Digital Image Processing Basics

Lab 01

---

01. Basics of OpenCV

OpenCV: Open Source Computer Vision library to implement image processing

In openCV we use arrays to store an image and Numpy library is used to handle arrays, image always store in 2D array

Images types

Gray scale Image: Single channel

True Color Image: RGB, BGR in Python, 3 channel

Binary Image: 1 bit, 0 or 1 black or white

Functions for Images in Python

In [2]:

```
import cv2    # OpenCV used as cv2 in python
import matplotlib.pyplot as plt # to plot inside
print(cv2.__version__)
```

```
4.6.0
```

Read an image and display

In [3]:

```
path = "C:\\Users\\hp\\Google Drive\\Fiverr Work\\2022\\33. Computer Vision Course\\pictures"
```

In [4]:

```
img = cv2.imread(path+"\\coloredChips.png")  # took path and name of image as an argument
```

In [5]:

```
print(img)
```

```
[[[157 182 197]
  [157 180 195]
  [155 178 192]
  ...
  [207 230 230]
  [204 228 229]
  [209 232 234]]

 [[150 171 192]
  [149 171 189]
```

```
  [149 173 187]
  ...
  [208 232 228]
  [204 229 229]
  [206 231 232]]

 [[146 165 185]
  [148 168 186]
  [149 172 187]
  ...
  [202 229 228]
  [201 226 227]
  [201 226 226]]

 ...

 [[143 163 174]
  [139 161 174]
  [135 159 171]
  ...
  [193 219 226]
  [197 224 231]
  [201 228 235]]

 [[141 161 169]
  [141 162 171]
  [137 159 169]
  ...
  [201 224 232]
  [200 222 230]
  [201 224 232]]

 [[144 162 169]
  [138 157 165]
  [142 160 170]
  ...
  [198 219 227]
  [196 217 225]
  [202 224 232]]]
```

In [6]:

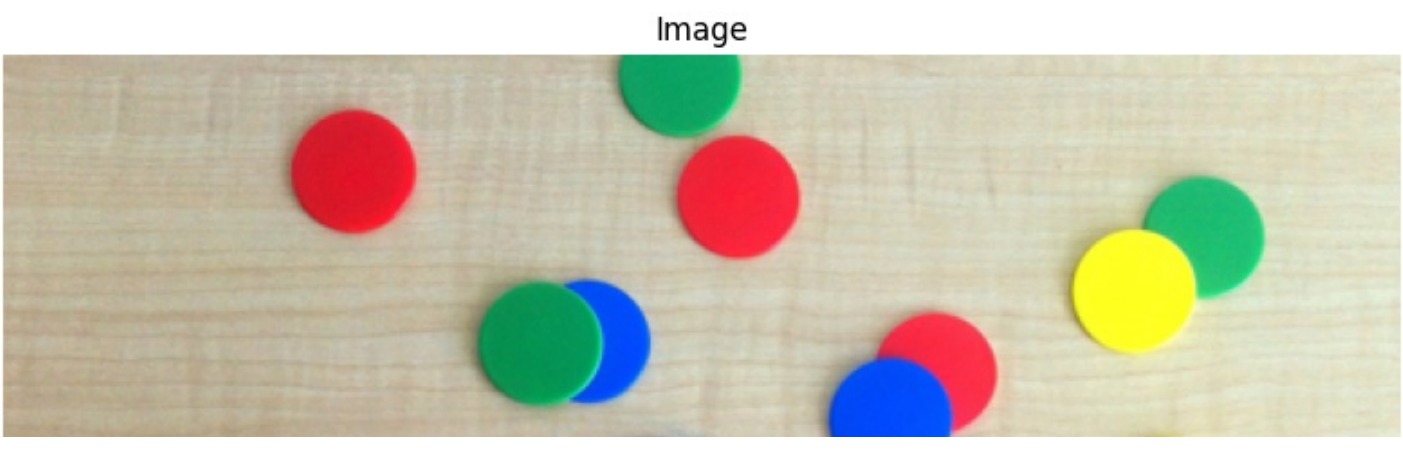
```
cv2.imshow("Image", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

In [10]:

```
# Display image using matplotlib
plt.figure(figsize=[10,10])
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title("Image")
plt.axis("off")
```

Out[10]:

```
(-0.5, 517.5, 390.5, -0.5)
```



Resize image

In [7]:

```
img_new = cv2.resize(img, (200, 200))
```

In [ ]:

```
plt.figure(1)
plt.imshow(img_new)
plt.title("Resized Image")
plt.axis("off")
```

In [8]:

```
cv2.imshow("Image", img)
cv2.imshow("NewImage", img_new)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Read image in grayscale

In [9]:

```
img1 = cv2.imread(path +"\\coloredChips.png", 0)   # 0 means in grayscale
```

In [10]:

```
cv2.imshow("Image1", img1)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Flip the image

In [12]:

```
img = cv2.imread(path+"\\football.jpg")
cv2.imshow("fliped image", cv2.flip(img, -1))   # it take parameters 0,-1,1
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Convert image to grayscale and save in current directory

In [13]:

```
import os

img2 = cv2.imread(path+"\\football.jpg", 0)   # 0 means in grayscale
cv2.imshow("Image2", img2)

k = cv2.waitKey()   # waits untill key press or wait for particular miliseocds time
if k == ord('s'):
    os.chdir("C:/Users/hp/Google Drive/Fiverr Work/2022/33. Computer Vision Course/pictur
es")
    cv2.imwrite('output.png', img2)
    cv2.destroyAllWindows()
else:
    cv2.destroyAllWindows()
```

Import video

In [14]:

```
cap = cv2.VideoCapture(path+"\\traffic.avi")

print(cap)
```

```
< cv2.VideoCapture 00000294AD3310D0>
```

In [15]:

```
while True:
    success, frame = cap.read()

    cv2.imshow('video', frame)
    key = cv2.waitKey(100)

    if key == ord('q'):
        break

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

In [ ]:

```
cv2.destroyAllWindows()
```

How to use webcam

In [16]:

```
import cv2
```

```
# 1. read video
cap = cv2.VideoCapture(0)

# now what is in cap, let's see
print('cap:', cap)  # so basically cap is an object of video capture

# as video is collection of frames
while True:

    status, frame = cap.read()    # status return boolean value True for read frame succ
essfully, false for not

    # 2. resize the frame
    frame = cv2.resize(frame, (700, 500))

    # 3. convert frames into grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    cv2.imshow("frames", frame)

    # showing gray frames
    cv2.imshow("gray frames", gray)

    # waits untill q pressed
    k = cv2.waitKey(100)
    if k == ord('q'):   # if not work the write k == ord('q') & 0xFF: this is mask
        break

cap.release() # relase the cap which capture the video
cv2.destroyAllWindows()
```

```
cap: < cv2.VideoCapture 00000294AD2EDF10>
```

Separate B G R from webcam frame

In [19]:

```
import cv2

# 1. read video
cap = cv2.VideoCapture(0)

# now what is in cap, let's see
print('cap:', cap)  # so basically cap is an object of video capture

# as video is collection of frames
while True:

    status, frame = cap.read()    # status return boolean value True for read frame succ
essfully, false for not

    # make the copy of orignal frame
    blueFrame = frame.copy()
    greenFrame = frame.copy()
    redFrame = frame.copy()

    # 2. resize the frame
    frame = cv2.resize(frame, (700, 500))

    # 3. convert frames into grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # for blue make 1, and 2, green, and red 0
    blueFrame[:,:,1] = 0
    blueFrame[:,:,2] = 0
```

```
    # for green make 0, and 2, blue, and red 0
    greenFrame[:,:,0] = 0
    greenFrame[:,:,2] = 0

    # for blue make 0, and 1, blue, and green 0
    redFrame[:,:,0] = 0
    redFrame[:,:,1] = 0

    cv2.imshow("Blue frames", blueFrame)
    cv2.imshow("Green frames", greenFrame)
    cv2.imshow("Red frames", redFrame)

    cv2.imshow("frames", frame)

    # showing gray frames
    cv2.imshow("gray frames", gray)

    # waits untill q pressed
    k = cv2.waitKey(100)
    if k == ord('q'):    # if not work the write k == ord('q') & 0xFF: this is mask
        break

cap.release() # relase the cap which capture the video
cv2.destroyAllWindows()
```

```
cap: < cv2.VideoCapture 00000294AD28FDD0>
```

How to draw shapes and lines on images

In [18]:

```
import cv2
import numpy as np

# How to draw shapes and lines on images
# we have to crate a matrix filled with zeros

# img = np.zeros((512,512)) # this is gray scale image
# print(img.shape)

img = np.zeros((512,512,3),np.uint8) # defining three colors
print(img.shape)

# obtaining color
img[:] = 255,0,0 # print blue
cv2.imshow("blue",img)

img[:] = 0,255,0 # print green
cv2.imshow("green",img)

img[:] = 0,0,255 # print red
cv2.imshow("red",img)

img[200:300,10:500] = 0,255,0 # print green
cv2.imshow("center red",img)

img[10:500,200:300] = 0,255,0 # print green
cv2.imshow("center red",img)

img[250:260,10:500] = 255,255,255 # print green
cv2.imshow("center red",img)

img[10:500,250:260] = 255,255,255 # print green
cv2.imshow("center red",img)

cv2.line(img,(0,0),(512,512),(0,0,0),3)
cv2.line(img,(0,512),(512,0),(0,0,0),3)
```

```
cv2.imshow("new",img)

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

```
(512, 512, 3)
```

Get Pixel value Using Cursor

In [ ]:

```
import cv2
import numpy as np

path = "C:\\Users\\hp\\Google Drive\\Fiverr Work\\2022\\33. Computer Vision Course\\pictures"

def draw(event, x,y,flag,param): # x,y for event call
    if event == cv2.EVENT_LBUTTONDBLCLK:
        font = cv2.FONT_HERSHEY_PLAIN
        text = str(x) + "," + str(y)

        cv2.putText(image, text, (x,y), font, 2, (255,0,255), 2)

# load image
image = cv2.imread(path+"\\car.png")

# create window and set mouse callback
cv2.namedWindow('image')
cv2.setMouseCallback('image', draw)
while True:
# show image
    cv2.imshow('image', image)
    key = cv2.waitKey(1)
    if key == ord("q"):
        break
cv2.destroyAllWindows()
```

Gaussian FIlter (Blur an Image)

A Gaussian Filter is a low pass filter used for reducing noise (high frequency components) and blurring regions of an image. The filter is implemented as an Odd sized Symmetric Kernel (DIP version of a Matrix) which is passed through each pixel of the Region of Interest to get the desired effect.

3x3

1/16

1	2	1
2	4	2
1	2	1

5x5

1/273

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

7x7

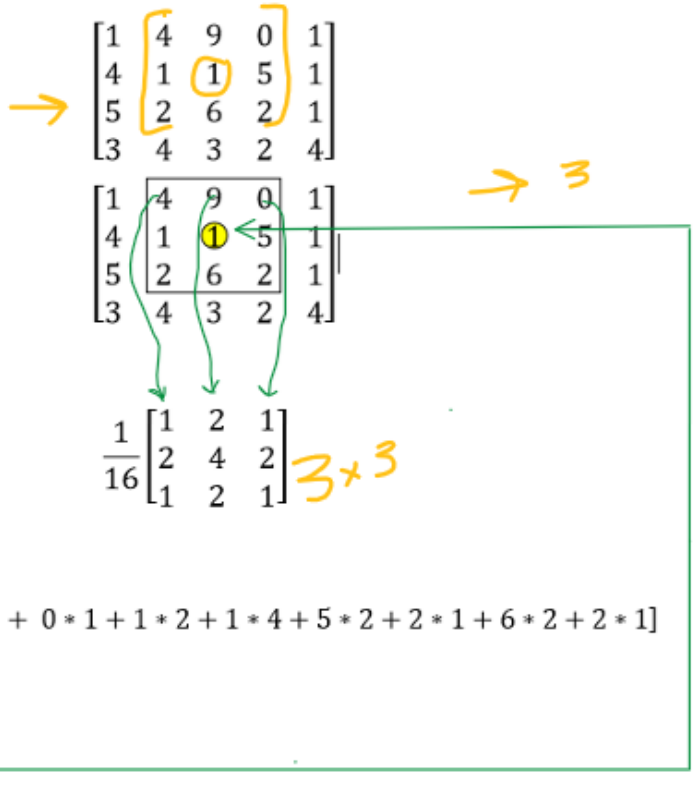
1/1003

0	0	1	2	1	0	0
0	3	13	22	13	3	0
1	13	59	97	59	13	1
2	22	97	159	97	22	2
1	13	59	97	59	13	1
0	3	13	22	13	3	0
0	0	1	2	1	0	0

Kernel

```
dst = cv2.GaussianBlur(src, ksize, 0)
```

Src: input image
Ksize: Gaussian Kernel Size. [height width]. height and width should be odd and can have different values. If ksize is set to [0 0], then ksize is computed from sigma values.
0: boarder type



$$= \frac{1}{16}[4*1+9*2+0*1+1*2+1*4+5*2+2*1+6*2+2*1]$$

$$= \frac{1}{16}[54]$$

$$= 3.3 = 3$$

Gaussian, Canny, Dilation, Erosion

In [ ]:

```
import cv2
import numpy as np

img = cv2.imread('onion.PNG')
cv2.imshow("original Image", img)

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

# convert to gray
img_gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

cv2.imshow('Gray Image',img_gray)


# convert to blur
img_blur = cv2.GaussianBlur(img_gray,(7,7),0)
cv2.imshow('Blur Image', img_blur)


# finding edges using canny functions
img_canny = cv2.Canny(img,200,200)
cv2.imshow('Canny Image', img_canny)
```

```
# Image dilation:  increase the thickness of edges
img_dilation = cv2.dilate(img_canny,kernel,iterations=1)
cv2.imshow('Dilation Image',img_dilation)


# Image Errosion
img_eroded = cv2.erode(img_dilation,kernel,iterations=1)
cv2.imshow('Eroded Image',img_eroded)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Find Types of an Image Using PIL Library

```
PIL.Image.open(fp, mode='r', formats=None)

    Opens and identifies the given image file.
    This is a lazy operation; this function identifies the file, but the file rema
ins open and the actual image data is not read from the file.

