OpenCV

* import cv2

VGA = 640 width x 480 height
HD = 1280 width x 720 height
FHD = 1920 width x 1080 height
4k = 3840 width x 2160 height

- Binary image - 2 levels of the image with pixels 0 black and 1 white
- For normal images we are using 8-bit values, 2^8 which is 256. So we have 254 colors of gray and 0 is black and 256 is white. Now it is known as a grayscale image.
- In RGB images, we have 3 layers of grayscale images with R, G, B pixels which give the full-color images.
- So, RGB VHD image is 640 width x 480 height x 3 pixels

Import Images

cv2.imread("path of the image") #imports the image

- Now we have to see what it imported

cv2.imshow("image window name", define image we want to display) #displays the image

- Now, the image will be shown but it'll vanish within milliseconds. So we need to add delay.

cv2.waitKey("milliseconds") - adds delay to show the image.
cv2.waitKey(0) - infinite delay the image will be seen till you stop running the code.

```
# READ AND PRINT THE IMAGE
cv2.imread("path of the image")
cv2.imshow("image window name", define image we want to display)
cv2.waitKey(0)
```

Import Videos

cv2.VideoCapture("path of the video") #imports the video

- Since video is a continuous frame of images. We need to use "while loop" in order to see the video.

```
while True:
    Success, img = cap.read() #save our image in an "img" variable and say whether it is successful or not.
    cv2.imshow("image window name", define image we want to display) # This shows the Video
```

```
# READ AND PRINT THE VIDEO
cv2.VideoCapture("path of the video")
while True:
    Success, img = cap.read()
    cv2.imshow("image window name", define image we want to display)
```

- Now we add the condition that takes the keyboard input to come out of the while loop.

```
If cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

the above lines will take the “q” as input, as soon as it takes “q” input it breaks the loop and stops the playing video.

```
# READ AND PRINT THE VIDEO
cv2.VideoCapture(“path of the video”)
while True:
        Success, img = cap.read()
        cv2.imshow(“image window name”, define image we want to display)
        if cv2.waitKey(1) & 0xFF == ord(‘q’):
                break
```

Access Webcam

- Using a webcam is similar to importing video.

```
cap = VideoCapture(0) # access the camera
```

- So, we want the video frame to be a specific size, so we declare some parameters.

```
cap.set(3,640) #defines width which is id number 3,
cap.set(4,480) #defines height which is id number 4,
cap.set(10,100) #defines brightness level which is id number 10,
```

```
# LIVE VIDEO with Brightness adjustment
cv2.VideoCapture(0)
cap.set(3,640)
cap.set(4,480)
while True:
        Success, img = cap.read()
        cv2.imshow(“image window name”, define image we want to display)
        if cv2.waitKey(1) & 0xFF == ord(‘q’):
                break
```

Basic Functions

1) Changing Colour Spaces

- Let’s see the basic functions that are required during projects.
- We convert RGB images into GRAYscale images.

```
ImgGray = cv2.cvtColor(image you want to convert , cv2.COLOR_BGR2GRAY)  #Converts the BGR image into gray scale image
```

- In OpenCV, images will be in BGR format.

```
#CHANGE THE COLOUR SPACE OF IMAGE
img= cv2.imread(“image path”)
ImgGray = cv2.cvtColor(image you want to convert , cv2.COLOR_BGR2GRAY)
cv2.imshow(“image window name”, define image we want to display)
cv2.waitKey(0)
```

---

2) Adding Blur

- In most of the cases we add Gaussian Blur to the image

cv2.GaussianBlur("image you want to add, (kernel size) , sigma(x) # adds the Gaussian blur to image

---

```
#ADDING BLUR TO THE GRAYSCALE IMAGE
img= cv2.imread("image path")
ImgGray = cv2.cvtColor(image you want to convert , cv2.COLOR_BGR2GRAY)
ImgBlur = cv2.GaussianBlur("imgGray", (7,7),  0)
cv2.imshow("img", define image we want to display)
cv2.imshow("imgGray",  window name", define image we want to display)
cv2.imshow("imgBlur", define image we want to display)
cv2.waitKey(0)
```

---

3) Edge Detection

- Let's see a "canny" edge detector.

imgCanny =cv2.Canny(the image you wanted to apply, Threshold value, Threshold value) #Find the edges.

- If we want to get fewer edges, we can modify the threshold value.The more the threshold value, the lesser the number of edges.

