Cv2.WINDOW\_AUTOSIZE- to create a window and load an image to it later and specify whether the window is resizable or not

Cv2.imread(imgpath,\_)🡪0- produces the grey scale image

-1-displays along with alpha transparency channel

1🡪default

Cv2.imwrite(outpath,cv2.imread)🡪saves an image to a particular path specified by its outpath

Cv2.waitkey(0)🡪waits for keyboard intervention

Each image is represented using a tuple

[123 34 212]🡪 BLUE GREEN RED channel intensities respectively with each TUPLE representing a PIXEL

Type(img1) is a numpy.ndarray data type 🡪composite data type 🡪array of n dimensions

Img1.dtype🡪uint8🡪unsigned integer of 8 bits

Values range from 0-255 in decimal number system

00000000-11111111 in binary number system

0🡪black

255🡪white

It results in 256\*256\*256 unique colours

Img1.shape🡪512\*512\*3 which signifies the resolution of the image where 3 represents the number of channels....for grey scale image it is 512\*512 where each number represents a pixel

Img1.ndim🡪number of dimensions in image....for grey scale image it comes out be 2 while for normal coloured images it is 3

Img1.size🡪512\*512\*3 size of image(no of images) in memory

**CREATION OF IMAGES USING NUMPY**

Np.zeros((512,512,3),np.uint8)

Cv2.line(img1🡪the image where the line is to be made,(0,99),(99,0)🡪coordinates of the end points of the line ,(255,0,0)🡪specifies the colour of the line,2🡪represents the thickness of the line)

..........for rectangles the coordinates represent the diagonally opposite points

The origin lies on the top left corner of the window

The most recent values gets overwritten over the previous values

For a circle one has to mention the centre of the circle and the radius in place of the end coordinates ..... and in place of thickness if -1 is written then it makes a solid circle of the colour specified

In case of an ellipse we have to additionally mention the the length of major and minor axis inplace of the radius and have to mention the angle of rotation(first one along x axis and the second one along y axis ) and also the portion of the ellipse to be displayed

Cv2.ellipse(img1,(160,260)🡪the centre of the ellipse,(50,20)🡪 the length of the major and the minor axis,20,50🡪the angle of rotation about the x and y axis respectively,360🡪the portion of the ellipse to be be displayed,(120,0,0)🡪the colour of the ellipse,3🡪the thickness of the ellipse or to make it solid

Cv2.polylines(img1,[points]🡪the array of points which needs to be connected to form a polygon ,TRUE,(0,255,255)🡪colour of the polygon

Cv2.putText(img1,text1🡪the text which needs to be displayed ,(100,100)🡪the coordinates at which the text needs to be displayed,cv2.FONT\_HERSHEY\_SIMPLEX🡪the font of the text,2🡪font size and then specify the colour in the usual manner

**BGR TRACKBAR(using trackbar GUI in opencv)**

while(True):

cv2.imshow(windowname,img1)

if cv2.waitKey(1)==27:#represent the escape button

break

This logic is generally used when we have to frequently refresh the images on the screen

cv2.createTrackbar('B'🡪The name of the trackbar,windowname🡪the name of the window its is associated with,0,255🡪minimum and the maximum value of the trackbar respectively,emptyfunction🡪the function it should call while moving the slider on the window associated with that colour)

blue=cv2.getTrackbarPos('B',windowname)#fetches the current position of the trackbar

**img1[:]=[blue,green,red]creates a composite image by merging all the colour channels**

**MOUSE EVENTS AND INTERACTIVE DRAWING**

events=[i for i in dir(cv2) if 'EVENT'in i ]# list of interactive mouse events

def draw\_circle(event, x, y, flags, param):#**custom defined function**

if event == cv2.EVENT\_LBUTTONDBLCLK:#on the event of left

button double click only

cv2.circle(img, (x, y), 40, (0, 255, 0), -1)

mode = True# tells whether the m key is pressed or not

if k == ord('m') or k == ord('M'):

mode = not mode #toggles the mode

plt.imshow()🡪cannot understand the data and doesn’t realise whether the image is grey scale or not

