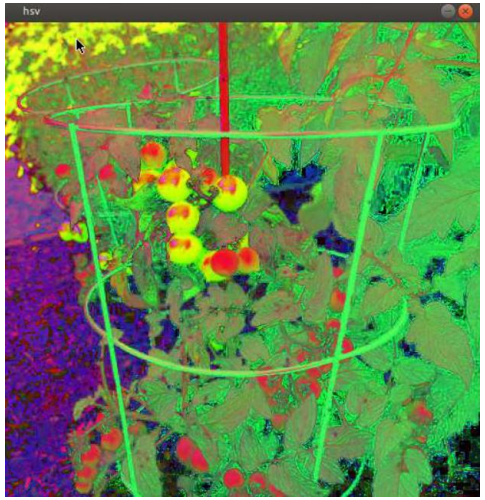
When converting to grayscale, each RGB channel is not weighted uniformly, we weight each channel differently to account for how much color we perceive of each:

$$Y = 0.299R + 0.587G + 0.114B$$

RGB

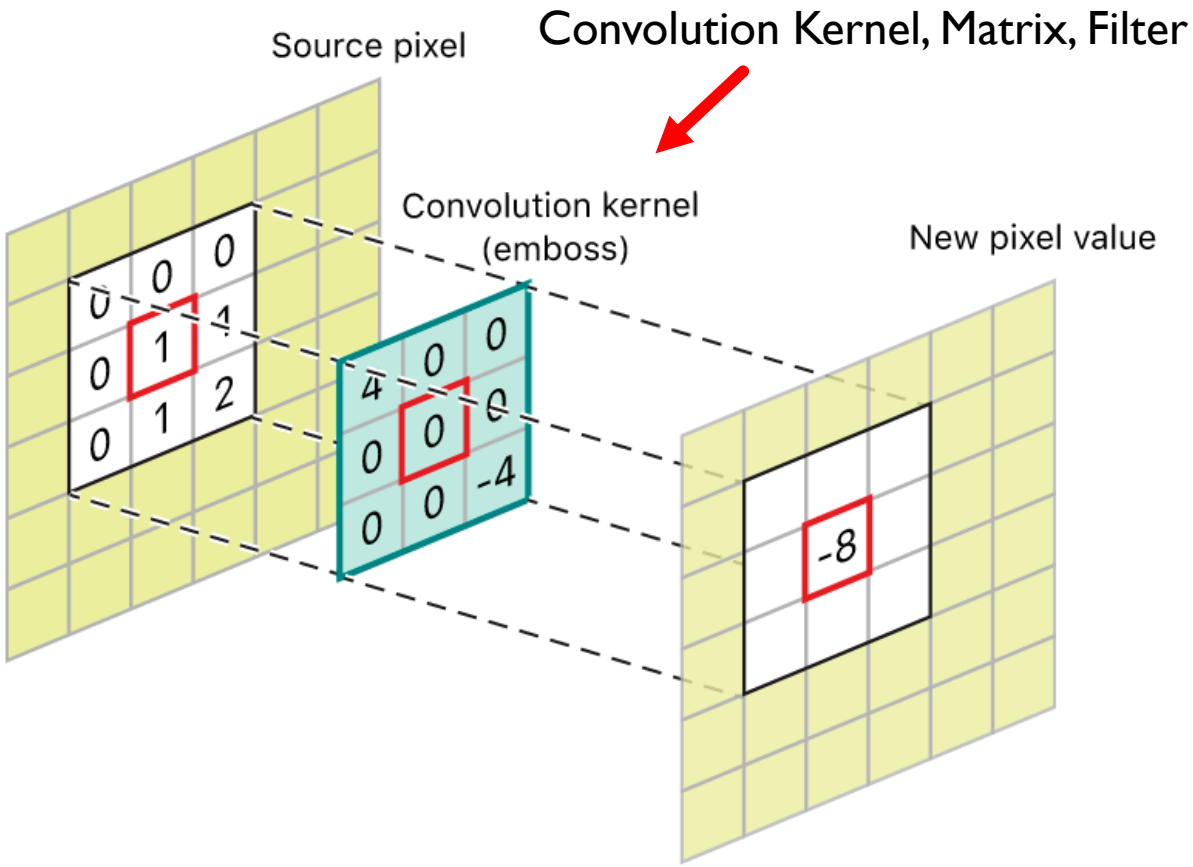


GRAY SCALE



HSV

IMAGE MANIPULATION : CONVOLUTION

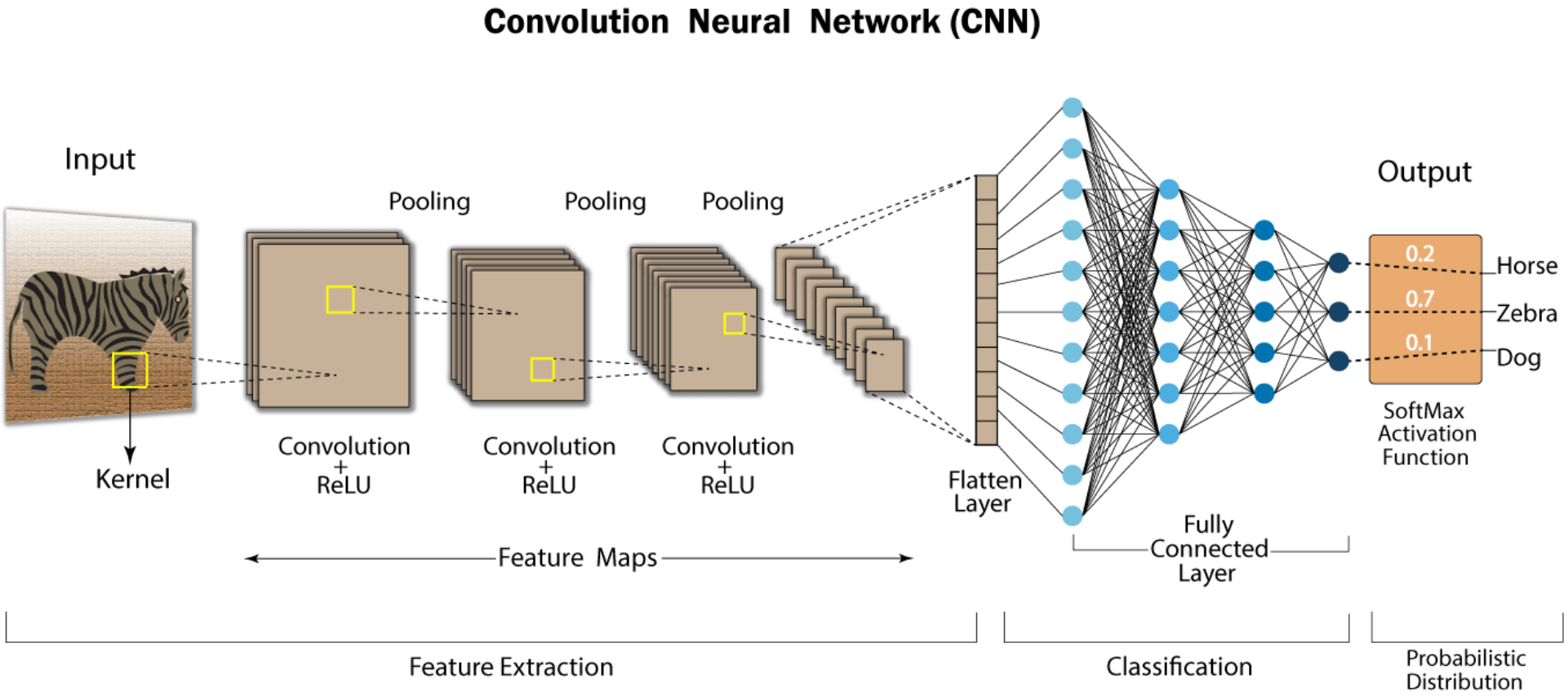


Application of Image Convolution

- Blurring
- Edge detection
- Sharpen
- CNN Feature Extraction
- Etc.

https://developer.apple.com/documentation/accelerate/blurring_an_image

IMAGE MANIPULATION : CONVOLUTION



https://developersbreach.com/convolution-neural-network-deep-learning/

IMAGE MANIPULATION : CONVOLUTION

Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Ridge or edge detection	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	

Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Gaussian blur 5 × 5 (approximation)	$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

https://en.wikipedia.org/wiki/Kernel_(image_processing)

SMOOTHING AND BLURRING

- many image processing and computer vision functions, such as thresholding and edge detection, perform better if the image is first smoothed or blurred.
- Simple blurring (cv2.blur)
- Weighted Gaussian blurring (cv2.GaussianBlur)
- Median filtering (cv2.medianBlur)
- Bilateral blurring (cv2.bilateralFilter)

SMOOTHING AND BLURRING

Original Image

Simple blurring

```
blur = cv2.blur(self.img,(3,3))
```

Weighted Gaussian blurring

```
blur = cv2.blur(self.img,(3,3),0)
```

SMOOTHING AND BLURRING

Original Image

Median filtering

```
blur = cv2.medianBlur(self.img,3)
```

Bilateral blurring

```
blur = cv2.bilateralFilter(self.img,11,41,21)
```

IMAGE THRESHOLDING

- For every pixel, the same threshold value is applied. If the pixel value is smaller than the threshold, it is set to 0, otherwise it is set to a maximum value. The function cv2.threshold is used to apply the thresholding.

```
ret,thresh1 = cv.threshold(src,thr,max,type)
```

