Experiment 3

Image Processing Basics in OpenCV

Aim:

1) To implement the following basic functions on an image or a video in Open CV:

Convert to Gray Scale, implement image intensity transformations, Blur, Draw shapes and adding Text, Mask or Crop, Histogram, Thresholding, Image Addition and Image Subtraction.

Software/ Packages Used:

1. Pycharm IDE
2. Libraries used:
    - NumPy
    - opencv-python
    - matplotlib
    - scipy

Programs:

1. To implement the basic functions of image processing in openCV

Image intensity transformations

```
# Read in an image
cv.imread("C:/Users/21r228\Downloads\download.jpg")

#Find the Width, Height and Color Channels of the image

# Converting to a shade

# Converting to grayscale
```

1. Image Negative

```
import cv2 as cv
import numpy as np
img=cv.imread("C:/Users/21r228\Downloads\download.jpg")
print("img.itype")
print("img")
img_res=255-img
cv.imshow("1",img)
cv.imshow("2",img_res)
```

IMAGE NEGATIVE:

INPUT:

OUTPUT:

PIXEL MATRIX:

IMAGE:

```
[[41 42  8]
 [41 42  8]
 [42 42  6]
 ...
 [35 34  6]
 [29 28  0]
 [29 28  0]]
```

IMAGE_REVERSE:

```
[[214 213 247]
 [214 213 247]
 [213 213 249]
 ...
 [220 221 249]
 [226 227 255]
 [226 227 255]]
```

```
    cv.waitKey(0)
    cv.destroyAllWindows()
```

2. Logarithmic Transformation

```
import cv2 as cv
import numpy as np

# Open the image.
img = cv.imread("C:/Users/21r228\Downloads\download.jpg")

# Apply log transform.
c = 255/(np.log(1 + np.max(img)))
log_transformed = c * np.log(1 + img)

# Specify the data type.
log_transformed = np.array(log_transformed, dtype = np.uint8)

# Save the output.
cv.imwrite('log_transformed.jpg', log_transformed)
cv.waitKey(0)
cv.destroyAllWindows()
```

3. Power Law (Gamma) Transformation

```
import cv2 as cv
import numpy as np

# Open the image.
img = cv.imread('C:/Users/21r228\Downloads\download.jpg')

# Trying 4 gamma values.
for gamma in [0.1, 0.5, 1.2, 2.2]:

    # Apply gamma correction.
    gamma_corrected = np.array(255 * (img / 255) ** gamma, dtype='uint8')

    # Save edited images.
    cv.imwrite('gamma_transformed' + str(gamma) + '.jpg', gamma_corrected)
cv.waitKey(0)
cv.destroyAllWindows()
```

LOGARITHMIC TRANSFORMATION

INPUT:

OUTPUT:

Power Law (Gamma) Transformation

INPUT:

OUTPUT:

0.1

0.5

1.2

2.2

4. Contrast stretching

```
import cv2
import numpy as np

# Function to map each intensity level to output intensity level.
def pixelVal(pix, r1, s1, r2, s2):
   if (0 <= pix and pix <= r1):
      return (s1 / r1)*pix
   elif (r1 < pix and pix <= r2):
      return ((s2 - s1)/(r2 - r1)) * (pix - r1) + s1
   else:
      return ((255 - s2)/(255 - r2)) * (pix - r2) + s2

# Open the image.
img = cv2.imread('C:/Users/21r228\Downloads\download.jpg')

# Define parameters.
r1 = 70
s1 = 0
r2 = 140
s2 = 255

# Vectorize the function to apply it to each value in the Numpy array.
pixelVal_vec = np.vectorize(pixelVal)

# Apply contrast stretching.
contrast_stretched = pixelVal_vec(img, r1, s1, r2, s2)

# Save edited image.
cv2.imwrite('contrast_stretch.jpg', contrast_stretched)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

CONTRAST STRETCHING

INPUT:

OUTPUT:

5. Bit plane slicing

```
import numpy as np
import cv2

# Read the image in greyscale
img = cv2.imread('C:/Users/21r228\Downloads\download.jpg', 0)

# Iterate over each pixel and change pixel value to binary using np.binary_repr() and store it in a list.
lst = []
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        lst.append(np.binary_repr(img[i][j], width=8))  # width = no. of bits

# We have a list of strings where each string represents binary pixel value. To extract bit planes we need to iterate over the strings and store the characters corresponding to bit planes into lists.
# Multiply with 2^(n-1) and reshape to reconstruct the bit image.
eight_bit_img = (np.array([int(i[0]) for i in lst], dtype=np.uint8) * 128).reshape(img.shape[0], img.shape[1])
seven_bit_img = (np.array([int(i[1]) for i in lst], dtype=np.uint8) * 64).reshape(img.shape[0], img.shape[1])
six_bit_img = (np.array([int(i[2]) for i in lst], dtype=np.uint8) * 32).reshape(img.shape[0], img.shape[1])
five_bit_img = (np.array([int(i[3]) for i in lst], dtype=np.uint8) * 16).reshape(img.shape[0], img.shape[1])
four_bit_img = (np.array([int(i[4]) for i in lst], dtype=np.uint8) * 8).reshape(img.shape[0], img.shape[1])
three_bit_img = (np.array([int(i[5]) for i in lst], dtype=np.uint8) * 4).reshape(img.shape[0], img.shape[1])
two_bit_img = (np.array([int(i[6]) for i in lst], dtype=np.uint8) * 2).reshape(img.shape[0], img.shape[1])
one_bit_img = (np.array([int(i[7]) for i in lst], dtype=np.uint8) * 1).reshape(img.shape[0], img.shape[1])