COLORMAPS🡪assigns colour values by itself in case of lack of

Information.........applies its default colormaps in

Case cmap is not mentioned

plt.imshow(img,cmap=”gray”)🡪produces the grey scale image

plt.title(‘’)🡪gives a title to the image displayed

plt.xticks([]) and plt.yticks([])🡪makes the scales disappear

cv2.imread🡪BGR method for storing the images

Matplotlib plt.imshow🡪RGB method for displaying the image

And thus misinterpretation of the color channels

**VARIOUS WAYS OF REPRESENTING COLOR CHANNELS ARE CALLED COLOR SPACES**

**GRAY**

BGR

RGB

HSV-HUE SATURATION VALUE

Img=cv2.cvtColor(img,cv2.COLOR\_BGR2RGB)#way of changing the color spacing of the image img from bgr to rgb and thus removing the misinterpretation

**WEB CAM IMAGE CAPTURE**

cap=cv2.VideoCapture(0)# 0 represents the very first web cam attached to the system

ret,frame=cap.read()🡪true or false value stored in ret depending on whether the web cam is on or off ......while the frame returns the current frame value

cap.release()🡪 releases the currently working webcam

cap.get(3)🡪 presents the width(resolution) of image captured by the webcam cap

cap.get(4)🡪 presents the height(resolution) of the image captured by the webcam cap

cap.set(3,1024)🡪 changes the resolution of the width to 1024

cap.set(4,768)🡪changes the resolution of the height to 768

if the set values adjusts itself to the closest p0ssible resolution

**CODEC** stands for coding and decoding......encoding of the video

FourCC=four character code

Codec=cv2.VideoWriter\_fourcc(‘X’,’V’,’I’,’D’)# creates the object for codec.....almost all codecs can be expressed with 4 variables

Frame rate is required for persistence of vision and at lower resolution the frame rate is better

framerate=30

resolution=(640,480)

VideoFileOutput=cv2.VideoWriter(filename,codec,framerate,resolution)🡪accepts the arguments mentioned and creates video file output object

VideoFileOutput.release()🡪closes the file on closing the window

**OPENCV VIDEO PLAYER**

cap=cv2.VideoCapture(filename) #captures the frames from a video file

as long as the value of the ret is true(i.e when the code is executed) it plays the video stored at the location

the frames per second (fps) with which the video was captured...at the same speed the video is played back every time the code is executed

cv2.waitkey(1🡪indicates in the no. Of milliseconds 1 frame has to be played)

to play 30 frames in 1 second i.e 1000 milliseconds

1000/30=33

**OBJECT TRACKING BY COLOUR**

BGR/RGB🡪not an appropriate colourspace for object tracking using colour due to only 3 available colours and none of their shades

**HSV-HUE SATURATION VALUE**

depending on the colour to be tracked user must define the lower and the higher

**#FOR BLUE COLOUR #** covers all the shades of blue

**Low=np.array([100,50,50])**

**high=np.array([140,255,255])**

**#FOR GREEN COLOUR #** covers all the shades of green

**Low=np.array([40,50,50])**

**high=np.array([80,255,255])**

image\_mask🡪returns a binary matrix based on the interval of the low to the high value passed

**image\_mask=cv2.inRange(hsv🡪color space used,low,high)....................**it masks the images its range of values matches with

output=cv2.bitwise\_and(frame,frame,mask=image\_mask)...........................does a bitwise and operation on the original binary video file and the image\_mask and therefore displays only the range colours passed between the high and the low

**IMAGE PROCESSIING USING MATPLOTLIB**

plt.subplot()🡪display multiple images can be displayed simultaneously in one window

plt.subplot(1🡪no of rows,2🡪no. Of columns,1🡪the position of the image) #a grid is defined of the no. Of rows and columns passed with the position variable starting from one

**BLENDING AND TRANSITIONING IMAGES**

addWeighted()🡪composite arithmetic operation using addition and multiplication to achieve the blending process

output=cv2.addWeighted(img1,alpha,img2,beta,gamma)🡪

output=img1\*alpha+img2\*beta+gamma

alpha +beta=1

for i in np.linspace(0,1,10):🡪denotes the linear space between 0 and 1 is divided into 10 equal parts