src = source image

the = threshold value

max = maximum value

type = thresholding type

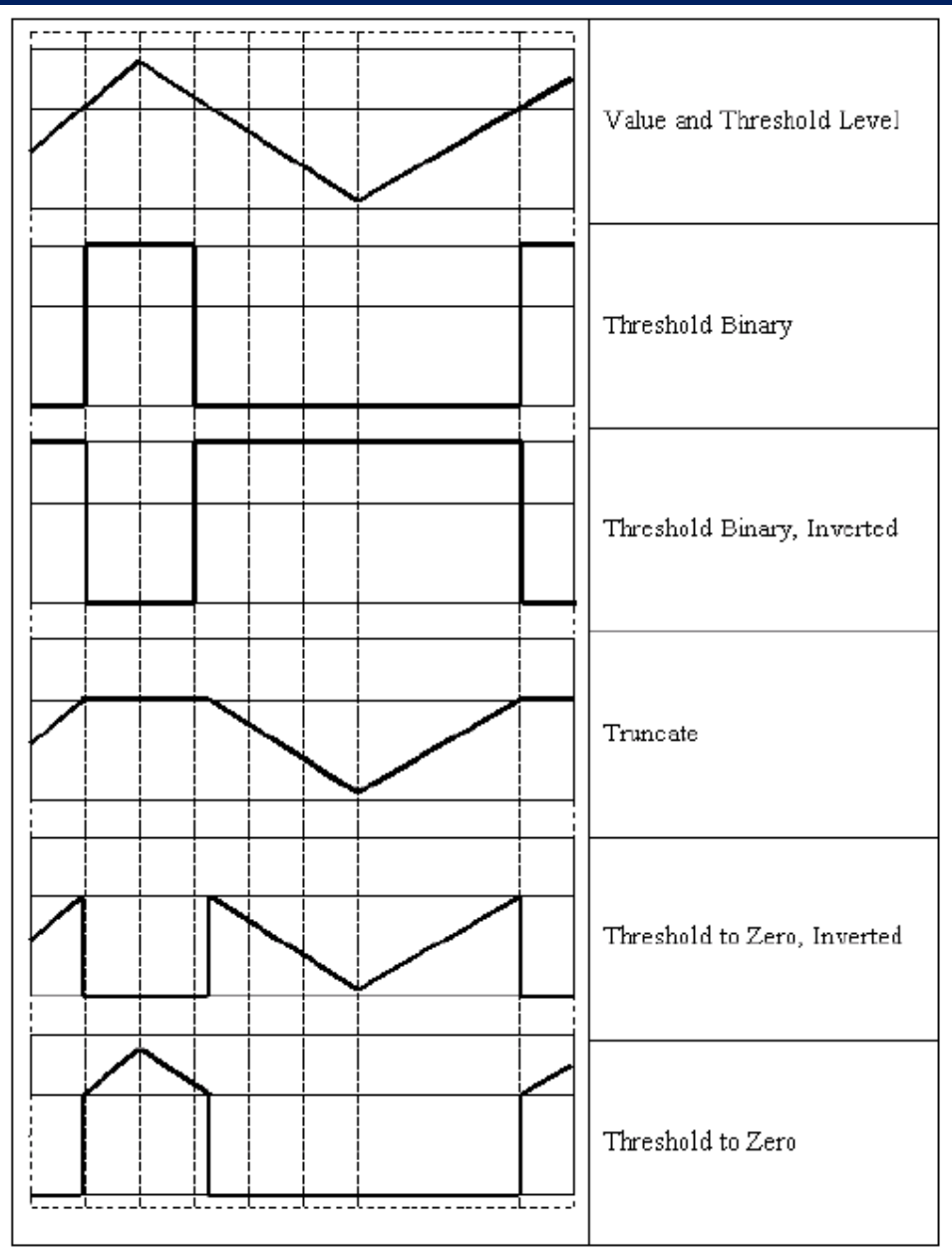
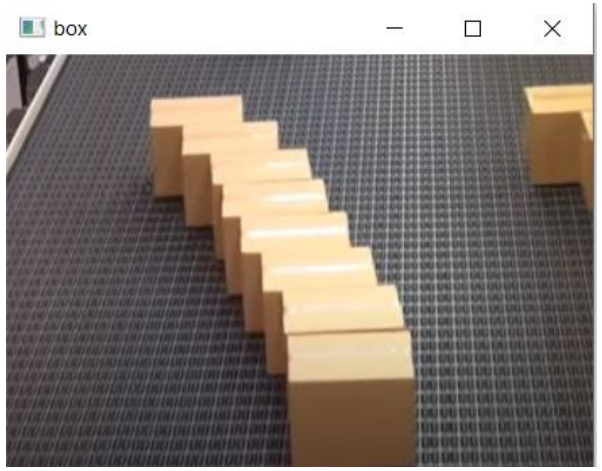
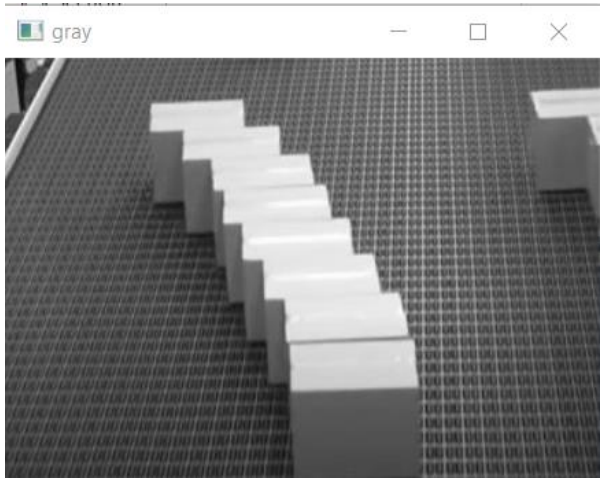


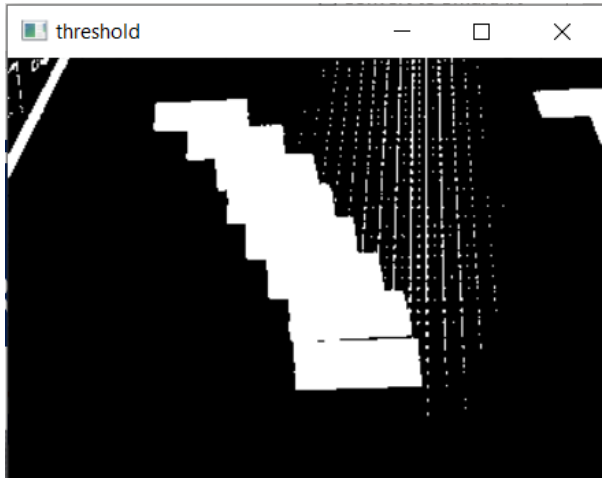
IMAGE THRESHOLDING



Original Image



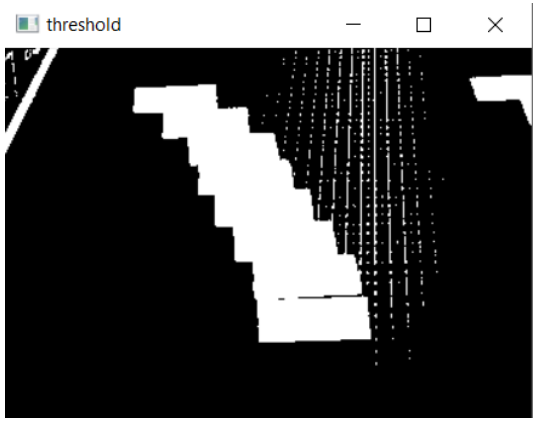
Grayscale Image



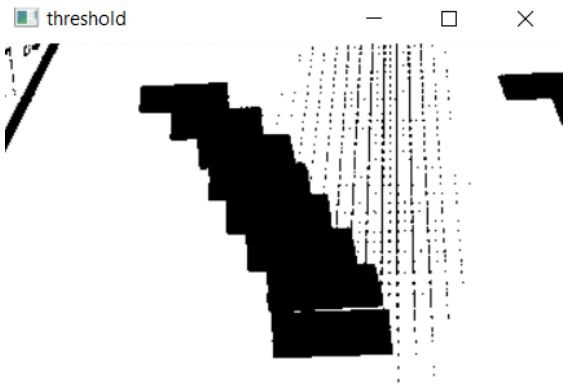
Binary Thresholding Image

```
img = cv2.imread("D:\AutonomousFLightEngineer\OpenCV\conv.jpg")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
ret,thresh = cv2.threshold(gray, 150, 255, cv2.THRESH_BINARY)

cv2.imshow("gray", gray)
cv2.imshow("threshold", thresh)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

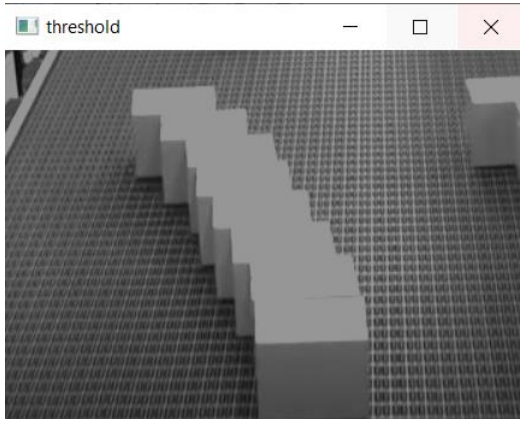
THRESHOLDING



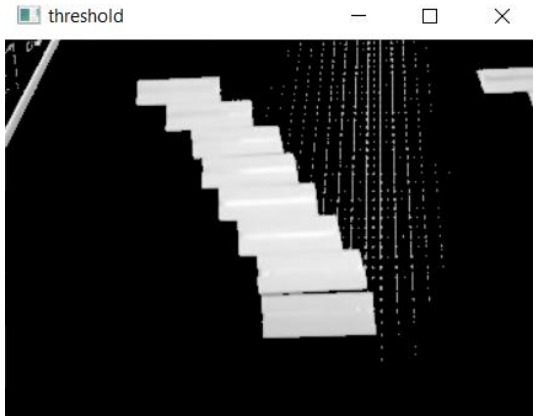

```
ret,thresh = cv2.threshold(gray, 150, 255, cv2.THRESH_BINARY)
```




```
ret,thresh = cv2.threshold(gray, 150, 255, cv2.THRESH_BINARY_INV)
```




```
ret,thresh = cv2.threshold(gray, 150, 255, cv2.THRESH_TRUNC)
```




```
ret,thresh = cv2.threshold(gray, 150, 255, cv2.THRESH_TOZERO)
```

MORPHOLOGICAL TRANSFORMATION

- The morphological operations we'll be covering include:

- Erosion

- Dilation

- Opening

- Closing

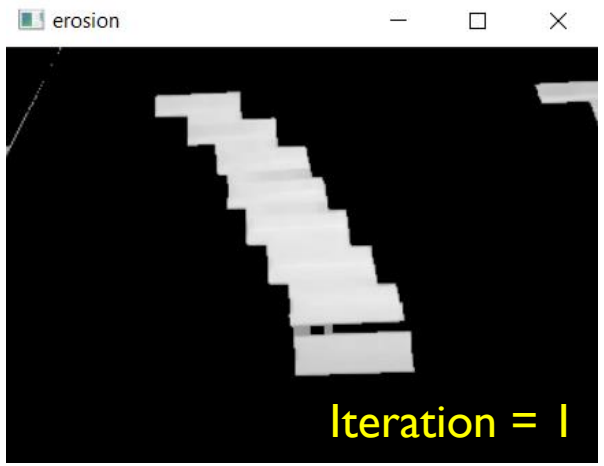
- These image processing operations are applied to grayscale or binary images

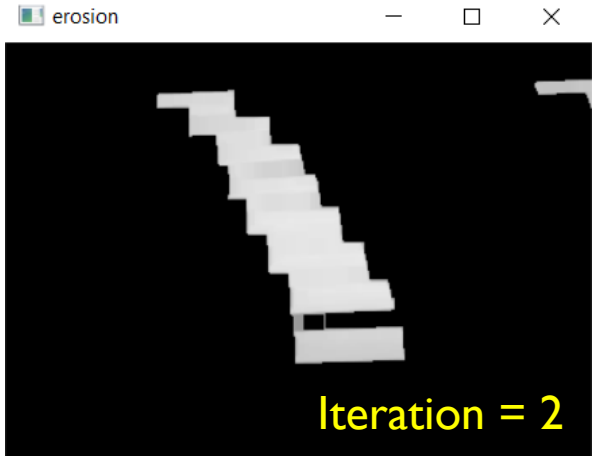
MORPHOLOGICAL TRANSFORMATION : EROSION

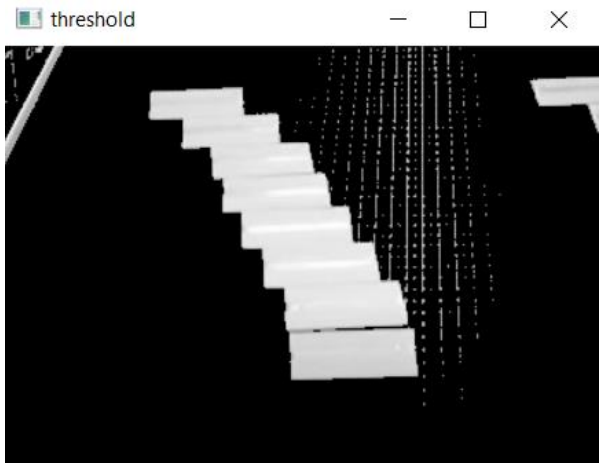
output = cv2.erode(src , kernel, iteration = 1)

- src = source image
- kernel = kernel specification

For erosion : Dark area grow up





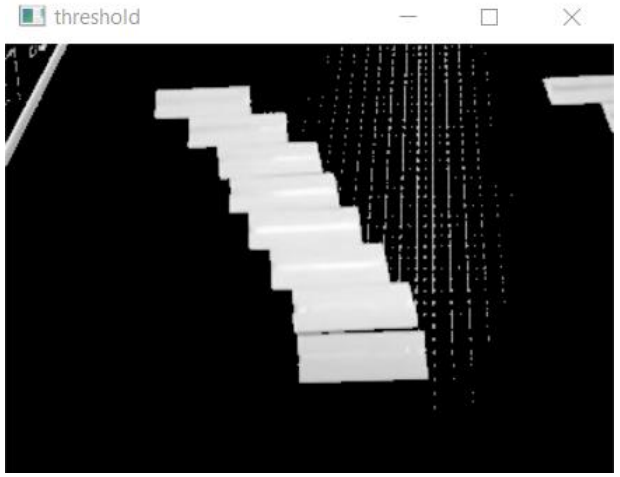



