List of Experiments

Exp. No.	Date	List of Experiments	Page No.	Marks (10)	Signature with Date
1		Basics of Python and Numpy			
2		Basic Concepts OpenCV Image and Video Handling			
3		Image Processing Basics in OpenCV			
4		Geometric Transformation and Filtering			
5		Edge Detection and Template Matching			
6		Stereo Matching and Point Cloud Reconstruction			
7		Camera Callibration and Pose Estimation Using Monocular Camera			
8		Object Detection Using CNN			
9		Parallel Programming Using CUDA C			
10		Basics of RoS			
11		Simulation of Path Planning Algorithms using ROS			
		Individual Report 1			
		Individual Report 2			

YUVAN R S
21R256

Experiment – 1

Basics of Python and Numpy

Aim:

1) Learn the procedure to Install Python, creating a Project in Pycharm and add packages

2) To code six examples for Python Programming

3) To code five Examples for using Numpy and Matplotlib Packages

Software/ Package Used:

1. Pycharm IDE with Python 3.7
2. Libraries used:
   a) NumPy
   b) Matplotlib

Procedure:

Install Python:

1. Download PyCharm community version from the official website.
2. Run the installer.
3. Provide the required location of installation.
4. Once done it is ready to use.

Create a Project in Pycharm:

5. Select new project from files.
6. Name the project and press enter.
7. Give some time to set up the environment and then it is ready to use.

Add Packages:

1. Search for the required package.
2. Press the install button to install the package.
3. Once installed then import it in the program to use it.

Programs:

1)

```
# define add function
def add(a,b):
        print(a+b)

if __name__ == '__main__':
        a = int(input("Enter first number: ")) # get first number
        b = int(input("Enter second number: ")) # get second number
        add(a,b) # printing the sum
```

Input

Enter first number: 4

Enter second number: 6

Output

10

---

2)

```
# condition statements
# define greater function
def greater(a,b,c):
        max=a
        for i in range(0,3):
        if max<b :
        max=b
        elif max<c :
        max=c
        print('The greatest number is ',max)

if __name__ == '__main__':
        a = int(input("Enter first number: ")) # get first number
        b = int(input("Enter second number: ")) # get second number
        c = int(input("Enter third number: ")) # get third number
        greater(a,b,c) # print the greatest of three
```

Input

Enter first number: 5

Enter second number: 2

Enter third number: 6

Output

The greatest number is 6

---

3)

```
# condition statements
 # define the function
 def patern(a,b):
        i = 0
        while i < a:
        i += 1
        if i%b==0:
          continue
        print(i)

 if __name__ == '__main__':
        a = int(input("Enter max number: ")) # get first number
        b = int(input("Enter the number whose multiple must be removed: ")) # get
second number
        patern(a,b) # print the patern
```

Input

Enter max number: 10

Enter the number whose multiple must be removed: 3

Output

1

2

4

5

7

8

10

4)

function definition

def my_function(country = "Norway"): # considers Norway as argument with no arqument is provide in function call

```
  print("I am from " + country)


my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

Input

Output

I am from Sweden

I am from India

I am from Norway

I am from Brazil

---

5)

```
# function definition
def my_function(fname, lname):
  print(fname + " " + lname)


my_function('Yuvan','Parker')
```

Input

Output

Yuvan Parkers

---

6)

```
# import numpy

import numpy as np


# input the arrays

a = np.array(42)

b = np.array([1, 2, 3, 4, 5])

c = np.array([[1, 2, 3], [4, 5, 6]])

d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])


# print the arrays with dimension

print(a,'\n','Dimension= ',a.ndim)

print(b,'\n','Dimension= ',b.ndim)

print(c,'\n','Dimension= ',c.ndim)

print(d,'\n','Dimension= ',d.ndim)
```

Input

Output

```
42

 Dimension=  0

[1 2 3 4 5]

 Dimension=  1

[[1 2 3]
```

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

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

---

7)

```
# import numpy
import numpy as np
# get the element index
x=int(input('Enter 1st index: '))
y=int(input('Enter 2nd index: '))
a = np.array([1, 2, 3, 4])
print(a[x] + a[y]) # print the sum of the elements
```

Input

Output

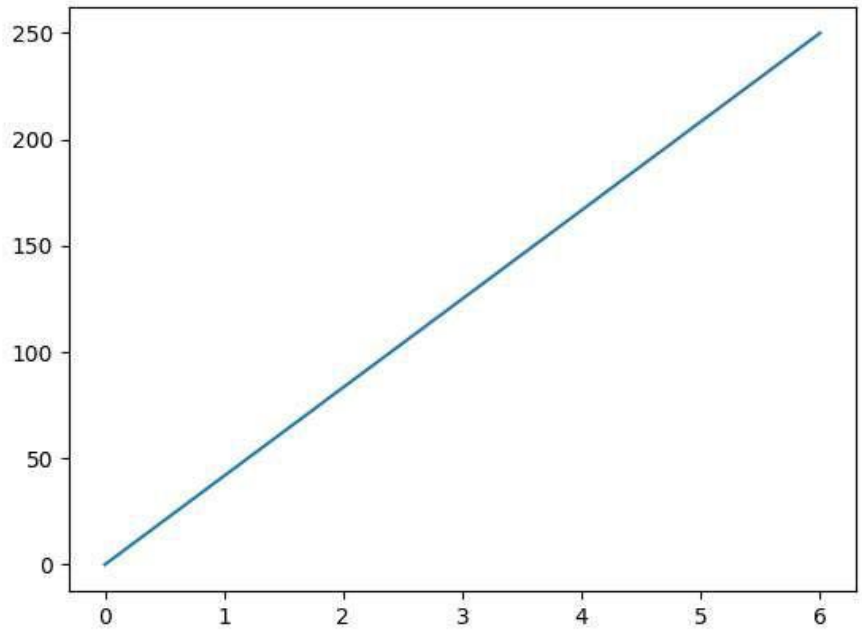
```
Enter 1st index: 2
Enter 2nd index: 3
7
```

8)

```
import matplotlib.pyplot as plt

import numpy as np


xpoints = np.array([0, 6])

ypoints = np.array([0, 250])


plt.plot(xpoints, ypoints)

plt.show()
```

Input

Output

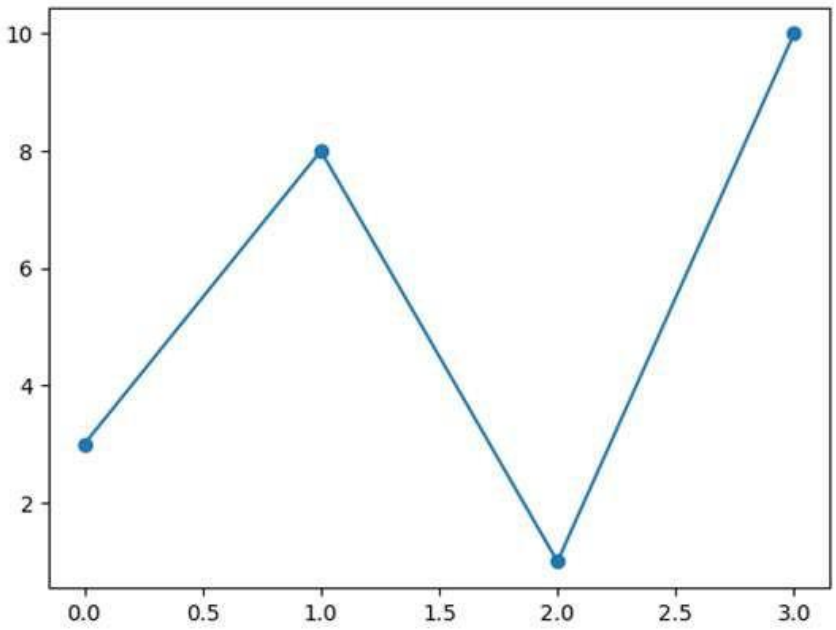


9)

```
import matplotlib.pyplot as plt

import numpy as np


ypoints = np.array([3, 8, 1, 10])


plt.plot(ypoints, marker = 'o')

plt.show()
```

Input

Output



10]

```
for x in "banana":
  print(x)
```

Input:

Output:

```
b
a
n
a
n
a
```

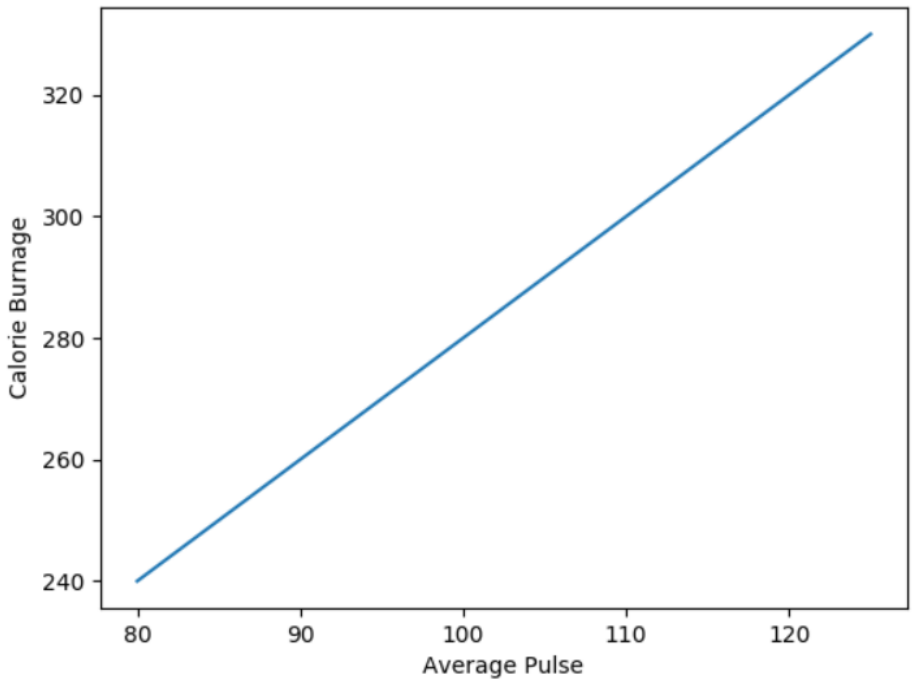
11]

```
#Three lines to make our compiler able to draw:
import sys
import matplotlib
matplotlib.use('Agg')

import numpy as np
import matplotlib.pyplot as plt

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.plot(x, y)

plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.show()

#Two  lines to make our compiler able to draw:
plt.savefig(sys.stdout.buffer)
sys.stdout.flush()
```

Input:

Output:

YUVAN R S
21R256



YUVAN R S
21R256

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 basics of python and Numpy had been done.

YUVAN R S
21R256

Experiment – 2

Basic Concepts in OpenCV Image and Video Handling

Aim:

1. To code the following tasks in OpenCV
   a) To read and display an image by using OpenCV
   b) To resize an image using OpenCV
2. To capture video from Camera, play a video from a file and save a video file.

Software/ Package Used:

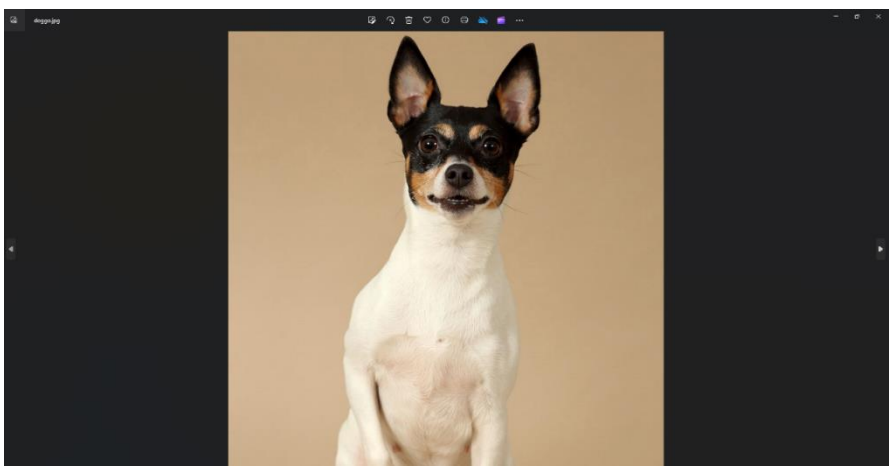
1. Pycharm IDE
2. Libraries used:
   a) NumPy
   b) opencv-python
   c) matplotlib
   d) scipy

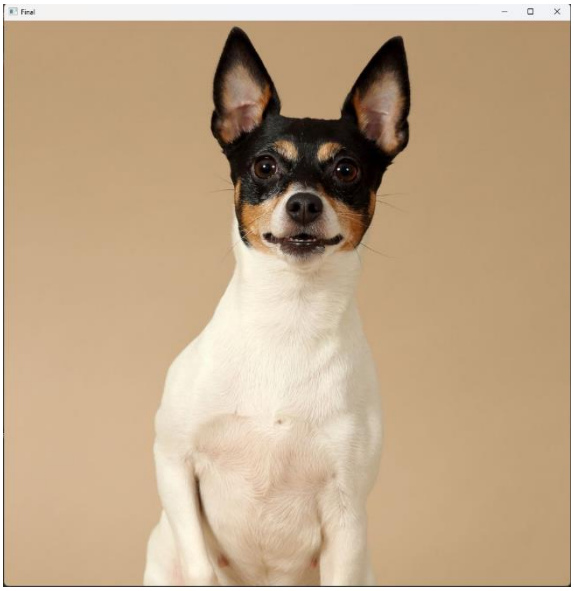
Programs:

1. **Read and Display image using OpenCV:**

```
import cv2 as cv    # import cv2
import sys      #import sys
img = cv.imread("X:\AI_V_lab\exp2\doggo.jpg")    # load image
if img is None:
    sys.exit("Couldn't read the image")     # prints this when image is not found
else:
    cv.imshow("Final", img)     # show image
cv.waitKey(0)   # holds the image in window
cv.destroyAllWindows()      # removes all the window
```

Color input image:

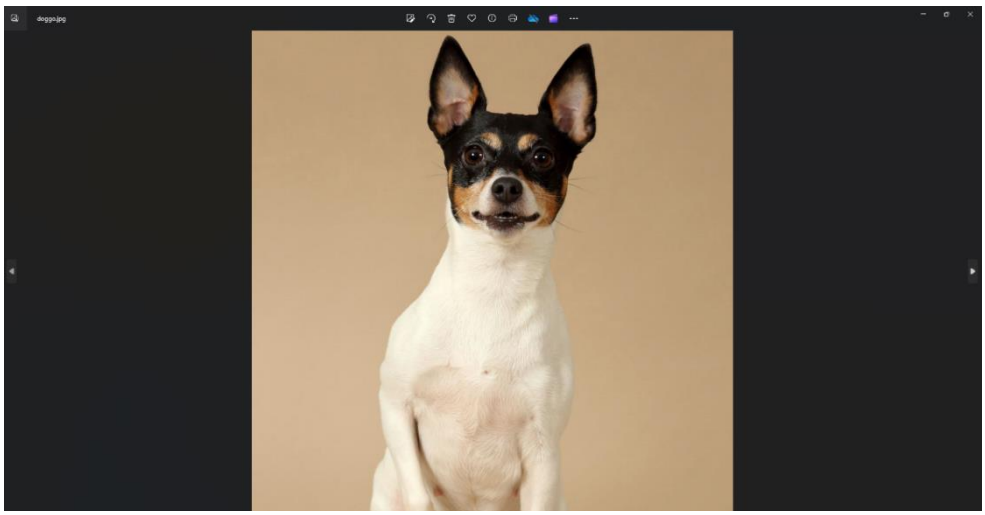


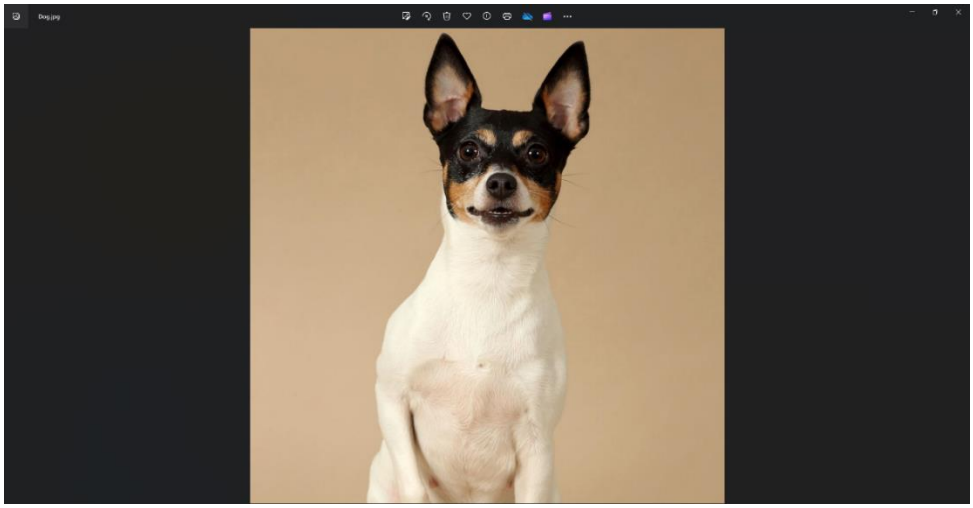


2. Writing an image:

```
import cv2 as cv    # import cv2
import sys      #import sys
img = cv.imread("X:\AI_V_lab\exp2\doggo.jpg")    # load image
if img is None:
    sys.exit("Couldn't read the image")     # prints this when image is not found
else:
    cv.imshow("Final", img)     # show image
k = cv.waitKey(0)   # holds the image in window
if k == ord('s'):
    cv.imwrite("X:\AI_V_lab\exp2\Dog.jpg", img)  # saves the image
cv.destroyAllWindows()      # removes all the window
```

Color input image:

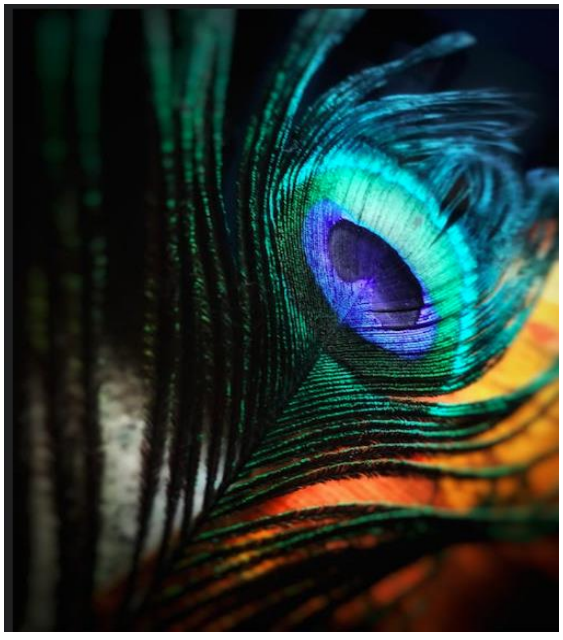




3. Gray Scaling:

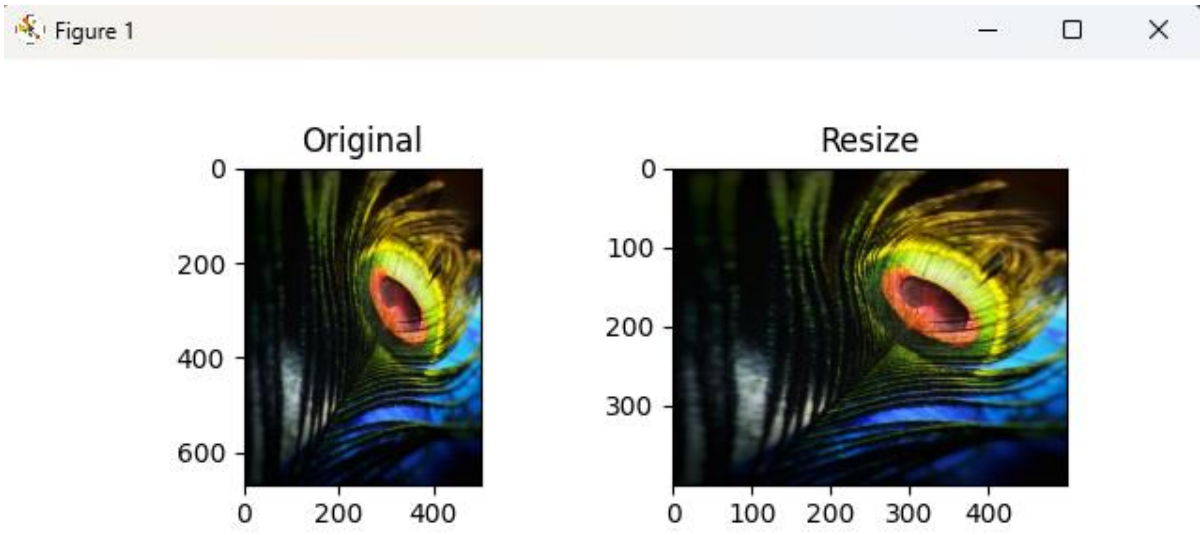
```
import cv2 as cv    # import cv2
import sys      #import sys
img = cv.imread("X:\AI_V_lab\exp2\P_F.jpeg")    # load image
if img is None:
    sys.exit("Couldn't read the image")     # prints this when image is not found
else:
    gray_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)  # convert image to gray scale
    cv.imshow("Final", gray_img)     # show image
cv.waitKey(0)   # holds the image in window
cv.destroyAllWindows()      # removes all the windo
```

Color input image:



4. Resize an Image Using OpenCV:

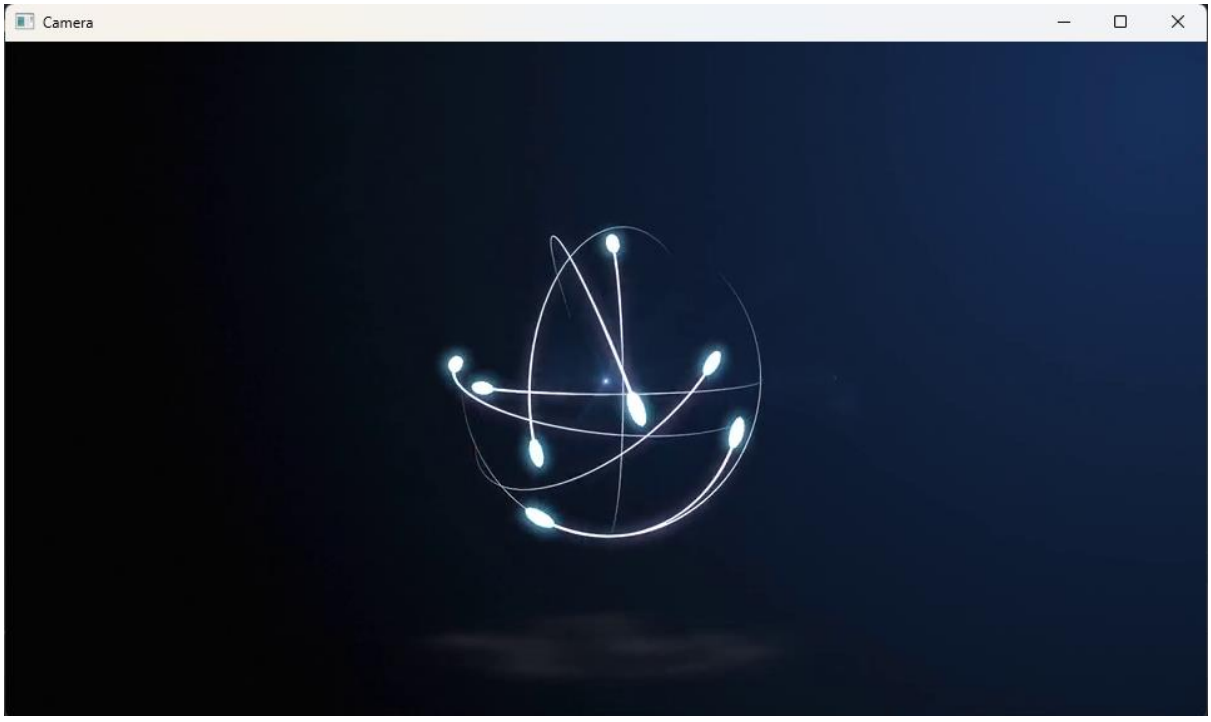
```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
image = cv.imread("X:\AI_V_lab\exp2\P_F.jpeg")
# Loading the image
resize = cv.resize(image, (500, 400), fx = 0.1, fy = 0.1, interpolation = cv.INTER_LINEAR)
Titles =["Original", "Resize"]
images =[image, resize]
count = 2
for i in range(count):
        plt.subplot(2, 2, i + 1)
        plt.title(Titles[i])
        plt.imshow(images[i])
plt.show()
```



5. Video Handling Recorded

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

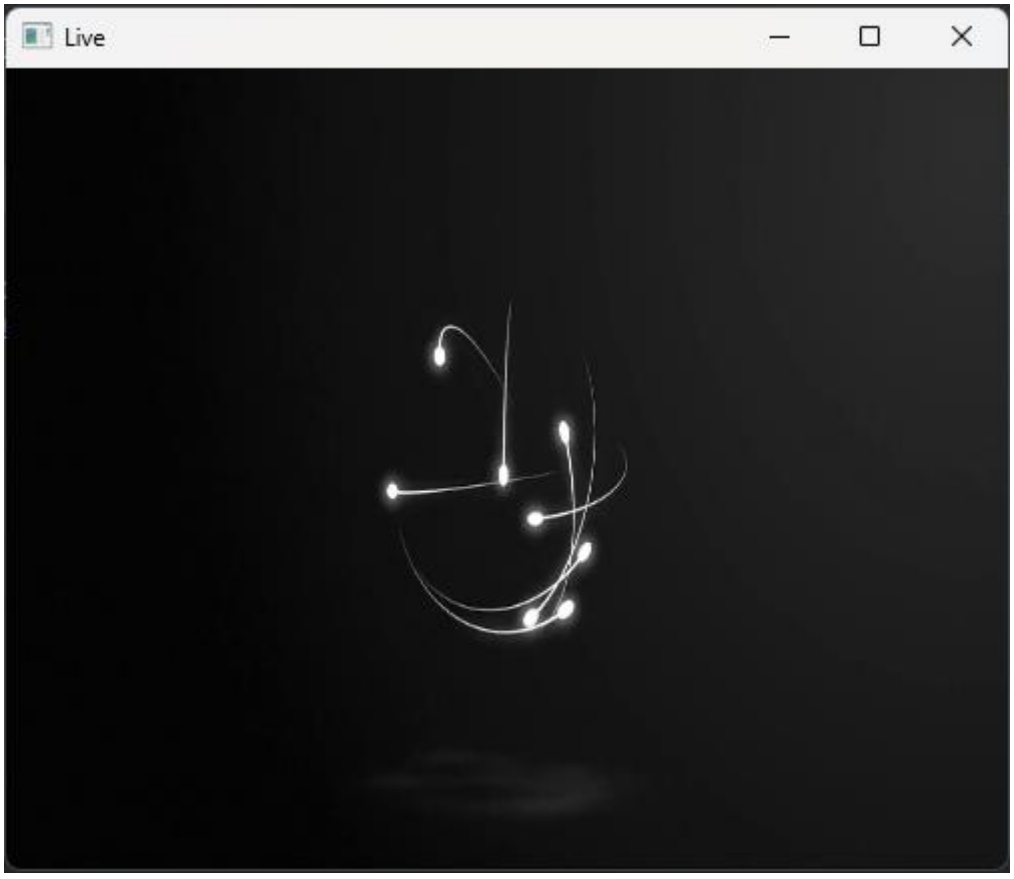
```
cap = cv.VideoCapture("X:/AI_V_lab/exp2/atoms_-_13232 (540p).mp4")                    # capture video

while cap.isOpened():

        ret, frame = cap.read()

        if not ret:

                print("Cant receive frame (stream end?). Exiting ...")

                break

        cv.imshow("Camera",frame)

        if cv.waitKey(1) == ord("q"):

                break

cap.release()

cv.destroyAllWindows()
```



6. Video Handling Recorded and Gray Scaled, Resized

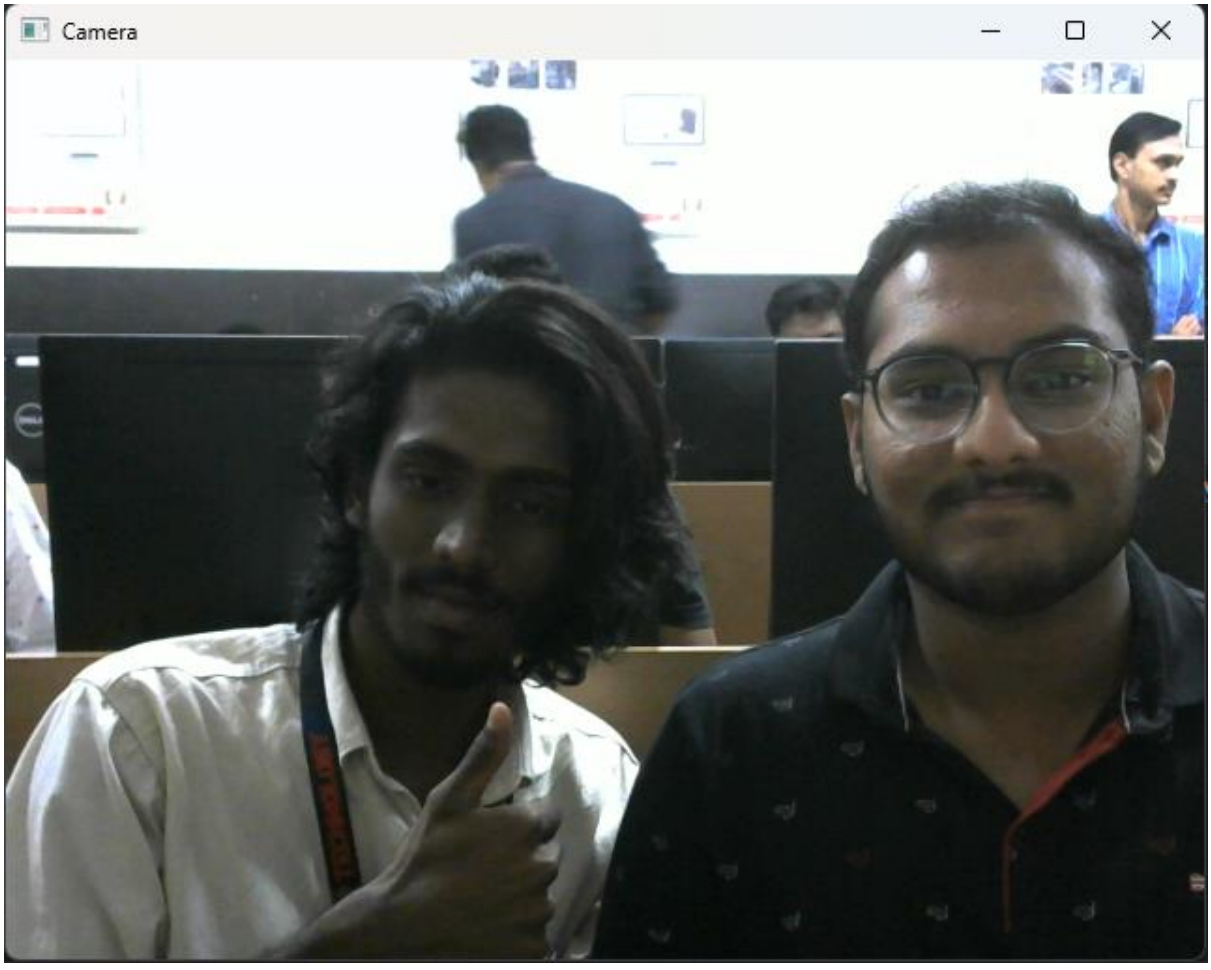
```
# importing the module

import cv2

# reading the video

cap = cv2.VideoCapture("X:/AI_V_lab/exp2/atoms_-_13232 (540p).mp4")

# running the loop

while True:
```

```
    # extracting the frames
    ret, img = cap.read()
    resize = cv2.resize(img, (500, 400))
    # converting to gray-scale
    gray = cv2.cvtColor(resize, cv2.COLOR_BGR2GRAY)
    # displaying the video
    cv2.imshow("Live", gray)
    # exiting the loop
    key = cv2.waitKey(1)
    if key == ord("q"):
        break
# closing the window
cv2.destroyAllWindows()
cap.release()
```



7. Video Handling Live

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
cap = cv.VideoCapture(0)                    # capture video
while cap.isOpened():s
        ret, frame = cap.read()
        if not ret:
                print("Cant receive frame (stream end?). Exiting ...")
                break
        cv.imshow("Camera",frame)
        if cv.waitKey(2) == ord("q"):
                break
cap.release()
cv.destroyAllWindows()
```



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

Result:

The basics of OpenCV Image and Video Handling were learnt using OpenCV- python in Pycharm ID

Post Lab Questions:

1. **List out the image and video file formats supported in OpenCV. What are the formats supported in OpenCV for writing an image into the computer?**

OpenCV supports various image file formats, including JPEG, PNG, TIFF, GIF, BMP, PPM, Sun Raster, HDR, and WEBP. For videos, it supports formats like AVI, MP4, MKV, MOV, FLV, and WMV. Writing images with OpenCV involves using `cv2.imwrite()`, commonly supporting JPEG, PNG, and TIFF formats based on the installation and configuration.