# Concatenate these images for ease of display using cv2.hconcat()
finalr = cv2.hconcat([eight_bit_img, seven_bit_img, six_bit_img, five_bit_img])
finalv = cv2.hconcat([four_bit_img, three_bit_img, two_bit_img, one_bit_img])

# Vertically concatenate
final = cv2.vconcat([finalr, finalv])

# Display the images
cv2.imshow('a', final)
cv2.waitKey(0)
```

Bit plane slicing

INPUT:

OUTPUT:

2. Paint and Draw on an image

```
# 1. Paint the image with certain color
# 2. Draw a Rectangle
# 3. Draw a circle
# 4. Draw a line
# 5. Write text
```

```
# Python3 program to draw solid-colored
# image using numpy.zeroes() function
import numpy as np
import cv2

# Creating a black image with 3 channels
# RGB and unsigned int datatype
img = cv2.imread("C:/Users/21r228\Downloads\long-exposure-photo-1024x576.jpg")
# Creating line
cv2.line(img, (120, 160), (500, 160), (0, 0, 255), 10)

# Creating rectangle
cv2.rectangle(img, (30, 30), (500, 400), (0, 255, 0), 5)

# Creating circle
cv2.circle(img, (200, 200), 80, (255, 0, 0), 3)

# writing text
font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(img, 'T.G.SON', (100, 100),
        font, 0.8, (0, 255, 0), 2, cv2.LINE_AA)

cv2.imshow('dark', img)

# Allows us to see image
# until closed forcefully
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Paint and Draw on an image