```
img = cv2.imread("D:\AutonomousFLightEngineer\OpenCV\conv.jpg")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
ret,thresh = cv2.threshold(gray, 150, 255, cv2.THRESH_TOZERO)

kernel = np.ones((5, 5), np.uint8)
eroded = cv2.erode(thresh, kernel,iterations = 1)
```

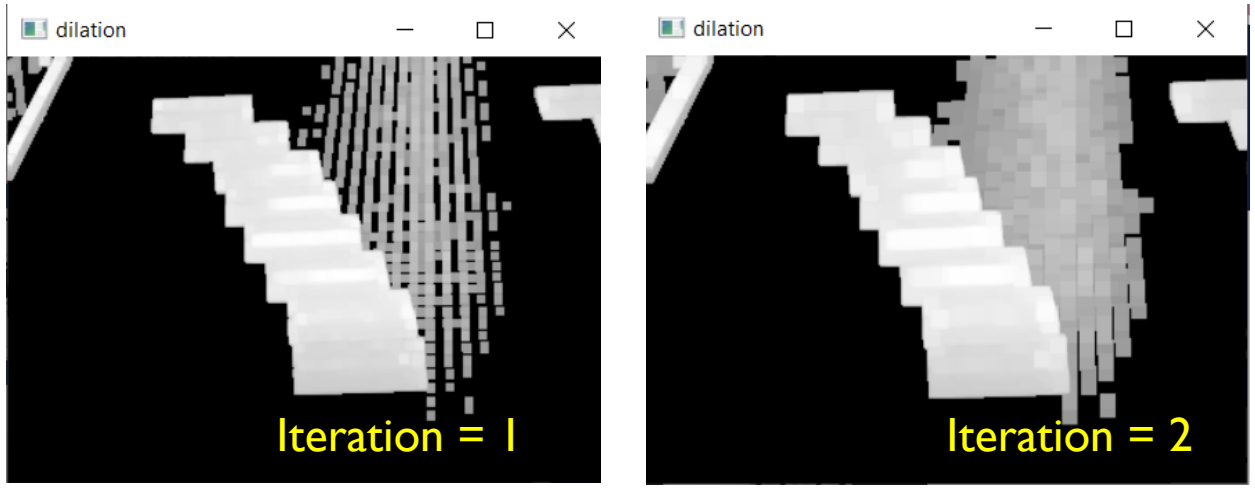
MORPHOLOGICAL TRANSFORMATION : DILATION

output = cv2.dilate(src , kernel, iterations = 1)

- src = source image
- kernel = kernel specification



For dilation : Bright area grow up




```
img = cv2.imread("D:\AutonomousFLightEngineer\OpenCV\conv.jpg")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
ret,thresh = cv2.threshold(gray, 150, 255, cv2.THRESH_TOZERO)

kernel = np.ones((5, 5), np.uint8)
dilated = cv2.dilate(thresh, kernel,iterations = 1)
```

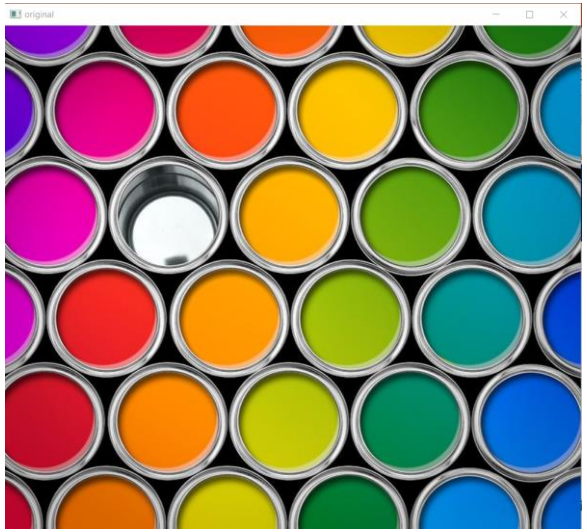
COLOR FILTERING

- One of image segmentation method is color filtering. We can filter interested color from others easily in openCV.

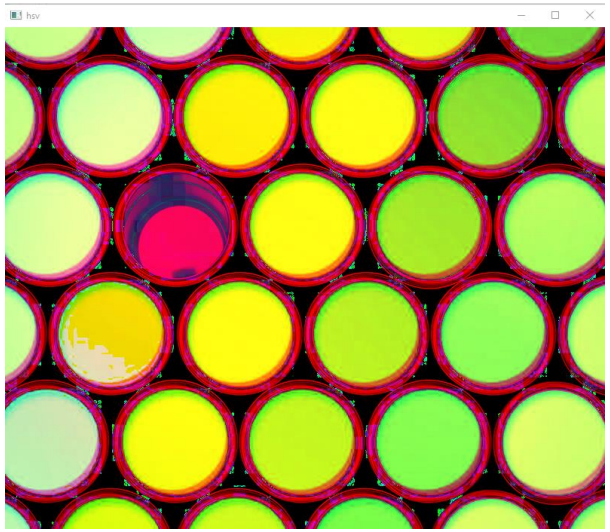
COLOR FILTERING

How to do color filtering

- Find the range of interested color which in the same color space.
  - If color space is RGB → use RGB range
  - If color space is HSV → use HSV range (Recommended)



RGB



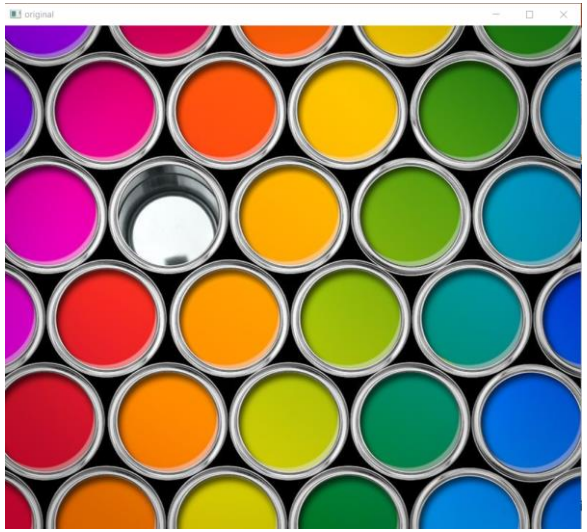
HSV in OpenCV

```
img = cv2.imread("D:\AutonomousFLightEngineer\OpenCV\color.jpg")
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

cv2.imshow("original", img)
cv2.imshow("hsv", hsv)
```

COLOR FILTERING

How to do color filtering

- Find the range of interested color which in the same color space.
  - If color space is RGB → use RGB range
  - If color space is HSV → use HSV range (Recommended)

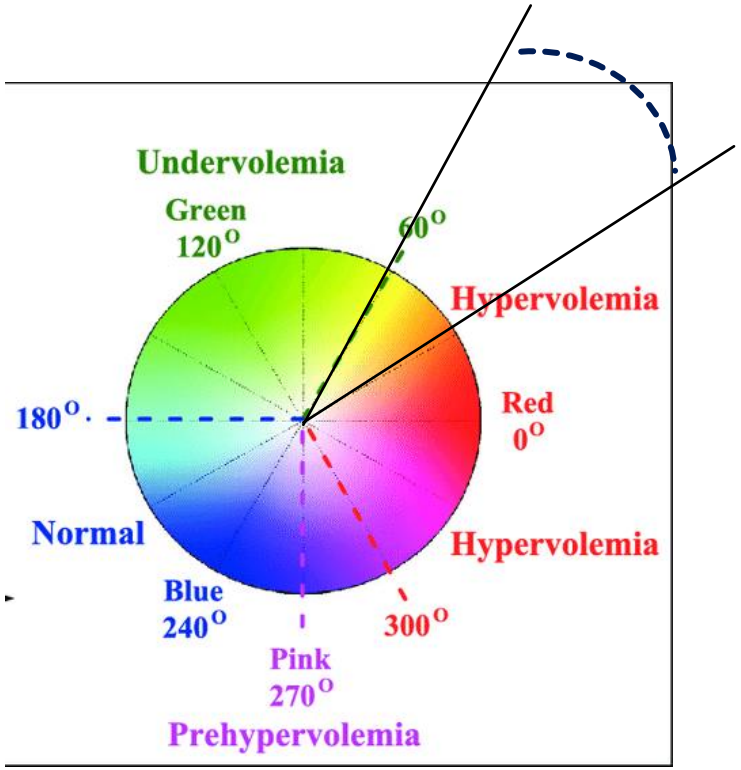


RGB

If target color is Yellow

Range of Hue value is between
30 to 60 degrees

But for HSV in OpenCV, don't forget
to use half of Hue value (15 to 30 degrees)



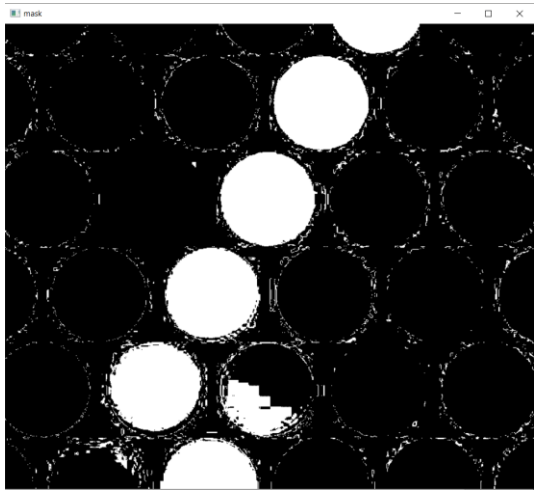
COLOR FILTERING

How to do color filtering

- Create HSV value lower and upper bound.
- Use function cv2.inRange to select the interested color ranges.

> Tip
> If first time setting, set S and V values to 0 and 255 for lower and upper bound, respectively.