PARAMETERS:

    fp – A filename (string), pathlib.Path object or a file object. The file objec
t must implement file.read, file.seek, and file.tell methods, and be opened in bina
ry mode.

    mode – The mode. If given, this argument must be "r".
```

Link: https://pillow.readthedocs.io/en/stable/handbook/tutorial.

- **Modes**

The mode of an image is a string which defines the type and depth of a pixel in the image. Each pixel uses the full range of the bit depth. So a 1-bit pixel has a range of 0-1, an 8-bit pixel has a range of 0-255 and so on. The current release supports the following standard modes:

- 1 (1-bit pixels, black and white, stored with one pixel per byte)
- L (8-bit pixels, black and white)
- P (8-bit pixels, mapped to any other mode using a color palette)
- RGB (3x8-bit pixels, true color)
- RGBA (4x8-bit pixels, true color with transparency mask)
- CMYK (4x8-bit pixels, color separation)
- YCbCr (3x8-bit pixels, color video format)
    - Note that this refers to the JPEG, and not the ITU-R BT.2020, standard
- LAB (3x8-bit pixels, the L*a*b color space)
- HSV (3x8-bit pixels, Hue, Saturation, Value color space)
- I (32-bit signed integer pixels)
- F (32-bit floating point pixels)

In [ ]:

```
from PIL import Image
import cv2 as cv

def findTypeOfImage(image):

    if image.mode == "1":
        imageType = "1-bit pixels, black and white, stored with one pixel per byte"
```

```
    elif image.mode == "L":
        imageType = "8-bit pixels, black and white"
    elif image.mode == "P":
        imageType = "8-bit pixels, mapped to any other mode using a color palette"
    elif image.mode == "RGB":
        imageType = "3x8-bit pixels, true color"
    elif image.mode == "CMYK":
        imageType = "4x8-bit pixels, color separation"
    elif image.mode == "YCbCr":
        imageType = "3x8-bit pixels, color video format"
    elif image.mode == "LAB":
        imageType = "3x8-bit pixels, the L*a*b color space"
    elif image.mode == "HSV":
        imageType ="3x8-bit pixels, Hue, Saturation, Value color space"
    elif image.mode == "I":
        imageType = "32-bit signed integer pixels"
    elif image.mode == "F":
        imageType = "32-bit floating point pixels"

    return imageType
```

In [ ]:

```
path ="C:\\Users\\hp\\Google Drive\\Fiverr Work\\2022\\15. Teaching OpenCV to Client\\Pi
cs+scripts\\Pictures"

### Part 01 open the images icons01.png .....#####

img1 = Image.open(path + "\\icons01.png")
img2 = Image.open(path + "\\icons02.png")

img1Type = findTypeOfImage(img1)
img2Type = findTypeOfImage(img2)

print(f"The Type of icons01.png is "+img1Type)
print(f"The Type of icons02.png is " +img2Type)
```

Find RGB Values

In [ ]:

```
def findRGBValuesByPIL(image, x, y):
    img = Image.open(image).convert("RGB")
    r, g, b = img.getpixel((x, y))

    value = (r, g, b)

    return value
```

In [ ]:

```
def findRGBValuesByOpenCV(image):
    blue = image[:,:,0]
    green = image[:,:,1]
    red = image[:,:,2]

    print(f"Blue color values: {blue} \nGreen Color Values: {green}\nRed Color Values: {
red}")
```

In [ ]:

```
img3 = cv.imread(path + "\\rgb01.png")
img03 = path + "\\rgb01.png"

img4 = cv.imread(path + "\\rgb02.png")
img04 = path + "\\rgb02.png"

findRGBValuesByOpenCV(img3)
findRGBValuesByOpenCV(img4)
```

```
rgbValue = findRGBValuesByPIL(img03, 10, 10)
print(rgbValue)

cv.imshow("imgae3", img3)
cv.imshow("image4", img4)
cv.waitKey()
cv.destroyAllWindows()
```

Find Resolution, dimension, class(data type), Number of channel, Image Type

In [ ]:

```
import tkinter as tk
from PIL import Image
import numpy as np
import cv2 as cv
import os

root = tk.Tk()

def findResolutionOfImage(image):

    # Image resolution is typically described in PPI, which refers to
    # how many pixels are displayed per inch of an image

    # Screen Dimensions from Monitor (or Display) Control Panel = 1024x768 pixels
    # Viewable Width of Monitor Screen Resolution = 12.5 inches
    # Screen Resolution = 1024/12.5 = 82 ppi

    # find the width of your monitor or laptop
    # it is in mm
    width_mm = root.winfo_screenmmwidth()
    # convert it to inches, To convert mm to inches, you must multiply the unit by 0.0393
7
    # because 1 mm = 0.03937 inches
    width_in = width_mm * 0.03937

    height = image.shape[0]
    width = image.shape[1]

    if height > width:
        resolution = round(height / width_in)
    else:
        resolution = round(width / width_in)

    return str(resolution) + " ppi"
```

In [ ]:

```
def completeInformation(image, img, name):

    resolution_img = findResolutionOfImage(image)
    dimension_img = image.shape

    height_img = dimension_img[0]
    width_img = dimension_img[1]

    channels_img = dimension_img[2]

    Type_img = findTypeOfImage(img)
    dataType_img = image.dtype


    print(f"\nThe resoltuion of {name} is " + resolution_img)
    print(f"The dimension {name} is {dimension_img}, height: {height_img}, width: {width
_img}")
    print(f"Channels: {channels_img}\nData Type: {dataType_img}\nType: {Type_img}\n")
```

In [ ]:

```
path ="C:\\Users\\hp\\Google Drive\\Fiverr Work\\2022\\15. Teaching OpenCV to Client\\Pi
cs+scripts\\Pictures"

img_1 = cv.imread(path + "\\flower.png")
img1 = Image.open(path + "\\flower.png")
image_1_name = os.path.basename(path + "\\flower.png")

img_2 = cv.imread(path + "\\eagle.png")
img2 = Image.open(path + "\\eagle.png")
image_2_name = os.path.basename(path + "\\eagle.png")

completeInformation(img_1, img1, image_1_name)

completeInformation(img_2, img2, image_2_name)
```

In [ ]:

```

```

Digital Image Processing

Lab 02

---

01. Convert BGR image to Grayscale Without using cvtColor Function

In [1]:

```
import cv2
```

In [2]:

```
path = "C:\\Users\\hp\\Google Drive\\Fiverr Work\\2022\\33. Computer Vision Course\\pict
ures"
```

In [3]:

```
img = cv2.imread(path+"\\coloredChips.png")  # took path and name of image as an argument
```

1.1 Split the channles

In [4]:

```
blueChannel = img[:,:,0]
greenChannel = img[:,:,1]
redChannel = img[:,:,2]
```

1.2 Convert it to Grayscale

In [9]:

```
grayImg = (0.299 * redChannel + 0.587 * greenChannel + 0.114 * blueChannel)/255
```

Why we need to multiply these values with each cahnnel?

When converting a BGR image to grayscale, we need to take into account the different contributions of each color channel to the perceived brightness of a pixel.

The human eye is more sensitive to green light than red or blue, so we should assign a higher weight to the green channel in the grayscale conversion formula. The coefficients 0.299, 0.587, and 0.114 that we use in the formula are based on the relative luminance of each color channel, taking into account the sensitivity of the human eye to each color.

Therefore, we multiply each color channel by its corresponding coefficient to compute the grayscale value for each pixel. This ensures that the resulting grayscale image accurately represents the relative brightness of the original color image, while also taking into account the human eye's sensitivity to different colors.

Why these specific values? where they come from?

The values 0.299, 0.587, and 0.114 that are commonly used in the grayscale conversion formula for BGR images are based on the relative luminance of each color channel, as well as the sensitivity of the human eye to different

colors.

The coefficients were originally derived from the CIE (Commission Internationale de l'Eclairage) color space, which defines the standard observer model for the human eye. The CIE model includes a set of color matching functions that describe the spectral sensitivity of the human eye to different colors.

The coefficients used in the grayscale conversion formula are based on the CIE color matching functions, and are designed to approximate the perceived brightness of a color image for a standard observer under typical viewing conditions. Specifically, the coefficients are designed to take into account the fact that the human eye is most sensitive to green light, followed by red and then blue light.

The specific values of 0.299, 0.587, and 0.114 were chosen to provide a good balance between accuracy and simplicity, and have become widely adopted as a standard for grayscale conversion in BGR images.

1.3 Display Grayscale Image

In [10]:

```
cv2.imshow("Original Image", img)
cv2.imshow("Grayscale Image", grayImg)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Full Code

In [17]:

```
import cv2

path = "C:\\Users\\hp\\Google Drive\\Fiverr Work\\2022\\33. Computer Vision Course\\pict
ures"

img = cv2.imread(path+"\\coloredChips.png")  # took path and name of image as an argument

blueChannel = img[:,:,0]
greenChannel = img[:,:,1]
redChannel = img[:,:,2]

grayImg = (0.299 * redChannel + 0.587 * greenChannel + 0.114 * blueChannel)/255

cv2.imshow("Original Image", img)
cv2.imshow("Grayscale Image", grayImg)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

---

02. Use Mobile Phone Camera as Webcam

- **Download app iriun Webcam app from play store into your mobile** https://play.google.com/store/apps/details?id=com.jacksoftw.webcam&hl=en&gl=US
- **Also download software iriun for windows into your laptop** https://iriun.com/

Full Code

In [ ]:

```
import cv2

cam = cv2.VideoCapture(1)
```

```
while True:
    Success, Frame = cam.read()

    cv2.imshow("Mobile Cam", Frame)
    k = cv2.waitKey(1)

    if k == ord("q"):
        break

cam.release()
cv2.destroyAllWindows()
```

---

03. Screen Recorder Using OpenCV

In [11]:

```
import cv2
import pyautogui
import numpy as np
```

3.1 Get Screen Resolution

In [12]:

```
# create resolution
screenResolution = pyautogui.size()      # return screen width and height
```

3.2 Get the file Name and Path where you want to store recording

In [13]:

```
# file name in which we want to store recording
fileName = input("Enter file name and path: ")
```

3.3 Define fourcc

cv2.VideoWriter_fourcc is a function in the OpenCV library for Python that is used to create a four-character code (fourcc) that specifies the video codec to be used when writing a video file using cv2.VideoWriter.

The fourcc code is a 32-bit integer that is used to identify the video codec used to encode the video frames. Different video codecs have different fourcc codes, and the fourcc code can be used to specify the codec when opening a video file for writing.

The function takes four arguments, which are the four characters that make up the fourcc code. These characters can be any ASCII characters, and they are combined into a 32-bit integer using bitwise operations.

In [14]:

```
# Now fix the frame rate
fps = 30

fourcc = cv2.VideoWriter_fourcc(*'XVID')
output = cv2.VideoWriter(fileName, fourcc, fps, screenResolution)
```

3.4 Creating Recording Window

In [15]:

```
# create recording module
cv2.namedWindow("Live Recording", cv2.WINDOW_NORMAL)
```

```
cv2.resizeWindow("Live Recording", (640, 480))
```

3.5 Start Recording

In [16]:

```
while True:
    img = pyautogui.screenshot()
    f = np.array(img)

    f = cv2.cvtColor(f, cv2.COLOR_BGR2RGB)

    output.write(f)
    cv2.imshow("Live Recording", f)

    k = cv2.waitKey(1)

    if k == ord("q"):
        break

output.release()
cv2.destroyAllWindows()
```

Full Code

In [ ]:

```
# Screen Recoder

import cv2
import pyautogui
import numpy as np

# create resolution
screenResolution = pyautogui.size()     # return screen width and height

# file name in which we want to store recording
fileName = input("Enter file name and path: ")

# Now fix the frame rate
fps = 30

fourcc = cv2.VideoWriter_fourcc(*'XVID')
output = cv2.VideoWriter(fileName, fourcc, fps, screenResolution)

# create recording module
cv2.namedWindow("Live Recording", cv2.WINDOW_NORMAL)
cv2.resizeWindow("Live Recording", (640, 480))

while True:
    img = pyautogui.screenshot()
    f = np.array(img)

    f = cv2.cvtColor(f, cv2.COLOR_BGR2RGB)

    output.write(f)
    cv2.imshow("Live Recording", f)

    k = cv2.waitKey(1)

    if k == ord("q"):
        break

output.release()
cv2.destroyAllWindows()
```

04. Extracting Frames from Video or Webcam

Full Code

In [20]:

```
# extracting the frames from video

import cv2

cam = cv2.VideoCapture("traffic.avi")
Success, frame = cam.read()
count = 0

while True:
    if Success:
        cv2.imwrite(f"frames\\imgn{count}.jpg", frame)

        # setting the frame speed
        cam.set(cv2.CAP_PROP_POS_MSEC, (count**100))

        Success, frame = cam.read()

        cv2.imshow("frame extraction", frame)

        count +=1
        k = cv2.waitKey(1)
        if k == ord('q'):
            break
            cv2.destroyAllWindows()

cam.release()
cv2.destroyAllWindows()
```

```
---------------------------------------------------------------------------
error                                     Traceback (most recent call last)
Cell In [20], line 18
     14 cam.set(cv2.CAP_PROP_POS_MSEC, (count**100))
     16 Success, frame = cam.read()
---> 18 cv2.imshow("frame extraction", frame)
     20 count +=1
     21 k = cv2.waitKey(1)

error: OpenCV(4.6.0) D:\a\opencv-python\opencv-python\opencv\modules\highgui\src\window.c
pp:967: error: (-215:Assertion failed) size.width>0 && size.height>0 in function 'cv::ims
how'
```

In [21]:

```
cam.release()
cv2.destroyAllWindows()
```

4.1 Save on pressing "S" button

In [22]:

```
import cv2

cam = cv2.VideoCapture(0)

count = 0

while True:

    Success, frame = cam.read()

    if Success:
```

```
        # # setting the frame speed
        # cam.set(cv2.CAP_PROP_POS_MSEC, (count**100))

        cv2.imshow("frame extraction", frame)

        count +=1
        k = cv2.waitKey(1)

        if k == ord('q'):
            cv2.destroyAllWindows()
            break
        elif k == ord("s"):
            cv2.imwrite(f"frames\\imgn{count}.jpg", frame)


cam.release()
cv2.destroyAllWindows()
```

In [ ]:

```
```

Digital Image Processing

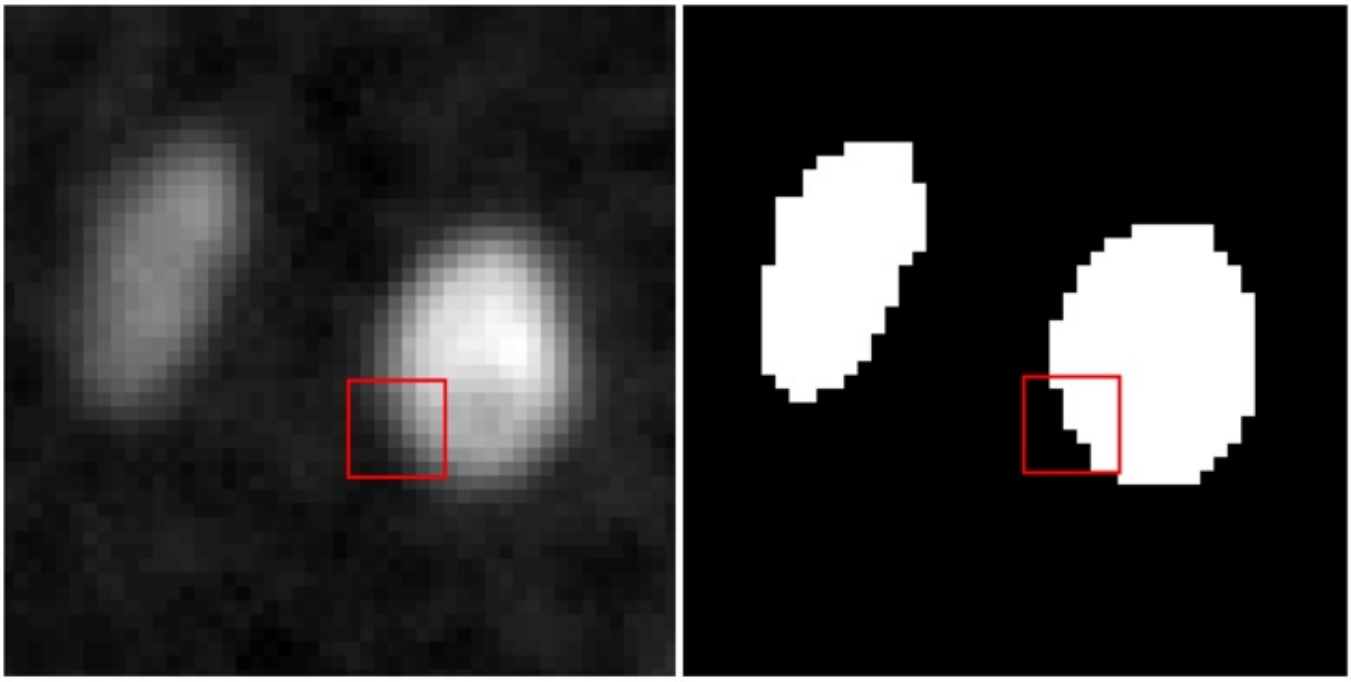
Lab 03

Binarization, image stacking, trackbars

---

01. Binarization / Threholding

- is a technique in OpenCV, which is the assignment of pixel values in relation to the threshold value provided.
- In thresholding, each pixel value is compared with the threshold value. If the pixel value is smaller than the threshold, it is set to 0, otherwise, it is set to a maximum value (generally 255).
- Thresholding is a very popular segmentation technique, used for separating an object considered as a foreground from its background.
- A threshold is a value which has two regions on its either side i.e. below the threshold or above the threshold.
- In Computer Vision, this technique of thresholding is done on grayscale images. So initially, the image has to be converted in grayscale color space.



61	66	75	86	96
59	64	73	83	92
59	64	73	83	92
56	60	68	79	88
54	58	64	72	81

0	0	1	1	1
0	0	1	1	1
0	0	1	1	1
0	0	0	1	1
0	0	0	0	1

`Syntax:` **cv2.threshold(source, thresholdValue, maxVal, thresholdingTechnique)**

`Parameters:`

- *`source:`* **Input Image array (must be in Grayscale).**
- *`thresholdValue:`* **Value of Threshold below and above which pixel values will change accordingly.**
- *`maxVal:`* **Maximum value that can be assigned to a pixel.**
- *`thresholdingTechnique:`* **The type of thresholding to be applied.**

`Simple Thresholding`

The basic Thresholding technique is Binary 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 different Simple Thresholding Techniques are:

`cv2.THRESH_BINARY:` ***If pixel intensity is greater than the set threshold, value set to 255, else set to 0 (black).***

In [38]:

```
import cv2
%matplotlib inline
import matplotlib.pyplot as plt
```

In [39]:

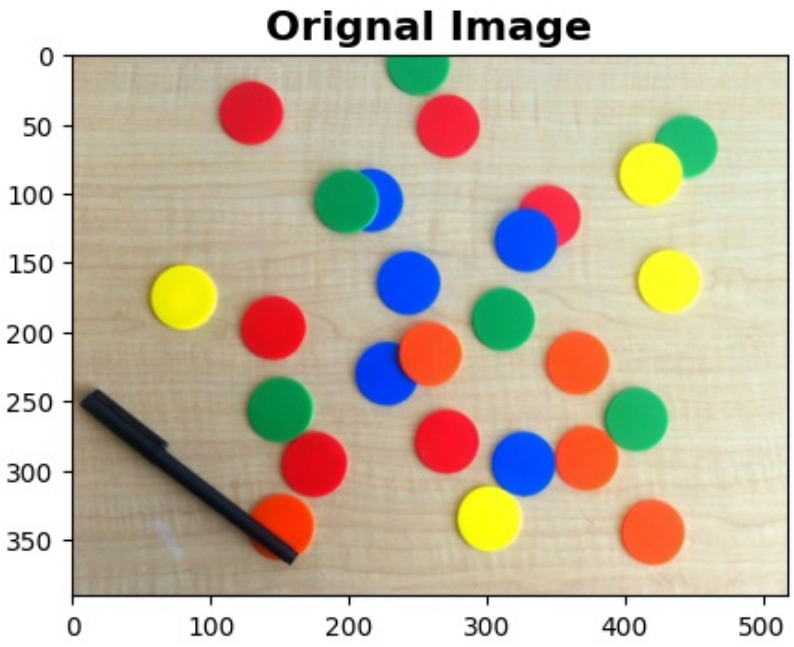
```
path = "C:\\Users\\hp\\Google Drive\\Fiverr Work\\2022\\33. Computer Vision Course\\pict
ures"
```

In [40]:

```
img = cv2.imread(path+"\\coloredChips.png")  # took path and name of image as an argument

RGBImage = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(5,5))
plt.imshow(RGBImage)
plt.title("Orignal Image", fontsize = 16, fontweight = 'bold')
```

Out[40]:

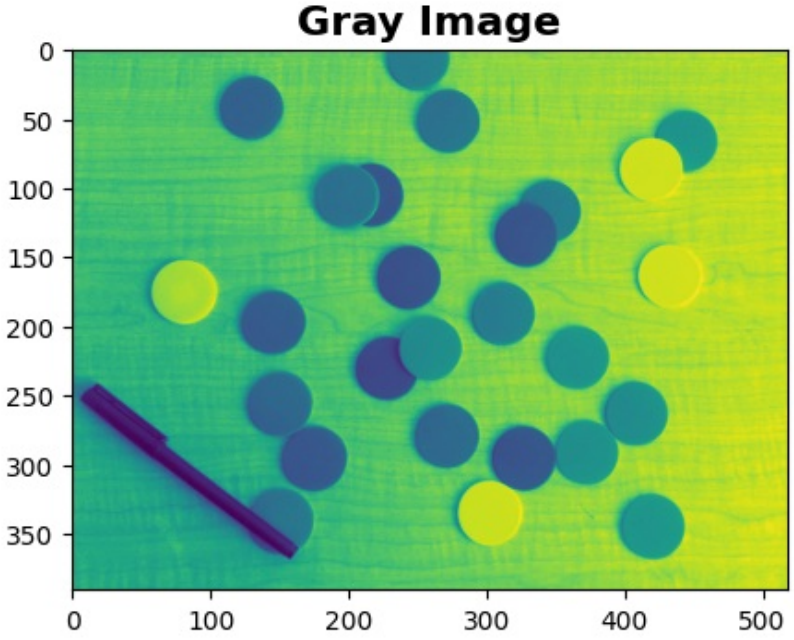
```
Text(0.5, 1.0, 'Orignal Image')
```



Convert it to Grayscale

In [41]:

```
imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

plt.figure(figsize=(5,5))
plt.imshow(imgGray)
plt.title("Gray Image", fontsize = 16, fontweight = 'bold')

cv2.imshow("Gray Image", imgGray)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Apply Binarization

In [44]:

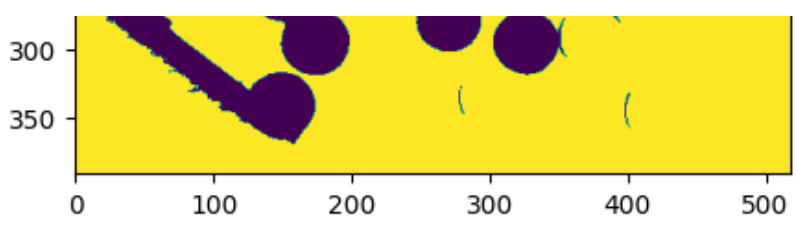
```
lb = 120
ub = 200
ret, thresh1 = cv2.threshold(imgGray, lb, ub, cv2.THRESH_BINARY)
ret
```

Out[44]:

```
120.0
```

In [49]:

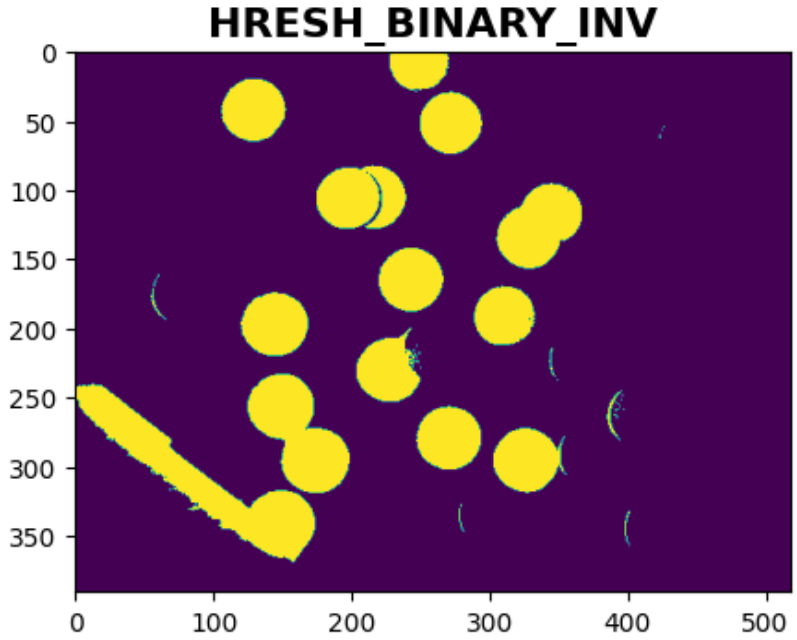
```
plt.figure(figsize=(5,5))
plt.imshow(thresh1)
plt.title("THRESH_BINAR", fontsize = 16, fontweight = 'bold')

cv2.imshow("THRESH_BINAR", thresh1)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



`cv2.THRESH_BINARY_INV:` ***Inverted or Opposite case of cv2.THRESH_BINARY.***

In [50]:

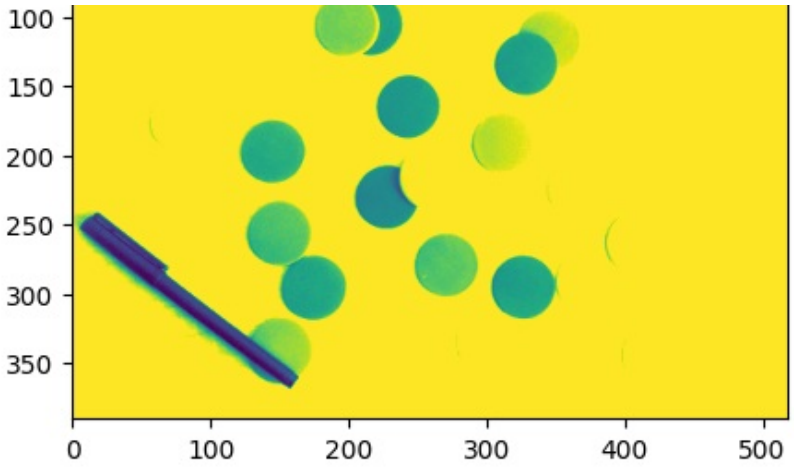
```
lb = 120
ub = 200
ret, thresh2 = cv2.threshold(imgGray, lb, ub, cv2.THRESH_BINARY_INV)

plt.figure(figsize=(5,5))
plt.imshow(thresh2)
plt.title("HRESH_BINARY_INV", fontsize = 16, fontweight = 'bold')

cv2.imshow("HRESH_BINARY_INV", thresh2)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



`cv.THRESH_TRUNC:` **If pixel intensity value is greater than threshold, it is truncated to the threshold. The pixel values are set to be the same as the threshold. All other values remain the same**

In [51]:

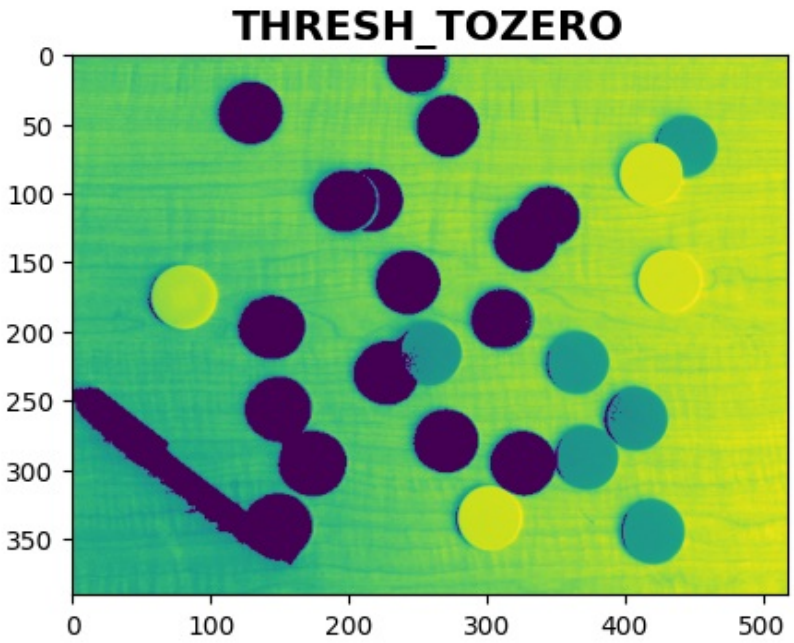
```
lb = 120
ub = 200
ret, thresh3 = cv2.threshold(imgGray, lb, ub, cv2.THRESH_TRUNC)

plt.figure(figsize=(5,5))
plt.imshow(thresh3)
plt.title("THRESH_TRUNC", fontsize = 16, fontweight = 'bold')

cv2.imshow("THRESH_TRUNC", thresh3)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



`cv.THRESH_TOZERO:` **Pixel intensity is set to 0, for all the pixels intensity, less than the threshold value.**

In [52]:

```
lb = 120
ub = 200
ret, thresh4 = cv2.threshold(imgGray, lb, ub, cv2.THRESH_TOZERO)

plt.figure(figsize=(5,5))
plt.imshow(thresh4)
plt.title("THRESH_TOZERO", fontsize = 16, fontweight = 'bold')

cv2.imshow("THRESH_TOZERO", thresh4)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

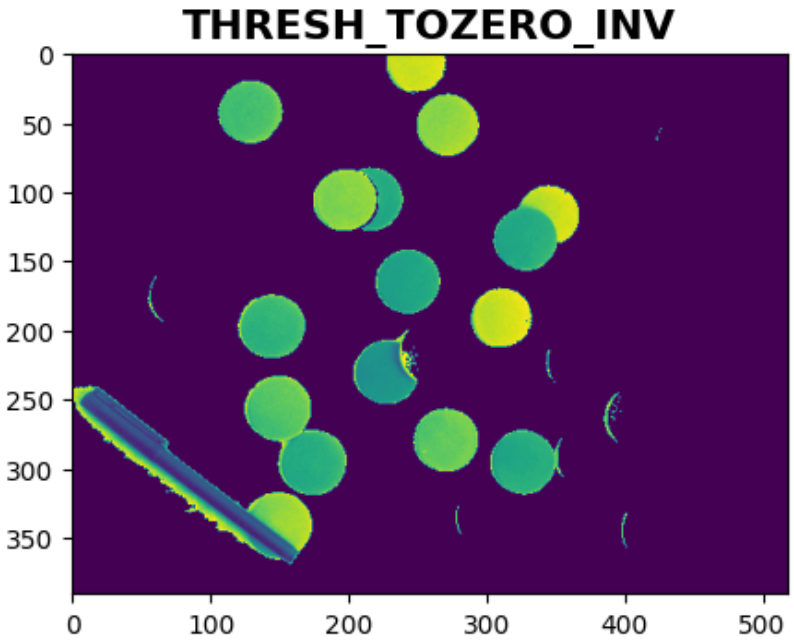


`cv.THRESH_TOZERO_INV`: **Inverted or Opposite case of cv2.THRESH_TOZERO.**

In [53]:

```
lb = 120
ub = 200
ret, thresh5 = cv2.threshold(imgGray, lb, ub, cv2.THRESH_TOZERO_INV)

plt.figure(figsize=(5,5))
plt.imshow(thresh5)
plt.title("THRESH_TOZERO_INV", fontsize = 16, fontweight = 'bold')

cv2.imshow("THRESH_TOZERO_INV", thresh5)
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```



---

02. OpenCV namedWindow(), createTrackbar(), getTrackbarPos() Function

- **is used to create a window with a suitable name and size to display images and videos on the screen.**

`Syntax`

```
cv2.namedWindow(window_name, flag)
```

- *`window_name:`* **Name of the window that will display image/video**
- *`flag:`* **Represents if window size is automatically set or adjustable.**

Some of the flag values are:

- *`WINDOW_NORMAL`* **– Allows to manually change window size**
- *`WINDOW_AUTOSIZE(Default)`* **– Automatically sets the window size**
- *`WINDOW_FULLSCREEN`* **– Changes the window size to fullscreen**

createTrackbar():

`Syntax`

```
cv2.createTrackbar(trackbarName, windowName, DefaultValue, maximumValue, functionN
ameWhichSimplyPass)
```

E.g.

```
cv2.createTrackbar('R', 'image', 0, 255, nothing)
```

`Return` **nothing**

getTrackbarPos():

`Syntax`

```
cv.getTrackbarPos(trackbarname, winname)
```

`Parameters`

`trackbarname:` **Name of trackbar** *`winname:`* **Name of the window that is the parent of the trackbar.**

`Return:` **Current position of the specified trackbar**

`Examaple: 1` **I want to change the color of image R, G, and B by using OpenCV trackbar functions**

Full Code

In [54]:

```
import cv2 as cv
import numpy as np

# you have to create a function which simply do nothing and pass
# because it is required as fifth argument of createTracbar function
def nothing(x):
    pass

# Creating a black image using numpy, with following dimension 400x500x3
img = np.zeros((200, 600, 3), "uint8")

# creating a display window with named "image"
cv.namedWindow("image")

# creating trackbars for red color change,
# R is name of trackbar,
# image is name of window on which it will display
# 0 is by default value of trackbar when it start
# 255 is the maximum value of the trackbar
# nothing is the function created above which simply do nothing and pass,
# it is requirement of createTrackbar function
cv.createTrackbar('R', 'image', 0, 255, nothing)

# creating trackbars for Green color change
cv.createTrackbar('G', 'image', 0, 255, nothing)

# creating trackbars for Blue color change
cv.createTrackbar('B', 'image', 0, 255, nothing)

# now i want to get the values of each trackbar
# for that i need a loop which runs continuously

while True:

    # get current positions or value of all Three trackbars
    # R is the name of trackbar created above and
    # image is the name of window
    r = cv.getTrackbarPos('R', 'image')
    g = cv.getTrackbarPos('G', 'image')
    b = cv.getTrackbarPos('B', 'image')

    # now assign these r, g, and b trackbar value to original image
    img[:] = [b, g, r]
```

```
    # Now show the created image (img) inside the above created window,
    # you have to mention the name of window, in this case the name of window is "image"

    cv.imshow("image", img)
    key = cv.waitKey(1)

    if key == ord("q"):
        cv.destroyAllWindows()
        break

cv.destroyAllWindows()
```

`Examaple:2` **- I want to convert the color (cvtColor) using Trackbar**

Full Code

In [59]:

```
import cv2 as cv
def nothing(x):
    pass

path = "C:\\Users\\hp\\Google Drive\\Fiverr Work\\2022\\15. Teaching OpenCV to Client\\p
ictures"

img = cv.imread(path + "\\eyes.jpg")

imgResized = cv.resize(img, (1000, 600))

# create a window
cv.namedWindow("image", cv.WINDOW_NORMAL)

# [4, 2, 0, 6, 40, 68, 66, 36, 44, 32, 50]
# each color code hase integer value
colorName = [cv.COLOR_BGR2RGB, cv.COLOR_BGR2RGBA, cv.COLOR_BGR2BGRA, cv.COLOR_BGR2GRAY,
             cv.COLOR_BGR2HSV, cv.COLOR_BGR2HLS_FULL,cv.COLOR_BGR2HSV_FULL,
             cv.COLOR_BGR2YCrCb, cv.COLOR_BGR2LAB, cv.COLOR_BGR2XYZ,
             cv.COLOR_BGR2LUV]

cv.createTrackbar('color', 'image', 0, 10, nothing)

while True:

    colorNumber = cv.getTrackbarPos('color', 'image')

    colorImage = cv.cvtColor(imgResized, colorName[colorNumber])

    cv.imshow("image", colorImage)

    k = cv.waitKey(1)

    if k == ord('q'):
        cv.destroyAllWindows()
        break

cv.destroyAllWindows()
```

---

03. Image Stacking

In [60]:

```
import numpy as np

a = np.array([1, 2, 3])
```

```
b = np.array([4, 5, 6])

c = np.hstack((a, b))

print(c)
```

```
[1 2 3 4 5 6]
```

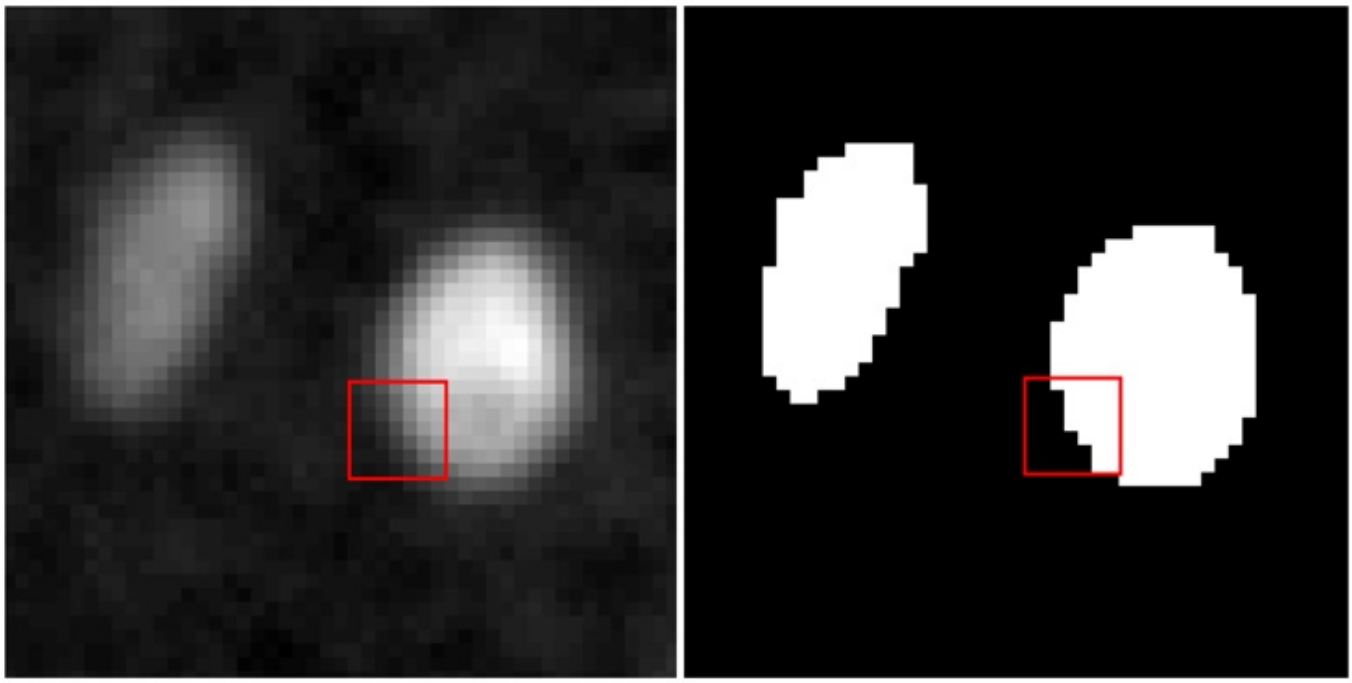
In [57]:

```
hstackw = np.hstack((imgGray, thresh1,thresh2,thresh3,thresh4,thresh5))
cv2.imshow("stack", hstackw)
cv2.waitKey()
cv2.destroyAllWindows()
```

LAB TASK

When workig with images, displaying them side by side after making manipulations to them can be useful to follow what is going on. Create a copy of the image (which ever you want) and using five methods below

- *cv2.THRESH_BINARY*
- *cv2.THRESH_BINARY_INV*
- *cv2.THRESH_TRUNC*
- *cv2.THRESH_TOZERO*
- *cv2.THRESH_TOZERO_INV*
- **Now you should have five different binarized versions of image. you can view this as having five different and independent matrices containing image data for each method .**



61	66	75	86	96
59	64	73	83	92
59	64	73	83	92
56	60	68	79	88
54	58	64	72	81

0	0	1	1	1
0	0	1	1	1
0	0	1	1	1
0	0	0	1	1
0	0	0	0	1

- **Now use a list to merge these images together and the NumPy stack functionto display them vertically or horizontally. Think about how imshow reads the picture after this operation, what happend?**
- **Lastly, create trackbars do be able to live adjust the boundary values to find the optimal lower and upper**

value for all images.

SOLUTION

In [63]:

```
# Exercise 03: Binarization, image stacking, trackbars

import cv2
import numpy as np

# as this is required by createTrackbar function
def nothing(x):
    pass

path ="C:\\Users\\hp\\Google Drive\\Fiverr Work\\2022\\15. Teaching OpenCV to Client\\Pi
cs+scripts\\Pictures"

img = cv2.imread(path +"\\piece03.png")
img = cv2.resize(img, (220, 600))
imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

cv2.namedWindow("stack", cv2.WINDOW_NORMAL)

cv2.createTrackbar('LB', 'stack', 0, 255, nothing)
cv2.createTrackbar('UB', 'stack', 255, 255, nothing)

while True:

    lb = cv2.getTrackbarPos('LB', 'stack')
    ub = cv2.getTrackbarPos('UB', 'stack')

    ret, thresh1 = cv2.threshold(imgGray, lb, ub, cv2.THRESH_BINARY)
    ret, thresh2 = cv2.threshold(imgGray, lb, ub, cv2.THRESH_BINARY_INV)
    ret, thresh3 = cv2.threshold(imgGray, lb, ub, cv2.THRESH_TRUNC)
    ret, thresh4 = cv2.threshold(imgGray, lb, ub, cv2.THRESH_TOZERO)
    ret, thresh5 = cv2.threshold(imgGray, lb, ub, cv2.THRESH_TOZERO_INV)

    # this print lower bound means threshold value
    # print(ret)

    # the window showing output images
    # with the corresponding thresholding
    # techniques applied to the input images

    hstackw = np.hstack((imgGray, thresh1,thresh2,thresh3,thresh4,thresh5))
    cv2.imshow("stack", hstackw)
    # cv2.imshow('Binary Threshold', thresh1)
    # cv2.imshow('Binary Threshold Inverted', thresh2)
    # cv2.imshow('Truncated Threshold', thresh3)
    # cv2.imshow('Set to 0', thresh4)
    # cv2.imshow('Set to 0 Inverted', thresh5)

    cv2.imshow("image", img)
    k = cv2.waitKey(1)
    if k == ord("q"):
        cv2.destroyAllWindows()
        break

cv2.destroyAllWindows()
```

Digital Image Processing

Lab 04

Image Preprocessing (Edge Detection)

---

01. Extracting Colors Channel

Read the image and extract its color channel as

- **grayscale image**
- **color images**

Then display the images to verify the operations has succeeded.

In [1]:

```
import cv2
%matplotlib inline
import matplotlib.pyplot as plt
```

In [2]:

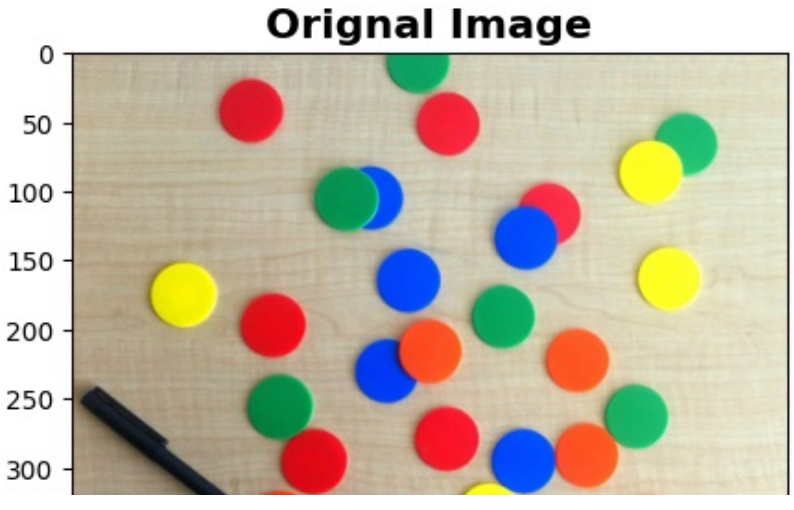
```
path = "C:\\Users\\hp\\Google Drive\\Fiverr Work\\2022\\33. Computer Vision Course\\pict
ures"
```

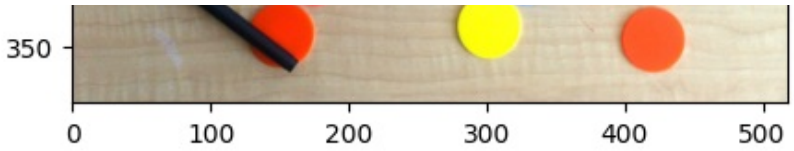
In [3]:

```
img = cv2.imread(path+"\\coloredChips.png")  # took path and name of image as an argument

RGBImage = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(5,5))
plt.imshow(RGBImage)
plt.title("Orignal Image", fontsize = 16, fontweight = 'bold')
```

Out[3]:

```
Text(0.5, 1.0, 'Orignal Image')
```





Get grayscale image of each channel

In [4]:

```
def getGrayscaleImages(image):

    b = image[:,:,0]
    g = image[:,:,1]
    r = image[:,:,2]

    return b, g, r
```

Get Colored Image of each channel (First Method)

In [5]:

```
import numpy as np

def colorChannelImages(image):

    dimension = image.shape
    height, width = dimension[0], dimension[1]

    zeroChannel = np.zeros((height, width), "uint8")

    b, g, r = getGrayscaleImages(image)

    blueImage = cv2.merge([b, zeroChannel, zeroChannel])
    greenImage = cv2.merge([zeroChannel, g, zeroChannel])
    redImage = cv2.merge([zeroChannel, zeroChannel, r])

    return blueImage, greenImage, redImage
```

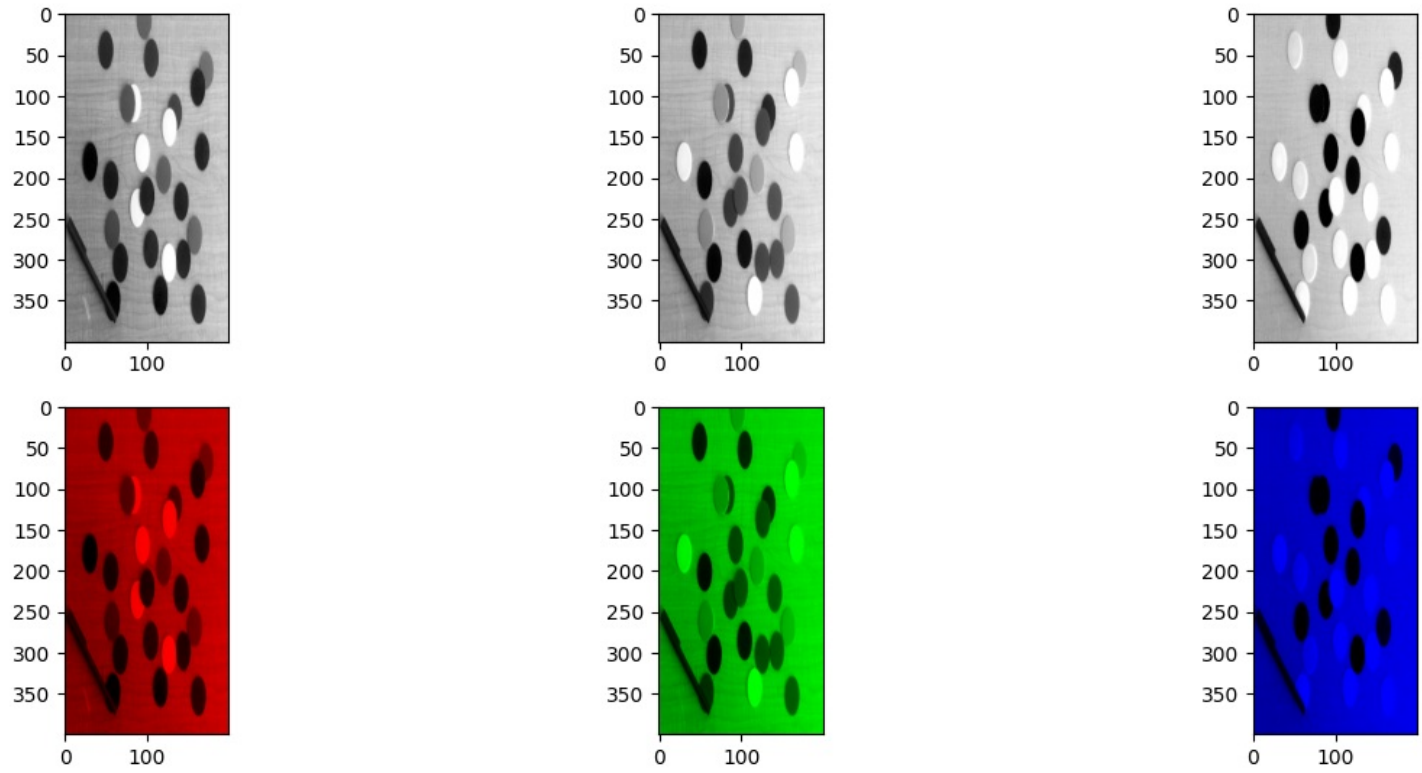
In [22]:

```
path = "C:\\Users\\hp\\Google Drive\\Fiverr Work\\2022\\33. Computer Vision Course\\pict
ures"

img = cv2.imread(path+"\\coloredChips.png")
imgResized = cv2.resize(img, (200, 400))

print(imgResized.shape)

b, g, r = getGrayscaleImages(imgResized)
blue, green, red = colorChannelImages(imgResized)

plt.figure(figsize=(15,10))
plt.subplot(3,3,1)
plt.imshow(b, cmap="gray")
plt.subplot(3,3,2)
plt.imshow(g, cmap="gray")
plt.subplot(3,3,3)
plt.imshow(r, cmap="gray")
plt.subplot(3,3,4)
plt.imshow(blue, cmap="gray")
plt.subplot(3,3,5)
plt.imshow(green, cmap="gray")
plt.subplot(3,3,6)
plt.imshow(red, cmap="gray")
```

```
(400, 200, 3)
```

Out[22]:

```
<matplotlib.image.AxesImage at 0x19fe1909220>
```



In [18]:

```
grayscalResult = np.hstack((b, g, r))
colorResult = np.hstack((blue, green, red))

cv2.imshow("image", imgResized)
cv2.imshow("grayScale channel Images", grayscalResult)
cv2.imshow("color channel Images", colorResult)

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

Simple method to Extract the channel

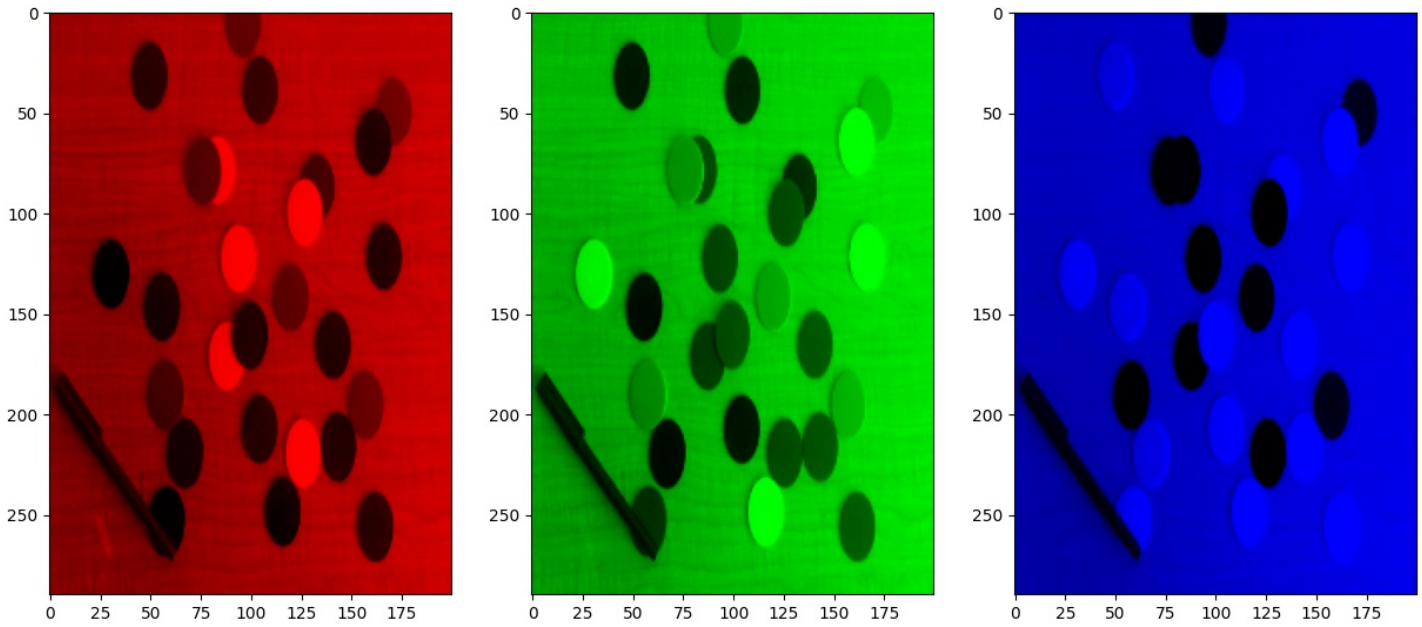
In [23]:

```
import cv2

path = "C:\\Users\\hp\\Google Drive\\Fiverr Work\\2022\\33. Computer Vision Course\\pict
ures"

image = cv2.imread(path+"\\coloredChips.png")
# image = cv2.imread(colorStripe)#pass correct object
image =cv2.resize(image, (200,290))
bw,gw,rw=cv2.split(image)#pass correct object

b = image.copy()
# set green and red channels to 0
b[:,:,1] = 0
b[:,:,2] = 0
g = image.copy()
# set blue and red channels to 0
g[:,:,0] = 0
g[:,:,2] = 0
r = image.copy()
# set blue and green channels to 0
r[:,:,0] = 0
r[:,:,1] = 0

plt.figure(figsize=(15,10))
plt.subplot(1,3,1)
```

```
plt.imshow(b, cmap="gray")
plt.subplot(1,3,2)
plt.imshow(g, cmap="gray")
plt.subplot(1,3,3)
plt.imshow(r, cmap="gray")
```

Out[23]:

```
<matplotlib.image.AxesImage at 0x19fe1c8e5b0>
```



In [24]:

```
# RGB - Blue
cv2.imshow('B-RGB', b)
cv2.imshow('BW B-RGB', bw)
# RGB - Green
cv2.imshow('G-RGB', g)
cv2.imshow('Gw-RGB', gw)
# RGB - Red
cv2.imshow('R-RGB', r)
cv2.imshow('w-RGB', rw)

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

Before Executing the part 02 of this plab you have to Open Countoring_01.ipynb file and uderstand how we find the contours in an image!

02. Image Preprocessing for Edge Detection

To detect objects in an image a common approach is to find the contours of the objects against the image background. The contours are referred to as edges in image processing, to find these edges a number of pre-processing steps can be done to an image to get better edge detection. Do the following steps to the image piece05.png.

- **Convert it to grayscale**
- **Blur the gray image using Gaussian Blur to reduce the noise in the image**
- **Use canny to find the edges**
- **Dilate the edges to fill in small gaps that might appear when using canny**
- **Now use the findContors on the diluted image, How many objects can you find?**

Simple

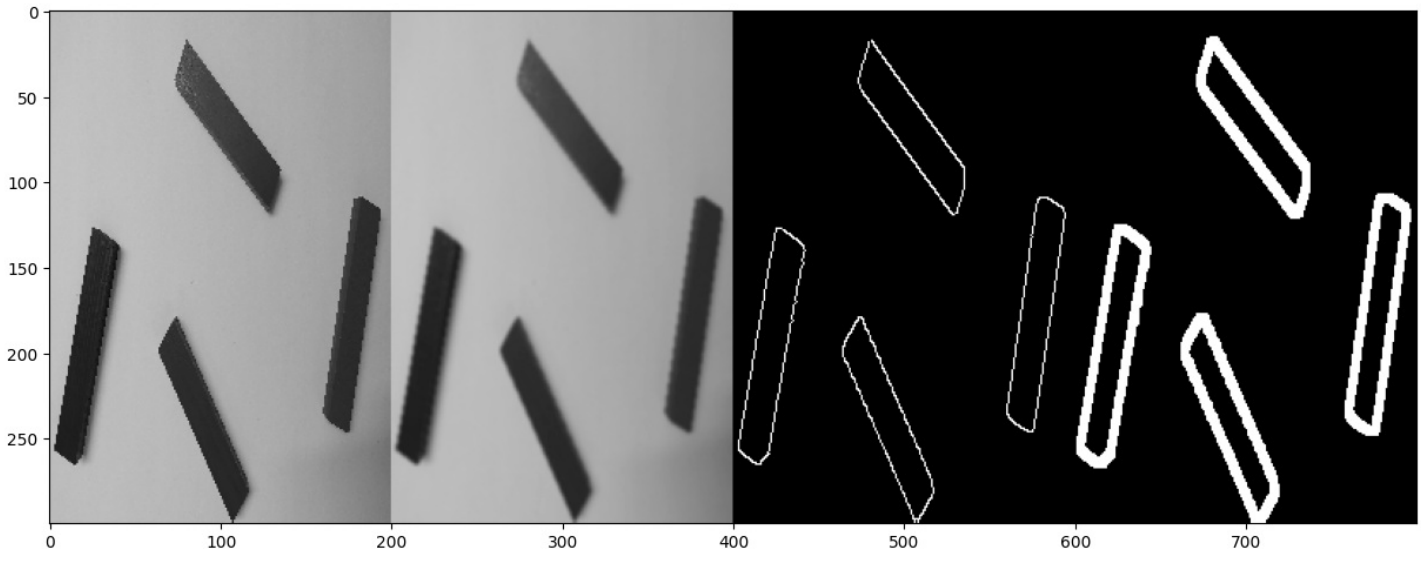
In [28]:

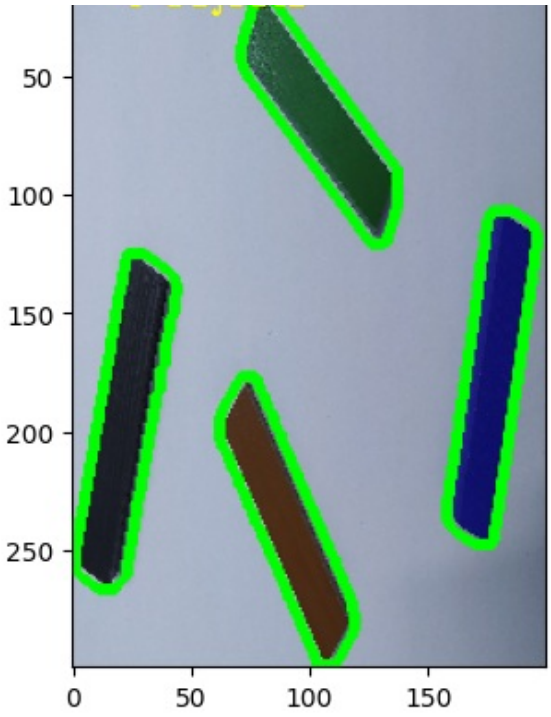
```
import cv2 as cv
import numpy as np

path ="C:\\Users\\hp\\Google Drive\\Fiverr Work\\2022\\15. Teaching OpenCV to Client\\Pi
cs+scripts\\Pictures"

img = cv.imread(path + "\\piece05.png")
imgResized = cv.resize(img, (200, 300))