**COLOUR CHANNELS**

CMY🡪CYAN MAGENTA YELLOW

CMYK🡪CYAN MAGENTA YELLOW KEY

Cv2.split()🡪returns the multiple channels....in other words separates the colour channels

Cv2.merge()🡪merges the individual colour channels

**img3=abs(255-img1)**# method of computing the image in negative.....each of the pixel is complemented

**BITWISE OPERATIONS ON IMAGES 🡪AND,OR,XOR,NOT(**negative of the image)....

Cv2.BITWISE\_AND.......

**SCALING and TRANSLATION (SHIFTING)OPERATIONS ON IMAGES**

Scaling/shrinking🡪(scaling down or shrinking)

Zooming🡪(scaling up)

Cv2.resize(img1,none,fx=1🡪scalling along x axis,fy=1🡪scaling along y axis,interpolation =cv2.LINEAR)

**INTERPOLATION🡪**determining how a pixel would look in relation to its surrounding by finding out the position of a point taking into consideration the position of the previous points

INTER\_AREA🡪shrinking

INTER\_CUBIC/INTER\_LANCZ0S4🡪zooming(4\*4/8\*8)

**Cv2.warpAffine()**🡪for translation and transformation of the images

T=np.float32([[1,0,50]🡪shifts image to the right by 50 pixels,[0,1,-50]🡪shifts image to downwards by 50 pixels])🡪transformation matrix for shifting of the image

output=cv2.warpAffine(img1,T🡪transformation matrix,(columns,rows)🡪columns and rows extracted after unpacking of a tuple (the image matrix))

**THRESHOLDING AND SEGMENTATION**

Dividing the image into several marked regions🡪segmentation

Thresholding is a basic form of segmentation used generally on gray scale images

[[10 24]

[34 67]]

Threshold=50

If threshold>50

Pixel=255 resulting output of the above matrix Else [[0 0]

Pixel=0 [0 255]]#binary algorithm

Cv2.threshold(img,th🡪the value of the threshold,max🡪maximum value in the input range,algo🡪algorithm which is to be implemented) returns ret 🡪threshold applied and output🡪output matrix

**AFFINE TRANSFORMATIONS🡪**maintains the parallelism between the lines if any in the image

Affine transformations require atleast 3 points and one has to make sure the points are not on the same line......as it maps one plane to another plane

**cv2.getAffineTransform**(point1🡪set of points in the input image,point2🡪set of points in the output image)

**PERSPECTIVE TRANSFORMATION**🡪parallelism between the line is preserved but the minimum number of points to be used is 4

**OTSUS BINARIZATION THRESHOLLDING**

Used when the threshold value cannot be decided upon by the user

Foreground and background images together are known as bimodal images......the image histogram can be separated into

Foreground and background and otsus binarisation is useful in bimodal images especially with the trunk algorithm

But not suitable in images having varying lighting......inwhich adaptive thresholding is to applied

**ADAPTIVE THRESHOLDING**

Block size is the neighbourhood of pixels on which the which the adaptive threshold algorithm works on

th1=cv2.adaptiveThreshold(img1,255,cv2.ADAPTIVE\_THRESH\_MEAN\_C,cv2.THRESH\_BINARY,block\_size,constant)

higher the value of the block size sharper are the edges and more intense are the colours of the foreground

**NOISE🡪unwanted signal**

**Gaussian noise**

**Salt and pepper noise**

# for i in range(rows):

for j in range(columns):

r=random.random()

if r<p/2:

output[i][j]=[0,0,0]#pepper sprinkled

elif r<p:

output[i][j]=[255,255,255]#salt sprinkled

else:

output[i][j]=img1[i][j]#salt and pepper sprinkled noise with 5 percent probability

**SNR**🡪signal power /noise power

**KERNEL AND CONVOLUTION**

Kernel🡪convolution matrix or mask

Kernel matrix is convoluted with the image matrix

IDENTITY KERNEL🡪when convoluted with the image matrix the same image is obtained...........[0 0 0