```
import cv2
import numpy as np
img = cv2.imread("D:\AutonomousFLightEngineer\OpenCV\color.jpg")
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
lower = np.array([15,0,0])
upper = np.array([30,255,255])

mask = cv2.inRange(hsv,lower,upper)

cv2.imshow("original", img)
cv2.imshow("mask", mask)
cv2.waitKey(0)
```



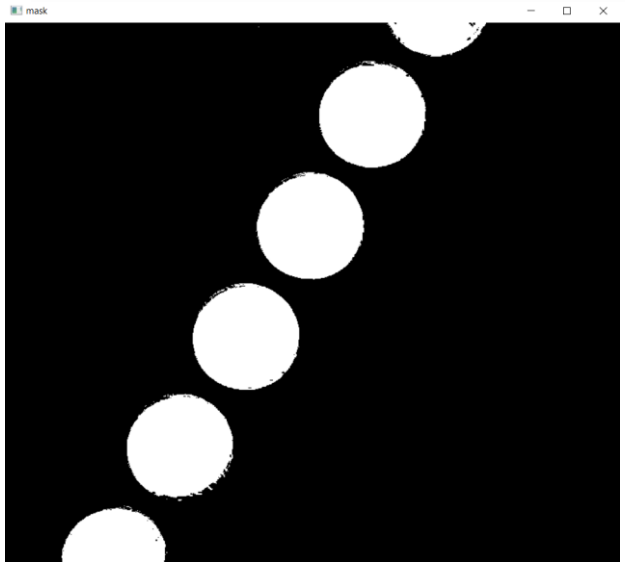
COLOR FILTERING

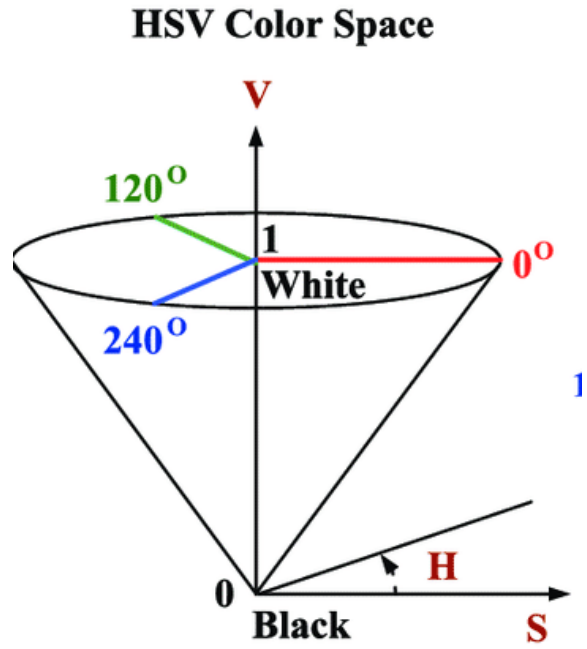
How to do color filtering

- Change Saturate (S) and Value(V) range to get more accurate area.
- Change H value again

```
lower = np.array([15,100,120])
upper = np.array([30,255,255])
```


```
lower = np.array([13,100,120])
upper = np.array([25,255,255])
```





COLOR FILTERING

After interested color is isolated, The white are in "mask" will more than zero pixel (Area > 0).

From this assumption, if there is the object in range of color, we can write a code to send the output as in the following :

```
img = cv2.imread("D:\AutonomousFLightEngineer\OpenCV\color.jpg")
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
lower = np.array([13,100,120])
upper = np.array([25,255,255])

mask = cv2.inRange(hsv,lower,upper)
mask_area = np.sum(mask)
if mask_area > 0:
    print("Yellow Detected")
```

How about multiple color Detection?

↓

Create more ranges (lower1, lower2….)
And masks (mask1, mask2, …)

OPEN CV & WEB CAMERA

- Video capture command in OpenCV

```
cap = cv2.VideoCapture(2)
```

Output Variable

Camera port number

- Get image frame from captured Video

```
ret, frame = cap.read()
```

Image frame

Read image from captured Video

CAPTURING IMAGE FROM WEB CAMERA

- We can save image from capturing with the command **cv2.imwrite("file name")**

Check Ascii Code of Pressed key
Ascii 27 is ESC

os.chdir(path)
used to set directory path for image saving

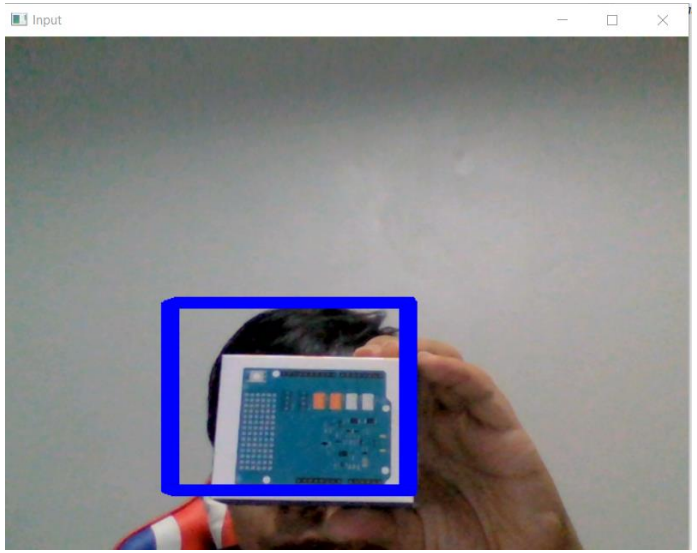
```
import cv2
import os
cap = cv2.VideoCapture(2)
while True:
    ret, frame = cap.read()
    cv2.imshow('Input', frame)
    c = cv2.waitKey(1)
    if c == 27:
        os.chdir("D:\AutonomousFLightEngineer\OpenCV")
        cv2.imwrite("test_cam.jpg",frame)
        break
cap.release()
cv2.destroyAllWindows()
```

TEMPLATE MATCHING

- Template matching in computer vision is the process of finding a smaller image (*template*) in a larger image.
- Template matching uses a **sliding window** method over source image and compare each patch with the template.
- This is helpful in cases when:
  - We want to find a particular object in an image or video frame.
  - We should exactly know what we are looking for.
  - What we want to detect should match the template in most of the cases.

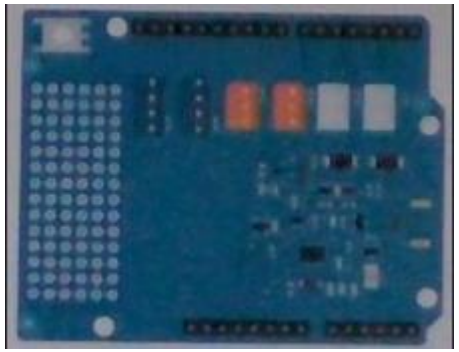
TEMPLATE MATCHING

- Template matching requires template image and matching image.

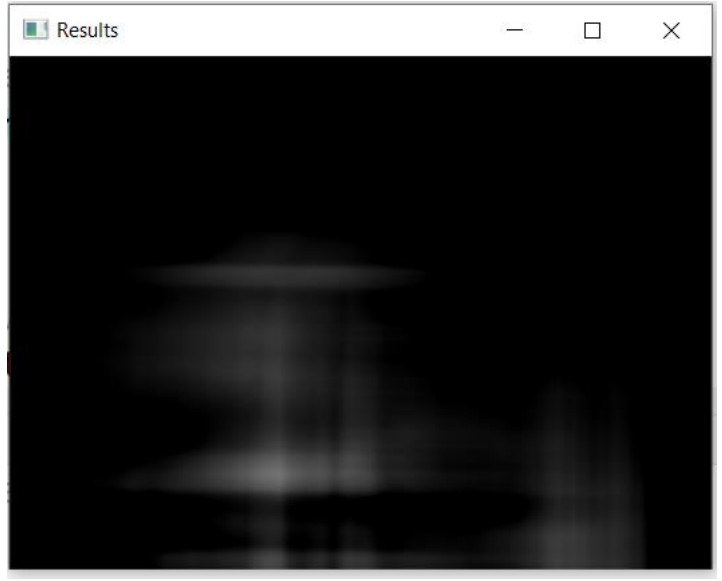


Detected Image Source

Template Image Must smaller than source image



Template Image



Matching Results

TEMPLATE MATCHING

- Template matching requires template image and matching image.

```
res = cv2.matchTemplate(img_gray,template,cv2.TM_CCOEFF_NORMED)
```

Output image — Source image (Gray scale) — Template (Gray scale) — Matching Score Method

cv2.TM_SQDIFF	$R(x,y) = \sum_{x',y'}(T(x',y') - I(x+x',y+y'))^2$
cv2.TM_SQDIFF_NORMED	$R(x,y) = \frac{\sum_{x',y'}(T(x',y') - I(x+x',y+y'))^2}{\sqrt{\sum_{x',y'}T(x',y')^2 \cdot \sum_{x',y'}I(x+x',y+y')^2}}$
cv2.TM_CCORR	$R(x,y) = \sum_{x',y'}(T(x',y') \cdot I(x+x',y+y'))$
cv2.TM_CCORR_NORMED	$R(x,y) = \frac{\sum_{x',y'}(T(x',y') \cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'}T(x',y')^2 \cdot \sum_{x',y'}I(x+x',y+y')^2}}$

Matching Function : I(x,y) = input R(x,y)= result T(x,y) = template

cv2.TM_CCOEFF	$R(x,y) = \sum_{x',y'}(T'(x',y') \cdot I'(x+x',y+y'))$ *where,* $T'(x',y') = T(x',y') - 1/(w \cdot h) \cdot \sum_{x'',y''} T(x'',y'')$ $I'(x+x',y+y') = I(x+x',y+y') - 1/(w \cdot h) \cdot \sum_{x'',y''} I(x+x'',y+y'')$
cv2.TM_CCOEFF_NORMED	$R(x,y) = \frac{\sum_{x',y'}(T'(x',y') \cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'}T'(x',y')^2 \cdot \sum_{x',y'}I'(x+x',y+y')^2}}$

TEMPLATE MATCHING

- Example code for template matching with Real-time web camera image

```
import cv2
import numpy as np
cap = cv2.VideoCapture(2)

while True:
    ret, frame = cap.read()
    img_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    template = cv2.imread('D:\\AutonomousFLightEngineer\\OpenCV\\test_template.jpg',0)
    w, h = template.shape[::-1]

    res = cv2.matchTemplate(img_gray,template,cv2.TM_CCOEFF_NORMED)
```

TEMPLATE MATCHING

- Example code for template matching with Real-time web camera image