```
#FINDING EDGES IN THE IMAGE
img= cv2.imread("image path")
ImgGray = cv2.cvtColor(image you want to convert , cv2.COLOR_BGR2GRAY)
ImgBlur = cv2.GaussianBlur("imgGray", (7,7),  0)
ImgCanny =cv2.Canny(img, 100, 100)
cv2.imshow("img", define image we want to display)
cv2.imshow("imgGray",  ImgGray)
cv2.imshow("imgBlur", ImgBlur)
cv2.imshow("imgCanny", ImgCanny)
cv2.waitKey(0)
```

---

4) Dilation

- Sometimes, Due to disturbances or discontinuity, we cannot detect edges properly. We can increase the thickness of the edges.

```
kernel = np.ones((5,5), np.unit8)
ImgDilation = cv2.Dilate(image on which we want to apply dilation, kernel, iterations)  #increases size of edges
```

- As the number of iterations increases, the Thickness of the edges will increase.

```
# INCREASING THE THICKNESS OF THE EDGES DETECTED ABOVE
img= cv2.imread("image path")
kernel = np.ones((5,5), np.unit8)   #kernel of size 5x5 (all are ones), with 8 unsigned bits
ImgGray = cv2.cvtColor(img , cv2.COLOR_BGR2GRAY)
ImgBlur = cv2.GaussianBlur("imgGray", (7,7),  0)
```

```
ImgCanny =cv2.Canny(img, 100, 100)
ImgDilation =cv2.Dilate(imgCanny, kernel, iterations=1)
cv2.imshow("img", define image we want to display)
cv2.imshow("imgGray",  ImgGray)
cv2.imshow("imgBlur", ImgBlur)
cv2.imshow("imgCanny", ImgCanny)
cv2.imshow("imgDilation", ImgDilation)
cv2.waitKey(0)
```

---

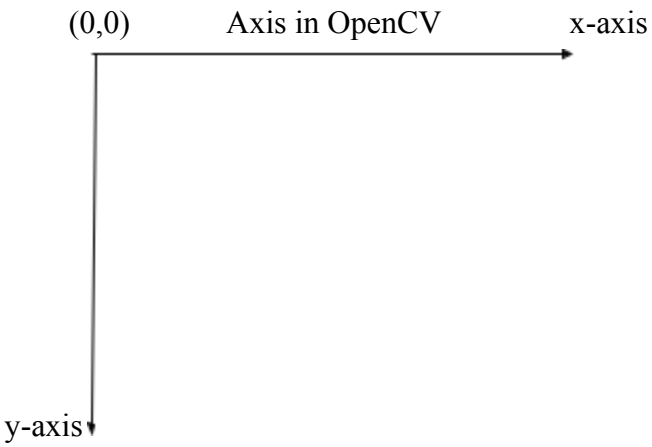
4) Erosion

- This is just the opposite of Dilation, We make edges thinner.

```
ImgEroded = cv2.erode(image on which we want to apply dilation, kernel, iterations)  #decreases size of edges.
```

```
#DECREASE THE SIZE OF EDGES
Img= cv2.imread("image path")
kernel = np.ones((5,5), np.unit8)   #kernel of size 5x5 (all are ones), with 8 unsigned bits
ImgGray = cv2.cvtColor(Img , cv2.COLOR_BGR2GRAY)
ImgBlur = cv2.GaussianBlur("ImgGray", (7,7),  0)
ImgCanny =cv2.Canny(Img, 100, 100)
ImgDilation =cv2.Dilate(ImgCanny, kernel, iterations=1)
ImgEroded = cv2.erode(ImgDilation, kernel, iterations=1)
cv2.imshow("img", define image we want to display)
cv2.imshow("imgGray",  ImgGray)
cv2.imshow("imgBlur", ImgBlur)
cv2.imshow("imgCanny", ImgCanny)
cv2.imshow("imgDilation", ImgDilation)
cv2.imshow("imgEroded", ImgEroded)
cv2.waitKey(0)
```

OpenCV Convention



1) Resize:

- We have to resize the image.

ImgResize = cv2.resize(image we want to resize, (width, height) # resizes the image

- It just increases or decreases the number of pixels, but does not affect the quality of the image.