0 1 0

0 0 0]

[1,0,-1],[0,0,0],[-1,0,1]#EDGE DETECTION

[0,1,0],[1,-4,1],[-0,1,0]#edge detection kernel

[0,-1,0],[-1,5,-1],[0,-1,0]#SHARPENING KERNEL

[1,1,1],[1,1,1],[1,1,1]#BOX BLUR KERNEL

Output=Cv2.filter2D(img,-1🡪depth of the output image (in this case it is same as of the input image),k🡪kernel matrix)

**LOW PASS FILTERS**

🡪allows only low frequency components to pass through

High frequency components are noise and edges

Low pass filters are able to perform

**BLURRING#** box=cv2.boxFilter(img1,-1,(53,53)🡪size of the

kernel matrix which is being internally

created)🡪box blur

blur = cv2.blur(img, (13, 13))

gaussian = cv2.GaussianBlur(img, (37, 37), 0)

**DEIONISING**

**SMOOTHING**

**MEDIAN PASS FILTERS**

Salt and pepper noise can be handled using medium pass filter

denoised =cv2.medianBlur(output,3)

**HIGH PASS FILTER**

Allows the high frequency components to pass through

Used in

**EDGE DETECTION**

High pass filters edge detection algorithms are

**Laplacian** # edges=cv2.Laplacian(img1,-1🡪depth of theoutput

,ksize=17🡪kernel size(has to be an odd number),scale=1,delta=0,borderType=cv2.BORDER\_DEFAULT)

**Sobel # x component** 🡪dx

Y component🡪dy

If dx=1 and dy=0🡪 it provides the vertical edges and the reverse gives the horizontal edges

edgesx=cv2.Sobel(img1,-1,dx=1,dy=0,ksize=11,scale=1,delta=0,borderType=cv2.BORDER\_DEFAULT)🡪**VERTICAL EDGES**

edgesy=cv2.Sobel(img1,-1,dx=0,dy=1,ksize=11,scale=1,delta=0,borderType=cv2.BORDER\_DEFAULT) 🡪**HORIZONATL EDGES**

**Scharr**

edgesx=cv2.Scharr(img1,-1,dx=1,dy=0,scale=1,delta=0,borderType=cv2.BORDER\_DEFAULT)

edgesy=cv2.Scharr(img1,-1,dx=0,dy=1,scale=1,delta=0,borderType=cv2.BORDER\_DEFAULT)

**CANNY EDGE DETECTION ALGORITHM**

**🡪**gaussian kernel 5\*5 applied to image for filtering noise

🡪intensity gradient calculated using l1 and l2 norm

🡪non maximum suppression is applied to output of 2

🡪using gradient threshold final edge set is calculated

->pixel less than gradient one is excluded (set a)and pixels more then gradient 2(set b) is included .pixels in between the 2 gradients are only included if only it is directly connected to set b

L1=cv2.Canny(img1,100🡪gradient 1,150🡪gradient 2,L2gradient=False🡪call edge detection with l1 norm)

l2=cv2.Canny(img1,100,150,L2gradient=True🡪calls edge detection with l2 norm)

**HOUGH LINE TRANSFORM**

lines=cv2.HoughLines(edges🡪image from which algorithm will compute the lines,1🡪rho value(distance accuracy of an accumulator , np.pi/180🡪(theta value)angle accuracy of an accumulator,200🡪threshold of the accumulator)

the lines calculated will be in polar coordinates and one has to convert them to Cartesian coordinates

if lines is not None:

for rho,theta in lines[0]:

a=np.cos(theta)

b=np.sin(theta)

x0=a\*rho

y0=b\*rho

pts1=(int(x0+1000\*(-b)),int(y0+1000\*(a)))

pts2=(int(x0-1000\*(-b)),int(y0-1000\*(a)))

cv2.line(frame,pts1,pts2,(0,255,0),3)

coversion of polar to Cartesian coordinates

**INPAINTING**

**🡪**restoration of images technique

Ouput1=cv2.inpaint(img1,mask🡪a mask of the original image is to be created which must contain only the dame=aged portion,5🡪size of the neighbourhood,cv2.INPAINT\_TELEA/INPAINT\_NS🡪image restoration algorithm)