```
13      threshold = 0.4
14      loc = np.where(res >= threshold)
15
16      for pt in zip(*loc[::-1]):
17          cv2.rectangle(frame, pt, (pt[0] + w, pt[1] + h), (255,0,0), 1)
18
19      cv2.imshow('Input', frame)
20      cv2.imshow('Results', res)
21      c = cv2.waitKey(1)
22      if c == 27:
23          break
24
25  cap.release()
26  cv2.destroyAllWindows()
```

Define criteria for decision making

Find result area which score more than threshold level

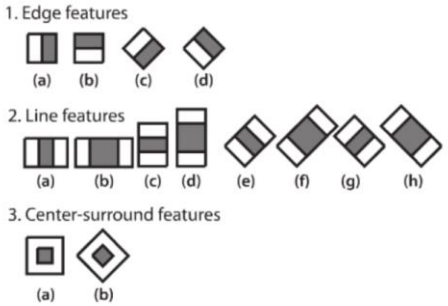
Loop to draw bounding box around detected area

Wait for ESC key to end the process

TEMPLATE MATCHING

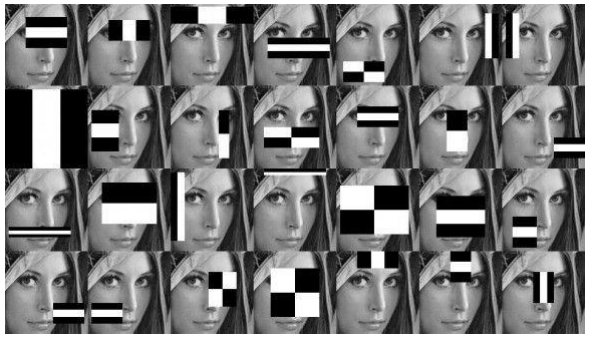
- Better way for object detection

Haar Cascade Classifier



Face Detection determines the locations and sizes of human faces in arbitrary (digital) images.

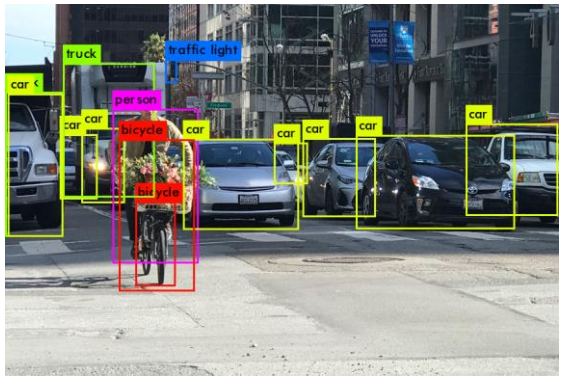
In **Face Recognition**, the use of Face Detection comes first to determine and isolate a face before it can be recognized.



https://becominghuman.ai/face-detection-using-opencv-with-haar-cascade-classifiers-941dbb25177

Deep Learning

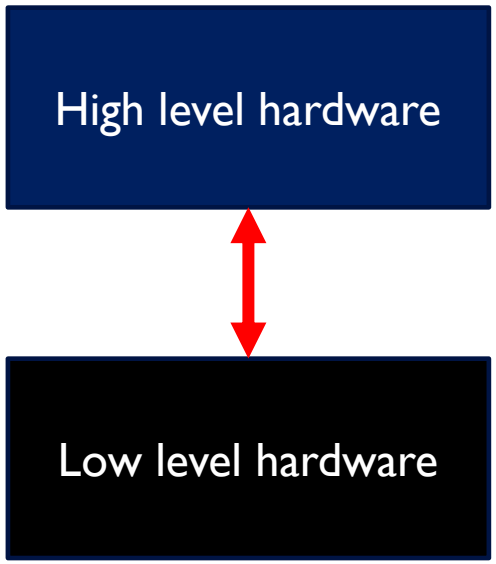
- Convolutional Neural Network (CNN)
- YOLO
- Etc.



https://dagshub.com/blog/yolov6/

COMPUTER VISION HARDWARE INTERFACING

- How to connect computer vision with low-level hardware?



High computing resource → Computer [Laptop, PC, Single Board Computer]

Raspberry Pi , Jetson SBC, Latte Panda, Beagle bone, etc.

Low computing resource,
High control capabilities,
Real-time performance → PLC, Microcontroller

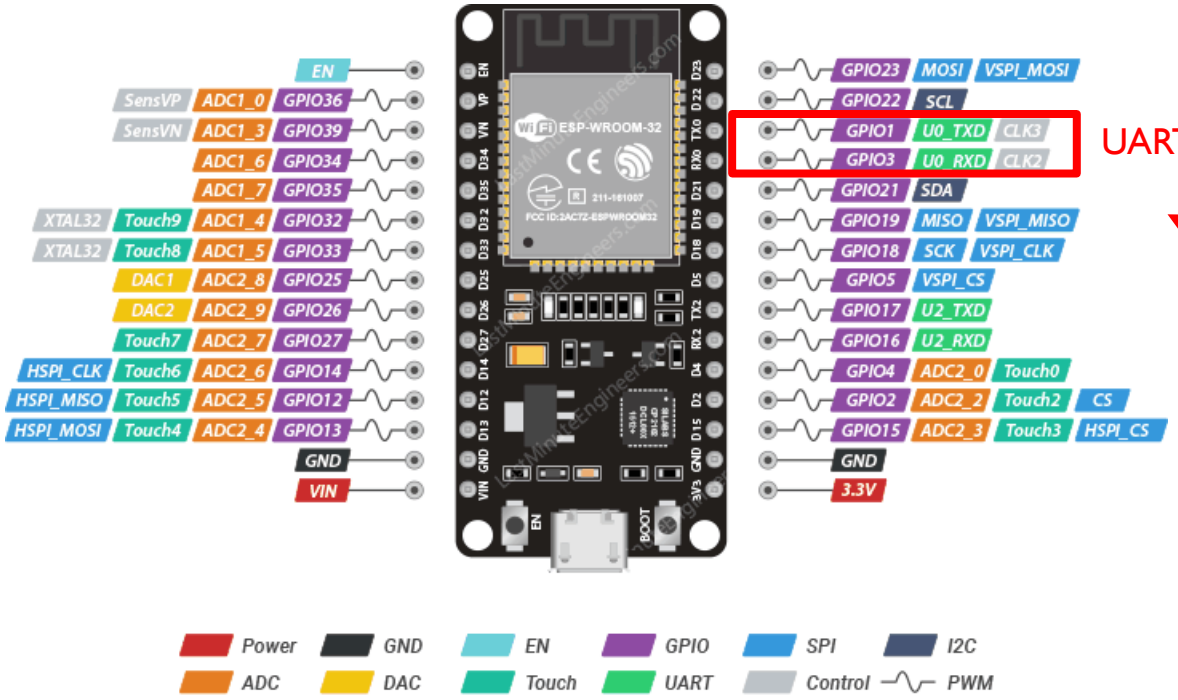
Arduino Board, ESP32, STM32, SAMD21, ATSAMD51, PIC32

CONNECTION BETWEEN HARDWARE

	Wired Connection	Available in Python	Wireless Connection	Available in Python
Microcontroller	UART RS232, RS485 Universal Serial Bus (USB)	Pyserial	MQTT TCP/UDP	Paho-mqtt socket
	I2C	Smbus		
	SPI	pyOnionSpi		
	CAN BUS	Python-can		
PLC	UART RS232, RS485	Pyserial	Modbus TCP/IP	pyModbusTCP
	OPC UA Server (LAN)	Python-opcua		
	MODBUS RTU	Pymodbus		
	CAN BUS	Python-can		

ESP32 MICRO PYTHON WIRED CONNECTION

- The simple and easiest wired connection between ESP32 MicroPython and Laptop is serial communication (UART)
- This method requires USB to TTL adapter.
- Connect RX , TX pin of ESP32 to adapter and then, connect the adapter to Computer's USB port.



UART0 port

USB port

ESP32 MICRO PYTHON WIRED CONNECTION

- In MicroPython, Write a code with UART Library
- In Python ,Write a code with Pyserial library

```
from machine import UART

uart1 = UART(1, baudrate=9600, tx=33, rx=32)
uart1.write('hello')  # write 5 bytes
uart1.read(5)         # read up to 5 bytes
```

MicroPython Example Code

```
import serial
import time

ser = serial.Serial('COM4', 9800, timeout=1)
time.sleep(2)

for i in range(50):
    # Reading all bytes available bytes till EOL
    line = ser.readline()
    if line:
        # Converting Byte Strings into unicode strings
        string = line.decode()
        # Converting Unicode String into integer
        num = int(string)
        print(num)

ser.close()
```

Pyserial Example code

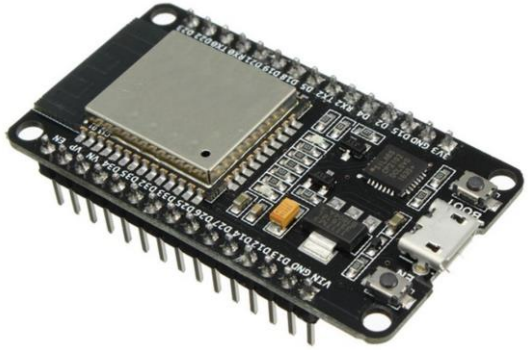
Recommended Source

https://www.pythonpool.com/python-serial-read/

ESP32 MICRO PYTHON WIRELESS CONNECTION

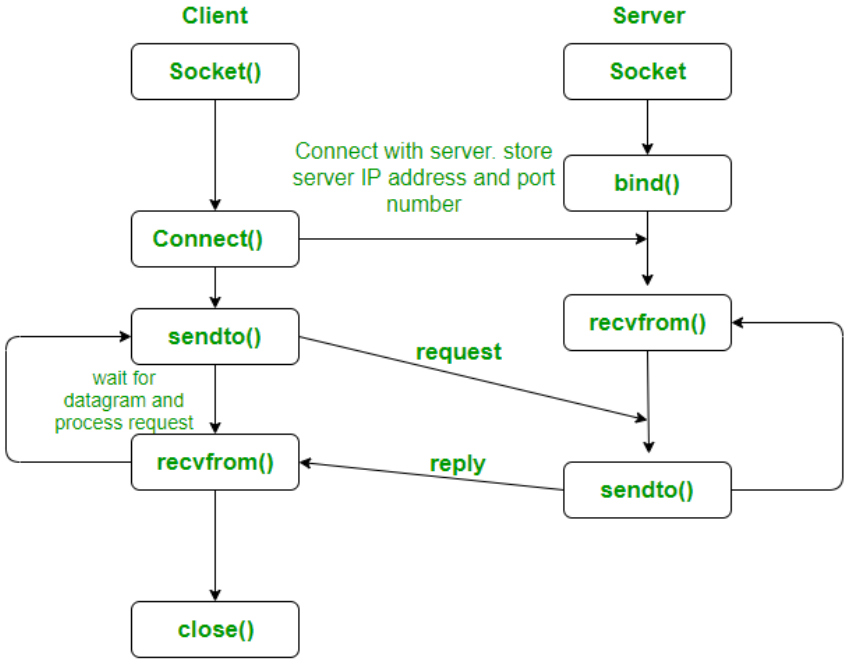
- We can use WiFi to establish the connection between ESP32 and other devices by using ESP32 as a server.

ESP32 as **Server**



μPython

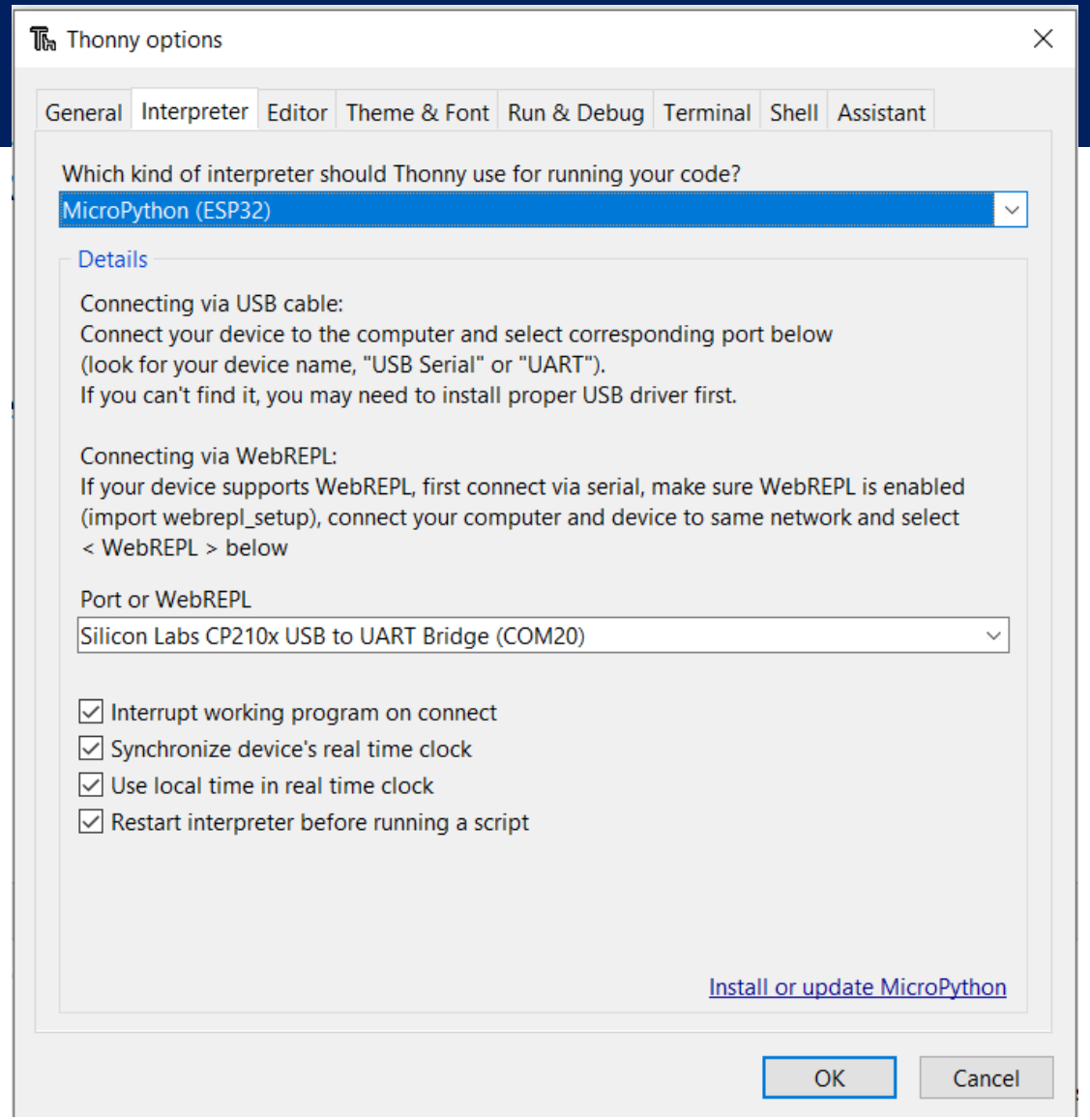
Python



UDP socket connection

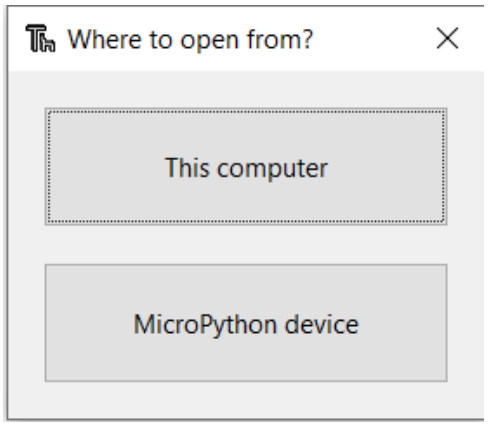
ESP32 SERVER SETUP

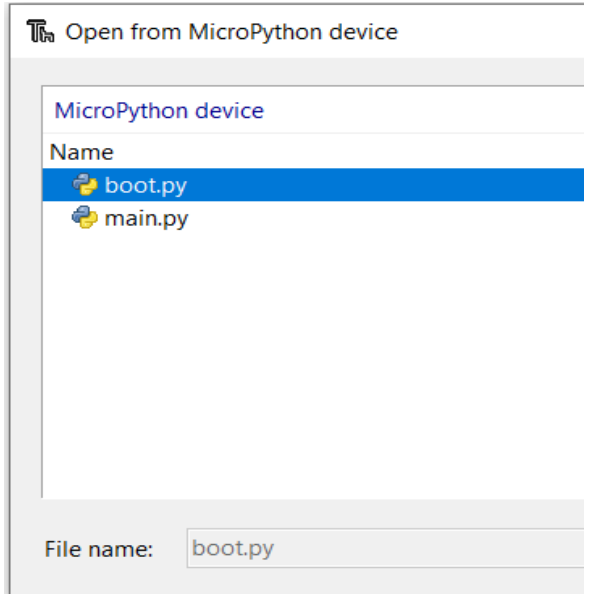
- Open Thonny IDE
- Connect ESP32 to laptop
- Change python interpreter to MicroPython(ESP32)

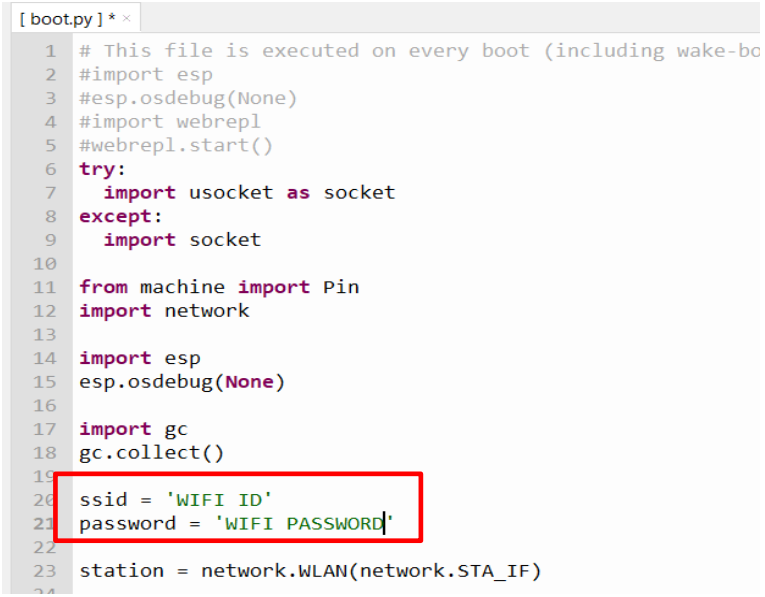


ESP32 SERVER SETUP

- Open file in MicroPython device.
- Open boot.py.
- Copy code from esp32_boot_setup.py to boot.py and enter SSID and password of WiFi.

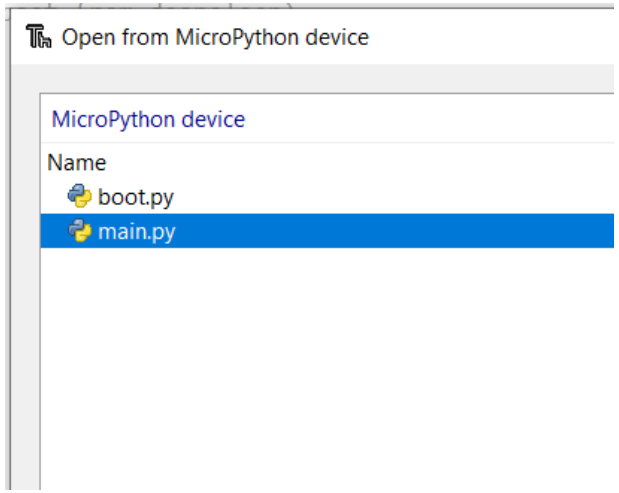






ESP32 SERVER SETUP

- Create main.py in MicroPython
- Write a code to start socket server connection in main.py