```
#RESIZE THE IMAGE
Img = cv2.imread("path of image")
print(Img.shape)
ImgResize = cv2.resize(Img,(300,200))
print(ImgResize.shape)
cv2.imshow("Image", Img)
cv2.imshow("Resize",ImgResize)
```

---

2) Crop

ImgCropped = image you want to crop.[height, width] #crops the images

---

```
#CROPPING THE IMAGE
Img = cv2.imread("path of image")
print(Img.shape)
ImgResize = cv2.resize(Img,(300,200))
print(ImgResize.shape)
ImgCropped = Img.[0:200, 200:500] #y-axis, x-axis
cv2.imshow("Croppedimage", ImgCropped)
cv2.imshow("Image", Img)
cv2.imshow("Resize",ImgResize)
cv2.waiKey(0)
```

Shapes and Texts

1) Line

- First, we create a matrix with zeros

Img = np.zeros((512,512,3), np.unit8) # This is a grayscale image
Img[height:width] = blue,green,red #colon is [height:width], Blue,Green, Red
cv2.line(Img,(starting point), (endingpoint),(colour)) # adds line on the image

---

```
#PRINTING THE BLACK IMAGE WITH GREEN LINE ON IT
Img = np.zeros((512,512,3), np.unit8) # This is a grayscale image, black
Img[:] = 255,0,0 #blue image
cv2.line(Img,(0,0), (Img[1],Img[0]),(0,255,0)) # adds diagonal line from (0,0) point to highest coordinate of the image
cv2.imshow("Image",Img)
cv2.waiKey(0)
```

---

2) Rectangle

- We want to keep the rectangle on the image.

cv2.rectangle(image,(startingpoint), (endingpoint), (colour), thickness) #keeps rectangular box
cv2.FILLED # fills the box we drew above.

```
#ADDING RECTANGLE ON THE IMAGE AND FILLING IT
Img = np.zeros((512,512,3), np.unit8) # This is a grayscale image, black
Img[:] = 255,0,0 #blue image
cv2.line(Img,(0,0), (Img[1],Img[0]),(0,255,0))
cv2.rectangle(Img,(0,0), (250,350), (0,0,255), cv2.FILLED) #keeps rectangular box and fill it with same colour of outline of box
cv2.imshow("Image",Img)
cv2.waiKey(0)
```

---

3) Circle

cv2.circle(image, (centerpoint), radius, (colour), thickness)

```
#ADDING CIRCLE ON THE IMAGE
Img = np.zeros((512,512,3), np.unit8) # This is a grayscale image, black
Img[:] = 255,0,0 #blue image
cv2.line(Img,(0,0), (Img[1],Img[0]),(0,255,0))
cv2.rectangle(Img,(0,0), (250,350), (0,0,255), cv2.FILLED)
cv2.circle(Img,(400,50),30,(255,255,0),5)
cv2.imshow("Image",Img)
cv2.waiKey(0)
```

---

4) Put Text

cv2.putText(image, "text we want to display",(starting point), font, scale, (colour), thickness)

```
#ADDING TEXT ON THE IMAGE
Img = np.zeros((512,512,3), np.unit8) # This is a grayscale image, black
Img[:] = 255,0,0 #blue image
cv2.line(Img,(0,0), (Img[1],Img[0]),(0,255,0))
cv2.rectangle(Img,(0,0), (250,350), (0,0,255), cv2.FILLED)
cv2.circle(Img,(400,50),30,(255,255,0),5)
cv2.putText(Img, "OpenCV",(300,150), cv2.FONT_HERSHEY_COMPLEX, 1, (0,255,0), 3)
cv2.imshow("Image",Img)
cv2.waiKey(0)
```

Warp Perspective

- We use “warp perspective” in order to get a birds eye view.

```
Img = cv2.imread(“file path”)
Width, height = 250,350
points = np.float32([[111,219],[287,188],[154,482][352,440]]) #numpy array of pixels of the part of the image.
```

- We can get the pixel values by opening the image in paint, and by moving the cursor we get the pixel values.
- We need to define which corner we are referring to.