2. **Differentiate the functions used to read the video inputs from a built-in camera, an external USB camera and from a video file that is saved in your local drive**

Video Source	Function	Description
Built-in Camera (Webcam)	cv2.VideoCapture(0) or cv2.VideoCapture(-1)	Initializes a video capture object for the default camera (index 0) or any available camera if the index is set to -1.
External USB Camera	cv2.VideoCapture(index)	Initializes a video capture object for an external camera by specifying the index corresponding to the camera device. Use ls /dev/video* on Linux to find available indices.
Video File from Local Drive	cv2.VideoCapture('file_path')	Initializes a video capture object for reading from a video file located on your local drive.

3. **How to find the size of an image?**

If already an image is loaded using OpenCV and converted it to a NumPy array, you can use the shape attribute to get its dimensions (height, width, and channels if it's a colored image).

4. **What are the different interpolation methods available in cv2.resize function?**

The cv2.resize() function supports several interpolation methods via the interpolation parameter. Here are the different interpolation methods available in cv2.resize():

- cv2.INTER_NEAREST: Nearest-neighbor interpolation. It picks the nearest pixel to the new pixel location.

- cv2.INTER_LINEAR: Bilinear interpolation. It calculates the new pixel value using linear interpolation based on the surrounding four pixels.
- cv2.INTER_CUBIC: Bicubic interpolation. It calculates the new pixel value using cubic interpolation based on a 4x4 pixel neighborhood.
- cv2.INTER_AREA: Resampling using pixel area relation. It resizes the image using pixel area relation, which can be a good choice for shrinking an image.
- cv2.INTER_LANCZOS4: Lanczos interpolation over 8x8 pixel neighborhood. It uses a Lanczos interpolation algorithm with a larger neighborhood.

5. **What is the significance of the waitKey() function in OpenCV?**
   - **Displaying Images/Videos:**

     When used with the imshow() function to display images or video frames, waitKey() helps to maintain the display window by waiting for a specified amount of time for a key event.
   - **Event Handling:**

     It's often used to detect and handle keyboard events. When a window is open and waitKey() is called, it waits for the specified delay in milliseconds for any key event. If a key is pressed within that time frame, it returns the ASCII value of the key.
   - **Control Flow in Real-Time Applications:**

     In real-time applications where video frames are continuously displayed, waitKey() allows for the control of the frame rate by introducing a delay between the display of frames. This is critical for smooth video playback.

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:

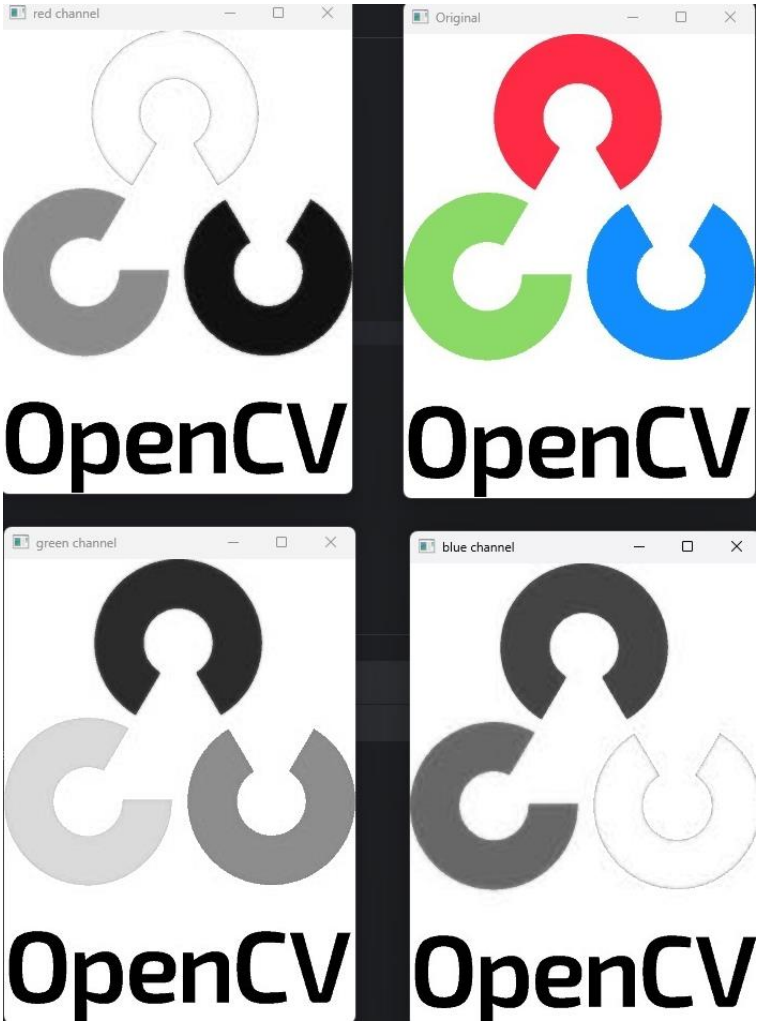
1. Pycharm IDE
2. Libraries used:
    - NumPy
    - opencv-python
    - matplotlib
    - scipy

Programs:

1] SPLITTING

```
import cv2
image = cv2.imread('X:\AI_V_lab\exp3\logo.jpg')
#split the image into its three channels
(b_channel, g_channel, r_channel) = cv2.split(image)
#display the images
cv2.imshow('Original',image)
cv2.imshow('blue channel',b_channel)
cv2.imshow('green channel',g_channel)
cv2.imshow('red channel',r_channel)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



2] LOG TRANSFORMATION

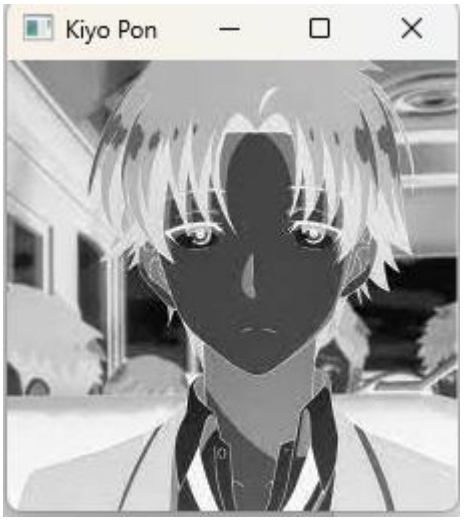
```
import cv2 as cv
import matplotlib as mpt
import numpy as np
# Open the image.
img = cv.imread('X:/AI_V_lab/exp3/a_k.jpg')
# Convert to grayscale
g_s = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
cv.imshow("Kiyo Pon gray", g_s)
# Apply log transform.
c = 255 / (np.log(1 + np.max(g_s)))
log_transformed = c * np.log(1 + g_s)
# Specify the data type.
log_transformed = np.array(log_transformed, dtype=np.uint8)
cv.imshow("Kiyo Pon", log_transformed)
cv.waitKey(0)
cv.destroyAllWindows()
```

INPUT: **OUTPUT:**

3] NEGATIVE

```
import cv2 as cv
import matplotlib as mpt
import numpy as np
# Open the image
img = cv.imread('X:/AI_V_lab/exp3/a_k.jpg')
print(img)
# Convert to grayscale
g_s = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
print(g_s)
# Convert to negative
neg = 255-g_s
print(neg)
neg_ = 255-img
print(neg_)
# Show the negative image
cv.imshow("Kiyo Pon", neg)
cv.imshow("Kiyo Pon_", neg_)
cv.waitKey(0)
cv.destroyAllWindows()
```

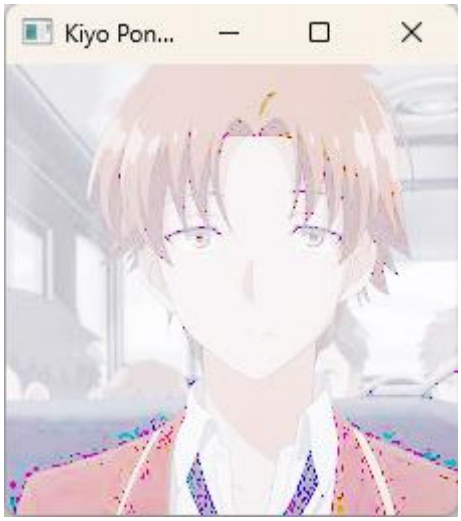
OUTPUT:

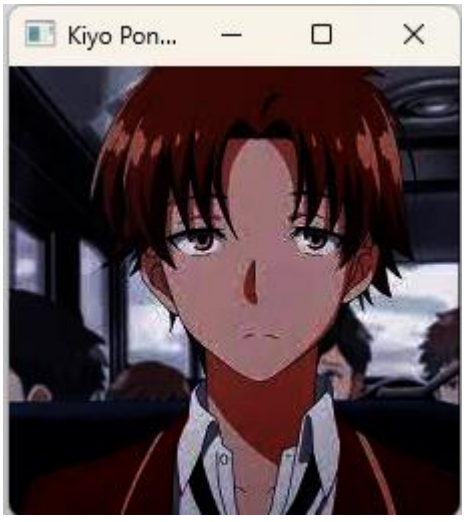


4] GAMMA

```
import cv2 as cv
import matplotlib as mpt
import numpy as np
# Open the image
img = cv.imread('X:/AI_V_lab/exp3/a_k.jpg')
# Convert to grayscale
g_s = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
# 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')
    # Show the negative image
    cv.imshow("Kiyo Pon" + str(gamma), gamma_corrected)
cv.waitKey(0)
cv.destroyAllWindows()
```

OUTPUT:

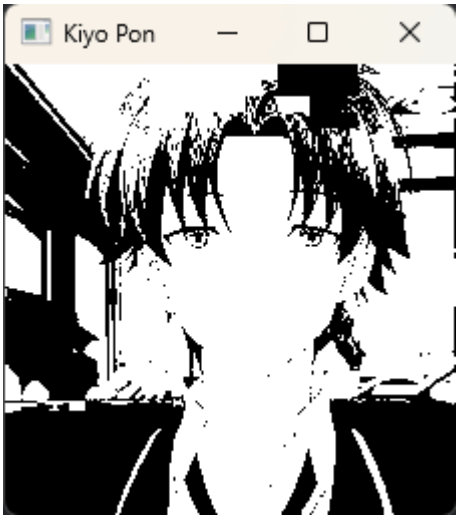




5] CONTRAST STRETCHING

```
import cv2 as cv
import matplotlib as mpt
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 = cv.imread('X:/AI_V_lab/exp3/a_k.jpg')
# Convert to grayscale
g_s = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
# 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(g_s, r1, s1, r2, s2)
# Show the negative image
cv.imshow("Kiyo Pon", contrast_stretched)
cv.waitKey(0)
cv.destroyAllWindows()
```

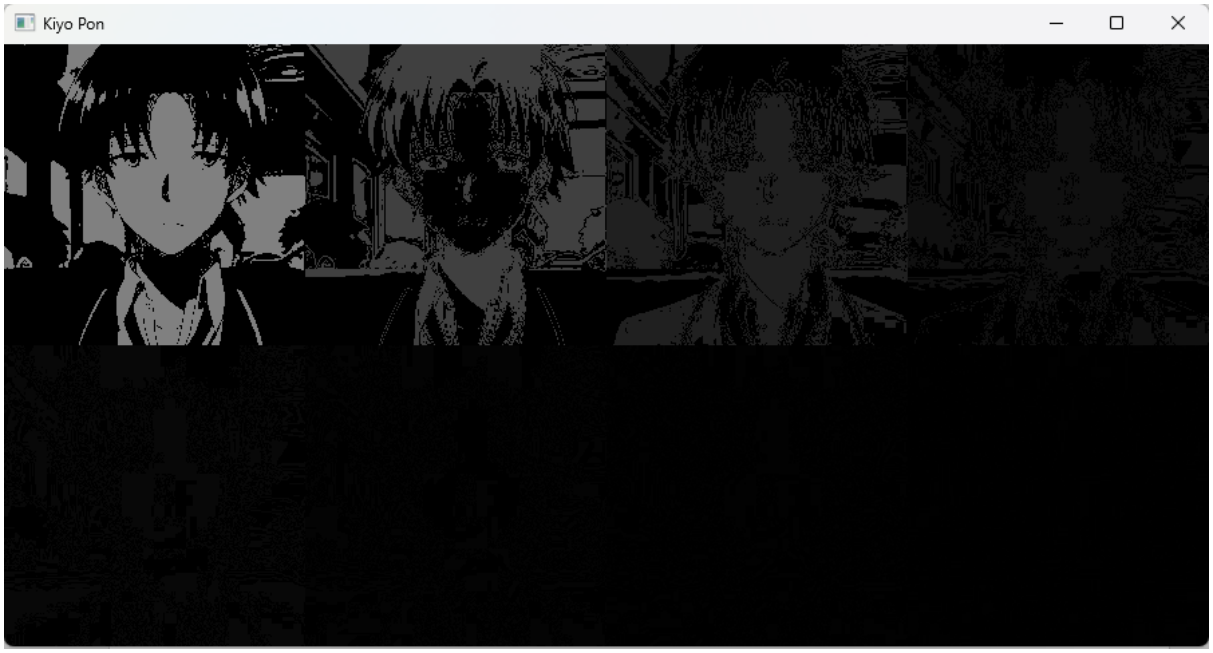
OUTPUT:



6] BIT PLANE SLICING:

```
import cv2 as cv
import matplotlib as mpt
import numpy as np
# Open the image
img = cv.imread('X:/AI_V_lab/exp3/a_k.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 = cv.hconcat([eight_bit_img, seven_bit_img, six_bit_img, five_bit_img])
finalv = cv.hconcat([four_bit_img, three_bit_img, two_bit_img, one_bit_img])
# Vertically concatenatess
final = cv.vconcat([finalr, finalv])
# Show the negative image
cv.imshow("Kiyo Pon", final)
cv.waitKey(0)
cv.destroyAllWindows()
```

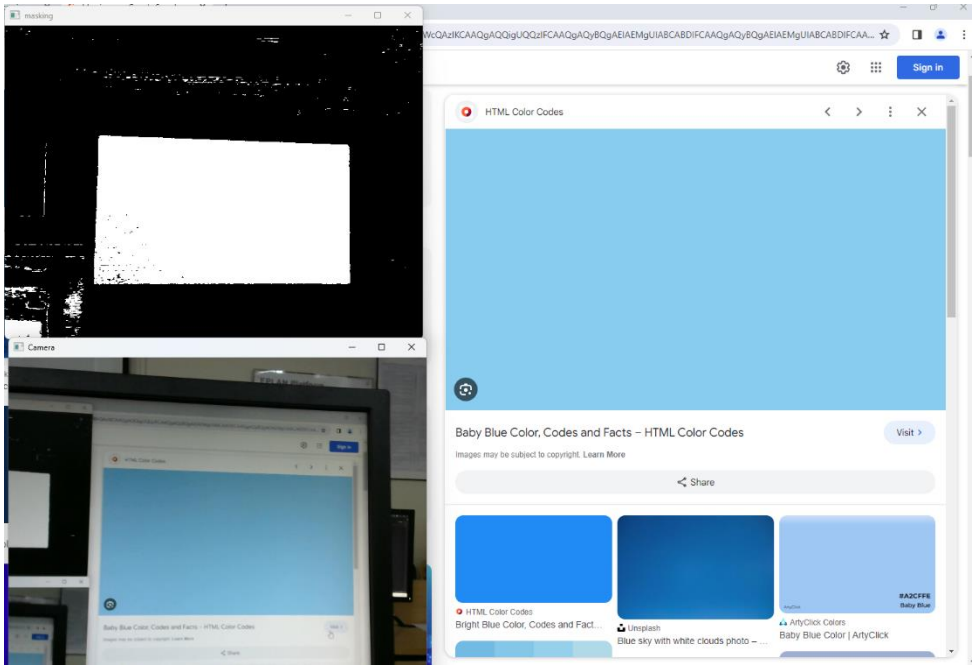
OUTPUT:



7] VIDEO MASKING

```
import cv2
import cv2 as cv
import matplotlib as mpt
import numpy as np
low_blue = np.array([52, 25, 72])
high_blue = np.array([102, 225, 255])
cap = cv.VideoCapture(0)
# loop to be executed when video is opened
while cap.isOpened():
   ret, frame = cap.read()
   if not ret:
      print("Cant receive frame (stream end?). Exiting ...")
      break
   cv.imshow("Camera", frame)
   hsv = cv.cvtColor(frame,cv2.COLOR_BGR2HSV)
   mask = cv.inRange(hsv, low_blue, high_blue)
   cv.imshow("masking",mask)
   if cv.waitKey(1) == ord("q"):
      break
cap.release()
cv.destroyAllWindows()
```

OUTPUT:



8] IMAGE MASKING

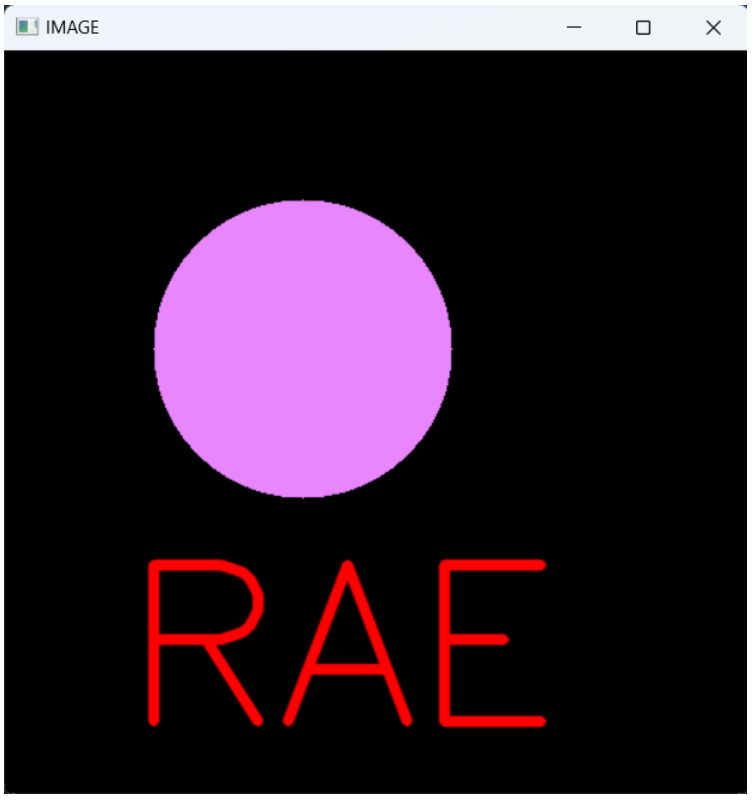
```
# import required libraries
import cv2
import numpy as np
# read input image
img = cv2.imread("X:\AI_V_lab\exp3\CAR.jpg")
cv2.imshow('Original',          img)
# Convert BGR to HSV
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
# define range of blue color in HSV
lower_yellow = np.array([15,50,180])
upper_yellow = 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.imshow('Masked Image',result)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:

9] CRICLE AND WORD

```
import cv2
import numpy as np
# Reading an image in default mode
Img = np.zeros((500, 500, 3), np.uint8)
# Center coordinates
center_coordinates = (200, 200)
# Radius of circle
radius = 100
# Red color in BGR
color = (255, 133, 233)
# Line thickness of -1 px
thickness = -1
# Using cv2.circle() method
# Draw a circle of red color of thickness -1 px
image = cv2.circle(Img, center_coordinates, radius, color, thickness)
# Using cv2.putText() method
wrd = cv2.putText(Img, 'RAE', (80,450), cv2.FONT_HERSHEY_SIMPLEX ,
                  5,(0, 0, 255), 5, cv2.LINE_AA)
# Displaying the image
cv2.imshow("IMAGE", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

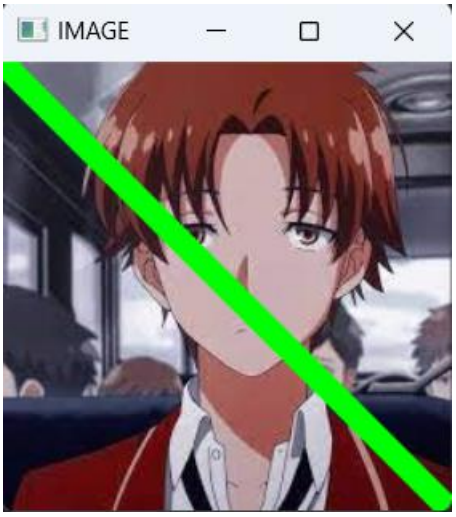
OUTPUT:



10] LINE AND IMAGE:

```
# importing cv2
import cv2
# Reading an image in default mode
image = cv2.imread("C:/Users/yuvan/Desktop/kiyoPon.jpg")
# Start coordinate, here (0, 0)
# represents the top left corner of image
start_point = (0, 0)
# End coordinate, here (250, 250)
# represents the bottom right corner of image
end_point = (250, 250)
# Green color in BGR
color = (0, 255, 0)
# Line thickness of 9 px
thickness = 9
# Draw a diagonal green line with thickness of 9 px
image = cv2.line(image, start_point, end_point, color, thickness)
# Displaying the image
cv2.imshow('IMAGE', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

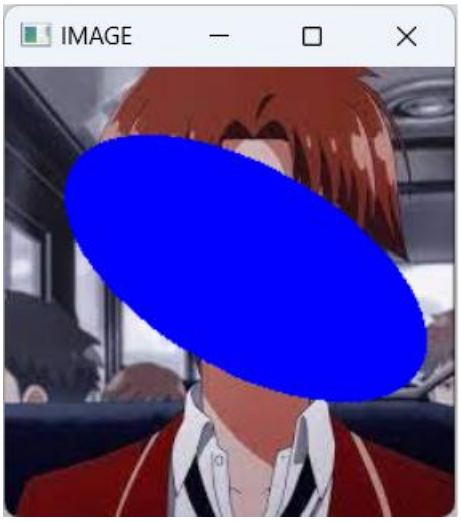
OUTPUT:



11] ELLIPSE:

```
# importing cv2
import cv2+
# Reading an image in default mode
image = cv2.imread("C:/Users/yuvan/Desktop/kiyoPon.jpg")
center_coordinates = (120, 100)
axesLength = (100, 50)
angle = 30
startAngle = 0
endAngle = 360
# Blue color in BGR
color = (255, 0, 0)
# Line thickness of -1 px
thickness = -1
# Draw a ellipse with blue line borders of thickness of -1 px
image = cv2.ellipse(image, center_coordinates, axesLength, angle,
                    startAngle, endAngle, color, thickness)

# Displaying the image
cv2.imshow("IMAGE", image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

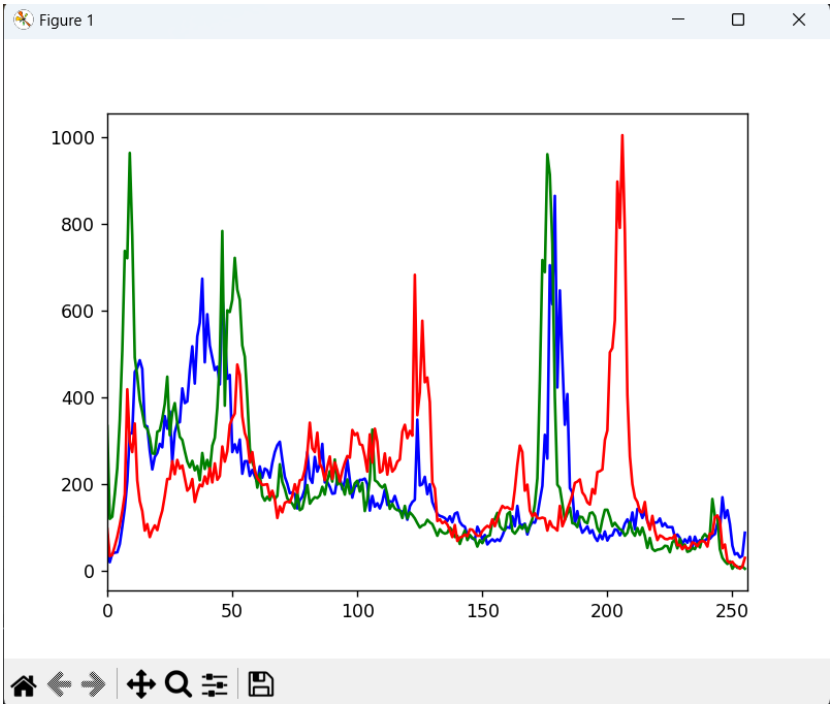
OUTPUT:



12] HISTOGRAM:

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('C:/Users/yuvan/Desktop/kiyoPon.jpg')
assert img is not None, "file could not be read, check with os.path.exists()"
color = ('b','g','r')
for i,col in enumerate(color):
    histr = cv.calcHist([img],[i],None,[256],[0,256])
    plt.plot(histr,color = col)
    plt.xlim([0,256])
plt.show()
```

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.