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

imgGray = cv.cvtColor(imgResized, cv.COLOR_BGR2GRAY)

blurImg = cv.GaussianBlur(imgGray, (5,5), 0)

cannyImge = cv.Canny(blurImg, 10, 150)

imgDilation = cv.dilate(cannyImge, kernel, iterations=1)

contours, hierarchy = cv.findContours(imgDilation,
    cv.RETR_EXTERNAL, cv.CHAIN_APPROX_NONE)

print(len(contours))


cv.drawContours(imgResized, contours, -1, (0, 255, 0), 3)
cv.putText(imgResized, str(len(contours))+" objects", (20, 20), cv.FONT_HERSHEY_COMPLEX,
            0.5, (255, 255, 0), 1)

result = np.hstack((imgGray, blurImg, cannyImge, imgDilation))

plt.figure(figsize=(15,10))
plt.imshow(result, cmap="gray")

plt.figure(figsize=(5,5))
plt.imshow(imgResized, cmap="gray")

# cv.imshow("image", imgResized)
# cv.imshow("output", result)

# cv.waitKey()
# cv.destroyAllWindows()
```

4

Out[28]:

<matplotlib.image.AxesImage at 0x19fe23ab130>





With trackbar

In [26]:

```
# Exercise 05: Preparing Image for edge detection

import cv2 as cv
import numpy as np

def nothing(x):
    pass

path ="C:\\Users\\hp\\Google Drive\\Fiverr Work\\2022\\15. Teaching OpenCV to Client\\Pics+scripts\\Pictures"

cv.namedWindow("output")

cv.createTrackbar("kernel1", "output", 0, 55, nothing)
cv.createTrackbar("kernel2", "output", 0, 55, nothing)
cv.createTrackbar("cannyLower", "output", 3, 255, nothing)
cv.createTrackbar("cannyUpper", "output", 255, 255, nothing)

while True:

    kernel1 = cv.getTrackbarPos("kernel1", "output")
    kernel2 = cv.getTrackbarPos("kernel2", "output")
    cannyLower = cv.getTrackbarPos("cannyLower", "output")
    cannyUpper = cv.getTrackbarPos("cannyUpper", "output")

    img = cv.imread(path + "\\piece05.png")
    imgResized = cv.resize(img, (200, 300))

    # # for dilation
    dilateKernel = np.ones((5,5), "uint8")

    imgGray = cv.cvtColor(imgResized, cv.COLOR_BGR2GRAY)

    # As kernel is size of odd dimension
    if (kernel1*kernel2)%2 == 1:
        dilateKernel = np.ones((kernel1,kernel2), "uint8")
        blurImg = cv.GaussianBlur(imgGray, (kernel1,kernel2), 0)
    else:
        blurImg = cv.GaussianBlur(imgGray, (3,3), 0)

    cannyImge = cv.Canny(blurImg, cannyLower, cannyUpper)
```

```
    imgDilation = cv.dilate(cannyImge, dilateKernel, iterations=1)

    contours, hierarchy = cv.findContours(imgDilation,
        cv.RETR_EXTERNAL, cv.CHAIN_APPROX_NONE)

    print(len(contours))

    cv.drawContours(imgResized, contours, -1, (0, 255, 0), 3)
    cv.putText(imgResized, str(len(contours))+" objects", (20, 20), cv.FONT_HERSHEY_COMP
LEX,
                0.5, (255, 255, 0), 1)

    result = np.hstack((imgGray, blurImg, cannyImge, imgDilation))

    cv.imshow("image", imgResized)
    cv.imshow("output", result)

    k = cv.waitKey(1)

    if k == ord("q"):
        cv.destroyAllWindows()
```

```
4
4
4
4
4
4
4
4
4
4
4
4
4
4
4
4
4
4
4
4
4
4
4
4
4
4
4
4
4
4
4
4
4
4
4
4
4
4
4
4
4
4
4
4
4

---------------------------------------------------------------------------
error                                     Traceback (most recent call last)
c:\Users\hp\Google Drive\Fiverr Work\2022\33. Computer Vision Course\Lab 04\Lab04Computer
Vision.ipynb Cell 22 in 2
```

```
vision.ipynb Cell 22 in 2
     <a href='vscode-notebook-cell:/c%3A/Users/hp/Google%20Drive/Fiverr%20Work/2022/33.%20Computer%20Vision%20Course/Lab%2004/Lab04ComputerVision.ipynb#Y104sZmlsZQ%3D%3D?line=15'>16</a> cv.createTrackbar("cannyUpper", "output", 255, 255, nothing)
     <a href='vscode-notebook-cell:/c%3A/Users/hp/Google%20Drive/Fiverr%20Work/2022/33.%20Computer%20Vision%20Course/Lab%2004/Lab04ComputerVision.ipynb#Y104sZmlsZQ%3D%3D?line=17'>18</a> while True:
---> <a href='vscode-notebook-cell:/c%3A/Users/hp/Google%20Drive/Fiverr%20Work/2022/33.%20Computer%20Vision%20Course/Lab%2004/Lab04ComputerVision.ipynb#Y104sZmlsZQ%3D%3D?line=19'>20</a>     kernel1 = cv.getTrackbarPos("kernel1", "output")
     <a href='vscode-notebook-cell:/c%3A/Users/hp/Google%20Drive/Fiverr%20Work/2022/33.%20Computer%20Vision%20Course/Lab%2004/Lab04ComputerVision.ipynb#Y104sZmlsZQ%3D%3D?line=20'>21</a>     kernel2 = cv.getTrackbarPos("kernel2", "output")
     <a href='vscode-notebook-cell:/c%3A/Users/hp/Google%20Drive/Fiverr%20Work/2022/33.%20Computer%20Vision%20Course/Lab%2004/Lab04ComputerVision.ipynb#Y104sZmlsZQ%3D%3D?line=21'>22</a>     cannyLower = cv.getTrackbarPos("cannyLower", "output")

error: OpenCV(4.6.0) D:\a\opencv-python\opencv-python\opencv\modules\highgui\src\window_w32.cpp:2581: error: (-27:Null pointer) NULL window: 'output' in function 'cvGetTrackbarPos'
```

Digital Image Processing

Lab 05

Finding Length & Width of Object

---

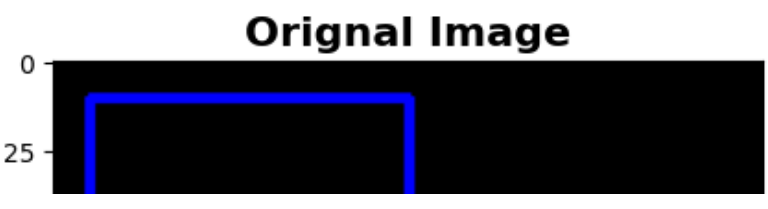
01. What is cv2.minAreaRect(), and how it can used?

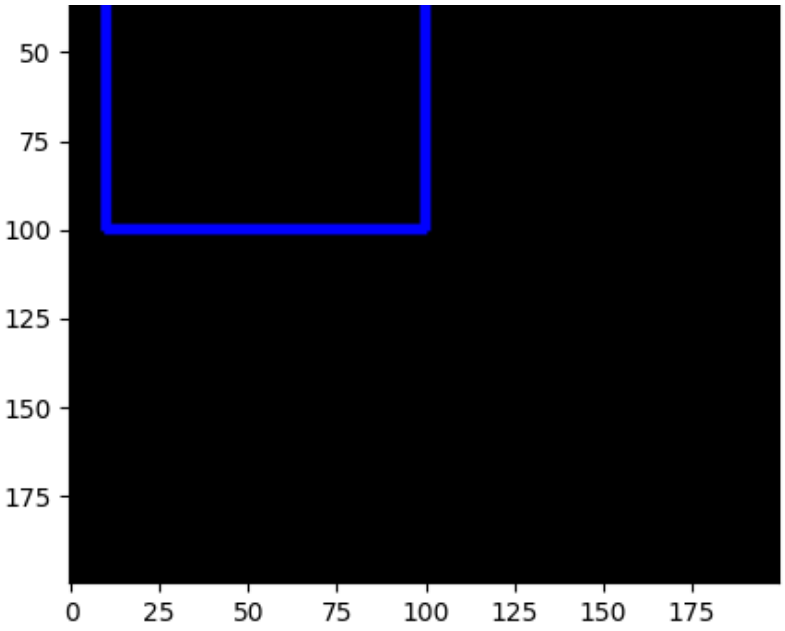
cv2.minAreaRect() is a function in the OpenCV (Open Computer Vision) library that can be used to find the minimum area rectangle that encloses a set of points. The function returns a rotated rectangle, which is defined by the center point, size, and orientation of the rectangle.

In [2]:

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

# Define a set of points
points = np.array([[10, 10], [100, 10], [100, 100], [10, 100]], np.int32)

# Find the minimum area rectangle that encloses the points
rect = cv2.minAreaRect(points)
print(rect)

# Extract the center, size, and orientation of the rectangle
(center_x, center_y), (width, height), angle = rect

# Draw the rectangle on an image
image = np.zeros((200, 200, 3), np.uint8)
box = cv2.boxPoints(rect)
print(box)
box = np.int0(box)
cv2.drawContours(image, [box], 0, (0, 0, 255), 2)

# Display the image

plt.figure(figsize=(5,5))
plt.imshow(image, cmap="gray")
plt.title("Orignal Image", fontsize = 16, fontweight = 'bold')

cv2.imshow('Image with rectangle', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
((55.0, 55.0), (90.0, 90.0), 90.0)
[[ 10.  10.]
 [100.  10.]
 [100. 100.]
 [ 10. 100.]]
```





In this example, points is a 2D NumPy array containing the coordinates of the points that define the region we want to enclose with a rectangle. The cv2.minAreaRect() function takes this array as an input and returns the minimum area rectangle that encloses the points.

The cv2.boxPoints() function is used to convert the rotated rectangle returned by cv2.minAreaRect() into a list of points that can be used to draw the rectangle on an image using cv2.drawContours().

02. Make a Function to Detect Contours Using Binary Thresholding

In [3]:

```
import cv2 as cv

def detectUsingThresh(img, threshVal = 140):

        # convert to grayscale
        imgGray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

        # invert grascale
        invertGray = cv.bitwise_not(imgGray)

        # apply binary thresholding
        ret, binaryThresh = cv.threshold(invertGray, threshVal, 255, cv.THRESH_BINARY)

        # detect contours
        contours, hierarchy = cv.findContours(binaryThresh, cv.RETR_LIST, cv.CHAIN_APPRO
X_SIMPLE)

        objectContours = []

        # save the contours with area atleast greater than 2000 pixels (50x40 = 2000)
        for cont in contours:
            area = cv.contourArea(cont)

            if area > 2000:
                objectContours.append(cont)

        return objectContours
```

In [5]:

```
import matplotlib.pyplot as plt

img = cv.imread("1.jpeg")
img = cv.resize(img, (400, 600))
```

```
contours = detectUsingThresh(img)
print(len(contours))

cv.drawContours(img, contours, -1, (0,0,255), 3)

plt.figure(figsize=(5,5))
plt.imshow(img, cmap='gray')
plt.axis(False)
# cv.imshow("image", img)
# cv.waitKey()

cv.destroyAllWindows()
```

```
1
```

03. Make a Function to Detect Contours Using Canny filter

In [6]:

```
import cv2 as cv
import numpy as np

def detectUsingCanny(img, lower= 120,upper= 160):

        # Blur the image to remove noise
        blurred_image = cv.GaussianBlur(img,(5,5),0)

        # Apply canny edge detection
        cannyImge = cv.Canny(blurred_image, lower, upper)

        # kernel for image dilation
        dilateKernel = np.ones((5,5), "uint8")

        # image dilation to enhance the edges
        imgDilation = cv.dilate(cannyImge, dilateKernel, iterations=1)

        contours, hierarchy = cv.findContours(imgDilation, cv.RETR_EXTERNAL, cv.CHAIN_AP
PROX_SIMPLE)

        cv.imshow("dilat", imgDilation)

        objectContours = []

        for cont in contours:
            area = cv.contourArea(cont)
```

```
            if area > 2000:
                objectContours.append(cont)

        return objectContours
```

In [9]:

```
import matplotlib.pyplot as plt

img = cv.imread("1.jpeg")
img = cv.resize(img, (400, 600))


contours = detectUsingCanny(img)
print(len(contours))

cv.drawContours(img, contours, -1, (0,0,255), 3)

plt.figure(figsize=(5,5))
plt.imshow(img)
plt.axis(False)
# cv.imshow("image", img)
# cv.waitKey()

cv.destroyAllWindows()
```

1

Which one is better Binary Thresholding or Canny filter!

Both binary thresholding and the Canny edge filter are commonly used techniques for detecting edges and contours in images. The choice of which method to use depends on the specific requirements of your application.

Binary thresholding is a simple and fast method for separating objects in an image from the background. It works by thresholding an image so that pixels with values above a certain threshold are set to one value (usually white), and pixels with values below the threshold are set to another value (usually black). This results in a binary image with white pixels representing object pixels and black pixels representing background pixels. Binary thresholding is suitable for images with clear and distinct object boundaries.