```python
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('', 80))
s.listen(5)
conn, addr = s.accept()
while True:

    data = conn.recv(1024)
    data = str(data,'utf-8')
    print(data)

    conn.sendall(str('loop','utf-8'))
conn.close()
```

ESP32 SERVER SETUP

Setup Part

[main.py]

```
1  import socket
2  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
3  s.bind(('', 80))
4  s.listen(5)
5  conn, addr = s.accept()
```

Import Socket library

Declare socket server

Bind socket server with unspecified IP and port 80

Prepare server for client request with 5 max clients

Accepts an incoming connection request and create socket object "conn"

Loop Part

```
6  while True:
7
8      data = conn.recv(1024)
9      data = str(data,'utf-8')
10     print(data)
11
12     conn.sendall(str('loop','utf-8'))
13 conn.close()
```

Receive data with 1024 bytes max size

Decode string data with 8 – bits Unicode Transformation Format (utf-8)

Send data through socket connection

PYTHON CLIENT

- Change Thonny Interpreter to **Local Python3.** or use other IDE.
- Create a new file for write a python socket client code.
- Save code to PC, Laptop, SBC.
- Connect the device to same WiFi with ESP32.
- Run the code to test connection.

```python
import socket
import time

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("192.168.1.63" , 80))
count = 0

while True:
    count+=1
    s.send(bytes(str(count),'utf-8'))
    data = s.recv(1024)

    print(data.decode())
    time.sleep(1)
```

ESP32 SERVER SETUP

Setup Part

```
1  import socket
2  import time
3
4  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
5  s.connect(("192.168.1.63" , 80))
6  count = 0
```

Connect client to socket at IP 192.168.1.63 (ESP32 IP address) on port 80

Loop Part

```
8  while True:
9      count+=1
10     s.send(bytes(str(count),'utf-8'))
11     data = s.recv(1024)
12
13     print(data.decode())
14     time.sleep(1)
```

Send data to the server in utf-8 format

Receive data with 1024 bytes max size

Decode received data

EXERCISE 1 : LED CONTROL OVER SOCKET CONNECTION

- Write a code to turn ON and OFF ESP32 built-in LED (Pin2) every 1 second.

EXERCISE 2 : COMPUTER VISION INTEGRATION

- Write a code to detect the object from webcam using template matching in real-time.
- If interested object is detected, Turn on ESP32 LED. Else, Turn OFF


ADDITIONAL INTERESTING TOPIC

CONTOUR

- Image segmentation is an image processing task in which the image is segmented or partitioned into multiple regions
- 3 Steps for contour finding:

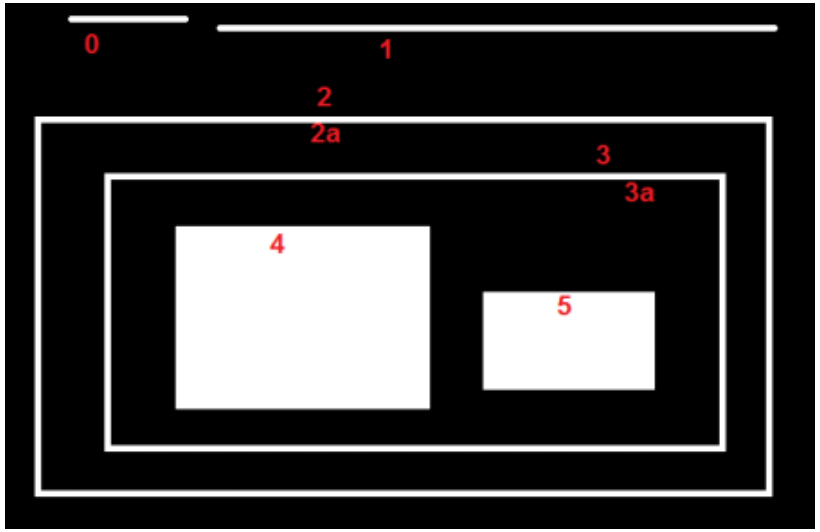
1. Convert image to gray scale image.
2. Separated interested area from the rest.
3. Find contours

```
contour,hierarchy = cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_NONE)
```

Input image **Mode** **Method**

CONTOUR

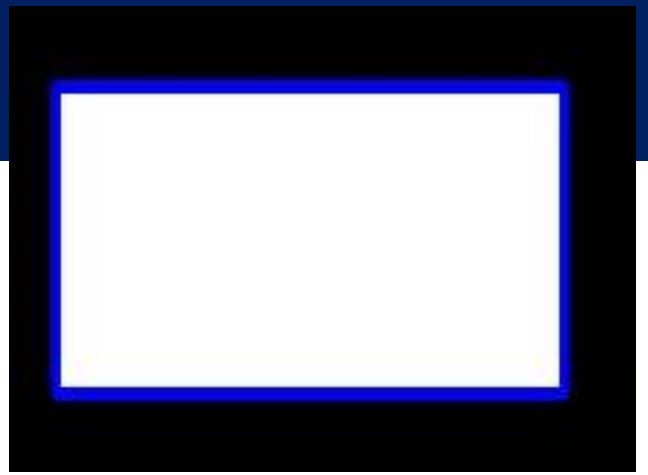
Mode	Description
cv.RETR_EXTERNAL	retrieves only the extreme outer contours. It sets hierarchy[i][2]=hierarchy[i][3]=-1 for all the contours.
cv.RETR_LIST	retrieves all of the contours without establishing any hierarchical relationships.
cv.RETR_CCOMP	retrieves all of the contours and organizes them into a two-level hierarchy. At the top level, there are external boundaries of the components. At the second level, there are boundaries of the holes. If there is another contour inside a hole of a connected component, it is still put at the top level.
cv.RETR_TREE	retrieves all of the contours and reconstructs a full hierarchy of nested contours.



Contour hierarchy

CONTOUR

Method	Description
cv.CHAIN_APPROX_NONE	stores absolutely all the contour points. That is, any 2 subsequent points (x1,y1) and (x2,y2) of the contour will be either horizontal, vertical or diagonal neighbors, that is, max(abs(x1-x2),abs(y2-y1))==1.
cv.CHAIN_APPROX_SIMPLE	compresses horizontal, vertical, and diagonal segments and leaves only their end points. For example, an up-right rectangular contour is encoded with 4 points.
cv.CHAIN_APPROX_TC89_L1	applies one of the flavors of the Teh-Chin chain approximation algorithm
cv.CHAIN_APPROX_TC89_KCOS	



None

Simple

CONTOUR

Original Image

Grayscale Image