Experiment 4

Geometric Transformation and Filtering Using OpenCV

Aim:

1) To implement the following Geometric Transformations a Filtering functions on an image in Open CV:
   a) Translation, Rotation, Affine Transformation and Perspective Transformation
   b) 2D convolution, Averaging and Blurring
   c) Thresholding

Software/ Packages Used:

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

Programs:

<u>Geometric Transformations</u>

```
#Translation

#Rotation

# Affine Transformation

#Perspective Transformation
```

<u>Filtering</u>

```
#Image Blurring
#Averaging
#Median Blurring
#Bilateral Filtering
```

<u>Thresholding</u>

```
#Simple
#Adaptive
#Ostu's
```

1] AVERAGE FILTERING:

```
import cv2 as cv
import matplotlib.pyplot as plt
import numpy as np
# Reading the image
image = cv.imread("X:/AI_V_lab/exp3/a_k.jpg")
# Applying the filter
averageBlur3 = cv.blur(image, (3, 3))
averageBlur5 = cv.blur(image, (5, 5))
averageBlur9 = cv.blur(image, (9, 9))
cv.imshow('Original',image)
cv.imshow('average Blur 3',averageBlur3)
cv.imshow('average Blur 5',averageBlur5)
cv.imshow('average Blur 9',averageBlur9)
cv.waitKey(0)
cv.destroyAllWindows()
```

OUTPUT:

2] GAUSSIAN FILTERING:

```
import cv2 as cv
import matplotlib.pyplot as plt
import numpy as np
# Reading the image
image = cv.imread("X:/AI_V_lab/exp3/a_k.jpg")
# Applying the filter
GaussianBlur3 = cv.GaussianBlur(image, (3, 3),0)
```

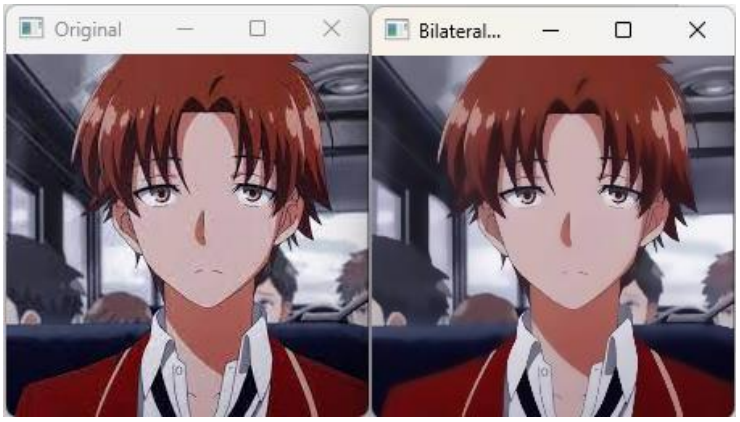
```
GaussianBlur5 = cv.GaussianBlur(image, (5, 5),0)
GaussianBlur9 = cv.GaussianBlur(image, (9, 9),0)
cv.imshow('Original',image)
cv.imshow('GaussianBlur 3',GaussianBlur3)
cv.imshow('GaussianBlur 5',GaussianBlur5)
cv.imshow('GaussianBlur 9',GaussianBlur9)
cv.waitKey(0)
cv.destroyAllWindows()
```

OUTPUT:

3] BILATERAL FILTERING:

```
import cv2 as cv
import matplotlib.pyplot as plt
import numpy as np
# Reading the image
image = cv.imread("X:/AI_V_lab/exp3/a_k.jpg")
# Applying the filter
bilateral = cv.bilateralFilter(image, 9, 75, 75)
# Showing the image
cv.imshow('Original', image)
cv.imshow('Bilateral blur', bilateral)
cv.waitKey(0)
cv.destroyAllWindows()
```

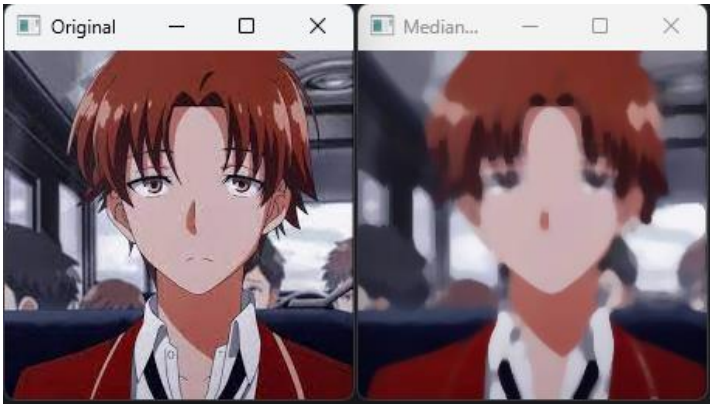
OUTPUT:



4] MEDIAN FILTERING:

```
import cv2 as cv
import matplotlib.pyplot as plt
import numpy as np
# Reading the image
image = cv.imread("X:/AI_V_lab/exp3/a_k.jpg"
# Applying the filter
medianBlur = cv.medianBlur(image, 9)
# Showing the image
cv.imshow('Original', image)
cv.imshow('Median blur', medianBlur)
cv.waitKey(0)
cv.destroyAllWindows()
```

OUTPUT:



5] BLUR FILTERING-MATRIX:

```
# import the important module in python
import cv2 as cv
import numpy as np
# make a matrix with numpy
matrix = np.matrix('[1, 2, 3; 4, 5, 6; 7, 8, 9]')
# Applying the filter
averageBlur3 = cv.blur(matrix, (3, 3))
averageBlur5 = cv.blur(matrix, (5, 5))
averageBlur9 = cv.blur(matrix, (9, 9))
print("Original\n",matrix)
print("averageBlur\n",averageBlur3)
```

```
print("averageBlur5\n",averageBlur5)
print("averageBlur9\n",averageBlur9)
```

OUTPUT:

```
Original
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
averageBlur3
 [[4 4 4]
 [5 5 5]
 [6 6 6]]
averageBlur5
 [[6 6 5]
 [5 5 5]
 [5 4 4]]
averageBlur9
 [[5 5 5]
 [5 5 5]
 [5 5 5]]
```

6] GAUSSIAN FILTERING-MATRIX:

```
# import the important module in python
import cv2 as cv
import numpy as np
# make a matrix with numpy
matrix = np.array([[1, 2, 3],[ 4, 5, 6],[ 7, 8, 9]],dtype=np.uint8)
# Applying the filter
GaussianBlur3 = cv.GaussianBlur(matrix,(3,3),0)
print("Original\n",matrix)
print("GaussianBlur3\n",GaussianBlur3)
```

OUTPUT:

```
Original
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
GaussianBlur3
 [[3 4 4]
 [5 5 6]
 [6 7 7]]
```

7] Bilateral FILTERING-MATRIX:

```
# import the important module in python
import cv2 as cv
import numpy as np
# make a matrix with numpy
matrix = np.array([[1, 2, 3],[ 4, 5, 6],[ 7, 8, 9]],dtype=np.uint8)
# Applying the filter
bilateral = cv.bilateralFilter(matrix, 9, 75, 75)
print("Original\n",matrix)
```

```
print("Bilateral\n",bilateral)
```

OUTPUT:

```
Original
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
Bilateral
 [[5 5 5]
 [5 5 5]
 [5 5 5]]
```

8] MEDIAN FILTERING-MATRIX:

```
# import the important module in python
import cv2 as cv
import numpy as np
# make a matrix with numpy
matrix = np.array([[1, 2, 3],[ 4, 5, 6],[ 7, 8, 9]],dtype=np.uint8)
# Applying the filter
medianBlur = cv.medianBlur(matrix, 9)
print("Original\n",matrix)
print("Median\n",medianBlur)
```

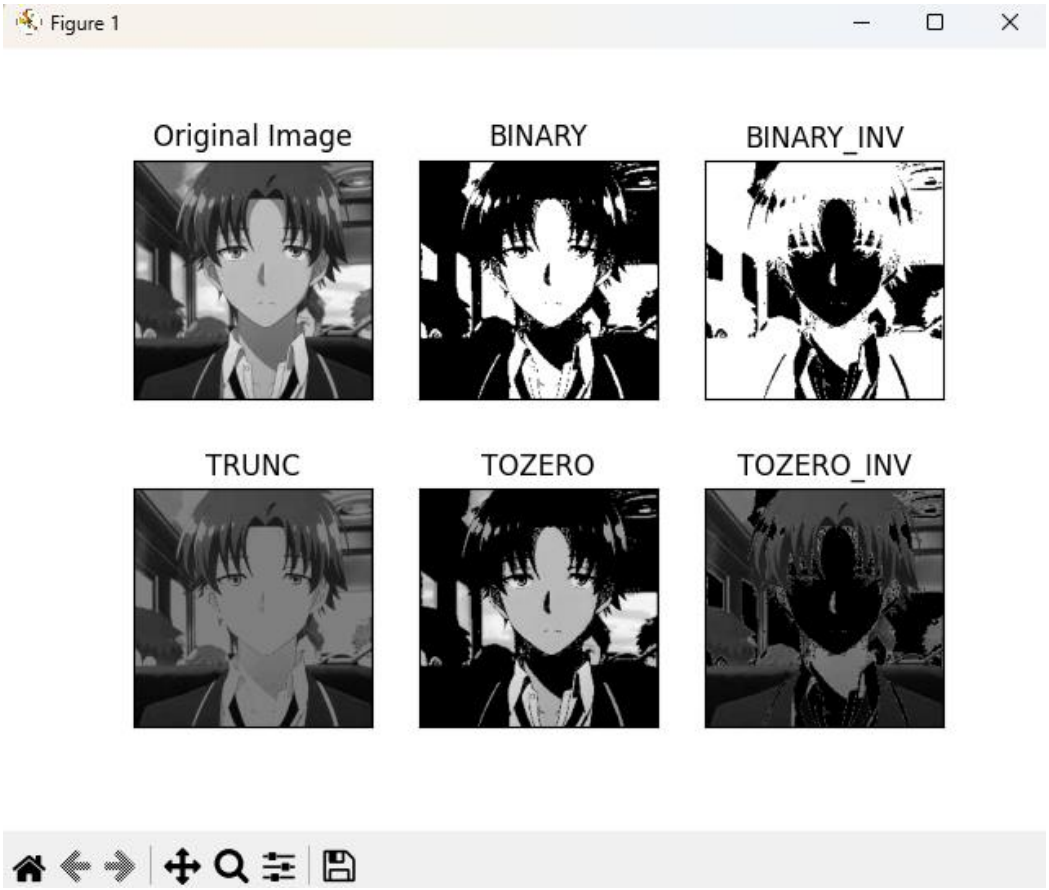
OUTPUT:

```
Original
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
Median
 [[3 3 3]
 [4 5 6]
 [7 7 7]]
```

9] SIMPLE THRESHOLD:

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img = cv.imread('X:/AI_V_lab/exp3/a_k.jpg', cv.IMREAD_GRAYSCALE)
ret,thresh1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
ret,thresh2 = cv.threshold(img,127,255,cv.THRESH_BINARY_INV)
ret,thresh3 = cv.threshold(img,127,255,cv.THRESH_TRUNC)
ret,thresh4 = cv.threshold(img,127,255,cv.THRESH_TOZERO)
ret,thresh5 = cv.threshold(img,127,255,cv.THRESH_TOZERO_INV)
titles = ['Original Image','BINARY','BINARY_INV','TRUNC','TOZERO','TOZERO_INV']
images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]
for i in range(6):
   plt.subplot(2,3,i+1),plt.imshow(images[i],'gray',vmin=0,vmax=255)
   plt.title(titles[i])
   plt.xticks([]),plt.yticks([])
plt.show()
```

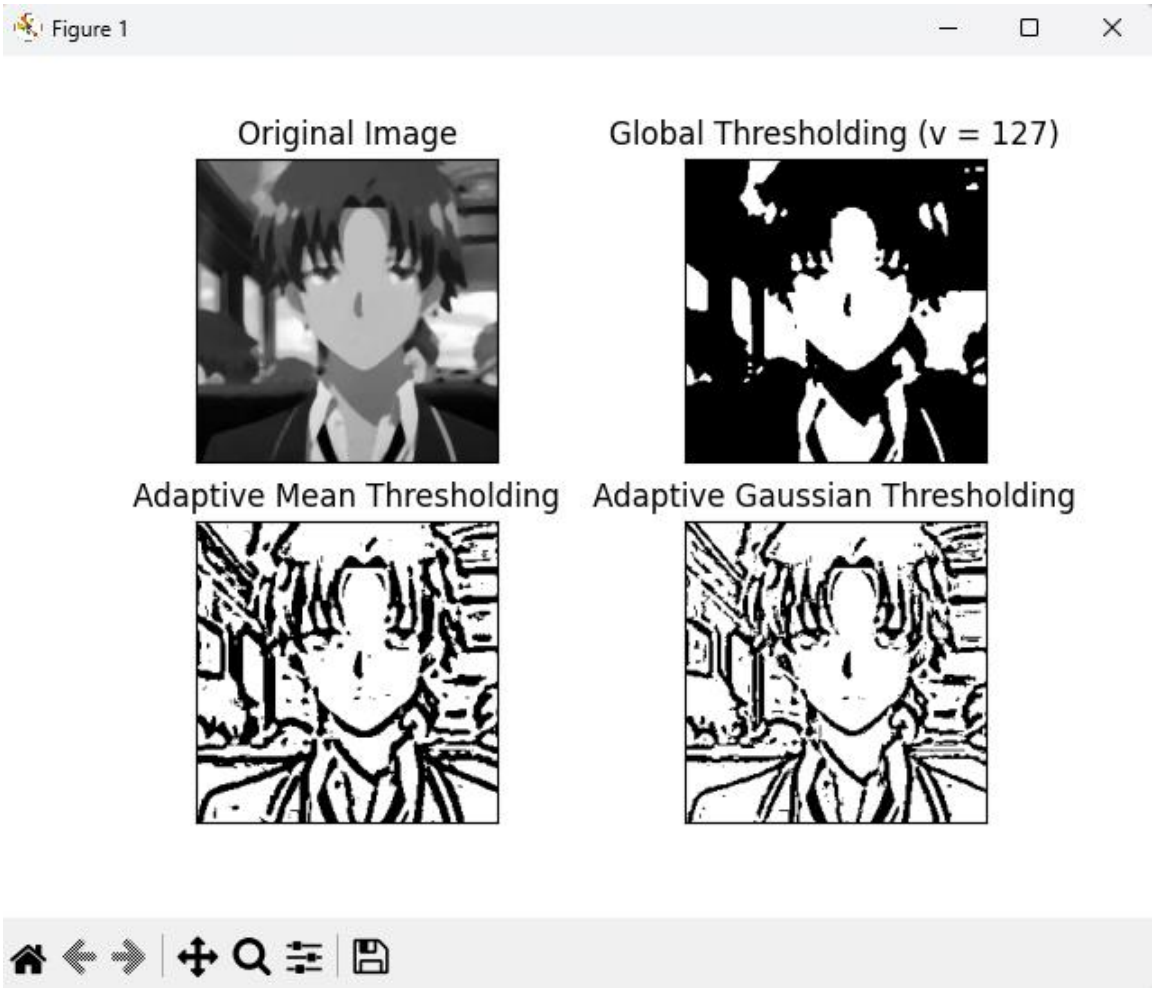
OUTPUT:



10] ADAPTIVE THRESHOLD:

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img = cv.imread('X:/AI_V_lab/exp3/a_k.jpg', cv.IMREAD_GRAYSCALE)
img = cv.medianBlur(img,5)
ret,th1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
th2 =
cv.adaptiveThreshold(img,255,cv.ADAPTIVE_THRESH_MEAN_C,cv.THRESH_BINARY,11,2)
th3 =
cv.adaptiveThreshold(img,255,cv.ADAPTIVE_THRESH_GAUSSIAN_C,cv.THRESH_BINARY,1
1,2)
titles = ['Original Image', 'Global Thresholding (v = 127)','Adaptive Mean Thresholding',
'Adaptive Gaussian Thresholding']
images = [img, th1, th2, th3]
for i in range(4):
   plt.subplot(2,2,i+1),plt.imshow(images[i],'gray')
   plt.title(titles[i])
   plt.xticks([]),plt.yticks([])
plt.show()
```

OUTPUT:

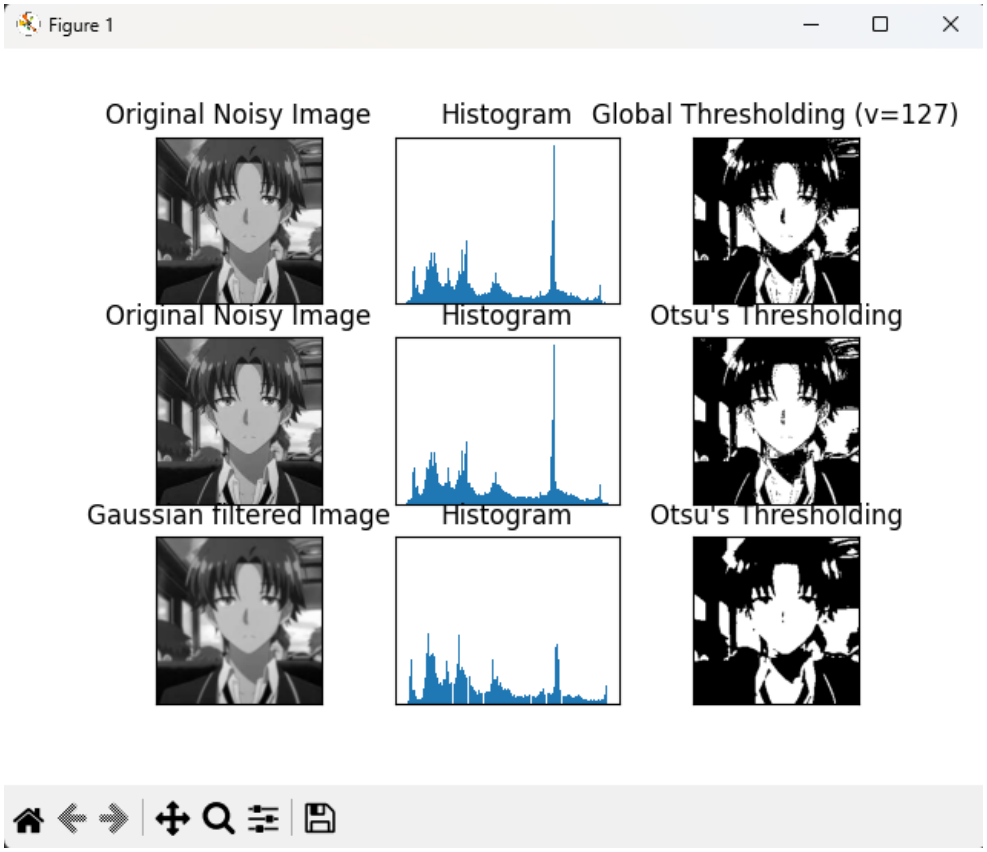


11] OSTU BINIRIZATION:

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img = cv.imread('X:/AI_V_lab/exp3/a_k.jpg', cv.IMREAD_GRAYSCALE)
# global thresholding
ret1,th1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
# Otsu's thresholding
ret2,th2 = cv.threshold(img,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
# Otsu's thresholding after Gaussian filtering
blur = cv.GaussianBlur(img,(5,5),0)
ret3,th3 = cv.threshold(blur,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
# plot all the images and their histograms
images = [img, 0, th1,
        img, 0, th2,
        blur, 0, th3]
titles = ['Original Noisy Image','Histogram','Global Thresholding (v=127)',
        'Original Noisy Image','Histogram',"Otsu's Thresholding",
        'Gaussian filtered Image','Histogram',"Otsu's Thresholding"]
for i in range(3):
   plt.subplot(3,3,i*3+1),plt.imshow(images[i*3],'gray')
```

```
    plt.title(titles[i*3]), plt.xticks([]), plt.yticks([])
    plt.subplot(3,3,i*3+2),plt.hist(images[i*3].ravel(),256)
    plt.title(titles[i*3+1]), plt.xticks([]), plt.yticks([])
    plt.subplot(3,3,i*3+3),plt.imshow(images[i*3+2],'gray')
    plt.title(titles[i*3+2]), plt.xticks([]), plt.yticks([])
plt.show()
```

OUTPUT:

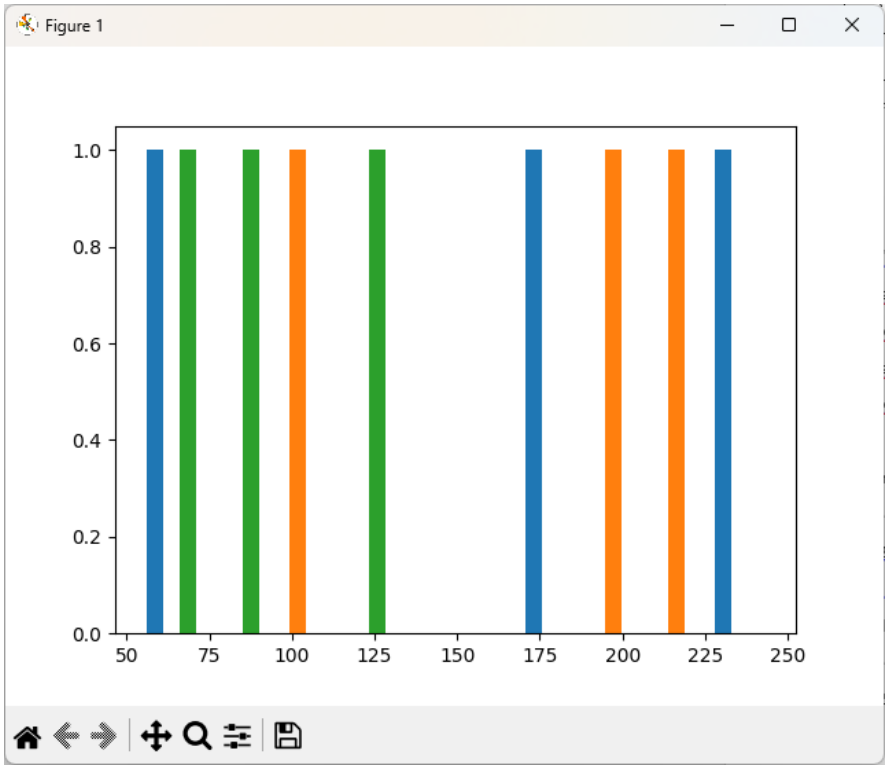


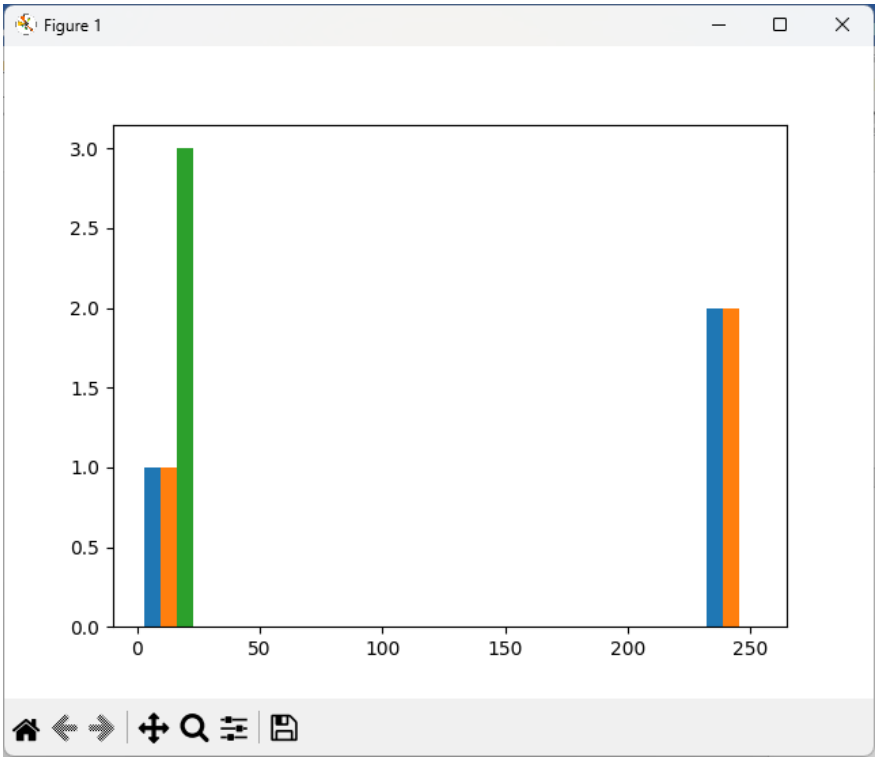
12] THRESHOLD-MATRIX:

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
# make a matrix with numpy
matrix = np.array([[245, 210, 130],[ 54, 205, 60],[ 170, 108, 90]],dtype=np.uint8)
thresh1 = np.empty((3,3),dtype=np.uint8)
print("Original\n",matrix)
for i in range(0,3):
   for j in range(0,3):
      if matrix[i][j] <= 150:
         thresh1[i][j] = 0
      else:
         thresh1[i][j] = 255
print("Threshold Matrix\n",thresh1)
plt.hist(matrix)
plt.show()
plt.hist(thresh1)
plt.show()
```

OUTPUT:

Original
[[245 210 130]
[54 205 60]
[170 108 90]]
Threshold Matrix
[[255 255 255]
[0 255   0]
[255   0   0]]





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 Geometrical Transformations and Filtering Techniques were learnt using OpenCV.

Post Lab Questions:

1. **What is the difference between affine transformation and perspective transformation?**
2. **What do you mean by cartooning?**
3. **What does the filter2D function do? Explain with the arguments**
4. **Write a program for Ostu's thresholding without using inbuilt function**

ANSWERS:

1.

Affine Transformation:

Affine transformations preserve straight lines and ratios of distances between points. It includes operations such as translation, rotation, scaling, and skewing. In affine transformations, parallel lines remain parallel after transformation.

Perspective Transformation:

Perspective transformations, also known as projective transformations, involve the transformation of a three-dimensional scene onto a two-dimensional plane. This type of transformation is used to represent three-dimensional objects in two dimensions and includes effects like foreshortening and depth.

2. Cartooning refers to the process of simplifying an image to emphasize its main features, reducing details, and often using bold, exaggerated lines and colors. It is a common technique in art and graphics to create a stylized or artistic representation of a subject. In image processing, cartooning filters are applied to images to achieve a similar effect, reducing the complexity and enhancing prominent features.

3. **filter2D** is a function commonly used in image processing, and it performs a 2D convolution operation on an image. The general syntax in OpenCV (assuming Python) is:

cv2.filter2D(src, ddepth, kernel[, dst[, anchor[, delta[, borderType]]]])

src: Input image.
ddepth: Depth of the output image.
kernel: Convolution kernel (a matrix).
dst: Output image (optional).
anchor: Anchor point of the kernel. Default is (-1, -1), meaning the anchor is at the kernel center.
delta: Optional value added to the filtered pixels before storing them in the output image.
borderType: Pixel extrapolation method.

4.

```
import cv2
import numpy as np

def otsu_thresholding(img):
    # Compute histogram
    hist, bins = np.histogram(img.flatten(), 256, [0, 256])
```

```
    # Calculate probabilities
    prob = hist / float(np.sum(hist))

    # Initialize variables
    sum_total = np.sum(np.arange(256) * prob)
    sum_back = 0
    w_back = 0
    max_variance = 0
    threshold = 0

    # Iterate through all possible thresholds
    for i in range(256):
        w_back += prob[i]
        if w_back == 0:
            continue
        w_fore = 1 - w_back

        sum_back += i * prob[i]

        mean_back = sum_back / w_back
        mean_fore = (sum_total - sum_back) / w_fore

        # Calculate between-class variance
        between_variance = w_back * w_fore * (mean_back - mean_fore)**2

        # Update threshold if variance is higher
        if between_variance > max_variance:
            max_variance = between_variance
            threshold = i

    # Apply threshold to the image
    _, thresholded_img = cv2.threshold(img, threshold, 255, cv2.THRESH_BINARY)

    return thresholded_img

# Example usage:
image = cv2.imread('your_image.jpg', cv2.IMREAD_GRAYSCALE)
result = otsu_thresholding(image)
cv2.imshow('Otsu Thresholding Result', result)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Experiment 5

Edge Detection and Template Matching

Aim:

To implement Edge detection techniques and template matching in OpenCV.

Software/ Packages Used:

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

Programs:

// https://docs.opencv.org/4.x/d5/d0f/tutorial_py_gradients.html
Edge Detection (Both for Image and Video)

#Sobel
#Prewit
#Laplacian
#Canny Edge Detection

Template Matching (Both for Image and Video)
a.Single Template
b.Multiple template
c.Video streaming

1] SOBEL EDGE DETECTION:

```
import cv2
# Read the original image
img = cv2.imread("X:/AI_V_lab/exp3/a_k.jpg")
# Display original image
cv2.imshow('Original', img)
# Convert to graycsale
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Blur the image for better edge detection
img_blur = cv2.GaussianBlur(img_gray, (3, 3), 0)
# Sobel Edge Detection
sobelx = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=1, dy=0, ksize=5)  # Sobel Edge
Detection on the X axis
```

```
sobely = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=0, dy=1, ksize=5)  # Sobel Edge Detection on the Y axis
sobelxy = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=1, dy=1, ksize=5)  # Combined X and Y Sobel Edge Detection
# Display Sobel Edge Detection Images
cv2.imshow('Sobel X', sobelx)
cv2.imshow('Sobel Y', sobely)
cv2.imshow('Sobel X Y using Sobel() function', sobelxy)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

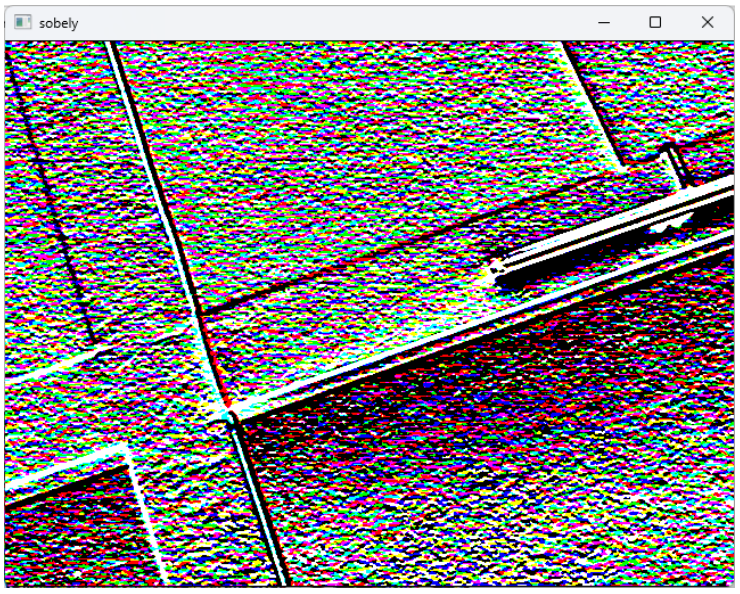
OUTPUT:

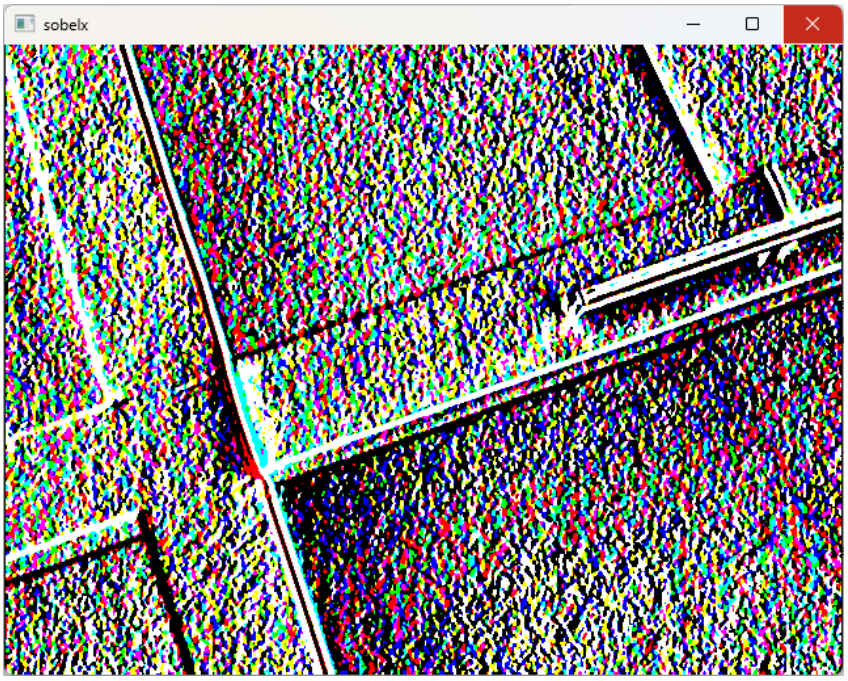
2] SOBEL EDGE DETECTION(CAMERA):