The Canny edge filter is a more sophisticated method for detecting edges and contours in images. It uses a multi-stage algorithm to detect edges that are more likely to correspond to object boundaries. The Canny edge

filter works by first smoothing the image using a Gaussian filter to reduce noise, then finding the gradient intensity and direction of the image using the Sobel operator. It then applies non-maximum suppression to thin the edges and removes false edges caused by noise. Finally, it applies hysteresis thresholding to suppress weak edges and retain only strong edges. The Canny edge filter is more robust than binary thresholding and can handle images with more complex and varied edge features. However, it is also slower and more computationally intensive.

In general, the Canny edge filter is a better choice for contour detection in images with complex edge features or low contrast, while binary thresholding is a good choice for images with clear and distinct object boundaries. Ultimately, the best method for contour detection will depend on the characteristics of the images you are working with and the specific requirements of your application.

04. Now make class with name MyDetectionMethods and include the above created function in it

In [11]:

```
import cv2 as cv
import numpy as np

class MyDetectionMethods():
    # def __init__(self):
    #     pass

    def detectUsingThresh(img, threshVal = 140):

        # convert to grayscale
        imgGray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

        # invert grascale
        invertGray = cv.bitwise_not(imgGray)

        # apply binary thresholding
        ret, binaryThresh = cv.threshold(invertGray, threshVal, 255, cv.THRESH_BINARY)

        # detect contours
        contours, hierarchy = cv.findContours(binaryThresh, cv.RETR_LIST, cv.CHAIN_APPRO
X_SIMPLE)

        objectContours = []

        # save the contours with area atleast greater than 2000 pixels (50x40 = 2000)
        for cont in contours:
            area = cv.contourArea(cont)

            if area > 2000:
                objectContours.append(cont)
        return objectContours

    def detectUsingCanny(img, lower= 120,upper= 160):

        # Blur the image to remove noise
        blurred_image = cv.GaussianBlur(img,(9,9),0)

        # Apply canny edge detection
        cannyImge = cv.Canny(blurred_image, lower, upper)

        # kernel for image dilation
        dilateKernel = np.ones((5,5), "uint8")

        # image dilation to enhance the edges
        imgDilation = cv.dilate(cannyImge, dilateKernel, iterations=1)

        contours, hierarchy = cv.findContours(imgDilation, cv.RETR_EXTERNAL, cv.CHAIN_AP
PROX_SIMPLE)

        # cv.imshow("dilat", imgDilation)
```

```
        objectContours = []

        for cont in contours:
            area = cv.contourArea(cont)

            if area > 2000:
                objectContours.append(cont)

        return objectContours
```

In [12]:

```
import matplotlib.pyplot as plt

# make object of above created class
detector = MyDetectionMethods

img = cv.imread("1.jpeg")
img = cv.resize(img, (400, 600))

# now call the above function from class using object detector
contours = detector.detectUsingCanny(img=img, lower=120, upper=160)
print(len(contours))

cv.drawContours(img, contours, -1, (0,0,255), 3)

plt.figure(figsize=(5,5))
plt.imshow(img)
plt.axis(False)
# cv.imshow("image", img)
cv.waitKey()

cv.destroyAllWindows()
```

```
1
```

05. Aruco Library of OpenCV

The Aruco library is a part of the open source computer vision library OpenCV (OpenCV stands for Open Source Computer Vision). It is a collection of tools and functions for detecting and identifying augmented reality (AR) markers in images and video streams. AR markers are small, square-shaped black and white patterns that are used to represent virtual objects in the real world.

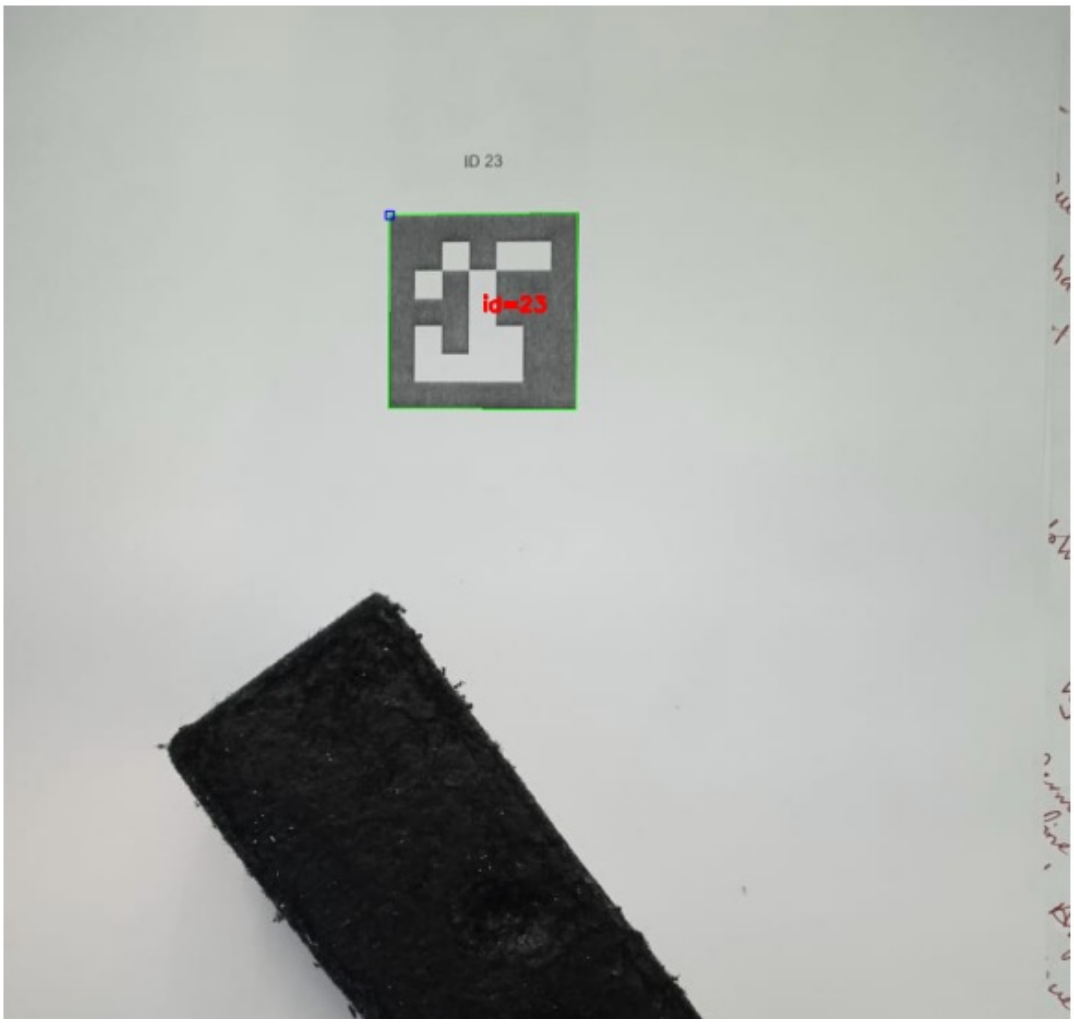
The Aruco library provides a set of functions for detecting and identifying AR markers in images. It can detect

markers of different sizes and types, and it can also estimate the pose (position and orientation) of the markers in the image. This information can be used to overlay virtual objects on top of the markers in real-time, creating the illusion of augmented reality.

To use the Aruco library in your Python code, you will need to install the OpenCV library and import the cv2.aruco module. Here is an example of how you can use the Aruco library to detect and identify AR markers in an image:

In [15]:

```
import cv2

# Load the image
image = cv2.imread("111.jpeg")

# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Create a dictionary of AR markers
dictionary = cv2.aruco.Dictionary_get(cv2.aruco.DICT_5X5_50)

# Detect the markers in the image
corners, ids, _ = cv2.aruco.detectMarkers(gray, dictionary)

# If markers were detected, draw them on the image
if ids is not None:
    image = cv2.aruco.drawDetectedMarkers(image, corners, ids)

# Show the image
plt.figure(figsize=(10,10))
plt.imshow(image)
plt.axis(False)

cv2.imshow("Image", image)
cv2.waitKey(0)

cv2.destroyAllWindows()
```



06. Detect and draw AR markers in video or webcam

In [16]:

```
import cv2

# capture webcam
cap = cv2.VideoCapture(0)

while True:
    _, frame = cap.read()

    # Convert the image to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Create a dictionary of AR markers
    dictionary = cv2.aruco.Dictionary_get(cv2.aruco.DICT_5X5_50)

    # Detect the markers in the image
    corners, ids, _ = cv2.aruco.detectMarkers(gray, dictionary)

    # If markers were detected, draw them on the image
    if ids is not None:
        frame = cv2.aruco.drawDetectedMarkers(frame, corners, ids)

    # Show the image
    cv2.imshow("Image", frame)
    k = cv2.waitKey(1)

    if k == ord("q"):
        cap.release()
        cv2.destroyAllWindows()
        break

cv2.destroyAllWindows()
```

This code will detect and draw the boundaries of any AR markers present in the image. You can then use the ids and corners variables to identify the markers and estimate their pose.

The Aruco library is a powerful tool for creating augmented reality applications. It is widely used in a variety of fields, including robotics, industrial automation, and entertainment.

07. cv.aruco.DetectorParameters_create()

cv2.aruco.DetectorParameters_create() is a function in the OpenCV Aruco library that creates and returns a cv2.aruco.DetectorParameters object. '

The cv2.aruco.DetectorParameters object is used to specify various parameters for the Aruco marker detector. These parameters can be used to fine-tune the behavior of the detector and improve its performance for a specific application.

Here is an example of how cv2.aruco.DetectorParameters_create() can be used in Python:

In [17]:

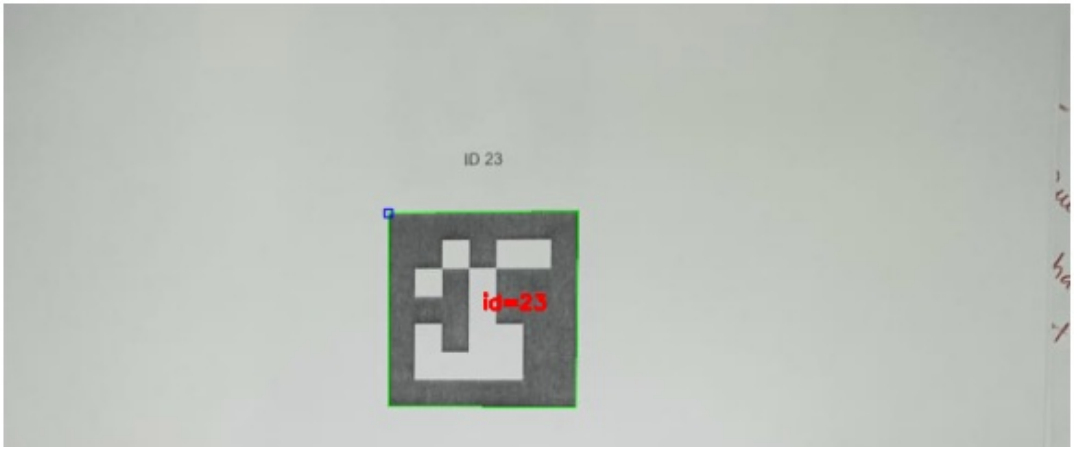
```
import cv2

# Create a DetectorParameters object with default values
params = cv2.aruco.DetectorParameters_create()

# Set the corner refinement method to "subpix"
params.cornerRefinementMethod = cv2.aruco.CORNER_REFINE_SUBPIX

# Use adaptive thresholding to detect markers
params.adaptiveThreshConstant = 10
```

This code creates a cv2.aruco.DetectorParameters object with default values and then modifies some of the parameters to customize the behavior of the detector. The maxErr parameter specifies the maximum allowable error in pixels between the detected corners and the true corners of the markers. The cornerRefinementMethod parameter specifies the method to use for refining the corners of the markers. The adaptiveThreshConstant parameter specifies the constant to use for adaptive thresholding, which is a method for detecting markers in images with varying illumination.

The cv2.aruco.DetectorParameters object can then be passed to the cv2.aruco.detectMarkers() function as an argument to specify the parameters to use for marker detection:

In [18]:

```
import cv2

# Load the image
image = cv2.imread("111.jpeg")

# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Create a dictionary of AR markers
dictionary = cv2.aruco.Dictionary_get(cv2.aruco.DICT_5X5_50)

# Detect the markers in the image using the specified parameters
corners, ids, _ = cv2.aruco.detectMarkers(gray, dictionary, parameters=params)

# If markers were detected, draw them on the image
if ids is not None:
    image = cv2.aruco.drawDetectedMarkers(image, corners, ids)

# Show the image
# Show the image
plt.figure(figsize=(10,10))
plt.imshow(image)
plt.axis(False)

cv2.imshow("Image", image)
cv2.waitKey(0)

cv2.destroyAllWindows()
```



This code will use the parameters specified in the params object to detect AR markers in the image. The cv2.aruco.DetectorParameters object can be used to fine-tune the behavior of the marker detector and improve its performance for a specific application.

08. cv.aruco.Dictionary_get(cv.aruco.DICT_5X5_50)

cv2.aruco.Dictionary_get(cv.aruco.DICT_5X5_50) is a function in the OpenCV Aruco library that returns a predefined dictionary of AR markers (augmented reality markers). The cv2.aruco.Dictionary_get() function takes a single argument, which specifies the type of dictionary to return. In this case, the argument cv.aruco.DICT_5X5_50 specifies that the function should return a dictionary of 50 unique 5x5 AR markers.

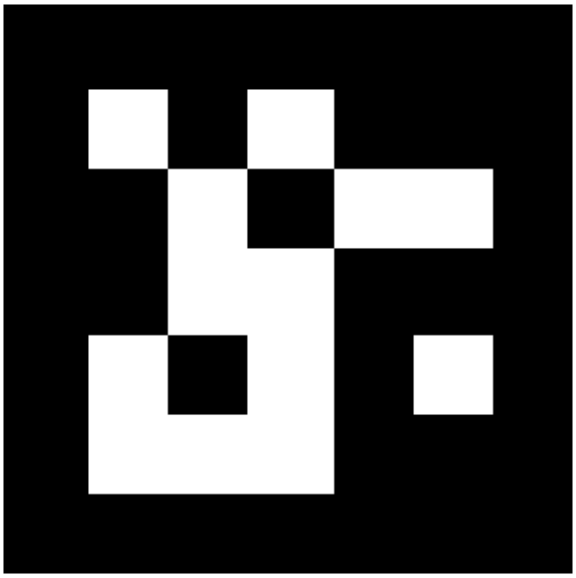
A dictionary of AR markers is a collection of unique markers that can be used to represent virtual objects in the real world. Each marker in the dictionary is a black and white pattern that can be detected and identified by a computer vision system. The Aruco library provides several predefined dictionaries of markers, each of which is optimized for different applications.