```
points2 = np.float32([[0,0][width,0][0,height][width,height]]) # refers to the points where they should locate.
matrix = cv2.getperspectiveTransform(points,points2)
ImgOut = cv2.warpperspective(image, matrix,(width, height))
```

```
#APPLYING WARP PERSPECTIVE ON THE IMAGE
Img = cv2.imread(“file path”)
Width, height = 250,350
points = np.float32([[111,219],[287,188],[154,482][352,440]]) #numpy array of pixels of the part of the image.
points2 = np.float32([[0,0][width,0][0,height][width,height]]) # refers to the points where they should locate.
matrix = cv2.getperspectiveTransform(points,points2)
ImgOut = cv2.warpperspective(Img, matrix,(width, height))
cv2.imshow(“Image”, Img)
cv2.imshow(“Output”, ImgOut)
cv2.waitKey(0)
```

Joining Images

```
#STACKING VERTICALLY AND HORIZONTALLY WITH NUMPY
import numpy as np
Img = cv2.imread(“file path”)
hor = np.hstack((Image,Image)) #horizontal stack function from numpy stacks/joins image side by side
ver = np.vstack((Img, Img)) #vertical stack function from  numpy stacks/joins the images vertically
cv2.imshow(“Horizontal”,hor)
cv2.imshow(“Vertical”,ver)
cv2.waitKey(0)
```

- There are few issues with the above method.
    1. We cannot resize the image.
    2. It’ll take lot of space
    3. If pixels of both the images are not RGB then this wont work.

Solution:

```
#SOLUTION FOR THE PROBLEM
 def stackImages(scale,imgArray):
    rows = len(imgArray)
    cols = len(imgArray[0])
```

```
    rowsAvailable = isinstance(imgArray[0], list)
    width = imgArray[0][0].shape[1]
    height = imgArray[0][0].shape[0]
    if rowsAvailable:
        for x in range ( 0, rows):
            for y in range(0, cols):
                if imgArray[x][y].shape[:2] == imgArray[0][0].shape [:2]:
                    imgArray[x][y] = cv2.resize(imgArray[x][y], (0, 0), None, scale, scale)
                else:
                    imgArray[x][y] = cv2.resize(imgArray[x][y], (imgArray[0][0].shape[1], imgArray[0][0].shape[0]),
None, scale, scale)
                if len(imgArray[x][y].shape) == 2: imgArray[x][y]= cv2.cvtColor( imgArray[x][y],
cv2.COLOR_GRAY2BGR)
        imageBlank = np.zeros((height, width, 3), np.uint8)
        hor = [imageBlank]*rows
        hor_con = [imageBlank]*rows
        for x in range(0, rows):
            hor[x] = np.hstack(imgArray[x])
        ver = np.vstack(hor)
    else:
        for x in range(0, rows):
            if imgArray[x].shape[:2] == imgArray[0].shape[:2]:
                imgArray[x] = cv2.resize(imgArray[x], (0, 0), None, scale, scale)
            else:
                imgArray[x] = cv2.resize(imgArray[x], (imgArray[0].shape[1], imgArray[0].shape[0]), None,scale,
scale)
            if len(imgArray[x].shape) == 2: imgArray[x] = cv2.cvtColor(imgArray[x], cv2.COLOR_GRAY2BGR)
        hor= np.hstack(imgArray)
        ver = hor
    return ver
```

---

imgStack = stackImages(scale of theimages,([matrices of images])) #stacks images horizontally
imgStack1 = stackImages(scale of theimages,([matrices of images], [matrices of images])) #stacks images vertically as well as horizontally
cv2.imshow("stack", imgStack)

- #You should have the same number of images in the top and bottom row.

#STACK IMAGES IN VERTICALLY AND HORIZONTALLY
import numpy as np
Img = cv2.imread("file path")
imgStack = stackImages(0.5,([Img,Img,Img])) #stacks images horizontally
imgStack1 = stackImages(0.5,([Img,Img,Img], [Img,Img,Img])) #stacks images vertically and horizontally
cv2.imshow("Horizontal",imgStack)
cv2.imshow("Vertical",imgStack1)
cv2.waitKey(0)

Color Detection

- Let's try to detect the orange colour in the image
- We first convert image into HSV using cvtColor
- We want some color values and some color ranges which we want our color to be, If the image color region falls within that range we will grab it.
- *We don't know the max and min values of orange colour, for that let's create track bars.

Img = cv2.imread("file path")
cv2.namedwindow("TrackBars") #creating new window named TrackBars
cv2.resizewindow ("TrackBars", 640, 240) #resize the trackbars
cv2.createTrackbar("what value we are going to change(just name)", "which window we are going to put this track bar", initial value of TrackBar, max value of TrackBar, function that runs every time we change trackbar.) #first track bar

- We need six track bars because Hue max and min, Saturation max and min, value max and min
- We read track bar values, so we can apply on image, using getTrackbarPos function

h_min = cv2.getTrackbaarPos('what value we are going to change(just name)', which trackbar window does it belong)

- We have 6 values, so we copy, paste the same thing 5 more times.
- Now we will apply a mask.

mask = cv2.inRange('which image are we talking about', lower limit, upper limit)

- Mask will give a filtered image.
- Finally, we need to get the image with original colours. So fir that we use "bitwise_and" function which adds the two images and create new image

cv2.bitwise_and(original image, image,mask=mask