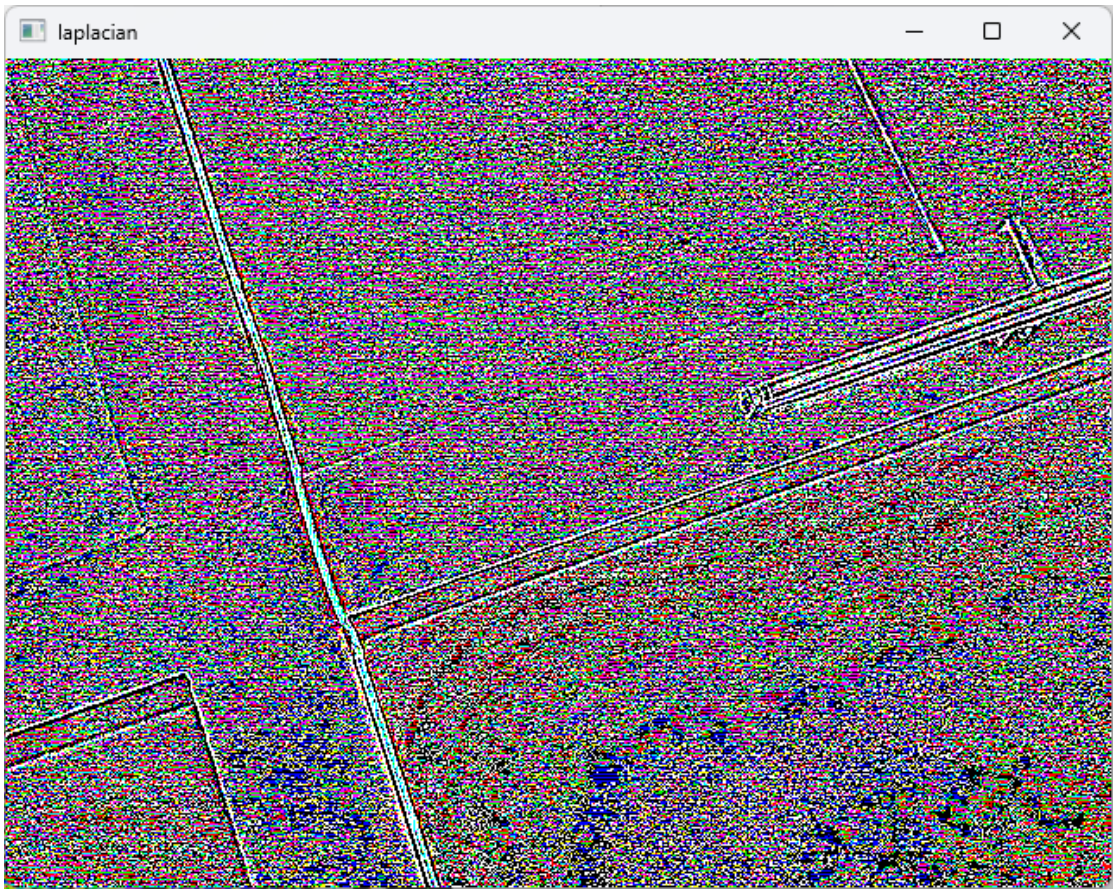
```
# Python program to Edge detection
# using OpenCV in Python
# using Sobel edge detection
# and laplacian method
import cv2
import numpy as np
# Capture livestream video content from camera 0
cap = cv2.VideoCapture(0)
while cap.isOpened():
    # Take each frame
    _, frame = cap.read()
 # Convert to HSV for simpler calculations
```

```
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    # Calculation of Sobelx
    sobelx = cv2.Sobel(frame, cv2.CV_64F, 1, 0, ksize=5)
    # Calculation of Sobely
    sobely = cv2.Sobel(frame, cv2.CV_64F, 0, 1, ksize=5)
    # Calculation of Laplacian
    laplacian = cv2.Laplacian(frame, cv2.CV_64F)
    cv2.imshow('sobelx', sobelx)
    cv2.imshow('sobely', sobely)
    cv2.imshow('laplacian', laplacian)
    cv2.waitKey(1)
cap.release()
cv2.destroyAllWindows()
# release the frame
cap.release()
```

OUTPUT:





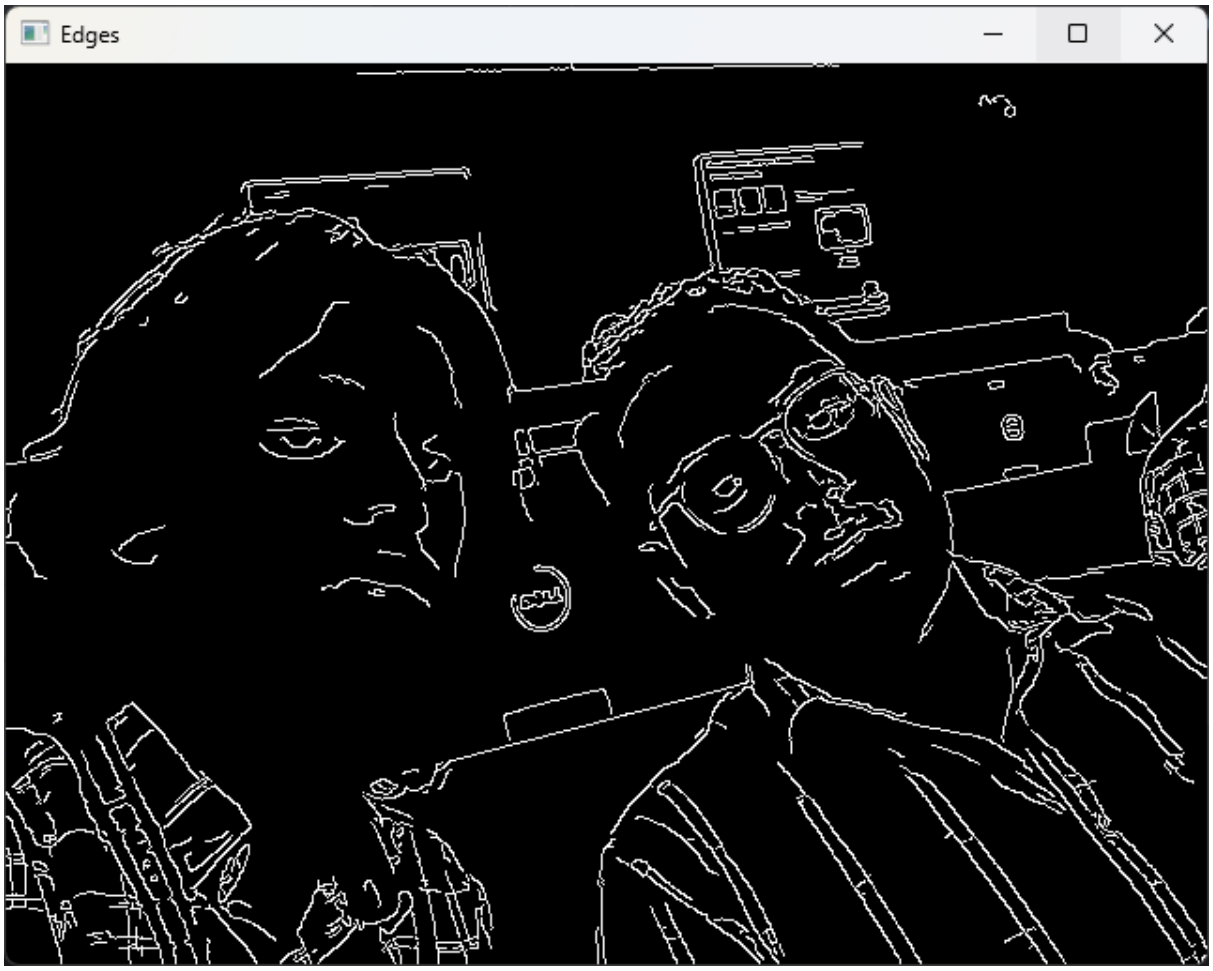


3] CANNY EDGE DETECTION(VIDEO):

```
import cv2
def canny_edge_detection(frame):
    # Convert the frame to grayscale for edge detection
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Apply Gaussian blur to reduce noise and smoothen edges
    blurred = cv2.GaussianBlur(src=gray, ksize=(3, 5), sigmaX=0.5)
    # Perform Canny edge detection
    edges = cv2.Canny(blurred, 70, 135)
    return blurred, edges
if __name__ == "__main__":
    cap = cv2.VideoCapture(0)
    while True:
        # Read a frame from the webcam
        ret, frame = cap.read()
        if not ret:
            print('Image not captured')
            break
        # Perform Canny edge detection on the frame
        blurred, edges = canny_edge_detection(frame)
        # Display the original frame and the edge-detected frame
        # cv2.imshow("Original", frame)
        cv2.imshow("Blurred", blurred)
        cv2.imshow("Edges", edges)
        # Exit the loop when 'q' key is pressed
        cv2.waitKey(1)
    # Release the webcam and close the windows
```

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

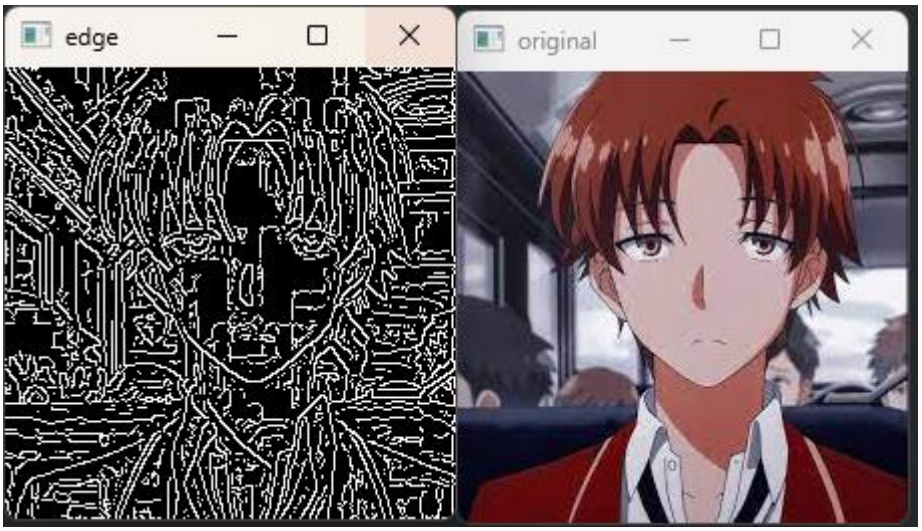
OUTPUT:



4] CANNY EDGE DETECTION:

```
import cv2
img = cv2.imread("X:/AI_V_lab/exp3/a_k.jpg") # Read image
# Setting All parameters
t_lower = 100 # Lower Threshold
t_upper = 200 # Upper threshold
aperture_size = 5 # Aperture size
# Applying the Canny Edge filter
# with Custom Aperture Size
edge = cv2.Canny(img, t_lower, t_upper, apertureSize=aperture_size)
cv2.imshow('original', img)
cv2.imshow('edge', edge)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:



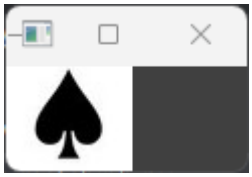
5] MULTI-TEMPLATE:

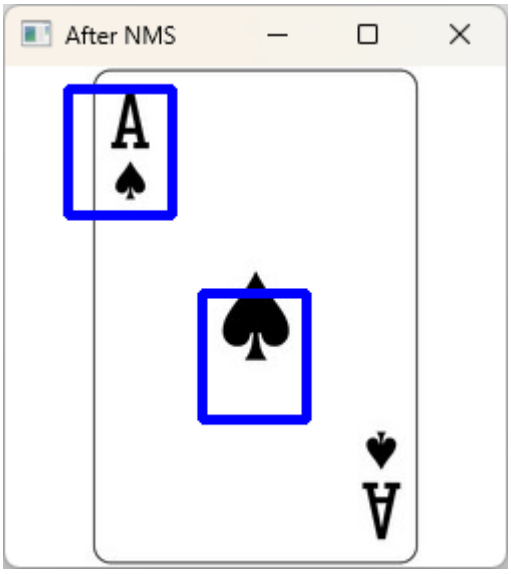
```
import cv2
import numpy as np
from imutils.object_detection import non_max_suppression
# Reading the image and the template
img = cv2.imread("X:/AI_V_lab/exp5/ace.jpg")
temp = cv2.imread("X:/AI_V_lab/exp5/ace_t.jpg")
# save the image dimensions
W, H = temp.shape[:2]
# Define a minimum threshold
thresh = 0.4
# Converting them to grayscale
img_gray = cv2.cvtColor(img,
                        cv2.COLOR_BGR2GRAY)
temp_gray = cv2.cvtColor(temp,
                         cv2.COLOR_BGR2GRAY)
# Passing the image to matchTemplate method
match = cv2.matchTemplate(
    image=img_gray, templ=temp_gray,
    method=cv2.TM_CCOEFF_NORMED)
# Select rectangles with
# confidence greater than threshold
(y_points, x_points) = np.where(match >= thresh)
# initialize our list of rectangles
boxes = list()
# loop over the starting (x, y)-coordinates again
for (x, y) in zip(x_points, y_points):
    # update our list of rectangles
    boxes.append((x, y, x + W, y + H))
# apply non-maxima suppression to the rectangles
# this will create a single bounding box
boxes = non_max_suppression(np.array(boxes))
# loop over the final bounding boxes
for (x1, y1, x2, y2) in boxes:
    # draw the bounding box on the image
```

```
    cv2.rectangle(img, (x1, y1), (x2, y2),
                  (255, 0, 0), 3)
# Show the template and the final output
cv2.imshow("Template", temp)
cv2.imshow("After NMS", img)
cv2.waitKey(0)
# destroy all the windows
# manually to be on the safe side
cv2.destroyAllWindows()
```

OUTPUT:

TEPPLETE:



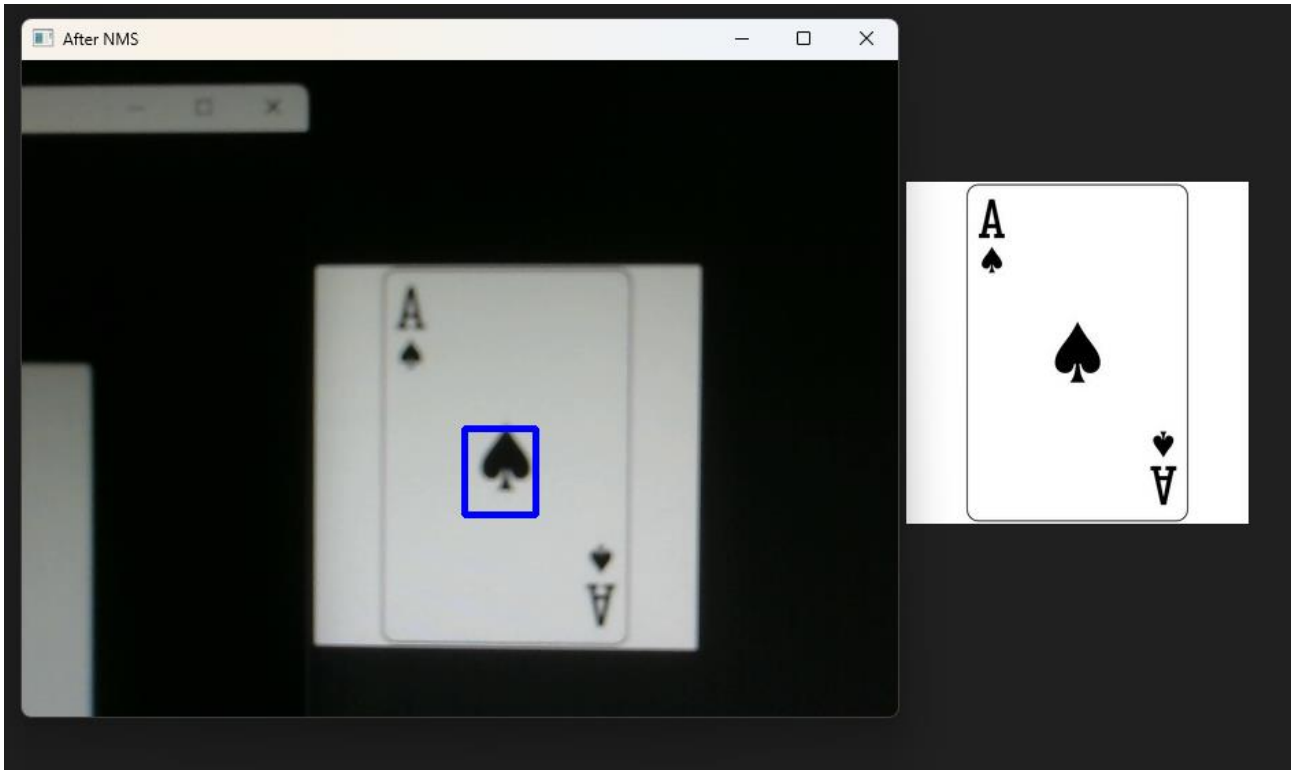


6] MULTI-TEMPLATE(VIDEO):

```
import cv2
import numpy as np
from imutils.object_detection import non_max_suppression
temp = cv2.imread("X:/AI_V_lab/exp5/ace_t.jpg")
cap = cv2.VideoCapture(0)
# save the template image dimensions
W, H = temp.shape[:2]
# Define a minimum threshold
thresh = 0.8
while cap.isOpened():
    # Take each frame
    ret, frame = cap.read()
    if not ret:
        break
    # Convert to HSV for simpler calculations
    img_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    temp_gray = cv2.cvtColor(temp, cv2.COLOR_BGR2HSV)
    # Passing the image to matchTemplate method
    match = cv2.matchTemplate(img_gray, temp_gray, cv2.TM_CCOEFF_NORMED)
    # Select rectangles with
    # confidence greater than threshold
    (y_points, x_points) = np.where(match >= thresh)
    # initialize our list of rectangles
```

```
    boxes = []
    # loop over the starting (x, y)-coordinates again
    for (x, y) in zip(x_points, y_points):
        # update our list of rectangles
        boxes.append((x, y, x + W, y + H))
    # apply non-maxima suppression to the rectangles
    # this will create a single bounding box
    boxes = non_max_suppression(np.array(boxes))
    # loop over the final bounding boxes
    for (x1, y1, x2, y2) in boxes:
        # draw the bounding box on the image
        cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 0, 0), 3)

    # Show the template and the final output
    cv2.imshow("Template", temp)
    cv2.imshow("After NMS", frame)

    # Break the loop if 'q' key is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
# release the frame
cap.release()
cv2.destroyAllWindows()
```

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 Edge Detection and Template Matching Techniques were learnt using OpenCV.

Post Lab Questions

1. What is the difference between convolution and correlation

2. 180 160 160 140 120
   110 110 120 140 120
   110 140 120 120 140
   120 160 160 170 170
   170 120 110 140 110

   For all the rows perform first order and second order derivative

3. Create a template and change the orientation of the template to different orientations and perform template matching for image of your choice

ANSWER:

1.

Convolution:

- In convolution, the kernel is flipped both horizontally and vertically before being applied to the image.
- The convolution operation is commutative, meaning that the order of the input image and the kernel does not affect the result.
- Convolution is often used in image processing for operations such as blurring, sharpening, and edge detection.

Correlation:

- In correlation, the kernel is not flipped before being applied to the image. It is directly used as it is.
- The correlation operation is commutative as well, but it is not equivalent to convolution.
- Correlation is commonly used in machine learning applications, especially in convolutional neural networks (CNNs) for tasks such as feature extraction.

2.

First derivative:

20 0 -20 -20 -20
0 10 10 20 -20
30 -10 0 20 -20
40 0 0 10 0
-50 -50 30 -30 -30

Second derivative:

20 -20 -40 0 -20
-20 10 0 30 -40

20 -50 40 -10 20
20 0 -10 10 0
-90 80 -70 0 10

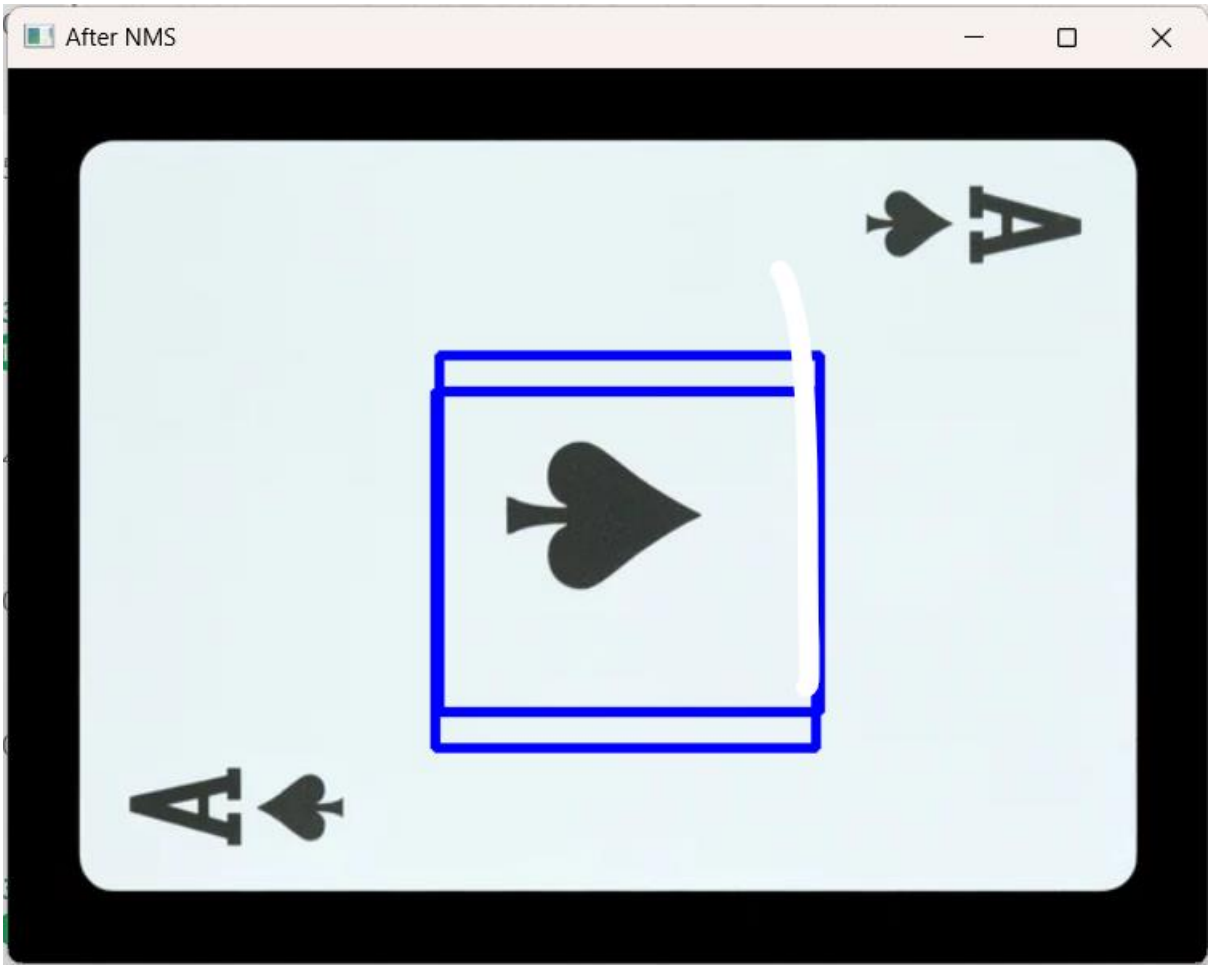
3.

```
import cv2
import numpy as np
from imutils.object_detection import non_max_suppression

# Function to rotate an image
def rotate_image(image, angle):
   (h, w) = image.shape[:2]
   center = (w // 2, h // 2)
   M = cv2.getRotationMatrix2D(center, angle, 1.0)
   rotated = cv2.warpAffine(image, M, (w, h))
   return rotated
# Reading the image
img = cv2.imread("img1.png")

# Create a template and change its orientation
template = cv2.imread("template.png")
orientations = [0, 90, 180, 270]  # Specify different orientations

for angle in orientations:
   rotated_template = rotate_image(template, angle)

   # Save the rotated template for reference or debugging
   cv2.imwrite(f"rotated_template_{angle}.png", rotated_template)

   # Save the image dimensions
   W, H = rotated_template.shape[:2]

   # Convert them to grayscale
   img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
   temp_gray = cv2.cvtColor(rotated_template, cv2.COLOR_BGR2GRAY)

   # Passing the image to matchTemplate method
   match = cv2.matchTemplate(
      image=img_gray, templ=temp_gray,
      method=cv2.TM_CCOEFF_NORMED)
```

```
    # Define a minimum threshold
    thresh = 0.4

    # Select rectangles with confidence greater than threshold
    (y_points, x_points) = np.where(match >= thresh)

    # Initialize the list of rectangles
    boxes = []

    # Loop over the starting (x, y)-coordinates
    for (x, y) in zip(x_points, y_points):
        # Update the list of rectangles
        boxes.append((x, y, x + W, y + H))

    # Apply non-maxima suppression to the rectangles
    # This will create a single bounding box
    boxes = non_max_suppression(np.array(boxes))

    # Loop over the final bounding boxes
    for (x1, y1, x2, y2) in boxes:
        # Draw the bounding box on the image
        cv2.rectangle(img, (x1, y1), (x2, y2), (255, 0, 0), 3)

# Show the template and the final output
cv2.imshow("Template", template)
cv2.imshow("After NMS", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:

TEMPLATE



EXPERIMENT 6

STEREO MATCHING AND POINT CLOUD RECONSTRUCTION

AIM:

To perform feature detection and matching, disparity map creation and point cloud reconstruction using OpenCV.