INPUT

OUTPUT

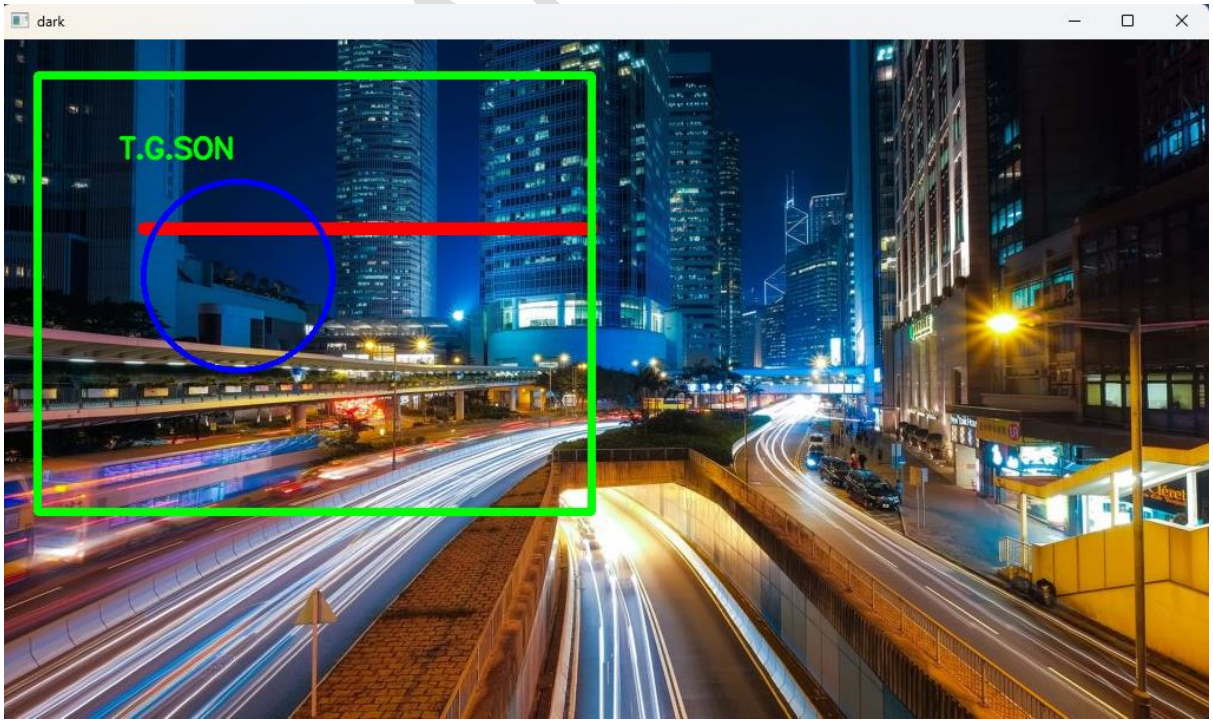


IMAGE MASKING:

```
# import required libraries
import cv2
import numpy as np

# read input image
img = cv2.imread('C:/Users/21r228\Downloads\download.jpg')

# Convert BGR to HSV
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

# define range of blue color in HSV
lower_yellow = np.array([15,50,180])
upper_yellow = np.array([40,255,255])

# Create a mask. Threshold the HSV image to get only yellow colors
mask = cv2.inRange(hsv, lower_yellow, upper_yellow)

# Bitwise-AND mask and original image
result = cv2.bitwise_and(img,img, mask= mask)

# display the mask and masked image
cv2.imshow('Mask',mask)
cv2.waitKey(0)
cv2.imshow('Masked Image',result)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

MASKING IMAGE:

INPUT:

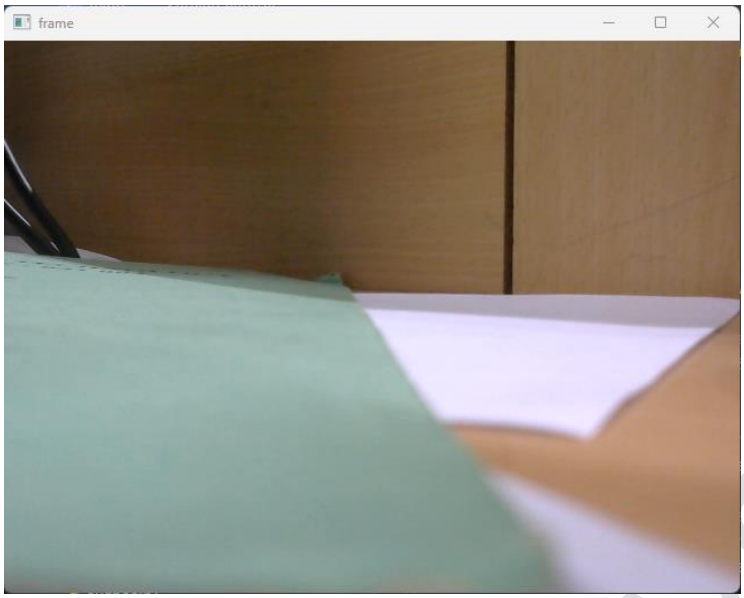
OUTPUT:

MASKING THE IMAGE VIDEO

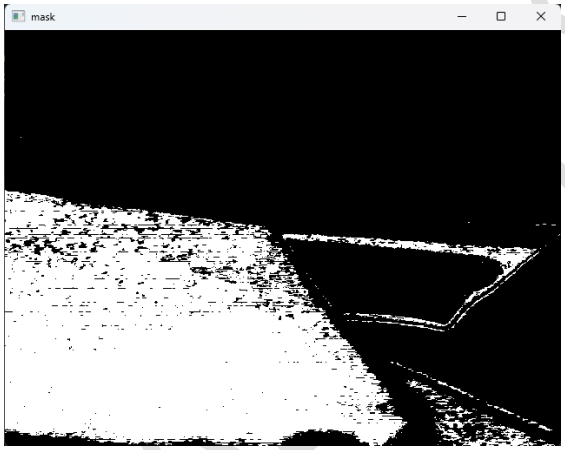
```
import cv2
import numpy as np

cap = cv2.VideoCapture(0)

while (1):
    _, frame = cap.read()
    # It converts the BGR color space of image to HSV color space
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # Threshold of blue in HSV space
    lower_blue = np.array([60, 35, 140])
    upper_blue = np.array([180, 255, 255])

    # preparing the mask to overlay
    mask = cv2.inRange(hsv, lower_blue, upper_blue)

    # The black region in the mask has the value of 0,
    # so when multiplied with original image removes all non-blue regions
    result = cv2.bitwise_and(frame, frame, mask=mask)

    cv2.imshow('frame', frame)
    cv2.imshow('mask', mask)
    cv2.imshow('result', result)

    cv2.waitKey(0)

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

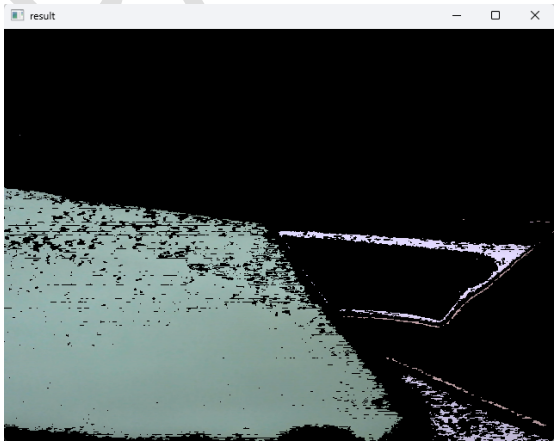
MASKING THE IMAGE VIDEO

INPUT:



OUTPUT:





Department of RAE			
Criteria	**Excellent (75% - 100%)**	**Good (50 – 75%)**	**Poor (<50%)**
Preparation (30)			
Performance (30)			
Evaluation (20)			
Report (20)			
Sign:		**Total (100)**	

Result:

Thus the basic image processing concepts were learnt using OpenCV in Pycharm IDE.