```
#STACK IMAGES
def empty(n):
    pass
cv2.namedwindow("TrackBars")
cv2.resizewindow ("TrackBars", 640, 240)
cv2.createTrackbar("Hue  Min", "TrackBars",0,179, empty)
cv2.createTrackbar("Hue  Max", "TrackBars",179,179, empty)
cv2.createTrackbar("Sat  Min", "TrackBars",0,255, empty)
cv2.createTrackbar("Sat  Max", "TrackBars",255,255, empty)
cv2.createTrackbar("Val  Min", "TrackBars",0,255, empty)
cv2.createTrackbar("Val  Max", "TrackBars",255,255, empty)
while True:  #We need to change Trackbar values, so we keep them in a loop
    Img = cv2.imread("file path")
    imgHSV = cv2.cvtColor(Img,cv2.COLOR_BGR2HSV)
    h_min = cv2.getTrackbaarPos('Hue  Min', 'TrackBars')
    h_max = cv2.getTrackbaarPos('Hue  Max', 'TrackBars')
    s_min = cv2.getTrackbaarPos('Sat  Min', 'TrackBars')
    s_max = cv2.getTrackbaarPos('Sat  Max', 'TrackBars')
    v_min = cv2.getTrackbaarPos('Val  Min', 'TrackBars')
    v_max = cv2.getTrackbaarPos('Val  Max', 'TrackBars')
```

```
print(h_min , h_max , s_min, s_max, v_min, v_max )

lower = np.array([h_min, sat_min,v_min])
upper= np.array([h_max, sat_max,v_max])
mask = cv2.inRange(imgHSV, lower, upper)
imgResult = cv2.bitwose_and(img, img, mask=mask)

cv2.imshow('Original', img)
cv2.imshow("HSV' imgHSV)
cv2.imshow("Mask",mask)
cv2.imshow("result",imgResult)
imgstack = stackImages(0.6,([img,imgHSV],[mask, imgResult]
cv2.waitKey(0)
```

Contour Detection

cv2.findContours(image we are working with, retrieval method, approximation) #find the extreme out contours
cv2.contourArea(contour we want to find the area for) #finds the area of the contours
cv2.drawContours(image we want to draw, contour, index, (colour), thickness) # draws the contours
cv2.arcLength(contour, True) #curve length of contour, it is True because it is closed
cv2.approxPoly(contour, (resolution),True)#finds the corners in the image,We want images to be closed so True
cv2.boundingRect() # keep bounding boxes around the objects
cv2.rectangle(where we ant to draw it, (starting point), (ending point), colour, thickness) #size and shape of bounding box

---

```
#CONTOURS AROUND THE OBJECTS
contours = cv2.findContours(Img, cv2.RETR.EXTERNAL,cv2.CHAIN_APPROXIMATION = None) #find the extreme out contours
for cnt in contours:
    cv2.contourArea(cnt)
    cv2.drawContours(Img, cnt, -1, (255,0,0), 3) #-1 is because we need to get all the contours.
    peri = cv2.arcLength(contour, True)
    approx = cv2.approxPoly(cnt, (0.02*peri),True)
    x,y,w,h = cv2.boundingRect(approx)
    cv2.rectangle(Img, (x,y), (x+w,y+h), (0,255,0),2)
```