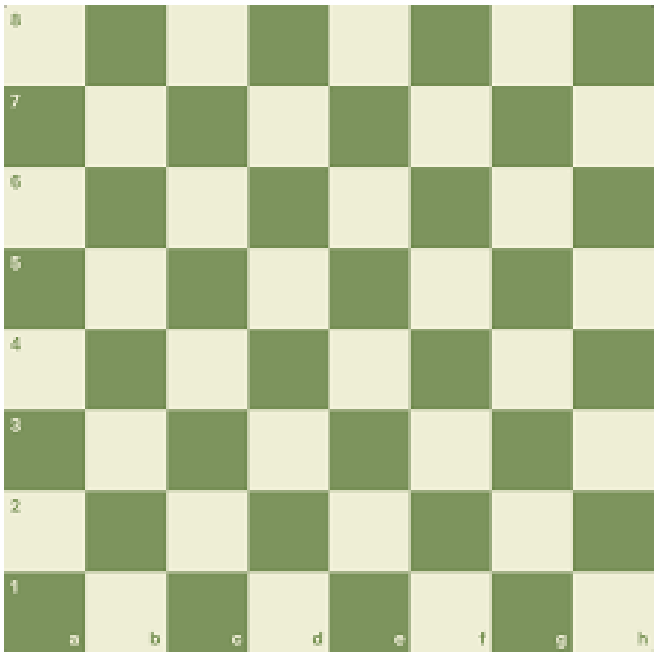
SOFTWARE/ PACKAGES USED:

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

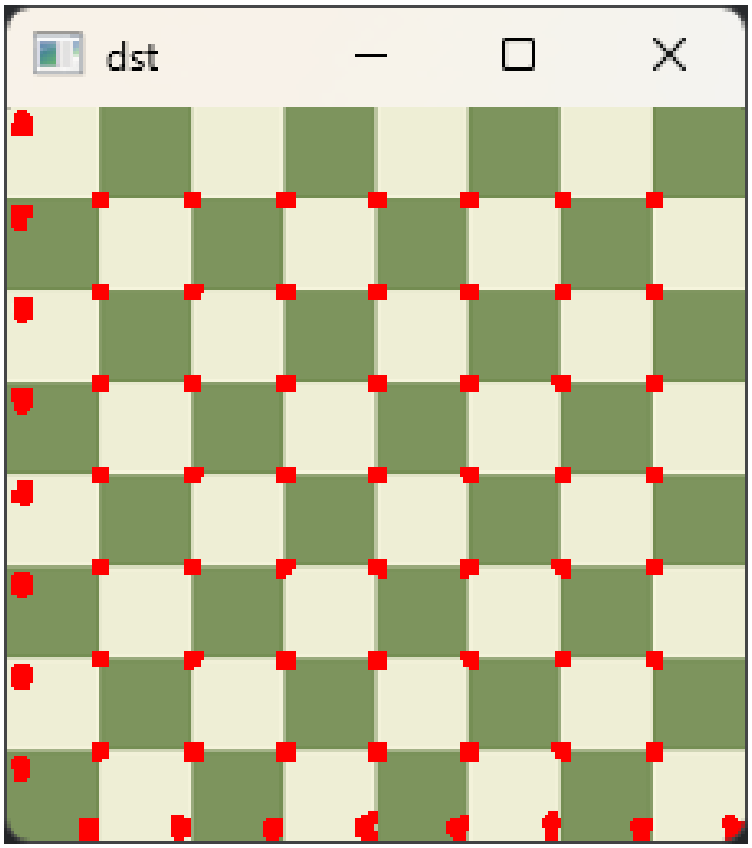
1] HARRIS CORNER:

```
import numpy as np
import cv2 as cv
img = cv.imread("X:/AI_V_lab/exp3/chess_board.png")
gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
gray = np.float32(gray)
dst = cv.cornerHarris(gray,2,3,0.04)
#result is dilated for marking the corners, not important
dst = cv.dilate(dst, None)
# Threshold for an optimal value, it may vary depending on the image.
img[dst>0.01*dst.max()]=[0,0,255]
cv.imshow('dst',img)
cv.waitKey(0)
cv.destroyAllWindows()
```

INPUT IMAGE:



OUTPUT:

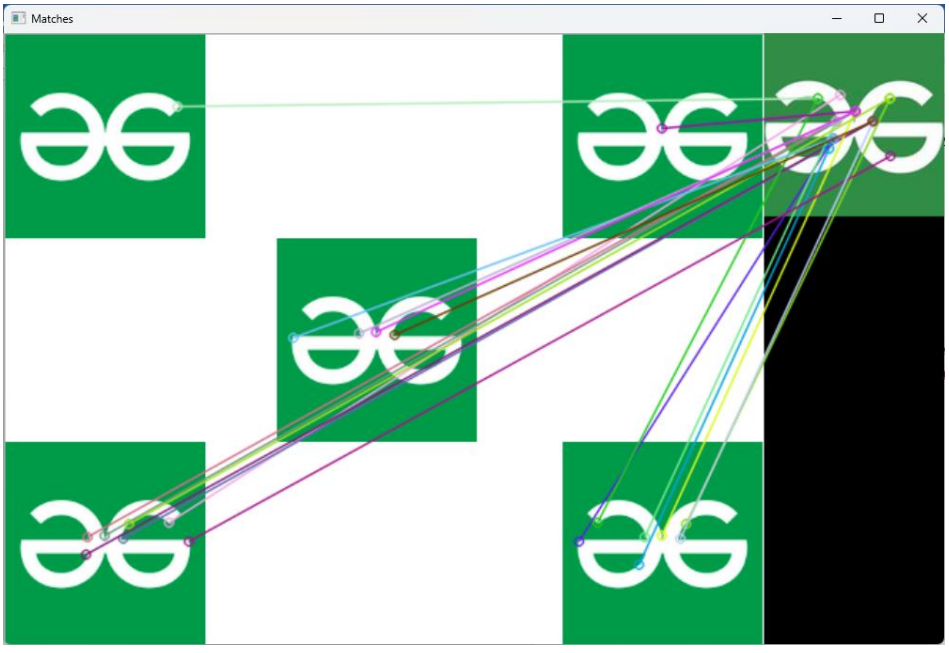


2] ORB:

```
import numpy as np
import cv2
# Read the query image as query_img
# and train image This query image
# is what you need to find in train image
# Save it in the same directory
# with the name image.jpg
query_img = cv2.imread('X:/AI_V_lab/exp5/img.png')
train_img = cv2.imread('X:/AI_V_lab/exp5/template.jpg')
# Convert it to grayscale
query_img_bw = cv2.cvtColor(query_img, cv2.COLOR_BGR2GRAY)
train_img_bw = cv2.cvtColor(train_img, cv2.COLOR_BGR2GRAY)
# Initialize the ORB detector algorithm
orb = cv2.ORB_create()
# Now detect the keypoints and compute
# the descriptors for the query image
# and train image
queryKeypoints, queryDescriptors = orb.detectAndCompute(query_img_bw, None)
trainKeypoints, trainDescriptors = orb.detectAndCompute(train_img_bw, None)
# Check if descriptors are empty
if queryDescriptors is None or trainDescriptors is None:
    print("No descriptors found.")
    exit()
# Convert descriptors to np.float32 if necessary
queryDescriptors = queryDescriptors.astype(np.float32)
trainDescriptors = trainDescriptors.astype(np.float32)
# Initialize the Matcher for matching
# the keypoints and then match the
# keypoints
```

```
matcher = cv2.BFMatcher()

matches = matcher.match(queryDescriptors, trainDescriptors)

# Extract keypoints locations

query_pts = np.float32([queryKeypoint.pt for queryKeypoint in queryKeypoints]).reshape(-1, 1, 2)

train_pts = np.float32([trainKeypoint.pt for trainKeypoint in trainKeypoints]).reshape(-1, 1, 2)

# draw the matches to the final image

# containing both the images the drawMatches()

# function takes both images and keypoints

# and outputs the matched query image with

# its train image

final_img = cv2.drawMatches(query_img, queryKeypoints,

                            train_img, trainKeypoints, matches[:20], None,

                            flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)

final_img = cv2.resize(final_img, (1000, 650))

# Show the final image

cv2.imshow("Matches", final_img)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

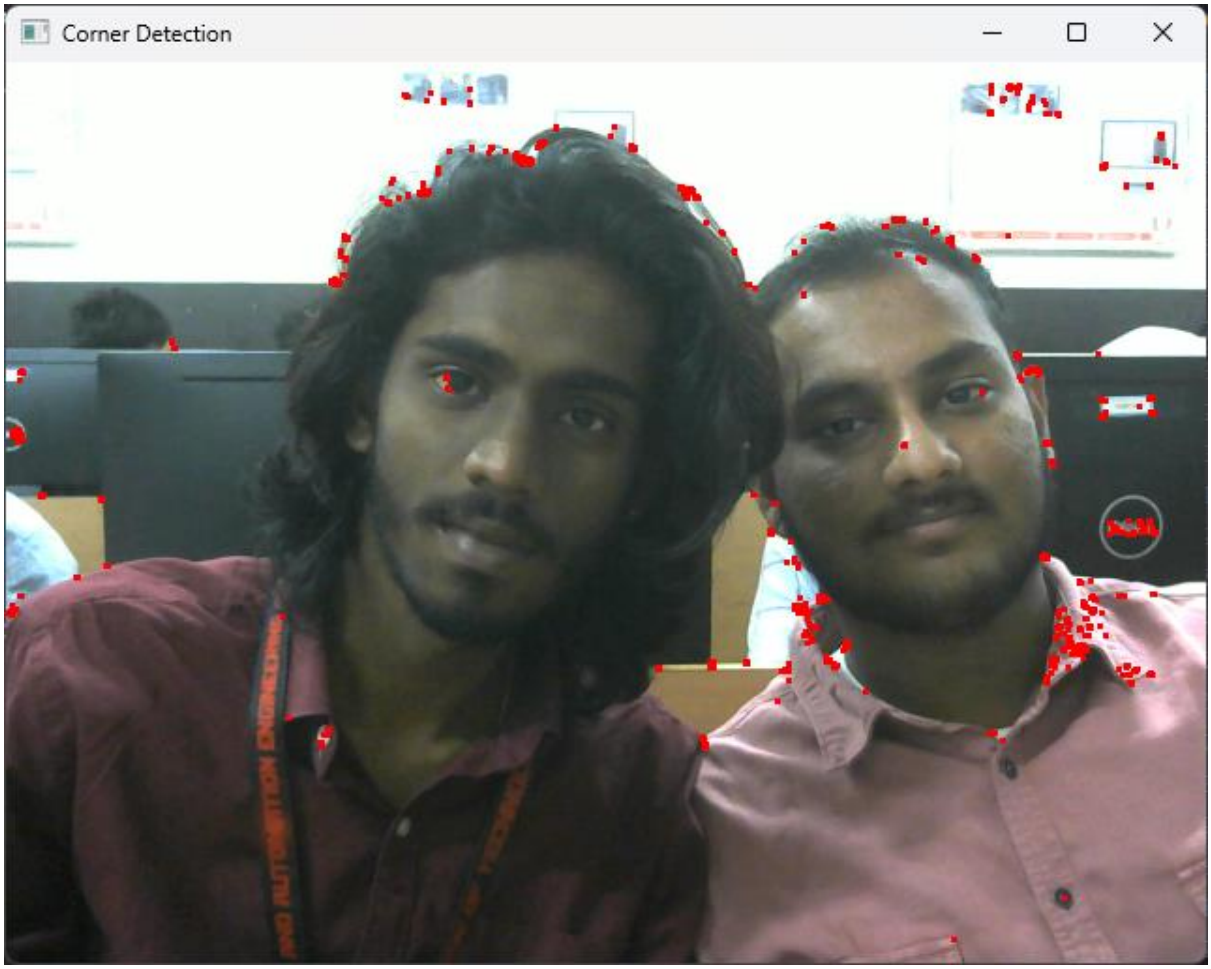
OUTPUT:



3] HARVID:

```
import numpy as np
import cv2 as cv
cap = cv.VideoCapture(0)
while True:
    ret, frame = cap.read()
    if not ret:
        print("Failed to capture frame")
        break
    gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    gray = np.float32(gray)
    dst = cv.cornerHarris(gray, 2, 3, 0.04)
    # Dilate the corner points to make them more visible
    dst = cv.dilate(dst, None)
    # Mark the corners on the original frame
    frame[dst > 0.01 * dst.max()] = [0, 0, 255]
    # Display the resulting frame
    cv.imshow('Corner Detection', frame)
    # Exit if 'q' is pressed
    if cv.waitKey(1) & 0xFF == ord('q'):
        break
# Release the capture
cap.release()
cv.destroyAllWindows()
```

OUTPUT:



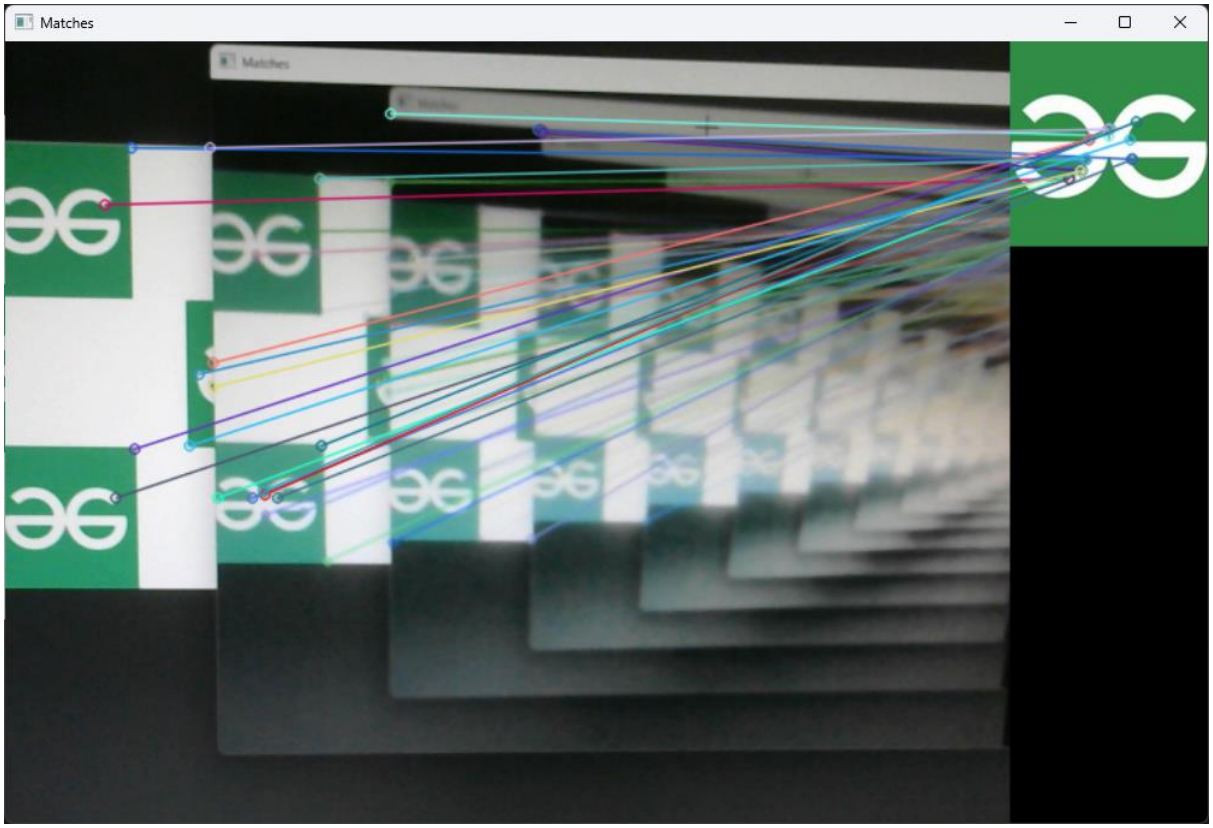
4] ORDVID:

```
import numpy as np
import cv2
cap = cv2.VideoCapture(0)
while True:
    ret, query_img = cap.read()
    if not ret:
        print("Failed to capture frame")
        break
    train_img = cv2.imread('X:/AI_V_lab/exp5/template.jpg')
    if train_img is None:
        print("Failed to load template image")
```

```
        break
    # Convert images to grayscale
    query_img_bw = cv2.cvtColor(query_img, cv2.COLOR_BGR2GRAY)
    train_img_bw = cv2.cvtColor(train_img, cv2.COLOR_BGR2GRAY)
    orb = cv2.ORB_create()
    queryKeypoints, queryDescriptors = orb.detectAndCompute(query_img_bw, None)
    trainKeypoints, trainDescriptors = orb.detectAndCompute(train_img_bw, None)
    if queryDescriptors is None or trainDescriptors is None:
        print("No descriptors found.")
        exit()
    queryDescriptors = queryDescriptors.astype(np.float32)
    trainDescriptors = trainDescriptors.astype(np.float32)
    matcher = cv2.BFMatcher()
    matches = matcher.match(queryDescriptors, trainDescriptors)
    # Extract keypoints locations
    query_pts = np.float32([queryKeypoint.pt for queryKeypoint in
queryKeypoints]).reshape(-1, 1, 2)
    train_pts = np.float32([trainKeypoint.pt for trainKeypoint in trainKeypoints]).reshape(-1,
1, 2)
    # Draw matches
    final_img = cv2.drawMatches(query_img, queryKeypoints,
                                train_img, trainKeypoints, matches[:20], None,
                                flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
    final_img = cv2.resize(final_img, (1000, 650))
    # Show the final image
    cv2.imshow("Matches", final_img)
    # Break the loop if 'q' is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
# Release the video capture and close all windows
cap.release()
```

cv2.destroyAllWindows()

OUTPUT:

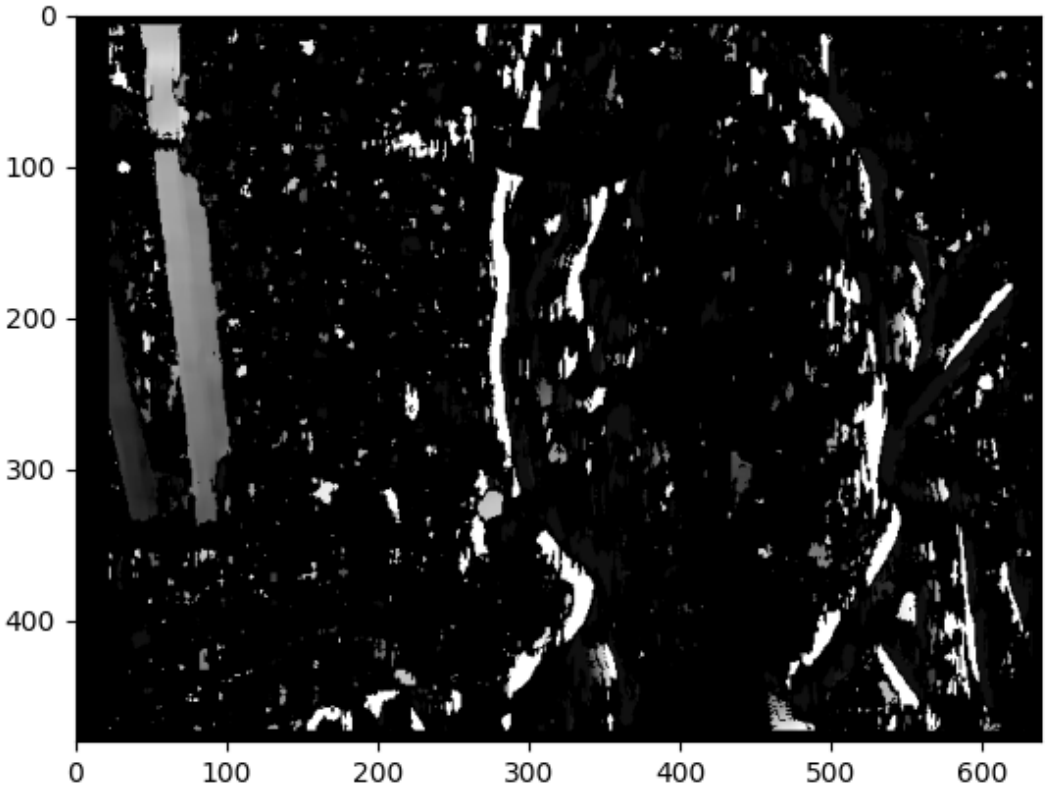


5] DISPARITY:

```
import cv2 as cv
from matplotlib import pyplot as plt
cap0 = cv.VideoCapture(0, cv.CAP_DSHOW)
cap1 = cv.VideoCapture(1, cv.CAP_DSHOW)
while True:
    ret1, frame0 = cap0.read()
    ret2, frame1 = cap1.read()
    if not ret1 or not ret2:
        print("Failed to capture frame")
        break
    # Convert frames to grayscale
    frame0_gray = cv.cvtColor(frame0, cv.COLOR_BGR2GRAY)
    frame1_gray = cv.cvtColor(frame1, cv.COLOR_BGR2GRAY)
```

```
    # Creates StereoBM object
    stereo = cv.StereoBM_create(numDisparities=16, blockSize=15)
    # Computes disparity
    disparity = stereo.compute(frame0_gray, frame1_gray)
    # Displays image as grayscale and plotted
    plt.imshow(disparity, 'gray')
    plt.show()
    # Break the loop if 'q' is pressed
    if cv.waitKey(1) & 0xFF == ord('q'):
        break
# Release the video capture and close all windows
cap0.release()
cap1.release()
cv.destroyAllWindows()s
```

OUTPUT:



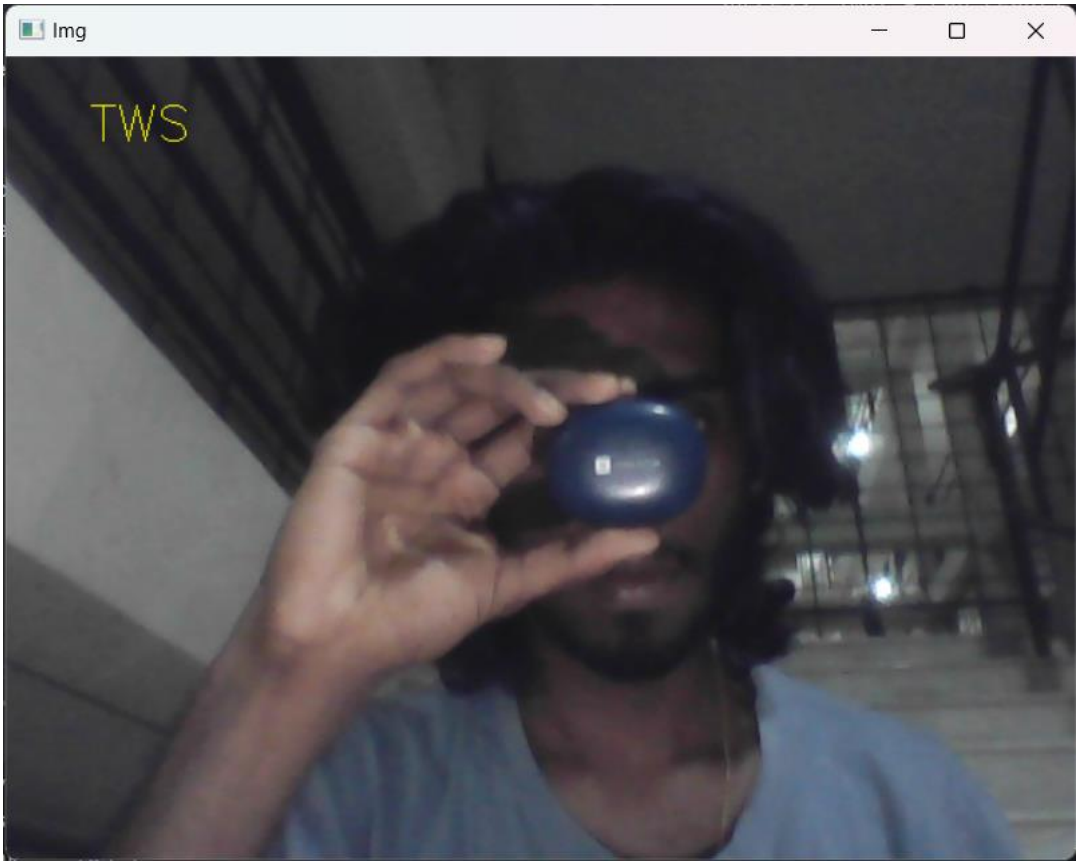
6] IMAGE CLASSIFIER:

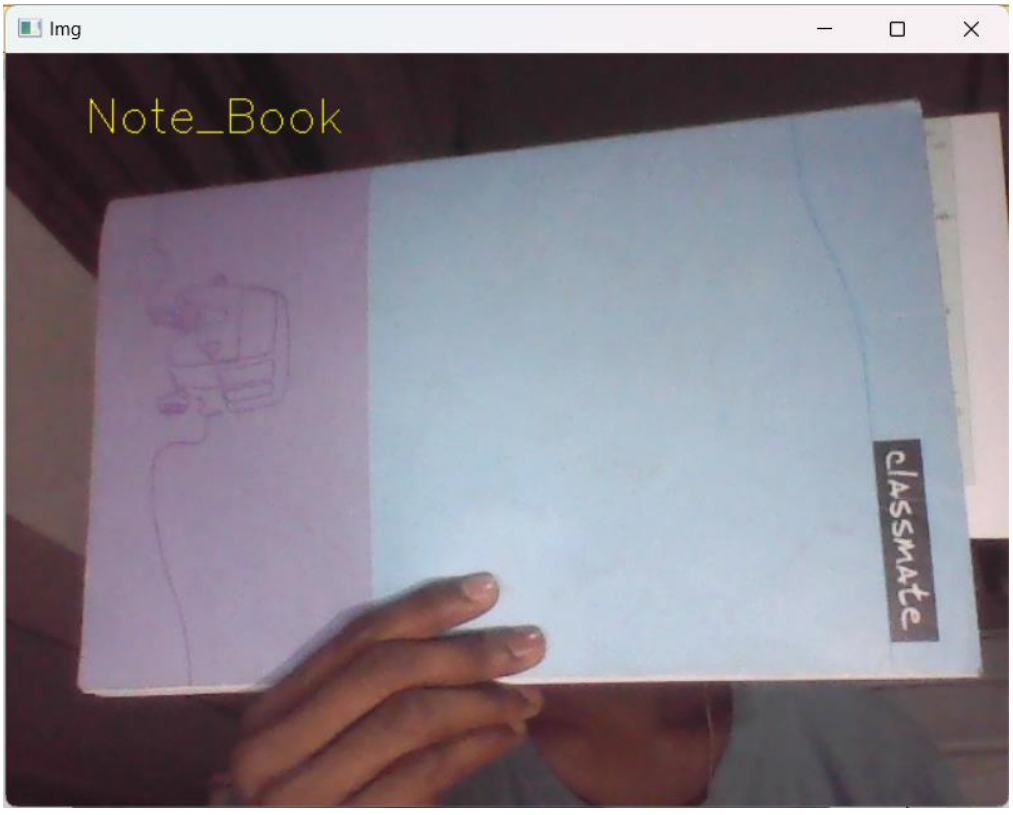
```
import cv2 as cv
import numpy as np
import os

#pre-trained images
path = "C:/Users/yuvan/Desktop/im_qry"
orb= cv.ORB_create(nfeatures=1000)
images=[]
classNames=[]
ml= os.listdir(path)
print("Total classes detected: ", len(ml))

for cl in ml:
    img= cv.imread(f'{path}/{cl}',0)

    images.append(img)
    classNames.append(os.path.splitext(cl)[0])
print(classNames)
#descriptors
def fd(images):
    desList=[]
    for img in images:
        kp,des = orb.detectAndCompute(img,None)
        desList.append(des)
    return desList
def fin(img,desList, thres=4):
    kp2,des2 = orb.detectAndCompute(img,None)
    bf = cv.BFMatcher()
    matchList=[]
    finval= -1
    try:
        for des in desList:
            matches = bf.knnMatch(des,des2,k=2)
            good=[]
            for m,n in matches:
                if m.distance<0.75 * n.distance:
                    good.append([m])
            matchList.append(len(good))
    except:
        pass
    if len(matchList)!=0:
        if max(matchList)>thres:
            finval = matchList.index(max(matchList))
    return finval
desList = fd(images)
print(len(desList))
cap = cv.VideoCapture(0)
while True:
```

```
    success, img2 = cap.read()
    imgo= img2.copy()
    img2= cv.cvtColor(img2, cv.COLOR_BGR2GRAY)
    ia= fin(img2,desList)
    if ia!=-1:

cv.putText(imgo,classNames[ia],(50,50),cv.FONT_HERSHEY_SIMPLEX,1,(0,255,255),1)

    cv. imshow('Img',imgo)
    cv.waitKey(1)
```

INPUT:

OUTPUT:





7] DISPARITY: (GENERATE POINT CLOUD DATA)

```
import numpy as np
import cv2 as cv
ply_header = '''ply
format ascii 1.0
element vertex %(vert_num)d
property float x
property float y
property float z
property uchar red
property uchar green
property uchar blue
end_header'''
def write_ply(fn, verts, colors):
verts = verts.reshape(-1, 3)
colors = colors.reshape(-1, 3)
verts = np.hstack([verts, colors])
with open(fn, 'wb') as f:
f.write((ply_header % dict(vert_num=len(verts))).encode('utf-8'))
np.savetxt(f, verts, fmt='%f %f %f %d %d %d ')
def main():
print('loading images...')

#imgL = cv.pyrDown(cv.imread(cv.samples.findFile('aloeL.jpg'))) # downscale images for faster
processing