Here is an example of how the cv2.aruco.Dictionary_get() function can be used in Python:

In [20]:

```
import cv2

# Create a dictionary of AR markers
dictionary = cv2.aruco.Dictionary_get(cv2.aruco.DICT_5X5_50)

# Generate a marker with id 0 from the dictionary
marker = cv2.aruco.drawMarker(dictionary, 0, 100)

# Show the marker
plt.figure(figsize=(4,4))
plt.imshow(marker, cmap='gray')
plt.axis(False)
cv2.imshow("Marker", marker)
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```



This code creates a dictionary of 50 unique 5x5 AR markers and generates a marker with id 0 from the dictionary. The marker is then displayed on the screen. You can use the cv2.aruco.drawMarker() function to generate markers with different sizes and IDs from the dictionary.

The cv2.aruco.Dictionary_get() function is commonly used in conjunction with the cv2.aruco.detectMarkers() function to detect and identify AR markers in images and video streams. The cv2.aruco.detectMarkers() function takes a dictionary of markers as an argument and uses it to identify the markers in the image.

Now Let's build the project to calculate the width and height of an object!

In [21]:

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

In [22]:

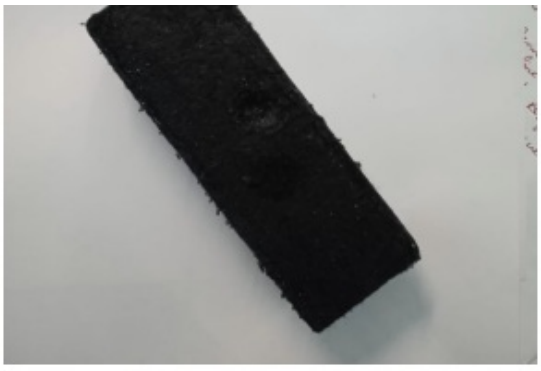
```
detector = MyDetectionMethods #make an object of MyDetectionMethods class created above
```

In [23]:

```
# read image and display

image = cv2.imread("111.jpeg")

plt.figure(figsize=(5,5))
plt.imshow(image)
plt.axis(False)
```

Out[23]:

```
(-0.5, 779.5, 1039.5, -0.5)
```



In [24]:

```
# Now find contours by using the obect created above detector

contours = detector.detectUsingCanny(image)
print(len(contours))
```

2

In [25]:

```
# Draw the contours

cv2.drawContours(image, contours, -1, (100,200,100), 7)

plt.figure(figsize=(5,5))
plt.imshow(image)
plt.axis(False)
```

Out[25]:

(-0.5, 779.5, 1039.5, -0.5)

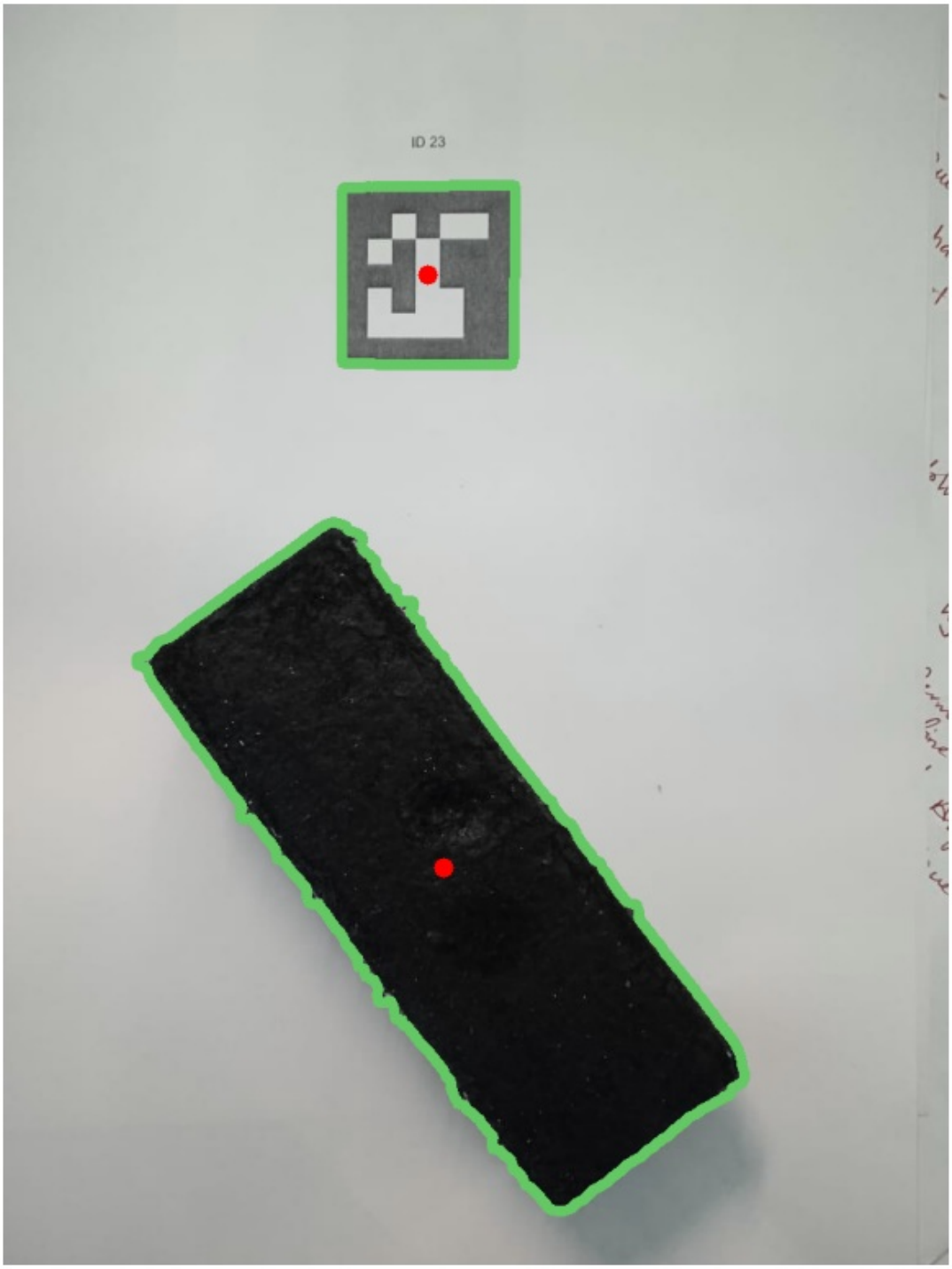
In [26]:

```
# Now find out the center of contours and draw the circle

for cont in contours:

    (centerX, centerY), (w, h), angle = cv2.minAreaRect(cont)

    cv.circle(image, (int(centerX),int(centerY)),8, (255,0,0),-1)
```

```
plt.figure(figsize=(10,10))
plt.imshow(image)
plt.axis(False)
```

Out[26]:

```
(-0.5, 779.5, 1039.5, -0.5)
```



In [27]:

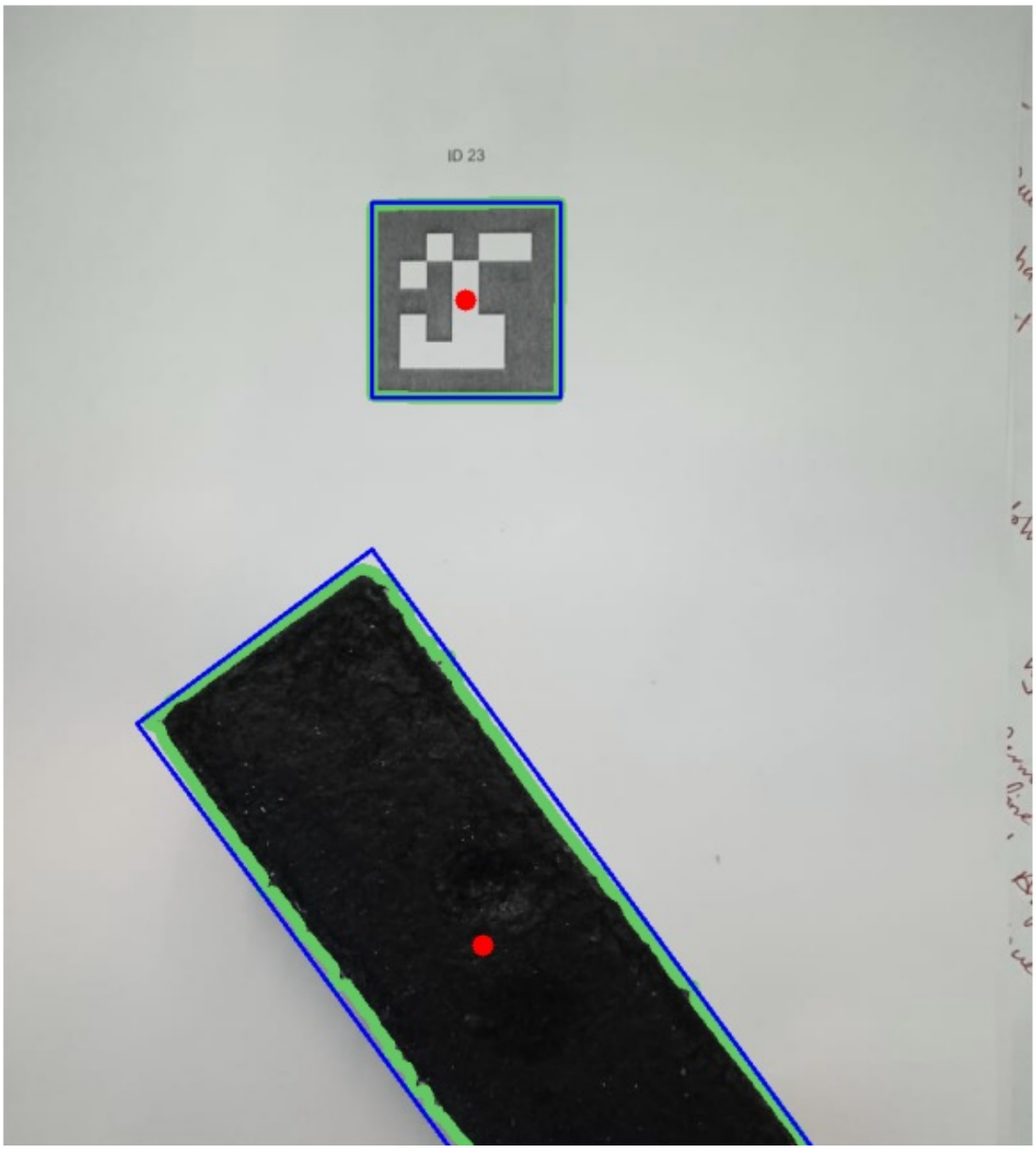
```
# The border line is not look good so in order to draw perfect line you have to do this

for cont in contours:


    result = cv2.minAreaRect(cont)

    (centerX, centerY), (w, h), angle = result

    cv.circle(image, (int(centerX),int(centerY)),8, (255,0,0),-1)

    # opposite of minAreaRect()
    boundingBox = cv.boxPoints(result)
    # #convert to integer
```

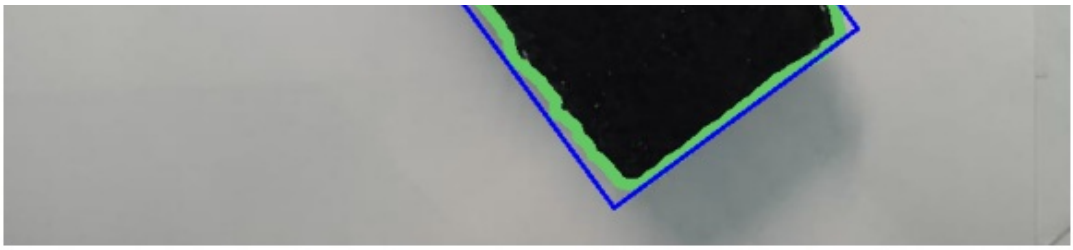
```
    boundingBox = np.int0(boundingBox)
    print(boundingBox)

     # # draw rectangle
    cv.line(image, (boundingBox[0][0], boundingBox[0][1]), (boundingBox[1][0],boundingBo
x[1][1]), (0,0,255),2)
    cv.line(image, (boundingBox[0][0], boundingBox[0][1]), (boundingBox[3][0],boundingBo
x[3][1]), (0,0,255),2)
    cv.line(image, (boundingBox[1][0],boundingBox[1][1]), (boundingBox[2][0],boundingBox
[2][1]), (0,0,255),2)
    cv.line(image, (boundingBox[3][0],boundingBox[3][1]), (boundingBox[2][0],boundingBox
[2][1]), (0,0,255),2)
    #




plt.figure(figsize=(10,10))
plt.imshow(image)
plt.axis(False)
```

```
[[ 101  544]
 [ 279  412]
 [ 624  881]
 [ 446 1012]]
[[279 149]
 [422 149]
 [422 297]
 [279 297]]
```

Out[27]:

```
(-0.5, 779.5, 1039.5, -0.5)
```





In [28]:

```
# Draw the width and height of each contour or object

for cont in contours:


    result = cv2.minAreaRect(cont)

    (centerX, centerY), (w, h), angle = result

    cv.circle(image, (int(centerX),int(centerY)),8, (255,0,0),-1)

    # opposite of minAreaRect()
    boundingBox = cv.boxPoints(result)
    # #convert to integer
    boundingBox = np.int0(boundingBox)
    print(boundingBox)

     # # draw rectangle
    cv.line(image, (boundingBox[0][0], boundingBox[0][1]), (boundingBox[1][0],boundingBo
x[1][1]), (0,0,255),2)
    cv.line(image, (boundingBox[0][0], boundingBox[0][1]), (boundingBox[3][0],boundingBo
x[3][1]), (0,0,255),2)
    cv.line(image, (boundingBox[1][0],boundingBox[1][1]), (boundingBox[2][0],boundingBox
[2][1]), (0,0,255),2)
    cv.line(image, (boundingBox[3][0],boundingBox[3][1]), (boundingBox[2][0],boundingBox
[2][1]), (0,0,255),2)
    #

    # # print height and width
    cv.putText(image, f'width: {w} cm', (int(centerX-50),int(centerY-30)),
                cv.FONT_HERSHEY_PLAIN, 2, (255,0,255), 2)
    cv.putText(image, f'height: {h} cm', (int(centerX-50),int(centerY-5)),
                cv.FONT_HERSHEY_PLAIN, 2, (255,0,255), 2)


plt.figure(figsize=(10,10))
plt.imshow(image)
plt.axis(False)
```

```
[[ 101  544]
 [ 279  412]
 [ 624  881]
 [ 446 1012]]
[[279 149]
 [422 149]
 [422 297]
 [279 297]]
```

Out[28]:

```
(-0.5, 779.5, 1039.5, -0.5)
```

But this width and height is not in cm it is in pixels, so we need to know how many pixels are in 1 cm