```
gray = cv2.cvtColor(self.img, cv2.COLOR_BGR2GRAY)
```

CONTOUR

```
contour,hierarchy = cv2.findContours(thresh,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_NONE)

cv2.drawContours(self.img,contour,-1,color = (0,255,0),thickness = 2)
```

IMAGE MOMENT

- image moments are a set of statistical parameters to measure the distribution of where the pixels are and their intensities (I)

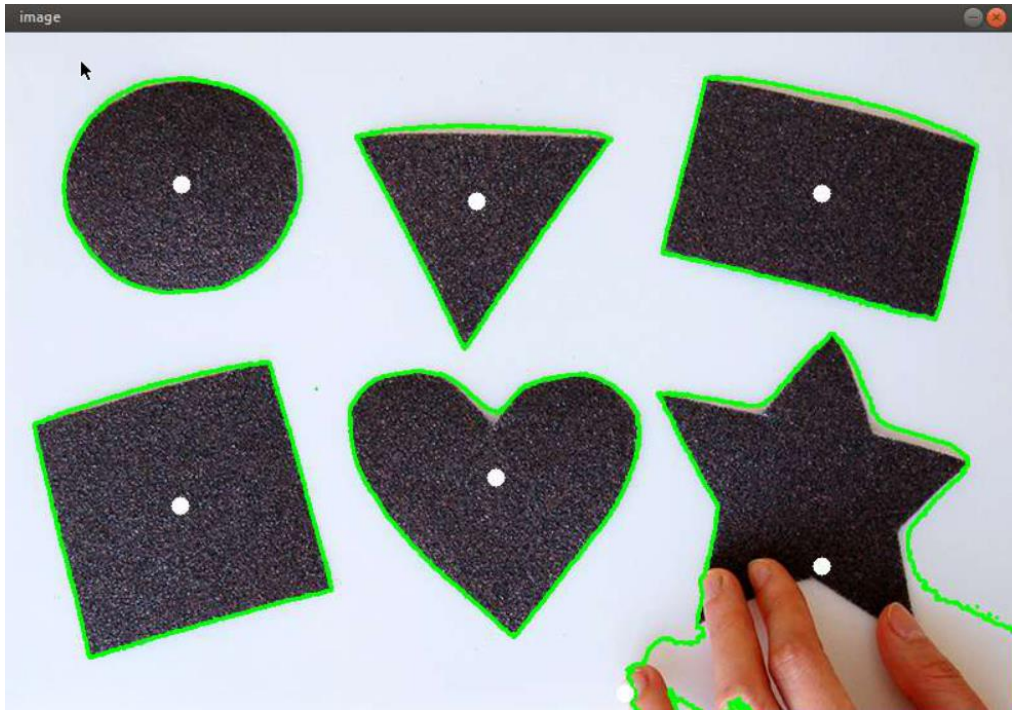
$$M_{ij} = \sum_x \sum_y x^i y^j I(x,y)$$

- For area, i and j = 0:

$$M_{00} = \sum_x \sum_y I(x,y)$$

- Centroid : $centroid\{\bar{x},\bar{y}\} = \left\{\frac{M_{10}}{M_{00}}, \frac{M_{01}}{M_{00}}\right\}$

IMAGE MOMENT




```
gray = cv2.cvtColor(self.img, cv2.COLOR_BGR2GRAY)

(t,thresh) = cv2.threshold(gray,200,255,cv2.THRESH_BINARY_INV)

contour = cv2.findContours(thresh,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)

cv2.drawContours(self.img,contour[0],-1,color = (0,255,0),thickness = 2)

contour = imutils.grab_contours(contour)
for c in contour:
        M = cv2.moments(c)
        if (M["m00"]!=0):


                area = M["m00"]
                print(area)
                cx = int(M["m10"]/M["m00"])
                cy = int(M["m01"]/M["m00"])

                cv2.circle(self.img, (cx,cy),7, (255,255,255),-1)

                self.display('image',self.img)
```

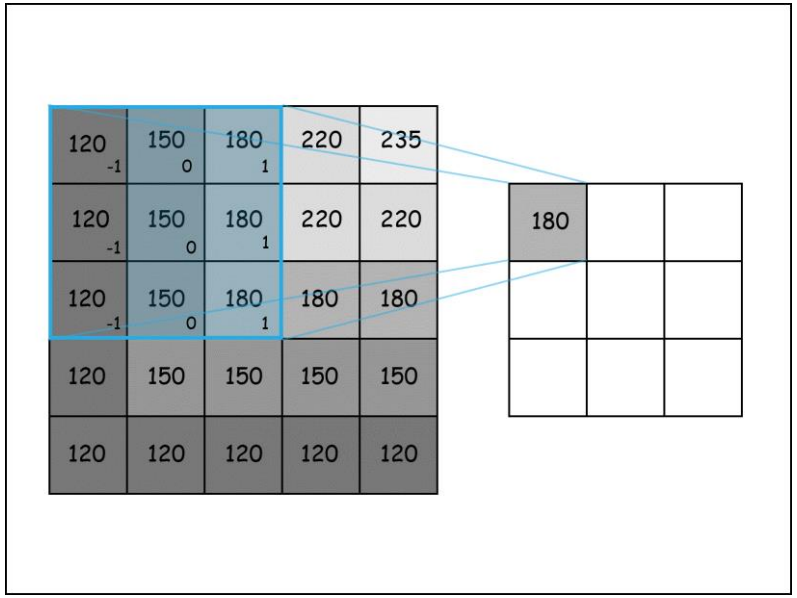
EDGE DETECTION

- Edge detection is an image-processing technique, which is used to identify the boundaries (edges) of objects, or regions within an image
- Two important method of edge detection are sobel and canny edge detection

120	150	180	220	235
120	150	180	220	220
120	150	180	180	180
120	150	150	150	150
120	120	120	120	120

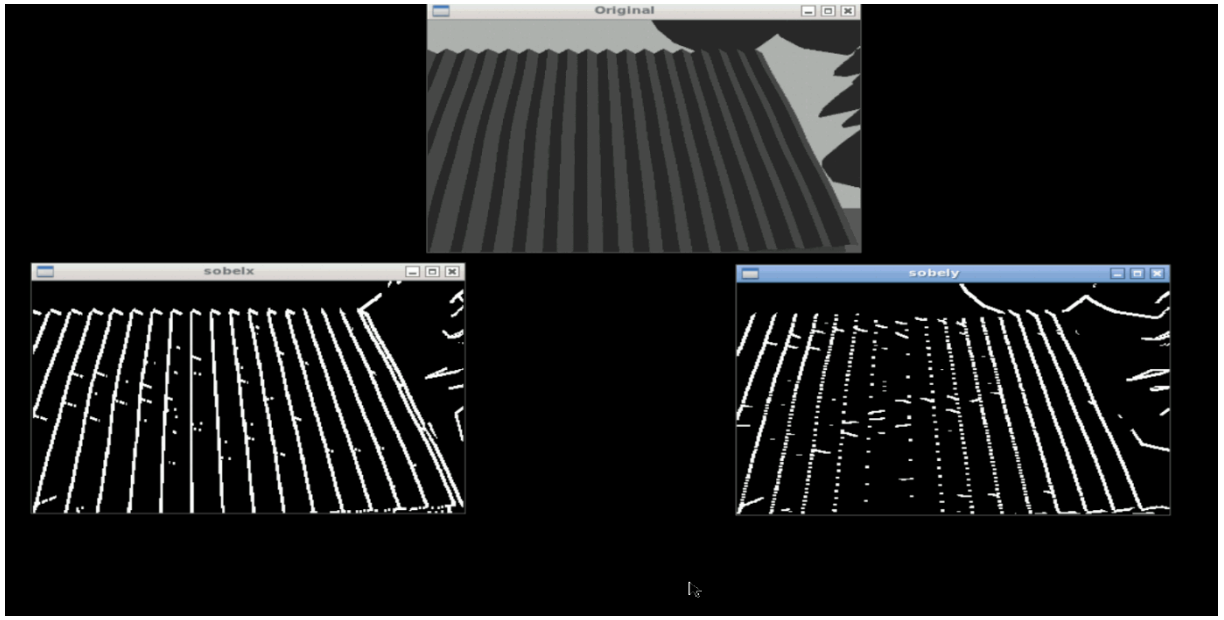
*

1	0	-1
1	0	-1
1	0	-1



EDGE DETECTION : SOBEL EDGE DETECTOR

- The Sobel operator is used in image processing, especially in edge detection algorithms. The operator calculates the intensity gradient of an image in every píxel using the convolution function



-1	-2	-1
0	0	0
1	2	1

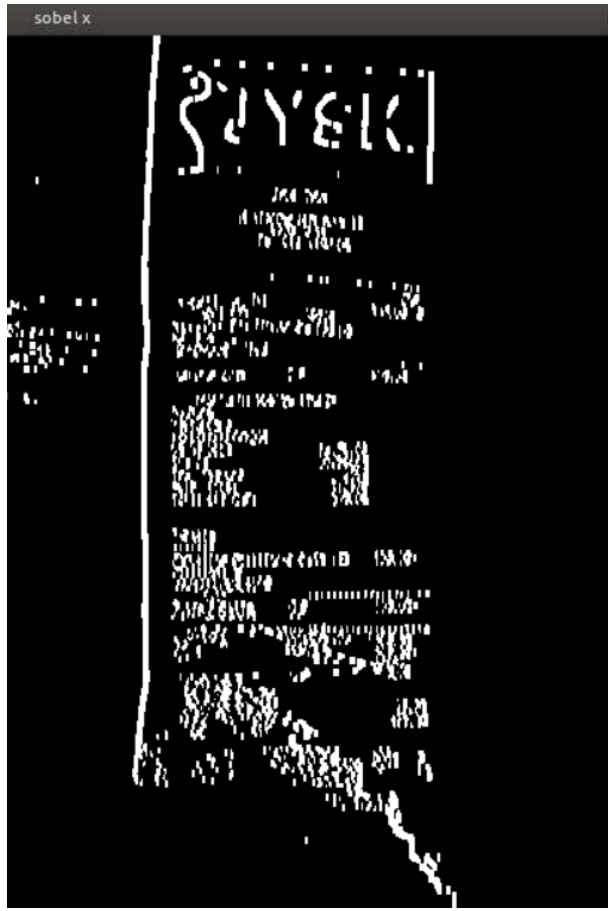
Sobel X

-1	0	1
-2	0	2
-1	0	1

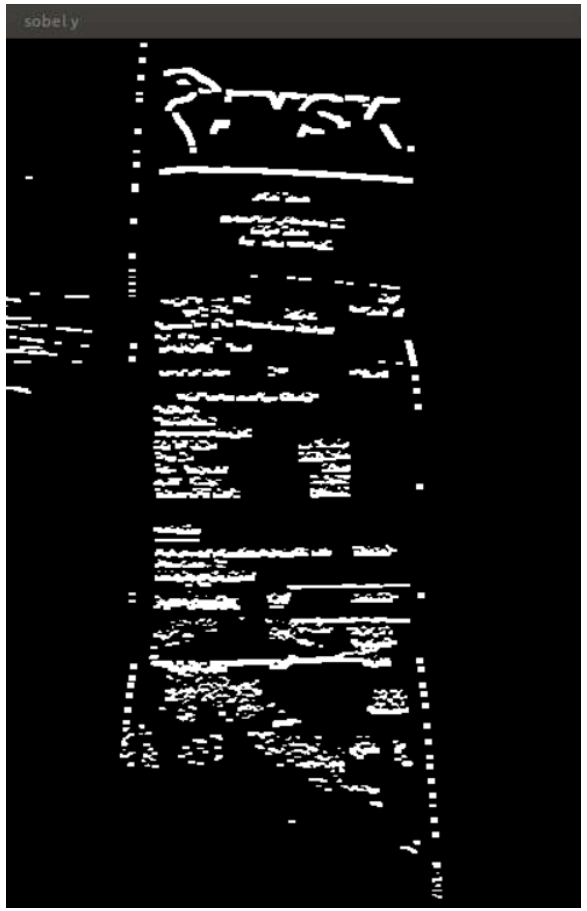
Sobel Y

EDGE DETECTION : SOBEL EDGE DETECTOR

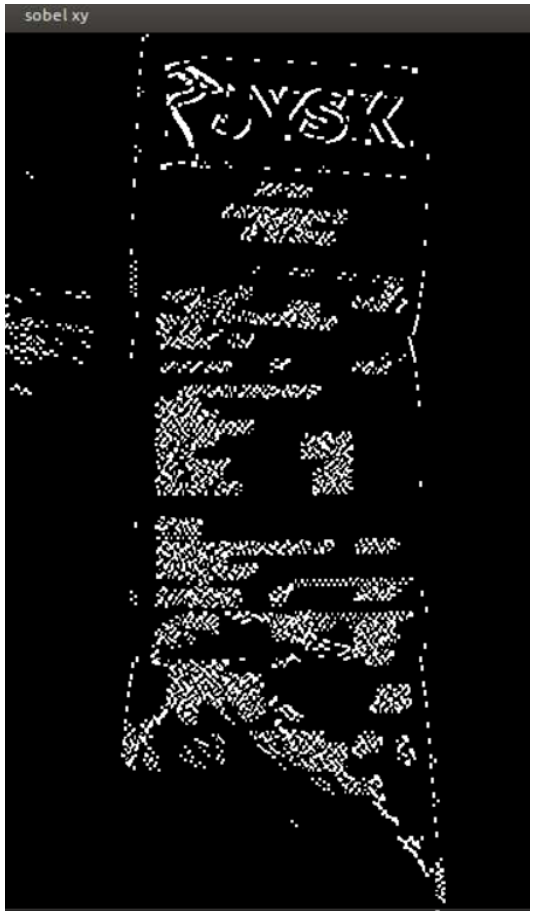
```
sobelx = cv2.Sobel(thresh, cv2.CV_64F, 1, 0, ksize = 5)
```