**IMAGE QUANTISATION**

**🡪**grey scale (0,255)

🡪coloured images(0,255)\*3(no. Of channels).......each pixel occupying 3 bits with one bit for each channel

The matrix representing a coloured image has 16 million colours .......in order to reduce it to fewer no of colours representing the image the image quantisation process is to be used

**K MEANS CLUSTERING🡪**method of image quantisation

[4,5,6,8,9,3,2,1,6,4,6,7] represent it within 4 numbers

[1,2,3,4,4,5,6,6,6,7,8,9]🡪ascending order

[2,4,6,8]🡪calculating average of the numbers in groups of 3 from the ascending order sets

Uint8 has to be converted float 32

**CRITERIA**

criteria=(cv2.TERM\_CRITERIA\_EPS+cv2.TERM\_CRITERIA\_MAX\_ITER,10,1.0)

ret,label1,center1=cv2.kmeans(z🡪image,k🡪no of colours it has to be quantised to,criteria,10🡪no of iterations,cv2.KMEANS\_RANDOM\_CENTERS)

**HISTOGRAM**

plt.hist(img1.ravel()🡪flattens the 2 or 3 dimensional matrix into 1 dimensional matrix,256🡪no of beats(for each intensity the no of pixels having that intensity level is calculated,[0,255]🡪range)🡪**MATPLOTLIB HISTOGRAM**

hist,bins=np.histogram(img1.ravel(),256,[0,255])🡪**NUMPY HISTOGRAM**

red\_hist=cv2.calcHist([img1],[0]🡪channel number,None🡪no masking,[256],[0,255]🡪range)

**HISTOGRAM EQUALISATION**

**🡪**image enhancement technique ......adjust the intensities of various colour channels of various pixels in the image

2 ways are

🡪Cv2.equalizehist(img1)

🡪contrast limited adaptive histogram#clahe=cv2.createCLAHE(cliiplimit=2.0,tileGridSize(8,8))🡪adjusts the histogram in a neighbourhood of 8\*8.....not valid on coloured images .....to implement on the coloured images one has to separate the colour channels and then implement the above algorithms and then merge the colour channels

**MORPHOLOGICAL TRANSFORMATIONS**

EROSION🡪borders of the foreground are being eroded by the background

erosion=cv2.erode(binary\_inv,k,iterations=3)

DILATION🡪the foreground encroaches on the background

dilation=cv2.dilate(binary\_inv,k,iterations=3)

GRADIENT🡪difference between the erosion and dilation

gradient=cv2.morphologyEx(binary\_inv,cv2.MORPH\_GRADIENT,k)

k=cv2.getStructuringElement(cv2.MORPH\_RECT,(5,5))#creates a kernel similar to np.ones() used before

k=cv2.getStructuringElement(cv2.MORPH\_ELLIPSE,(5,5))#kernel matrix is of an elliptical shape

k=cv2.getStructuringElement(cv2.MORPH\_CROSS,(5,5))#kernel matrix is of a cross shape

**CONTOURS**

**🡪j**oining all continuous points having the same colour value

Cv2.findcontours(image ,contour retrieval mode,contour approximation mode)

🡪**RETREIVAL MODES**

RETR\_EXTERNAL

RETR\_LIST

RETR\_CCOMP

RETR\_TREE

🡪**APPROXIMATION MODES**

CHAIN\_APPROX\_NONE

CHAIN\_APPROX\_SIMPLE

CHAIN\_APPROX\_TC89\_L1

CHAIN\_APPROX\_TC89\_KCOS

It is is easier to find out the contours in a binary image

img2,contours🡪ist of contours,hierarchy=cv2.findContours(thresh,cv2.RETR\_TREE,cv2.CHAIN\_APPROX\_SIMPLE)

Cv2.drawContours