#imgR = cv.pyrDown(cv.imread(cv.samples.findFile('aloeR.jpg')))
frameWidth = 700
frameHeight = 524
imgL = cv.imread("left.png",0)
imgR = cv.imread("right.png",0)
imgL = cv.resize(imgL, (frameWidth, frameHeight))
imgR = cv.resize(imgR, (frameWidth, frameHeight))
# disparity range is tuned for 'aloe' image pair
window_size = 3
min_disp = 16
num_disp = 112-min_disp
stereo = cv.StereoSGBM_create(minDisparity = min_disp,
numDisparities = num_disp,
blockSize = 16,
P1 = 8*3*window_size**2,
P2 = 32*3*window_size**2,
disp12MaxDiff = 1,
uniquenessRatio = 10,
speckleWindowSize = 100,
speckleRange = 32
)
print('computing disparity...')
disp = stereo.compute(imgL, imgR).astype(np.float32) / 16.0
print('generating 3d point cloud...',)
h, w = imgL.shape[:2]
f = 0.8*w # guess for focal length
Q = np.float32([[1, 0, 0, -0.5*w],
[0,-1, 0, 0.5*h], # turn points 180 deg around x-axis,
```

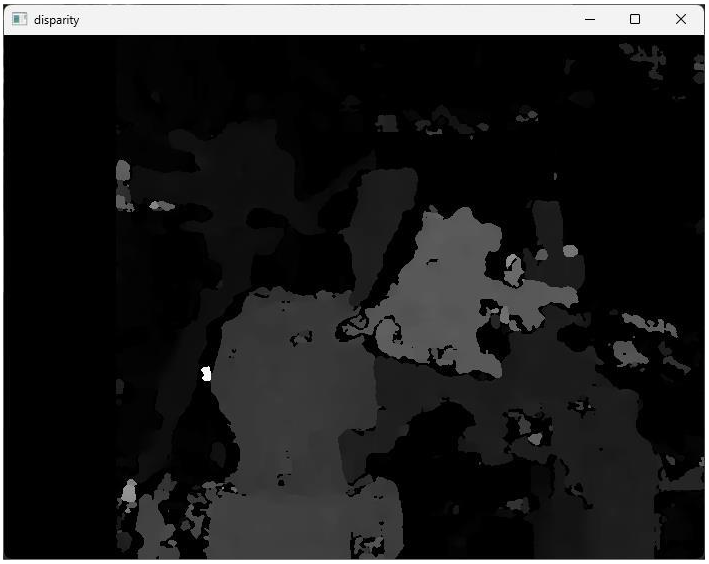
```
[0, 0, 0, -f], # so that y-axis looks up
[0, 0, 1, 0]])
points = cv.reprojectImageTo3D(disp, Q)
colors = cv.cvtColor(imgL, cv.COLOR_BGR2RGB)
mask = disp > disp.min()
out_points = points[mask]
out_colors = colors[mask]
out_fn = 'out.ply'
write_ply(out_fn, out_points, out_colors)
print('%s saved' % out_fn)
cv.imshow('Left', imgL)
cv.imshow('Right', imgR)
cv.imshow('disparity', (disp-min_disp)/num_disp)
cv.waitKey()
print('Done')
if __name__ == '__main__':
print(__doc__)
main()
cv.destroyAllWindows()
```

INPUT:

LEFT:

RIGHT:

OUTPUT:

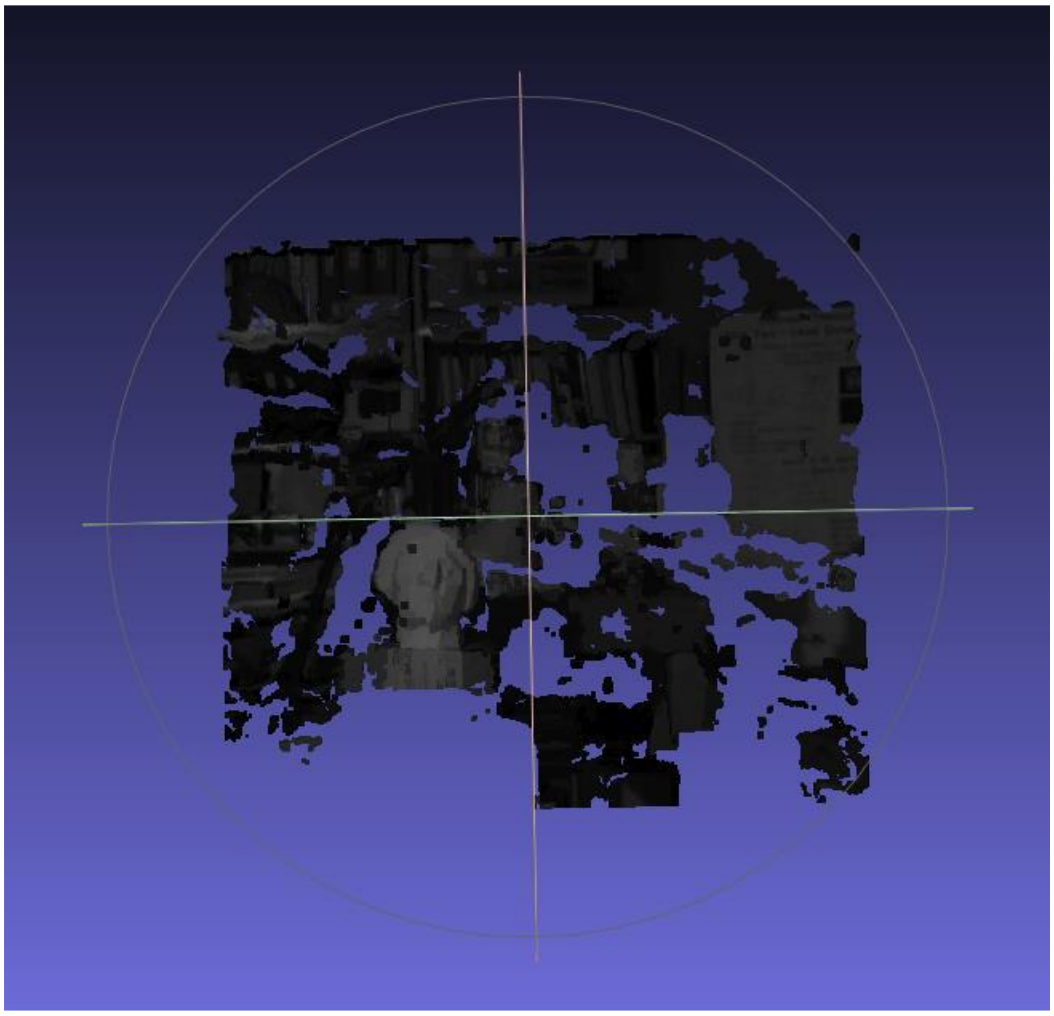


Points generated:

-13.430657 15.299270 -32.700729 164 164 164
-13.372263 15.299270 -32.700729 164 164 164
-13.313869 15.299270 -32.700729 164 164 164
-13.255474 15.299270 -32.700729 164 164 164
-13.197081 15.299270 -32.700729 164 164 164
-13.138686 15.299270 -32.700729 164 164 164
-13.080292 15.299270 -32.700729 164 164 164

-12.974545 15.243636 -32.581818 164 164 164
-12.916364 15.243636 -32.581818 164 164 164
-12.858182 15.243636 -32.581818 164 164 164
-12.846715 15.299270 -32.700729 164 164 164
-12.788321 15.299270 -32.700729 164 164 164
.
.
.

DISPARITY IMAGE (MESH LAB):



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 Feature Detection and Matching, Disparity Map Creation and Point Cloud Reconstruction were done using OpenCV.

POST LAB QUESTIONS:

1. **What is Correspondence problem? Which approach is used for finding corresponding pixels?**
2. **How 2D image is converted to 3D?**
3. **How ORB differs from FAST?**
4. **Compare ORB, Sift and Surf.**

ANSWERS:

1. The correspondence problem in computer vision refers to the challenge of matching points or features in one image with their corresponding points or features in another image. This is crucial for tasks like image stitching, object recognition, and 3D reconstruction. One common approach for finding corresponding pixels is using feature-based methods, which detect distinctive points or features in both images and then match them based on certain criteria such as similarity in appearance or spatial proximity.

2. Converting a 2D image to 3D typically involves depth estimation or reconstruction techniques. One common method is stereo vision, where multiple 2D images of a scene are taken from different viewpoints (using multiple cameras or a single moving camera) and then depth information is inferred by triangulating corresponding points in these images. Another approach is using structured light or depth sensors like LiDAR to directly measure depth information from the scene.

3. ORB (Oriented FAST and Rotated BRIEF) and FAST (Features from Accelerated Segment Test) are both feature detection and description algorithms used in computer vision. However, ORB improves upon FAST by adding orientation estimation and using a binary descriptor (BRIEF) for feature description. This makes ORB more robust to rotation and scale changes compared to FAST.

4. ORB, SIFT (Scale-Invariant Feature Transform), and SURF (Speeded Up Robust Features) are all feature detection and description algorithms commonly used in computer vision:
   ORB: It combines the FAST keypoint detector with the BRIEF descriptor. ORB is fast and efficient, making it suitable for real-time applications. However, it may not be as robust to viewpoint changes or occlusion compared to SIFT and SURF.
   SIFT: SIFT detects keypoints at multiple scales and orientations using Difference of Gaussians (DoG) and then describes these keypoints using histograms of gradient orientations in local image patches. SIFT is known for its robustness to various transformations like scaling, rotation, and affine distortion, but it can be computationally expensive.
   SURF: SURF also detects keypoints at multiple scales but uses box filters and integral images for speed. It describes keypoints using Haar wavelet responses in local image patches. SURF is faster than SIFT and still fairly robust to scale and rotation changes, but it may not perform as well under severe viewpoint changes or image noise compared to SIFT.

Experiment 7

Camera Calibration and Pose Estimation using Monocular Camera

Aim:

To calibrate a Monocular Camera and estimate the pose using OpenCV.

Software/ Packages Used:

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

Programs:

Camera Calibration

Code:

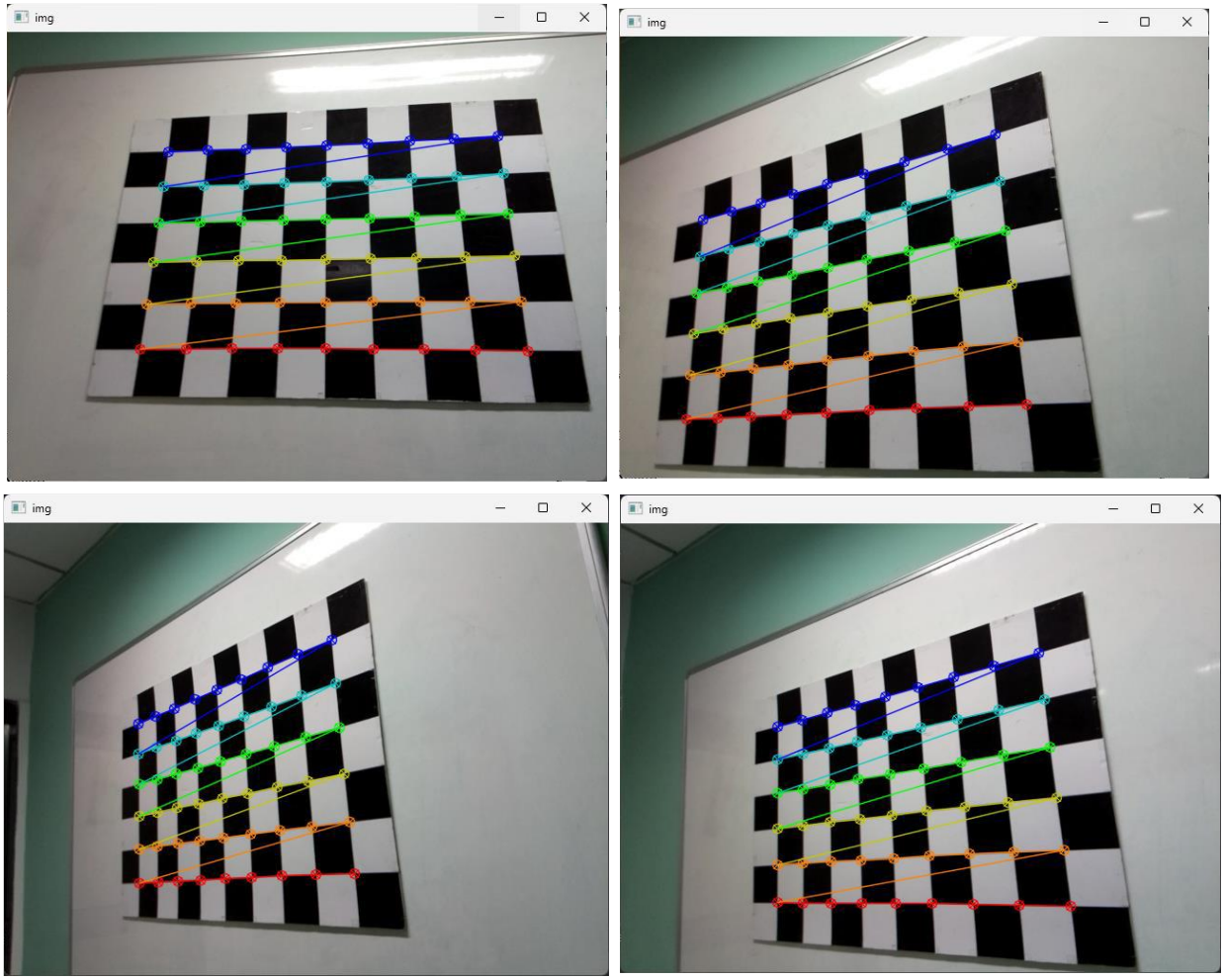
```
import numpy as np
import cv2 as cv
import glob

frameSize = (816,459)
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)

# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(5,5,0)
objp = np.zeros((7*5,3), np.float32)
objp[:,:2] = np.mgrid[0:7,0:5].T.reshape(-1,2)

# Arrays to store object points and image points from all the images.
objpoints = [] # 3d point in real world space
imgpoints = [] # 2d points in image plane.
images = glob.glob(r'C:\Python files\clg\Experiment7\Experiment7\img_cal\20240226_065*.jpg')

for image in images:
    img = cv.imread(image)
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    ret, corners = cv.findChessboardCorners(gray, (7,5), None)
    # If found, add object points, image points (after refining them)
    if ret == True:
        objpoints.append(objp)
        corners2 = cv.cornerSubPix(gray, corners, (11,11), (-1,-1), criteria)
        imgpoints.append(corners)
```

```
        # Draw and display the corners
        cv.drawChessboardCorners(img, (7,5), corners2, ret)
        cv.imshow('img', img)
        cv.waitKey(100)

cv.destroyAllWindows()

ret, mtx, dist, rvecs, tvecs = cv.calibrateCamera(objpoints, imgpoints, frameSize, None, None)
np.savez('B.npz', mtx=mtx, dist=dist, rvecs=rvecs, tvecs=tvecs)
print('Calibration Completed')

img= cv.imread(r'C:\Python files\clg\Experiment7\Experiment7\img_cal\calib1.jpg')
cv.imshow('img', img)
h, w = img.shape[:2]
newCameraMatrix, roi = cv.getOptimalNewCameraMatrix(mtx, dist, (w,h), 1, (w,h))
# Undistort
dst = cv.undistort(img, mtx, dist, None, newCameraMatrix)
# crop the image
x, y, w, h = roi
dst = dst[y:y+h, x:x+w]
cv.imwrite('caliResult3.jpg', dst)
mapx, mapy = cv.initUndistortRectifyMap(mtx, dist, None, newCameraMatrix, (w,h), 5)
dst = cv.remap(img, mapx, mapy, cv.INTER_LINEAR)
# crop the image
x, y, w, h = roi
dst = dst[y:y+h, x:x+w]
cv.imwrite('caliResult4.jpg', dst)

# Reprojection Error
mean_error = 0
for i in range(len(objpoints)):
    imgpoints2, _ = cv.projectPoints(objpoints[i], rvecs[i], tvecs[i], mtx, dist)
    error = cv.norm(imgpoints[i], imgpoints2, cv.NORM_L2)/len(imgpoints2)
    mean_error += error

print("total error: {}".format(mean_error/len(objpoints)))
np.savez('B.npz', mtx=mtx, dist=dist, rvecs=rvecs, tvecs=tvecs)

with np.load('B.npz') as file:
    mtx, dist, rvecs, tvecs = [file[i] for i in ('mtx','dist','rvecs','tvecs')]

def draw(img, corners, imgpts):
    corner = tuple(corners[0].ravel())
    img = cv.line(img, corner, tuple(imgpts[0].ravel()), (255,0,0), 10)
```

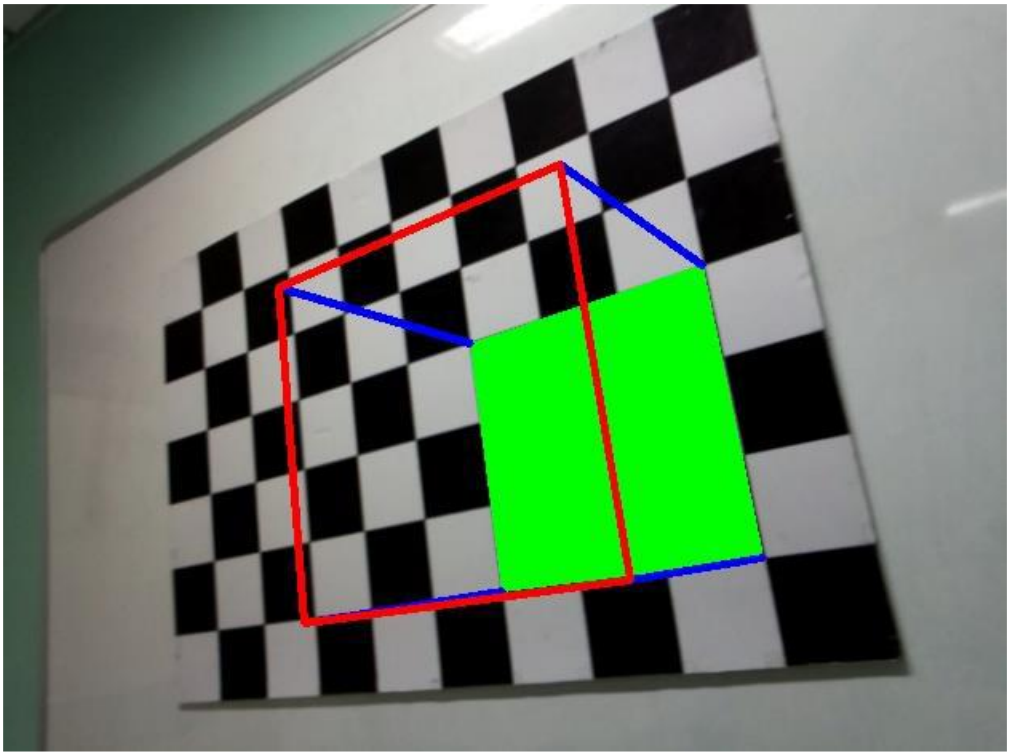
```
    img = cv.line(img, corner, tuple(imgpts[1].ravel()), (0,255,0), 10)
    img = cv.line(img, corner, tuple(imgpts[2].ravel()), (0,0,255), 10)
    return img

def drawBoxes(img, corners, imgpts):
    imgpts = np.int32(imgpts).reshape(-1,2)
    # draw ground floor in green
    img = cv.drawContours(img, [imgpts[:4]],-1,(0,255,0),-3)
    # draw pillars in blue color
    for i,j in zip(range(4),range(4,8)):
        img = cv.line(img, tuple(imgpts[i]), tuple(imgpts[j]),(255),3)
    # draw top layer in red color
    img = cv.drawContours(img, [imgpts[4:]],-1,(0,0,255),3)
    return img

criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)
objp = np.zeros((7*5,3), np.float32)
objp[:,:2] = np.mgrid[0:7,0:5].T.reshape(-1,2)
axis = np.float32([[3,0,0], [0,3,0], [0,0,-3]]).reshape(-1,3)
axisBoxes = np.float32([[0,0,0], [0,3,0], [3,3,0], [3,0,0], [0,0,-3],[0,3,-3],[3,3,-3],[3,0,-3] ])

for image in glob.glob('C:\Python files\clg\Experiment7\Experiment7\caliResult*.jpg'):
    img = cv.imread(image)
    gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
    ret, corners = cv.findChessboardCorners(gray, (7,5), None)
    if ret == True:
        print('Pose Estimation Started')
        corners2 = cv.cornerSubPix(gray,corners,(11,11),(-1,-1), criteria)
        ret, rvecs, tvecs = cv.solvePnP(objp, corners2, mtx, dist)
        imgpts, jac = cv.projectPoints(axisBoxes, rvecs, tvecs, mtx, dist)
        img = drawBoxes(img,corners2,imgpts)
        cv.imshow('img',img)
        cv.imwrite('pose12.jpg', img)
        cv.waitKey(0)

cv.destroyAllWindows()
```

INPUT 1:

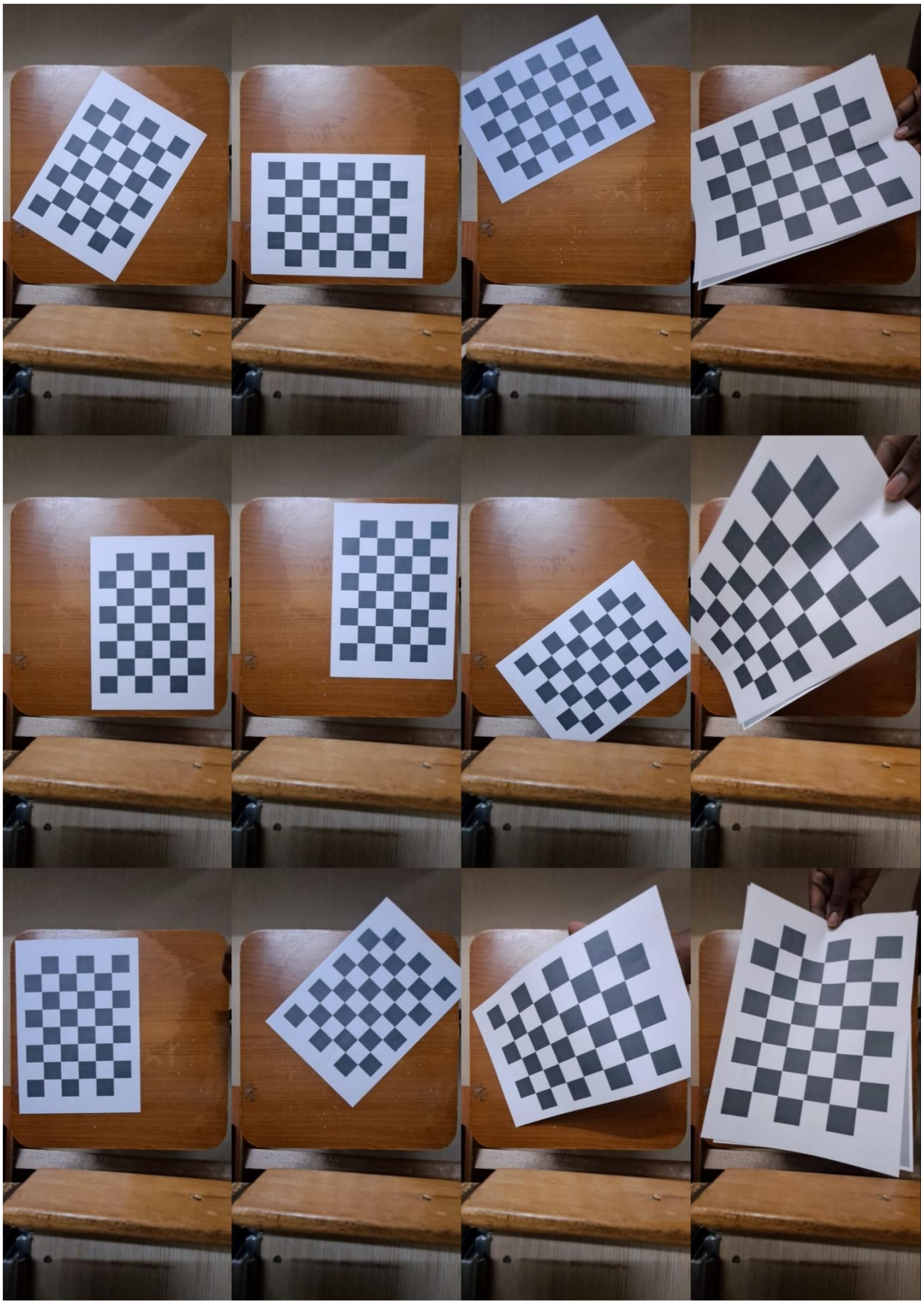
OUTPUT 1:



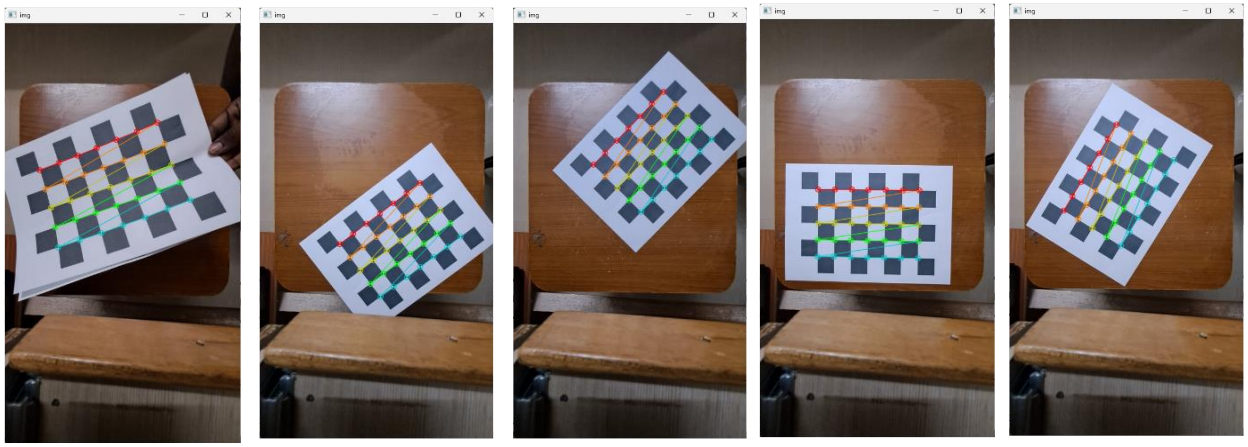
POST ESTIMATION:



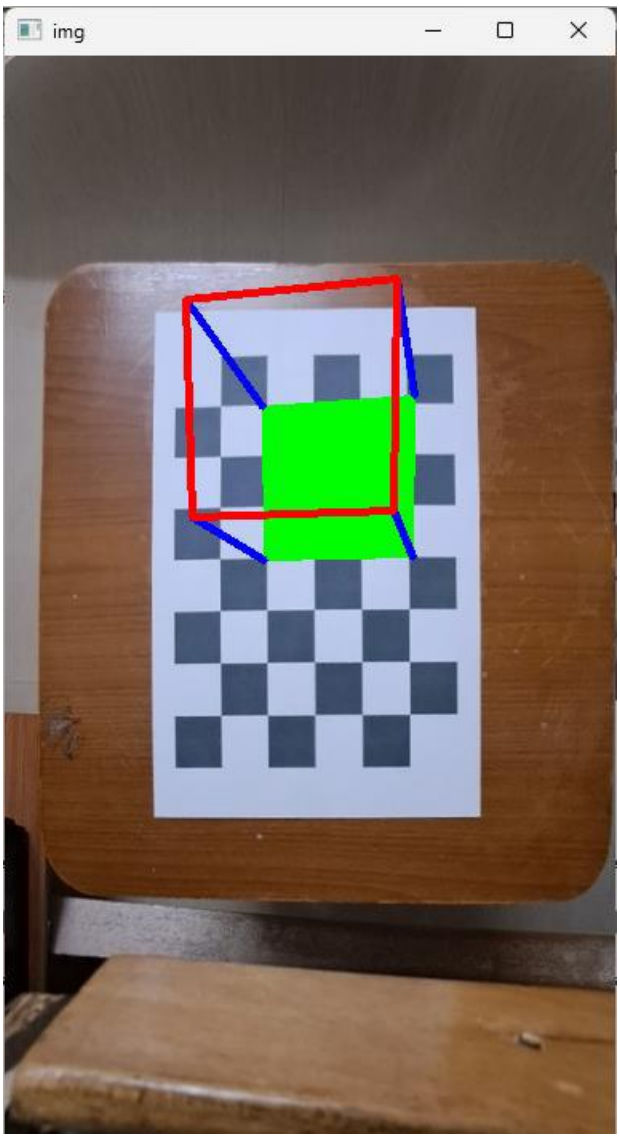
INPUT 2:



OUTPUT 2:



POST ESTIMATION:



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 calibration of a monocular camera and pose estimation has been completed using OpenCV.

POST LAB QUESTIONS

1. What are the challenges in camera calibration?

Challenges in camera calibration include lens distortion, image noise, accurate feature detection, and proper handling of calibration patterns.

2. What is the function inbuilt function for camera calibration? Explain each argument.

In OpenCV, the inbuilt function for camera calibration is cv.calibrateCamera(). It takes object points, image points, camera matrix, and distortion coefficients as arguments. Object points represent 3D points in the world, image points are corresponding 2D points in the image, camera matrix is intrinsic camera parameters, and distortion coefficients model lens distortion.

3. What is the difference between disparity and depth? Explain how is disparity is related to pose estimation.

Disparity is the difference in pixel coordinates between the same 3D point in stereo images, while depth is the actual distance of that point from the camera. Disparity is related to pose estimation as it provides information about the relative positions of stereo cameras, aiding in determining 3D structure.

4. What do you mean by the following code?
"criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)"

The code sets termination criteria for optimization algorithms in OpenCV, specifying both maximum iterations (30) and the desired accuracy (0.001) for convergence. It is commonly used in functions like cv.findChessboardCorners() or cv.solvePnP() during camera calibration

Experiment 8

Face and Object Detection

Aim:

To perform Face and Object Detection using Haar Cascade and Object Detection using YOLO V5 Deep Learning Library.

Software/ Packages Used:

1. Google Colaboratory
2. Libraries used:
    - Opencv – python
    - Numpy
    - Matplotlib
    - tensorflow

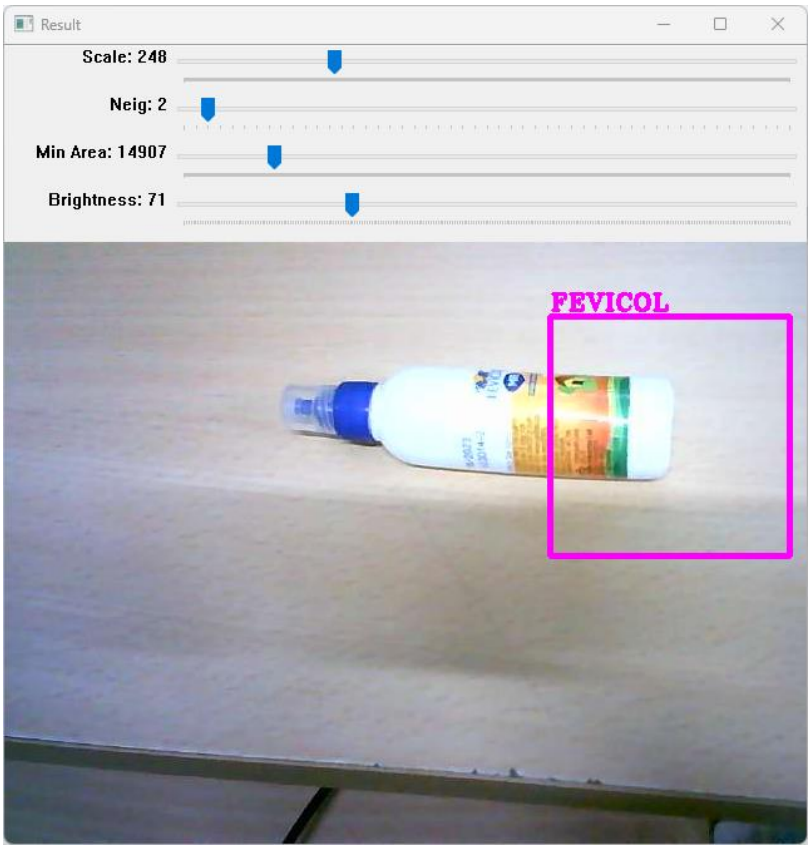
Programs:

1] Haar Cascade Based Face Detection:

POSITIVE IMAGES:

NEGATIVE IMAGES:

OUTPUT:



2] YOLO v5 Based Object Detection: (both for pretrained and custom data- (i/p -Video, Image, Live Video))

CODE:

Setup:

```
!git clone https://github.com/ultralytics/yolov5  # clone
%cd yolov5
%pip install -qr requirements.txt comet_ml  # install

import torch
import utils
display = utils.notebook_init()  # checks

from google.colab import drive
drive.mount('/content/drive')
```

Detect:

```
!python detect.py --source "/content/drive/MyDrive/datasets/chicken video.mp4"
!python detect.py --weights yolov5s.pt --img 640 --conf 0.25 --source data/images
# display.Image(filename='runs/detect/exp/zidane.jpg', width=600)
```

Validate:

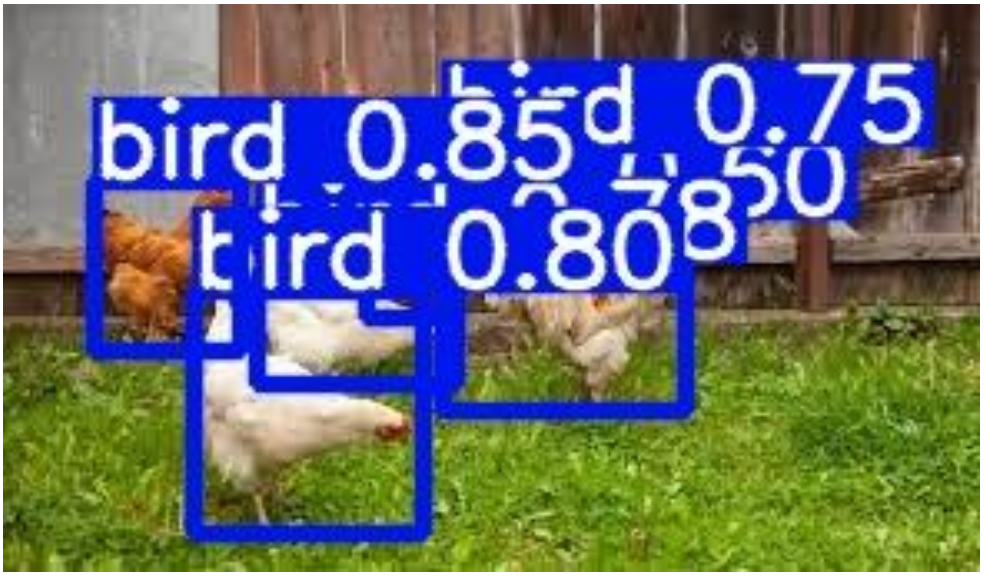
```
# Download COCO val
torch.hub.download_url_to_file('https://ultralytics.com/assets/coco2017val.zip', 'tmp.zip')
# download (780M - 5000 images)
!unzip -q tmp.zip -d ../datasets && rm tmp.zip # unzip
# Validate YOLOv5s on COCO val
!python val.py --weights yolov5s.pt --data coco.yaml --img 640 --half
```

Train:

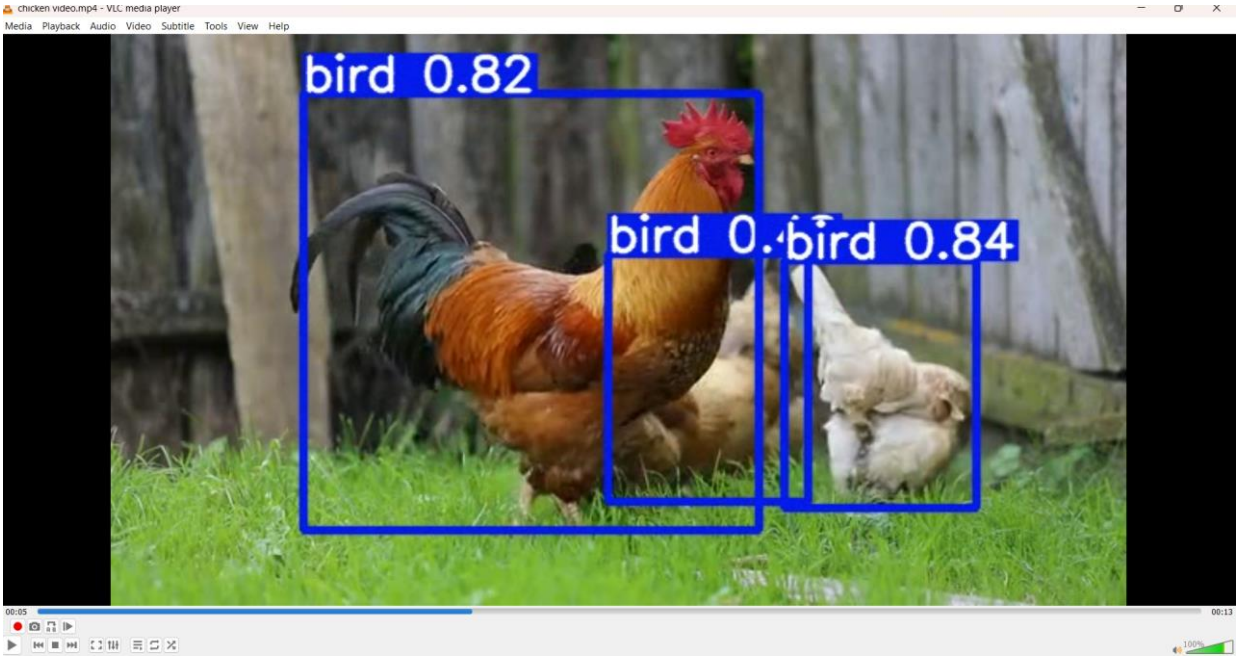
```
#@title Select YOLOv5
logger = 'Comet' #@param ['Comet', 'ClearML', 'TensorBoard']

if logger == 'Comet':
%pip install -q comet_ml
import comet_ml; comet_ml.init()
elif logger == 'ClearML':
%pip install -q clearml
import clearml; clearml.browser_login()
elif logger == 'TensorBoard':
%load_ext tensorboard
%tensorboard --logdir runs/train
# Train YOLOv5s on COCO128 for 3 epochs
!python train.py --img 640 --batch 16 --epochs 3 --data coco128.yaml --weights yolov5s.pt
        --cache
```

OUTPUT:

IMAGE:

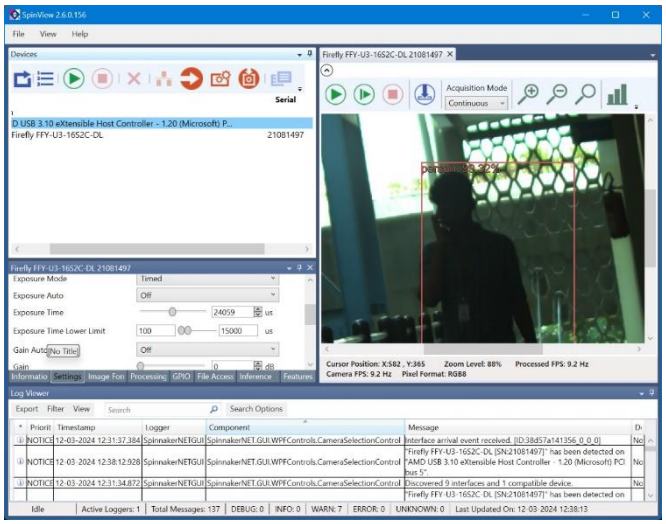


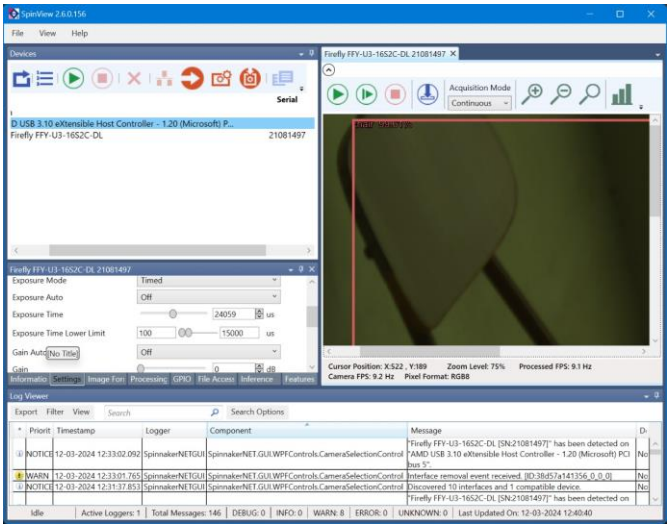
VIDEO:

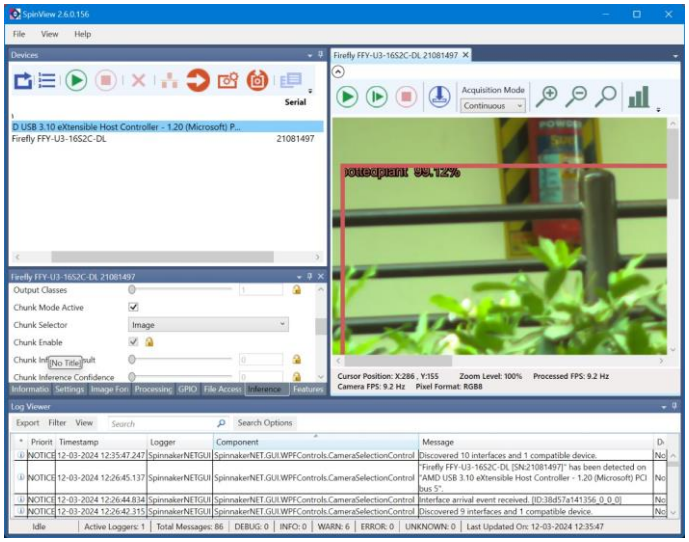


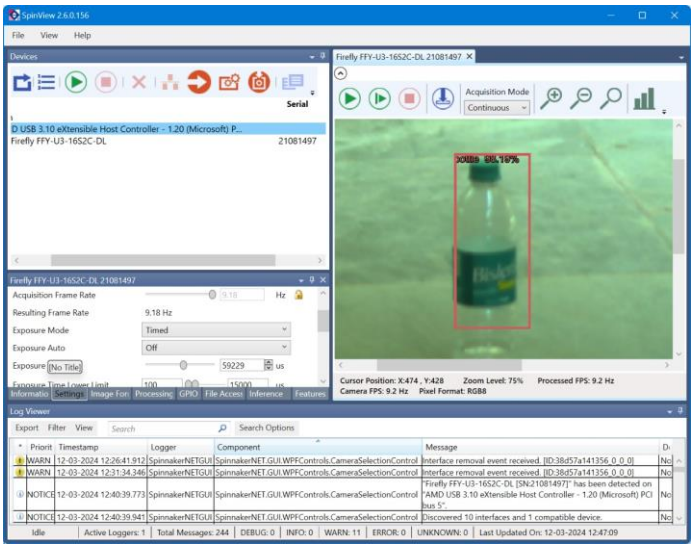
3] Object Detection using Deep Learning camera:

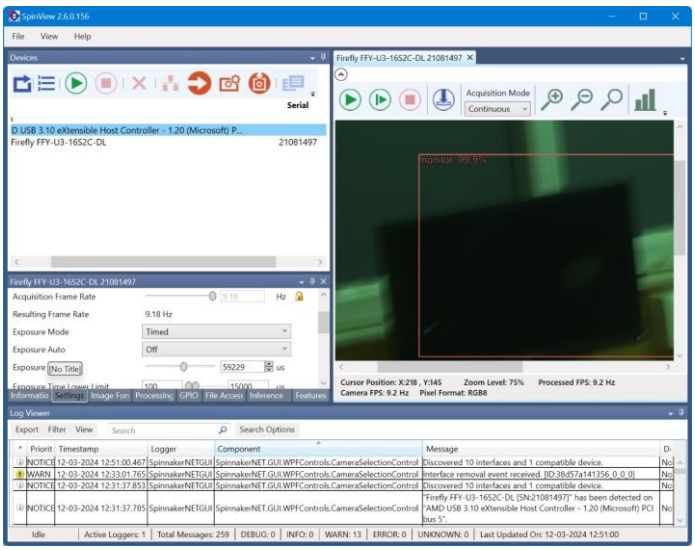
OUTPUT:

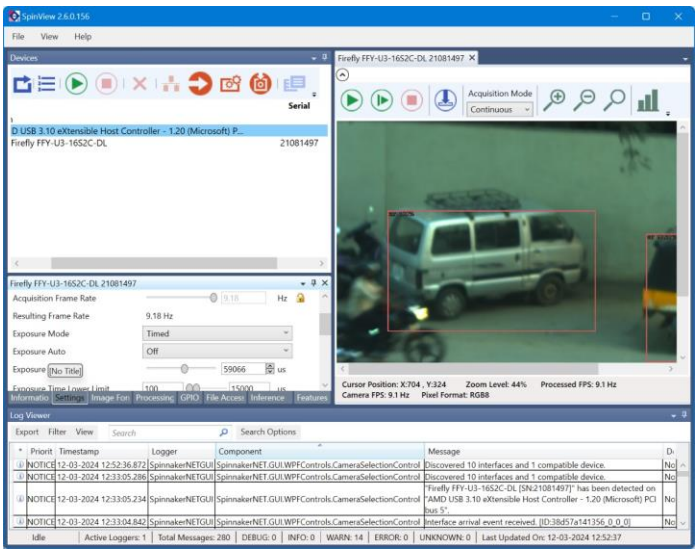


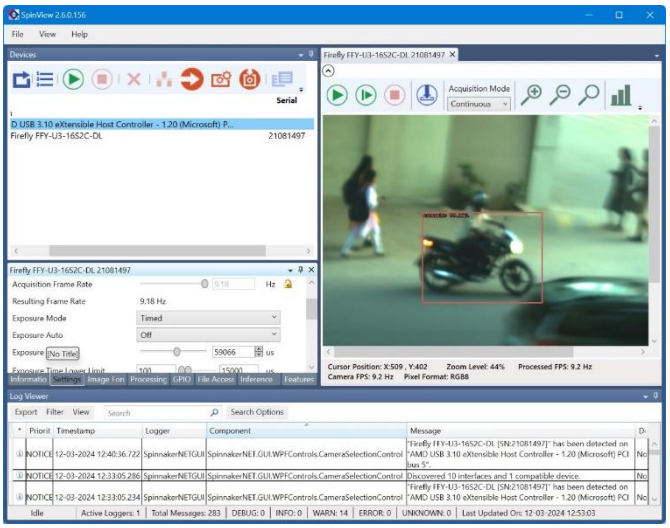


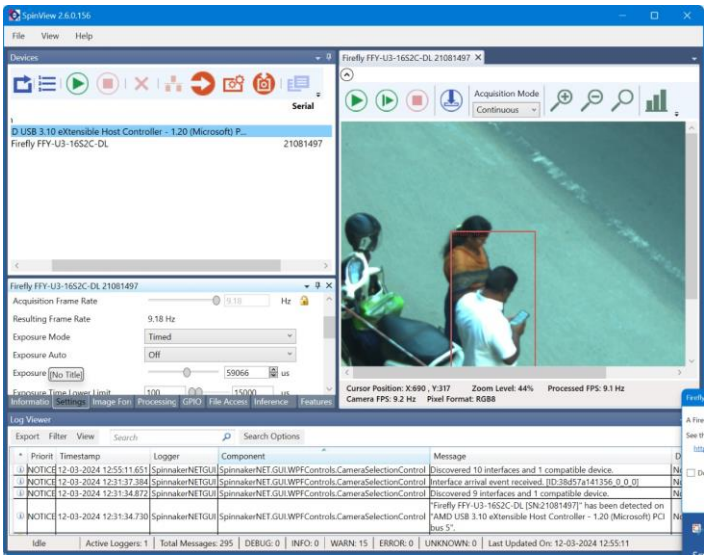












Department of RAE			
Criteria	**Excellent (75% - 100%)**	**Good (50 – 75%)**	**Poor (<50%)**
Preparation (30)			
Performance (30)			
Evaluation (20)			
Report (20)			
Sign:		**Total (100)**	

Result:

Thus, Face Detection using Haar Cascade and Object Detection using Yolo V5 were performed.

Post Lab Questions

1. **What are the key advantages of using Haar Cascades for face detection compared to other methods?**
   - Haar cascades use simple rectangular features, enabling rapid image scanning for real-time face detection, even on low-powered devices.
   - They achieve high face detection rates while maintaining relatively low false positive rates, suitable for security systems and video surveillance.
   - Haar cascades exhibit robustness to lighting conditions and facial expressions, ensuring reliable detection across diverse environments.
   - They are relatively easy to train, allowing for adaptation to specific use cases and environments.
   - Haar cascades are versatile and applicable in a range of scenarios, enhancing their utility for various applications.
2. **What is the difference between ANN & CNN.**
   - ANNs consist of interconnected layers of nodes, while CNNs utilize convolutional layers for feature extraction.
   - ANNs are used for tasks like regression, classification, and pattern recognition, while CNNs are specialized for processing grid-like data such as images.
   - CNNs employ convolutional layers to extract features from input images efficiently.
   - CNNs utilize hierarchical structures to learn increasingly complex features, particularly effective for tasks like image classification and object detection.
   - CNNs excel in tasks involving visual data due to their tailored architecture for image processing.
3. **For the following image perform the convolution operation. Also perform Max pooling, Min pooling and Average pooling on the input image.**

6	5	4	3	2	1
7	6	5	4	3	2
8	7	6	5	4	3
9	8	7	6	5	4
10	9	8	7	6	5
10	10	9	8	7	6

3	2	4
2	0	2
4	2	3

Input Matrix:

[[6 5 4 3 2 1]
[7 6 5 4 3 2]
[8 7 6 5 4 3]
[9 8 7 6 5 4]
[10 9 8 7 6 5]
[10 10 9 8 7 6]]

Filter Matrix:

[[3 2 4]
[2 0 2]
[4 2 3]]

Output Matrix after Convolution:

```
[[132. 110.  88.  66.]
 [154. 132. 110.  88.]
 [176. 154. 132. 110.]
 [194. 176. 154. 132.]]
```

Input Matrix:

```
[[ 6  5  4  3  2  1]
 [ 7  6  5  4  3  2]
 [ 8  7  6  5  4  3]
 [ 9  8  7  6  5  4]
 [10  9  8  7  6  5]
 [10 10  9  8  7  6]]
```

Max Pooled Matrix:

```
[[ 7.  5.  3.]
 [ 9.  7.  5.]
 [10.  9.  7.]]
```

Min Pooled Matrix:

```
[[5. 3. 1.]
 [7. 5. 3.]
 [9. 7. 5.]]
```

Average Pooled Matrix:

```
[[6.   4.   2.  ]
 [8.   6.   4.  ]
 [9.75 8.   6.  ]]
```

Experiment 9

Parallel Programming Using CUDA

Aim:

To study parallel programming concepts using CUDA and understand the difference between GPU and CPU processing.

Software/ Packages Used:

1. Google Colaboratory
2. Libraries used:
    - Opencv – python
    - Numpy
    - Matplotlib
    - tensorflow

Programs:
Coding:
Installation:

```
!apt-get --purge remove cuda nvidia* libnvidia-*

!dpkg -l | grep cuda- | awk '{print $2}' | xargs -n1 dpkg --purge

!apt-get remove cuda-*

!apt autoremove

!apt-get update
```

Install CUDA Version 9:

```
!wget https://developer.nvidia.com/compute/cuda/9.2/Prod/local_installers/cuda-repo-ubuntu1604-9-2-local_9.2.88-1_amd64 -O cuda-repo-ubuntu1604-9-2-local_9.2.88-1_amd64.deb
!dpkg -i cuda-repo-ubuntu1604-9-2-local_9.2.88-1_amd64.deb
!dpkg -i cuda-repo-ubuntu1604-9-2-local_9.2.88-1_amd64.deb
!apt-key add /var/cuda-repo-9-2-local/7fa2af80.pub
!apt-get update
!apt-get install cuda-9.2
```

Check the Version of CUDA by : running the command below to get the following output :

```
!export PATH=/usr/local/cuda/bin${PATH:+:${PATH}}
!export LD_LIBRARY_PATH=/usr/local/cuda/lib64\${LD_LIBRARY_PATH:+:${ LD_LIBRARY_PATH}}
!/usr/local/cuda/bin/nvcc --version
```

Execute the given command to install a small extension to run nvcc from Notebook cells:
!git config --global url."https://github.com/".insteadOf git://github.com/
!pip install git+git://github.com/andreinechaev/nvcc4jupyter.git
Load the extension using this code:
%load_ext nvcc_plugin

CUDA Program – 1

```
%%cu
#include <stdio.h> #include <stdlib.h>
        global  void add(int *a, int *b, int *c) {
*c = *a + *b;
}
int main() { int a, b, c;
// host copies of variables a, b & c int *d_a, *d_b, *d_c;
// device copies of variables a, b & c int size = sizeof(int);
// Allocate space for device copies of a, b, c cudaMalloc((void **)&d_a, size);
cudaMalloc((void **)&d_b, size); cudaMalloc((void **)&d_c, size);
// Setup input values c = 0;
a = 3;
b = 5;
// Copy inputs to device
cudaMemcpy(d_a, &a, size, cudaMemcpyHostToDevice); cudaMemcpy(d_b, &b, size,
cudaMemcpyHostToDevice);
// Launch add() kernel on GPU add<<<1,1>>>(d_a, d_b, d_c);
// Copy result back to host
cudaError err = cudaMemcpy(&c, d_c, size, cudaMemcpyDeviceToHost);
if(err!=cudaSuccess) {
printf("CUDA error copying to Host: %s\n", cudaGetErrorString(err));

}
printf("result is %d\n",c);
// Cleanup cudaFree(d_a); cudaFree(d_b); cudaFree(d_c); return 0;
}
```

CUDA Program – 2

```
%%cu
#include <stdio.h> #define N 64
inline cudaError_t checkCudaErr(cudaError_t err, const char* msg) { if (err != cudaSuccess) {
fprintf(stderr, "CUDA Runtime error at %s: %s\n", msg, cudaGetErrorString(err)
);
}
```

```
return err;
}
        global  void matrixMulGPU( int * a, int * b, int * c )
{
/*
* Build out this kernel.
*/
int row = threadIdx.y + blockIdx.y * blockDim.y; int col = threadIdx.x + blockIdx.x *
blockDim.x;

int val = 0;
if (row < N && col < N) { for (int i = 0; i < N; ++i) {
val += a[row * N + i] * b[i * N + col];
}

c[row * N + col] = val;
}
}

/*
*       This CPU function already works, and will run to create a solution matrix
*       against which to verify your work building out the matrixMulGPU kernel.
*/
void matrixMulCPU( int * a, int * b, int * c )
{
int val = 0;
for( int row = 0; row < N; ++row ) for( int col = 0; col < N; ++col )
{
val = 0;
for ( int k = 0; k < N; ++k )
val += a[row * N + k] * b[k * N + col]; c[row * N + col] = val;
}
}
int main()
{
int *a, *b, *c_cpu, *c_gpu; // Allocate a solution matrix for both the CPU and the GPU
operations
int size = N * N * sizeof (int); // Number of bytes of an N x N matrix
// Allocate memory cudaMallocManaged (&a, size); cudaMallocManaged (&b, size);
cudaMallocManaged (&c_cpu, size); cudaMallocManaged (&c_gpu, size);
// Initialize memory; create 2D matrices for( int row = 0; row < N; ++row ) for( int col = 0; col
< N; ++col )
{
a[row*N + col] = row; b[row*N + col] = col+2; c_cpu[row*N + col] = 0; c_gpu[row*N + col]
```

```
= 0;
}
/*
*       Assign `threads_per_block` and `number_of_blocks` 2D values
*       that can be used in matrixMulGPU above.
*/
dim3 threads_per_block(32, 32, 1);

dim3 number_of_blocks(N / threads_per_block.x + 1, N / threads_per_block.y + 1, 1);
matrixMulGPU <<< number_of_blocks, threads_per_block >>> ( a, b, c_gpu );
checkCudaErr(cudaDeviceSynchronize(), "Syncronization");
checkCudaErr(cudaGetLastError(), "GPU");
// Call the CPU version to check our work matrixMulCPU( a, b, c_cpu );
// Compare the two answers to make sure they are equal bool error = false;
for( int row = 0; row < N && !error; ++row ) for( int col = 0; col < N && !error; ++col )
if (c_cpu[row * N + col] != c_gpu[row * N + col])
{
printf("FOUND ERROR at c[%d][%d]\n", row, col); error = true;
break;
}
if (!error) printf("Success!\n");
// Free all our allocated memory cudaFree(a); cudaFree(b);
cudaFree( c_cpu ); cudaFree( c_gpu );
}
```

CUDA Program – 3

```
%%cu #include<stdio.h> #include<cuda.h>

int main()
{
cudaDeviceProp p; int device_id;
int major; int minor;

cudaGetDevice(&device_id); cudaGetDeviceProperties(&p,device_id);

major=p.major; minor=p.minor;

printf("Name of GPU on your system is %s\n",p.name);

printf("\n Compute Capability of a current GPU on your system is %d.%d",majo r,minor);
```

```
return 0;
}
```

CUDA Program – 4

```
%%cu #include<stdio.h> #include<cuda.h>
#define row1 2 /* Number of rows of first matrix */ #define col1 3 /* Number of columns of first matrix */ #define row2 3 /* Number of rows of second matrix */ #define col2 2 /* Number of columns of second matrix */

        global  void matproductsharedmemory(int *l,int *m, int *n)
{
int x=blockIdx.x; int y=blockIdx.y;
        shared  int p[col1];

int i;
int k=threadIdx.x; n[col2*y+x]=0; p[k]=l[col1*y+k]*m[col2*k+x];



        syncthreads();

for(i=0;i<col1;i++) n[col2*y+x]=n[col2*y+x]+p[i];
}

int main()
{
int a[row1][col1]; int b[row2][col2]; int c[row1][col2]; int *d,*e,*f;
int i,j;

a[0][0]=2;
a[0][1]=6;
a[0][2]=2;
a[1][0]=4;
a[1][1]=7;
a[1][2]=3;
b[0][0]=2;
b[0][1]=5;
b[1][0]=7;
b[1][1]=1;
b[2][0]=8;
b[2][1]=5;

cudaMalloc((void **)&d,row1*col1*sizeof(int)); cudaMalloc((void
```

**)&e,row2*col2*sizeof(int)); cudaMalloc((void **)&f,row1*col2*sizeof(int));

cudaMemcpy(d,a,row1*col1*sizeof(int),cudaMemcpyHostToDevice);
cudaMemcpy(e,b,row2*col2*sizeof(int),cudaMemcpyHostToDevice);

dim3 grid(col2,row1);
/* Here we are defining two dimensional Grid(collection of blocks) structure. Synt ax is dim3 grid(no. of columns,no. of rows) */

matproductsharedmemory<<<grid,col1>>>(d,e,f);

cudaMemcpy(c,f,row1*col2*sizeof(int),cudaMemcpyDeviceToHost);

printf("\n Product of two matrices:\n "); for(i=0;i<row1;i++)
{
for(j=0;j<col2;j++)
{
printf("%d\t",c[i][j]);
}
printf("\n");
}

cudaFree(d); cudaFree(e); cudaFree(f);

return 0;
}

CUDA Program – 5

%%cu #include<stdio.h> #include<cuda.h>

constant int d[5];

global void add(int *c)
{
int id=threadIdx.x;

c[id]=c[id]+d[id];
}

int main()
{

```
int a[5];
int b[5]={1,2,3,4,5};
int *c; int i;



a[0]=1;a[1]=8;a[2]=9;a[3]=6;a[4]=3;
cudaMalloc((void **)&c,5*sizeof(int));
cudaMemcpy(c,a,5*sizeof(int),cudaMemcpyHostToDevice);
cudaMemcpyToSymbol(d,b,5*sizeof(int)); /*copying contents of array b to cons
tant array d */ add<<<1,5>>>(c);
cudaMemcpy(a,c,5*sizeof(int),cudaMemcpyDeviceToHost);

printf("Elements of your array after addition with constant array {1,2,3,4,5} :\n")
;

for(i=0;i<5;i++)
{
printf("%d\t",a[i]);
}

cudaFree(c); cudaFree(d);

}
```

2. CPU VS GPU:

```
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers

from google.colab.patches import cv2_imshow # Model / data parameters

num_classes = 10
input_shape = (28, 28, 1)

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Scale images to the [0, 1] range x_train = x_train.astype("float32") / 255 x_test =
x_test.astype("float32") / 255
# Make sure images have shape (28, 28, 1) x_train = np.expand_dims(x_train, -1) x_test =
np.expand_dims(x_test, -1) print("x_train shape:", x_train.shape) print(x_train.shape[0], "train
samples") print(x_test.shape[0], "test samples")
```

```
# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)

y_test = keras.utils.to_categorical(y_test, num_classes)

model = keras.Sequential( [
keras.Input(shape=input_shape),
layers.Conv2D(32, kernel_size=(3, 3), activation="relu"), layers.MaxPooling2D(pool_size=(2, 2)), layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
layers.MaxPooling2D(pool_size=(2, 2)),
layers.Flatten(), layers.Dropout(0.5),
layers.Dense(num_classes, activation="softmax"),
]
)


model.summary()

import datetime print("Training the Model  ")
start_time=datetime.datetime.now() print("Training started at: {}".format(start_time))
batch_size = 128
epochs = 15


model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)
end_time= datetime.datetime.now() print("Training ended at: {}".format(end_time)) duration = end_time - start_time print("Training Duration: {}".format(duration))

score = model.evaluate(x_test, y_test, verbose=0) print("Test loss:", score[0])
```

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 parallel programming concepts and the difference between CPU and GPU programming were studied.

EXPERIMENT 10
Basics of ROS

Aim:

To install the ROS Noetic version and learn to create a workspace in ROS and to run simple program.

Software/ Package Used:

- Ubuntu 18.04
- ROS - Noetic

Programs:

1. Write a ROS program to configure a node and send a message and configure two different users to receive the same message.

PUBLISHER CODE:

```
#!/usr/bin/env python3
import rospy
from std_msgs.msg import String
def talker():
        pub = rospy.Publisher('chatter', String, queue_size=10)
        rospy.init_node('talker', anonymous=True)
        rate = rospy.Rate(10) # 10hz
        while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()
if __name__ == '__main__':
        try:
        talker()
        except rospy.ROSInterruptException:
        pass
```

SUBSCRIBER 1 CODE:

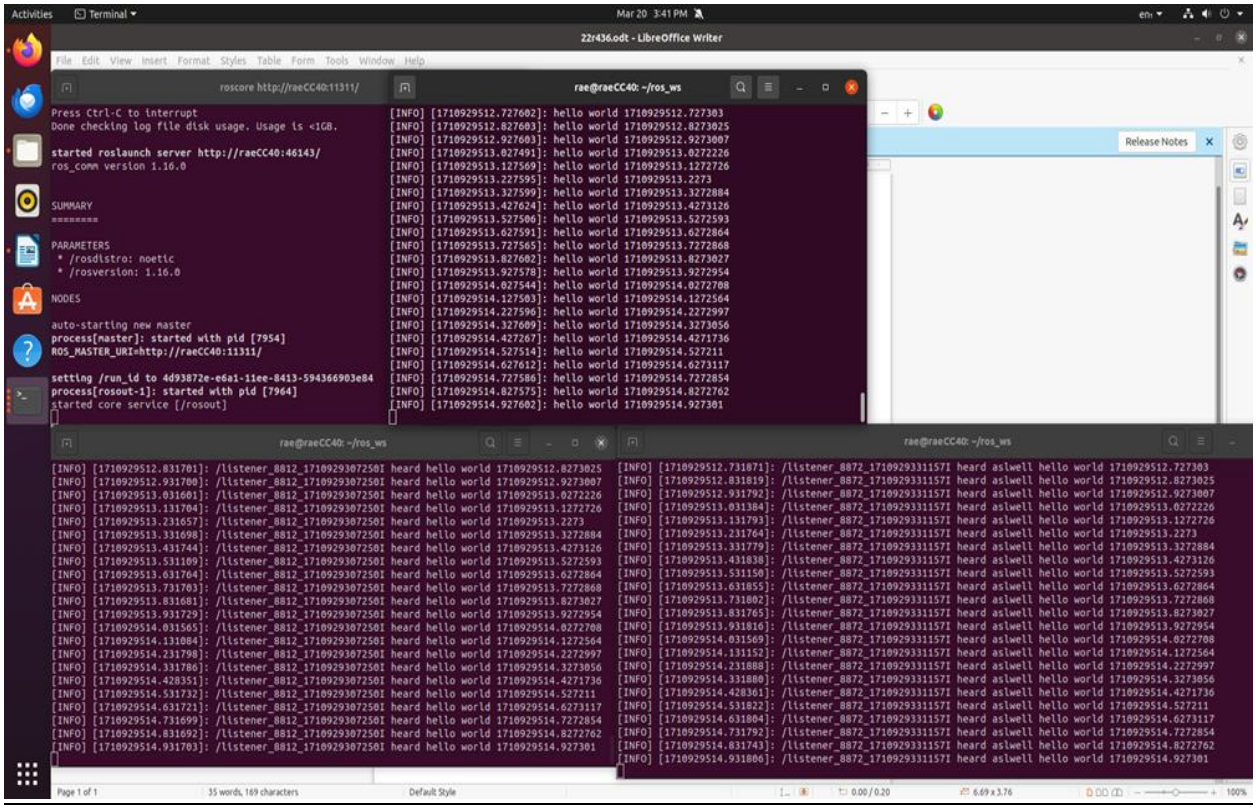
```
#!/usr/bin/env python3
import rospy
from std_msgs.msg import String
def callback(data):
        rospy.loginfo(rospy.get_caller_id() + 'I heard %s', data.data)
def listener():
```

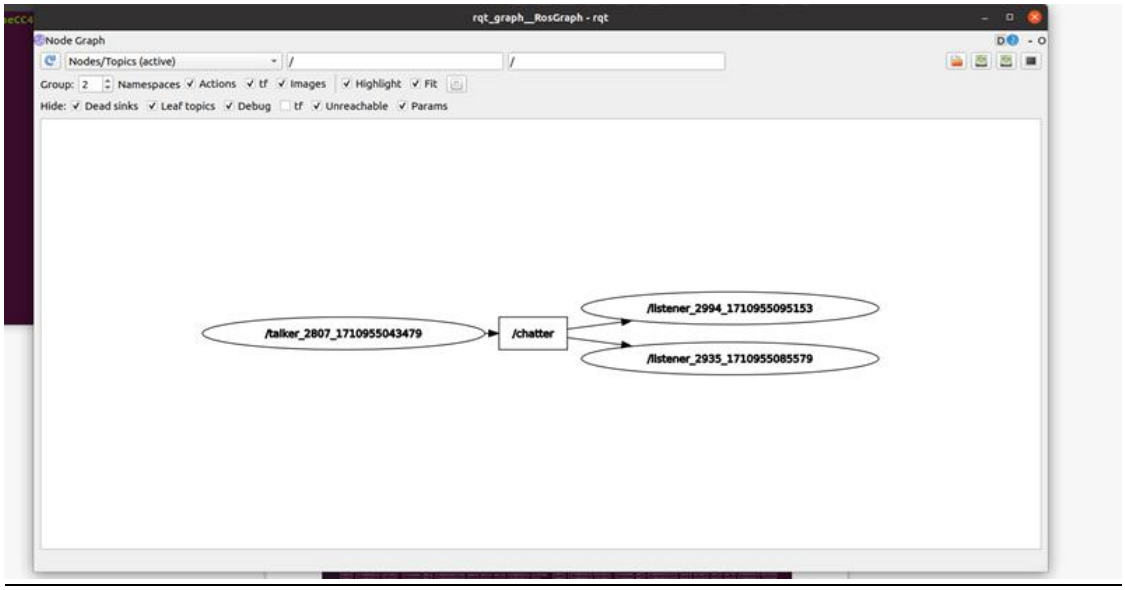
```
        rospy.init_node('listener', anonymous=True)
        rospy.Subscriber('chatter', String, callback)
        # spin() simply keeps python from exiting until this node is stopped
        rospy.spin()
if __name__ == '__main__':
        listener()
```

SUBSCRIBER 2 CODE:

```
#!/usr/bin/env python3
import rospy
from std_msgs.msg import String
def callback(data):
        rospy.loginfo(rospy.get_caller_id() + 'I heard as well %s', data.data)
def listener():
        rospy.init_node('listener', anonymous=True)
        rospy.Subscriber('chatter', String, callback)
        rospy.spin()
if __name__ == '__main__':
        listener()
```

OUTPUT:



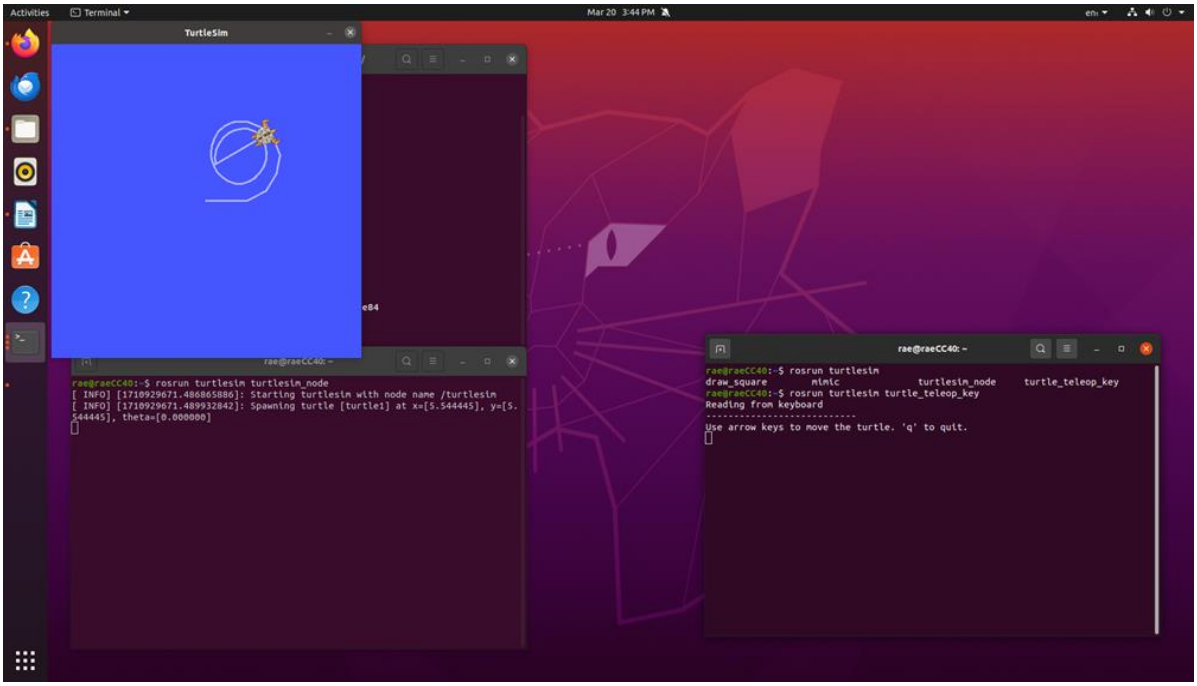


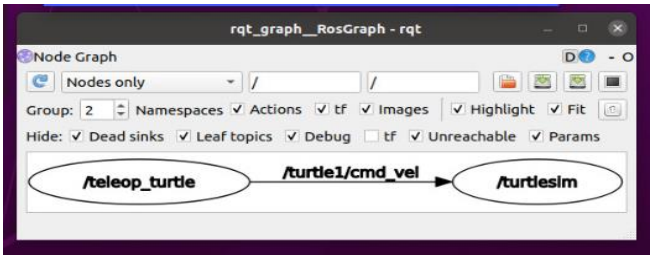
2. Run turtlesim.

COMMANDS:

- rosrun turtlesim turtlesim_node
- rosrun turtlesim turtle_teleop_key
- rqt_graph

OUTPUT:



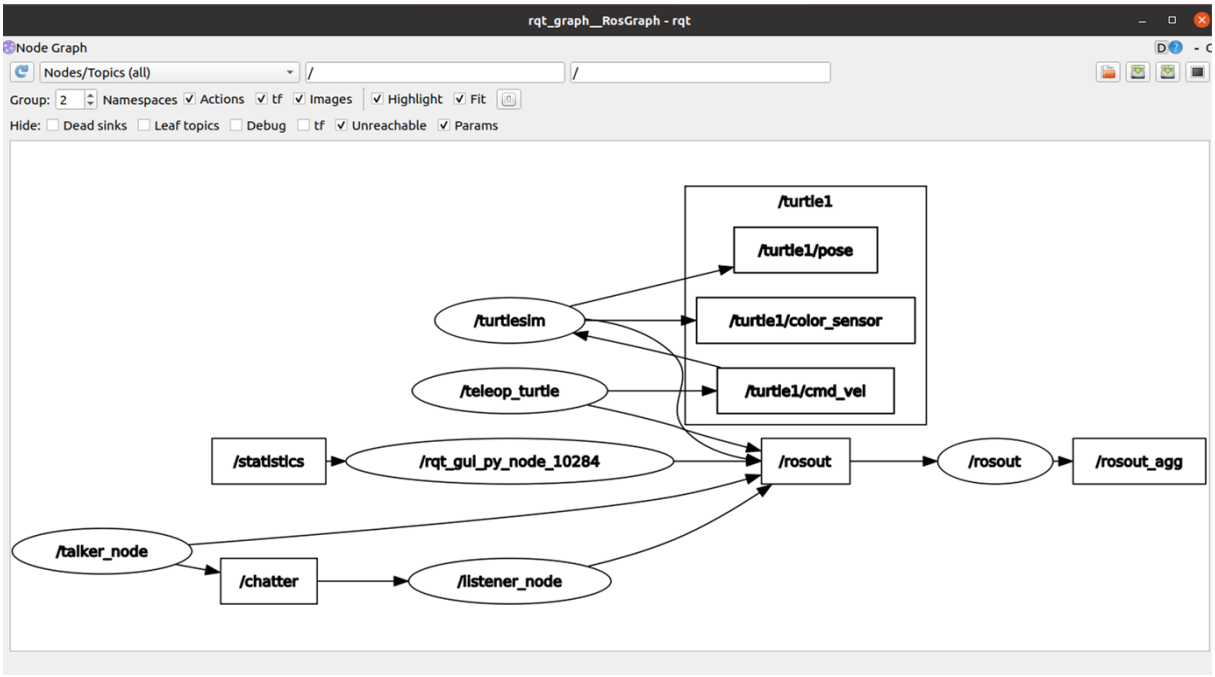


3. RQT Graph.

COMMANDS:

- rqt_graph

OUTPUT:



4. Bridge ROS with openCV. Read an image into ROS and rotate the image.

CODE:

```
#!/usr/bin/env python3
import rospy # Python library for ROS
from sensor_msgs.msg import Image # Image is the message type
from cv_bridge import CvBridge # Package to convert between ROS and OpenCV
Images
import cv2 # OpenCV library
def callback(data):
  br = CvBridge()
    rospy.loginfo("receiving video frame")
    current_frame = br.imgmsg_to_cv2(data)
  current_frame=cv2.circle(current_frame,(60,60),10,(0,255,255),-1)
  current_frame=cv2.rotate(current_frame,cv2.ROTATE_90_CLOCKWISE)
  cv2.imshow("camera", current_frame)
  cv2.waitKey(0)
def receive_message():
  rospy.init_node('video_sub_py', anonymous=True)
```

```
  rospy.Subscriber('video_frames', Image, callback)
  rospy.spin()
  cv2.destroyAllWindows()
if _name_ == '_main_':
  receive_message()
```

OUTPUT:

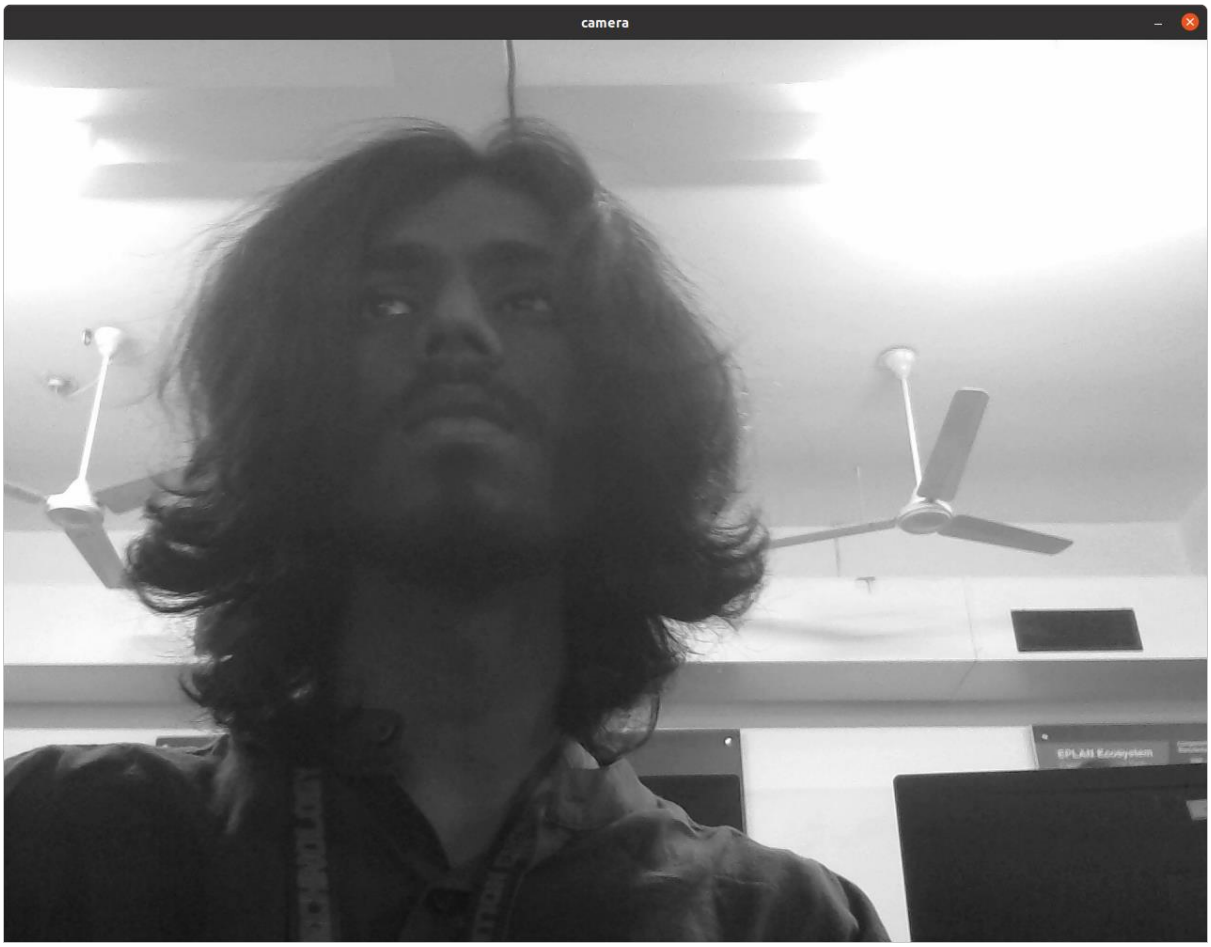
5. Read an image into ROS and perform color conversions on an image.

CODE:

```
#!/usr/bin/env python3
import rospy # Python library for ROS
from sensor_msgs.msg import Image # Image is the message type
from cv_bridge import CvBridge # Package to convert between ROS and OpenCV
Images
import cv2 # OpenCV library
 def publish_message():
  pub = rospy.Publisher('video_frames', Image, queue_size=10)
  rospy.init_node('video_pub_py', anonymous=True)
  rate = rospy.Rate(10) # 10hz
  cap = cv2.imread('/home/rae/Downloads/test.png',0)
  br = CvBridge()
   while not rospy.is_shutdown():
```

```
            if True:
            rospy.loginfo('publishing video frame')
            pub.publish(br.cv2_to_imgmsg(cap))
            rate.sleep()
if _name_ == '_main_':
  try:
            publish_message()
  except rospy.ROSInterruptException:
            pass
```

OUTPUT:



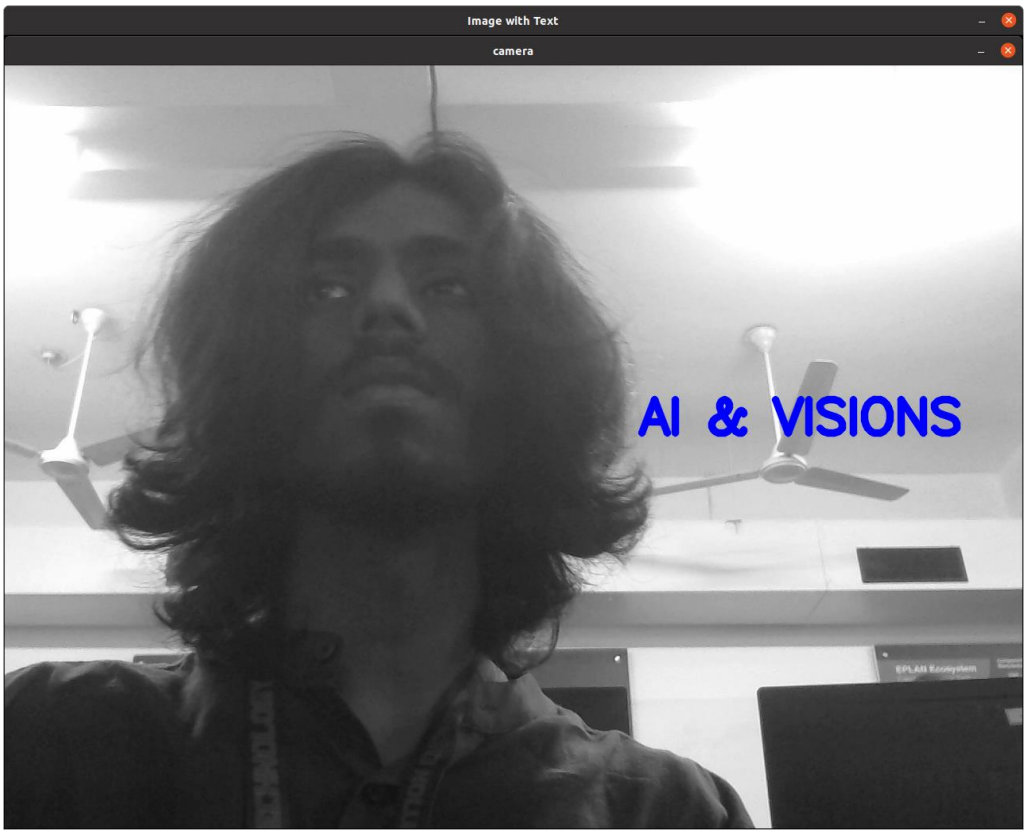
6. Write AI and vison on an image.

CODE:

```
#!/usr/bin/env python3
import rospy # Python library for ROS
from sensor_msgs.msg import Image # Image is the message type
from cv_bridge import CvBridge # Package to convert between ROS and OpenCV
Images
```

```
import cv2 # OpenCV library
 def publish_message():
  pub = rospy.Publisher('video_frames', Image, queue_size=10)
  rospy.init_node('video_pub_py', anonymous=True)
  rate = rospy.Rate(10) # 10hz
  cap = cv2.imread('/home/rae/Downloads/black screen.png')
current_frame=cv2.putText(current_frame,"AI &
VISION",(50,50),cv2.FONT_HERSHEY_SIMPLEX,1,(255,0,255),2,cv2.LINE_AA)
  br = CvBridge()
   while not rospy.is_shutdown():
          if True:
          rospy.loginfo('publishing video frame')
          pub.publish(br.cv2_to_imgmsg(cap))
          rate.sleep()
if _name_ == '_main_':
  try:
          publish_message()
  except rospy.ROSInterruptException:
          pass
```

OUTPUT:

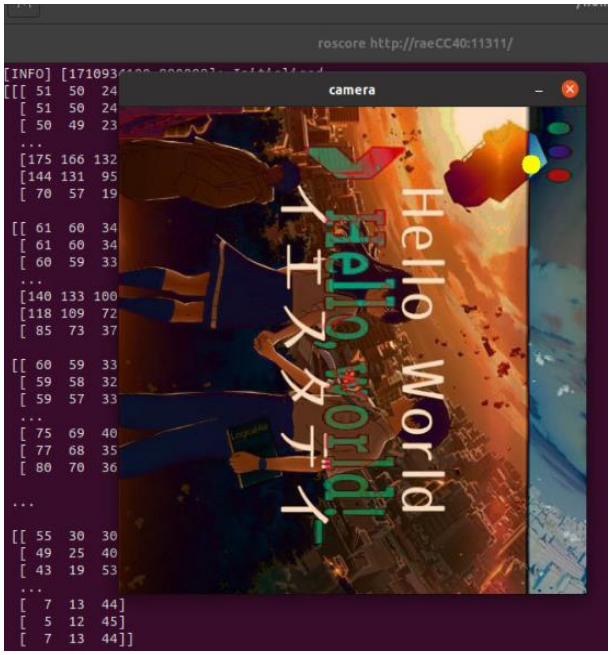


7. Find the difference between the two images.

CODE:

```
import rospy # Python library for ROS
from sensor_msgs.msg import Image # Image is the message type
from cv_bridge import CvBridge # Package to convert between ROS and OpenCV
Images
import cv2 # OpenCV library
def publish_message():
  pub = rospy.Publisher('video_frames', Image, queue_size=10)
  rospy.init_node('video_pub_py', anonymous=True)
  rate = rospy.Rate(10) # 10hz
  cap = cv2.imread('/home/rae/Downloads/test.png',0)
  br = CvBridge()
  while not rospy.is_shutdown():
        rospy.loginfo('publishing video frame')
        pub.publish(br.cv2_to_imgmsg(cap))
        rate.sleep()
if _name_ == '_main_':
  try:
        publish_message()
  except rospy.ROSInterruptException:
        pass
```

OUTPUT:

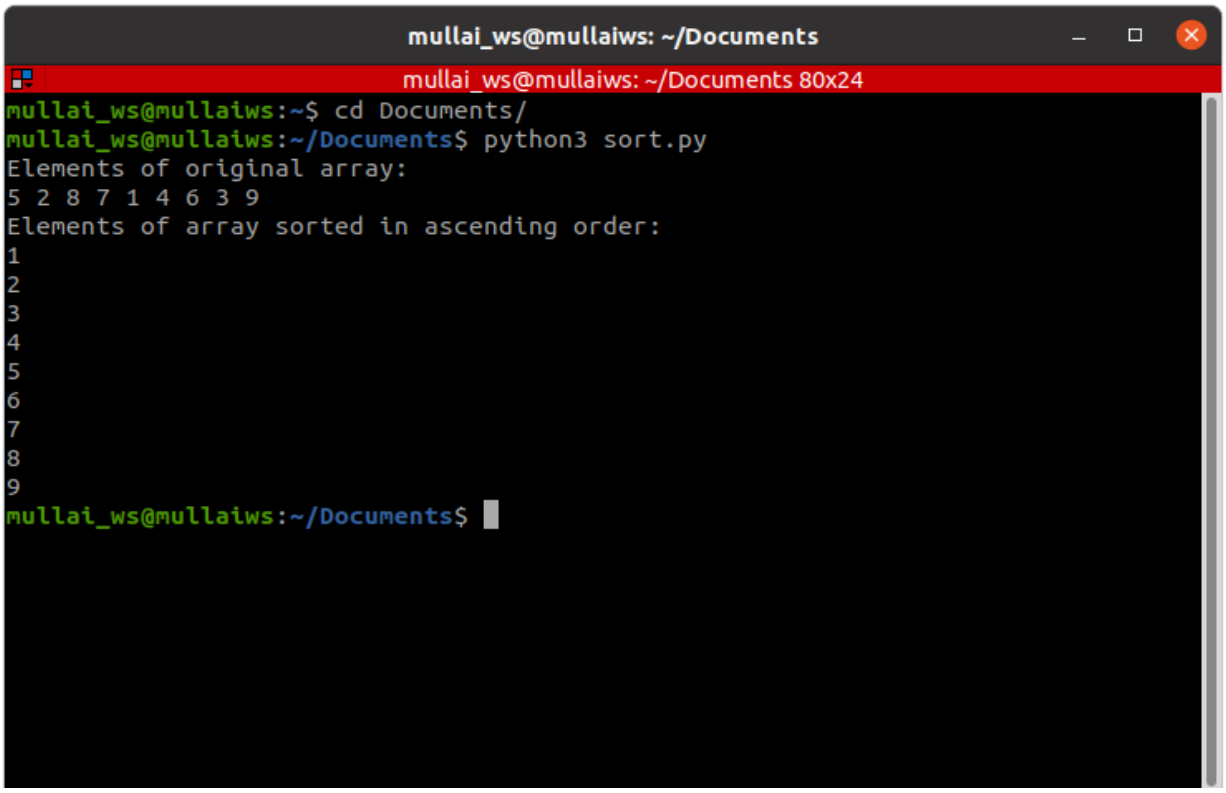


8. Write a python program in ROS to sort a given set of numbers.

CODE:

```
#Initialize array
arr = [5, 2, 8, 7, 1];
temp = 0;
#Displaying elements of original array
print("Elements of original array: ");
for i in range(0, len(arr)):
   print(arr[i], end=" ");
#Sort the array in ascending order
for i in range(0, len(arr)):
   for j in range(i+1, len(arr)):
      if(arr[i] > arr[j]):
         temp = arr[i];
         arr[i] = arr[j];
         arr[j] = temp;
print();
#Displaying elements of the array after sorting
print("Elements of array sorted in ascending order: ");
for i in range(0, len(arr)):
   print(arr[i], end=" ");
```

OUTPUT:



9. Stream the video from USB camera in RoS and write your name on the Stream.

CODE:

```
#!/usr/bin/env python3
import rospy # Python library for ROS
from sensor_msgs.msg import Image # Image is the message type
from cv_bridge import CvBridge # Package to convert between ROS and OpenCV
Images
import cv2 # OpenCV library
def callback(data):
  br = CvBridge()
  rospy.loginfo("receiving video frame")
  current_frame = br.imgmsg_to_cv2(data)
current_frame=cv2.putText(current_frame,"hiiiii",(50,50),cv2.FONT_HERSHEY_SI
MPLEX,1,(255,0,255),2,cv2.LINE_AA)
  current_frame=cv2.circle(current_frame,(60,60),10,(0,255,255),-1)
  cv2.imshow("camera", current_frame)
  cv2.waitKey(1)
def receive_message():
   rospy.init_node('video_sub_py', anonymous=True)
   rospy.Subscriber('video_frames', Image, callback)
   rospy.spin()
  cv2.destroyAllWindows()
if _name_ == '_main_':
  receive_message()
```

OUTPUT:

10. Simulate a world of your own in Gazebo and Rviz and spawn a turtlebot on it. Environment with turtlebot 3 has been setup:

SETUP:

Download link:
https://github.com/SakshayMahna/Robotics-Playground/tree/main/turtlebot3_ws

Unzip into home dir.
Open terminal
    roscore
Open another terminal
    sudo apt-get install ros-noetic-navigation

Open another terminal
    cd turtlebot3_ws/
    catkin_make
    source devel/setup.bash

TO RUN TURTLEBOT3:

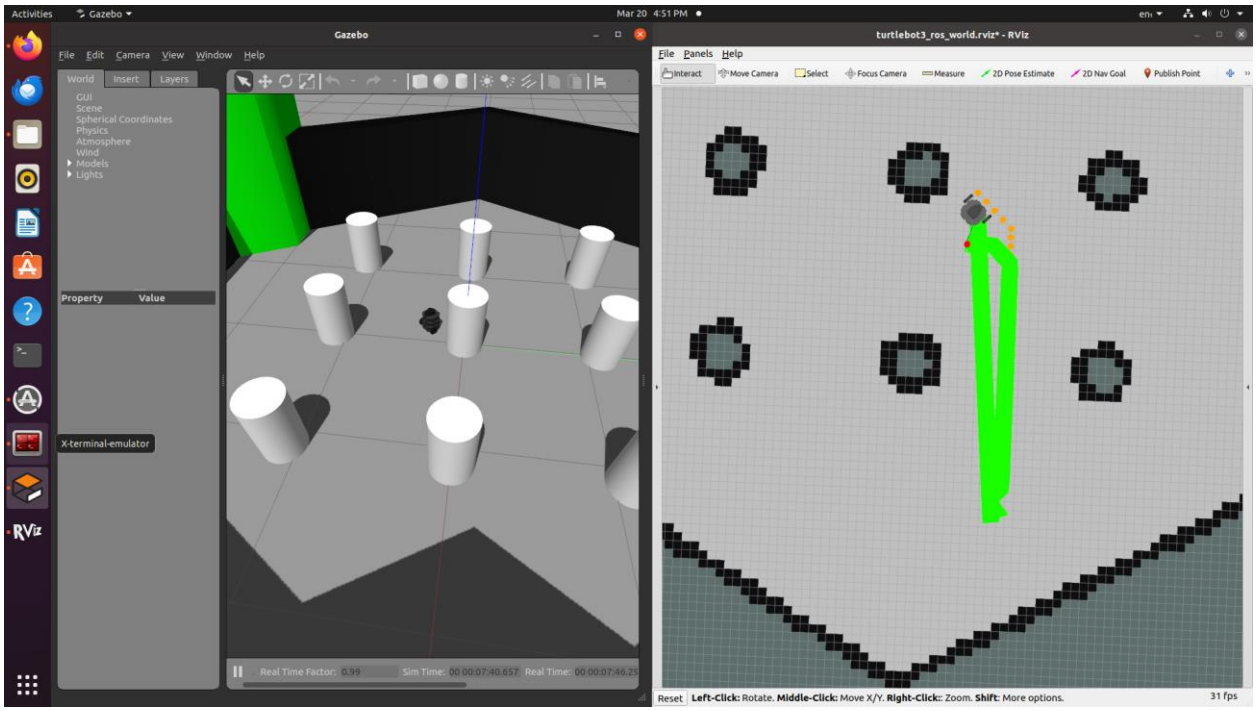
Terminal 1
    roscore

Terminal 2
    cd turtlebot3_ws/
    catkin_make
    source devel/setup.bash
    roslaunch ros_world turtlebot3_world.launch

Terminal 3
    cd turtlebot3_ws/
    catkin_make
    source devel/setup.bash
    roslaunch global_path_planning turtlebot3_ros_world.launch

Terminal 4
    cd turtlebot3_ws/
    catkin_make
    source devel/setup.bash
    rosrun global_path_planning path_planning_server.py

OUTPUT:


Activities
Gazebo
Mar 20 4:51 PM
Gazebo
File Edit Camera View Window Help
World Insert Layers
GUI
Scene
Spherical Coordinates
Physics
Atmosphere
Wind
Models
Lights
Property Value
X-terminal-emulator
RViz
Real Time Factor: 0.99 Sim Time: 00 00:07:40.657 Real Time: 00 00:07:46.25
turtlebot3_ros_world.rviz* - RViz
File Panels Help
Interact Move Camera Select Focus Camera Measure 2D Pose Estimate 2D Nav Goal Publish Point
Reset Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click:: Zoom. Shift: More options.
31 fps

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 ROS basics has been successfully implemented.