```
sobely = cv2.Sobel(thresh, cv2.CV_64F, 0, 1, ksize = 5)
```



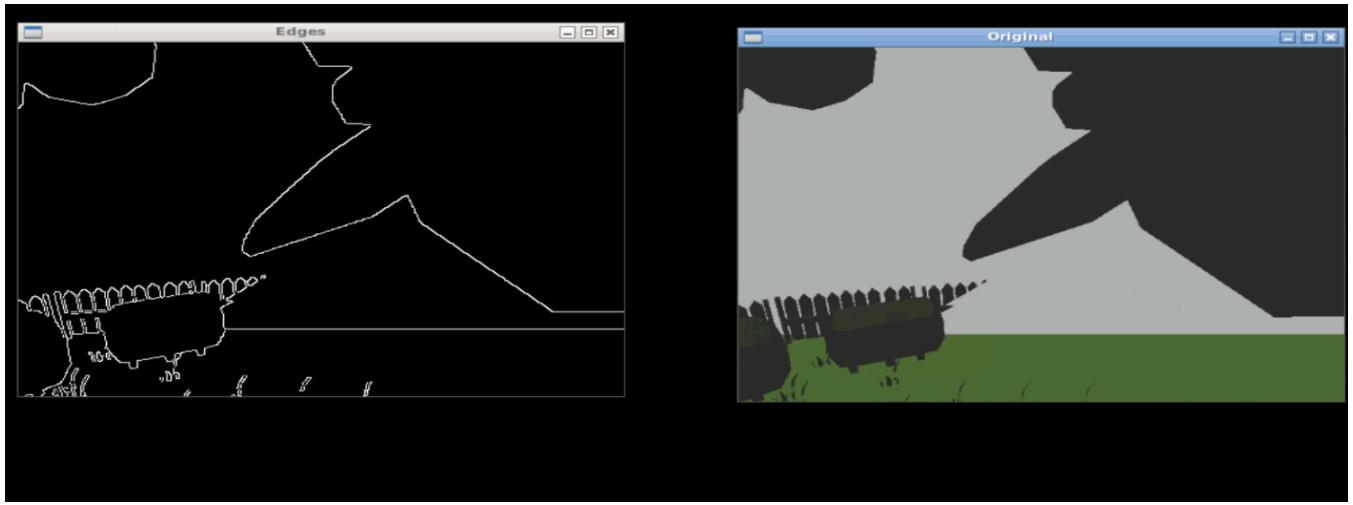

```
sobelxy = cv2.Sobel(thresh, cv2.CV_64F, 1, 1, ksize =5)
```



EDGE DETECTION : CANNY EDGE DETECTOR

- Canny Edge Detection is one of the most popular edge-detection methods in use today because it is so robust and flexible
- Four stage of canny edge detection are

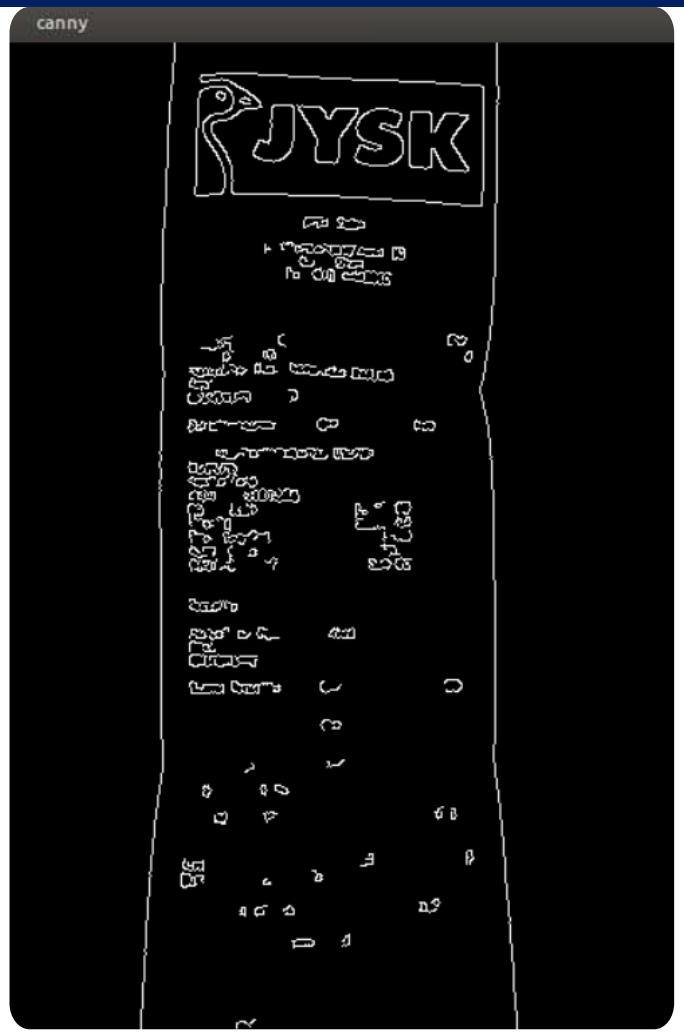
1. Noise Reduction
2. Calculating Intensity Gradient of the Image
3. Suppression of False Edges
4. Hysteresis Thresholding



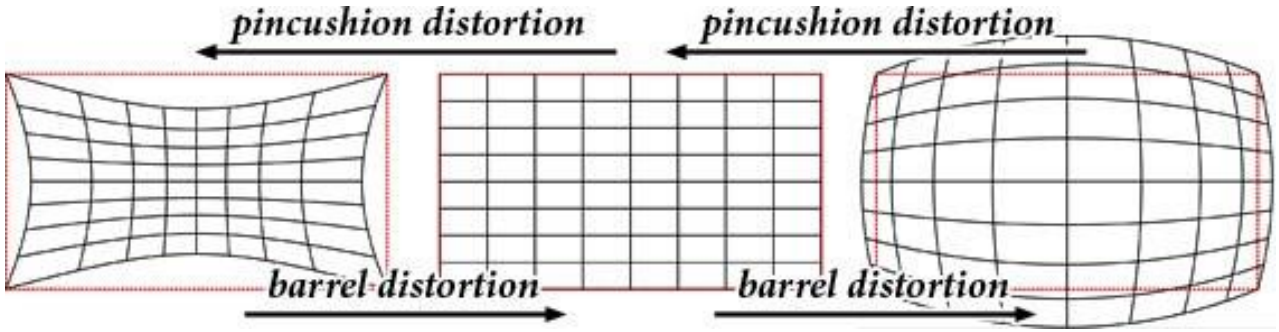
EDGE DETECTION : CANNY EDGE DETECTOR

- Canny(image, threshold1, threshold2)
- Don't forget to blur the image, before calling the Canny() function. It is a highly-recommended preprocessing step

```
blur = cv2.blur(self.img,(3,3))

canny = cv2.Canny(blur,100,200)
```



CAMERA CALIBRATION



Before calibration

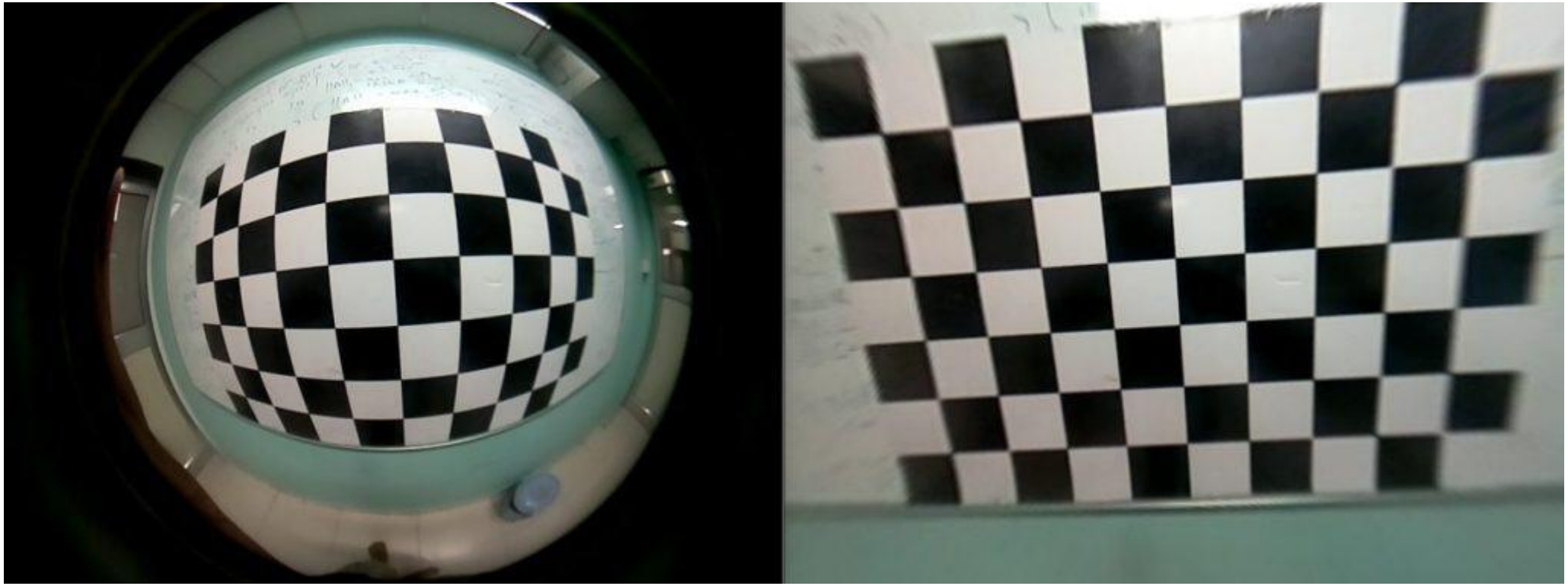
After calibration

Before calibration

After calibration

https://aliyasineser.medium.com/opencv-camera-calibration-e9a48bdd1844

CAMERA CALIBRATION



Effect of geometric correction on distorted image.

https://learnopencv.com/camera-calibration-using-opencv/

THANKS FOR YOUR ATTENTION