**Practical No. 1**

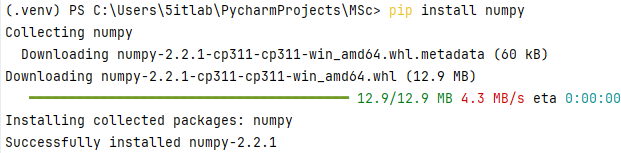
**Part A**

**Aim: Perform Geometric Transformation Packages Used: numpy, cv2**

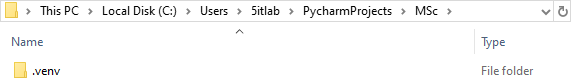
**Command:**

**pip install numpy**

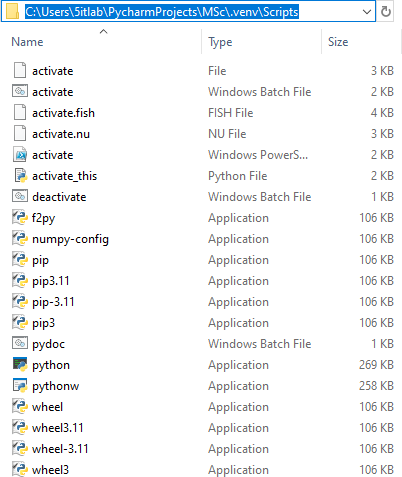
On Pycharm Terminal



**Note:** If an error is found in installing packages from Pycharm Terminal, then open the project in the File Explorer.

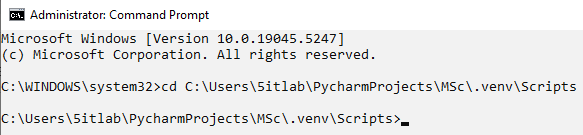


Open the **Scripts** folder and copy the path

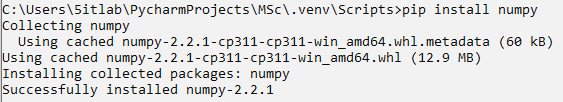


Open a command propt with administrative privileges, and change the working directory using the following command:

**cd **

****

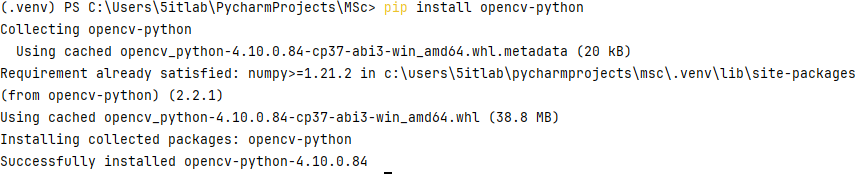
Try using the command again



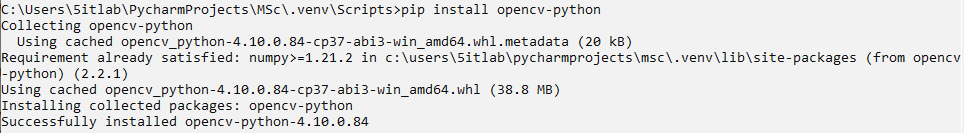




On Pycharm Terminal



On Command Propt (Administrative Privileges)



**Images Used:**

img.jpeg, moon.jpg

**Code (Translation, Rotation, Scaling and Shearing):**

import numpy as np import cv2

def translate(image, x, y): M = np.float32([

[1, 0, x],

[0, 1, y]])

translated\_image = cv2.warpAffine(image, M, (image.shape[1], image.shape[0])) return translated\_image

def rotate(image, angle, center=None, scale=1.0): (h, w) = image.shape[:2]

if center is None:

center = (w // 2, h // 2)

M = cv2.getRotationMatrix2D(center, angle, scale) rotated\_image = cv2.warpAffine(image, M, (w, h)) return rotated\_image

def scale(image, scale\_x, scale\_y):

scaled\_image = cv2.resize(image, None, fx=scale\_x, fy=scale\_y) return scaled\_image

def shear(image, shear\_x, shear\_y): M = np.float32([

[1, shear\_x, 0],

[shear\_y, 1, 0]])

sheared\_image = cv2.warpAffine(image, M, (image.shape[1], image.shape[0])) return sheared\_image

image = cv2.imread('img.jpg') #Main Logic

translated\_image = translate(image, 100, 100)

rotated\_image = rotate(image, 45) scaled\_image = scale(image, 0.5, 0.5)

sheared\_image = shear(image, 0.2, 0.3)

#Display of images cv2.imshow('Original',image)

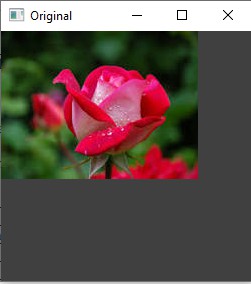
cv2.imshow('translated\_image',translated\_image) cv2.imshow('rotated\_image',rotated\_image) cv2.imshow('scaled\_image',scaled\_image) cv2.imshow('sheared\_image',sheared\_image)

#Resize Images cv2.resizeWindow('Original',250,250) cv2.resizeWindow('translated\_image',250,250) cv2.resizeWindow('rotated\_image',250,250) cv2.resizeWindow('scaled\_image',250,250) cv2.resizeWindow('sheared\_image',250,250)

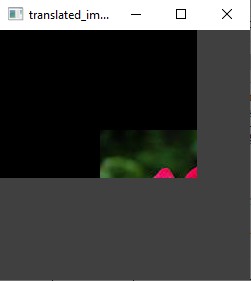
cv2.waitKey(0) cv2.destroyAllWindows()

**Output:**

**Original**

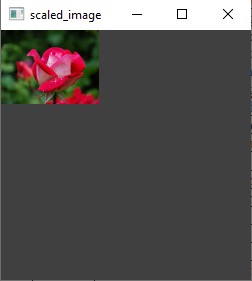


**translated\_image**

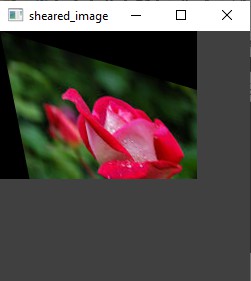


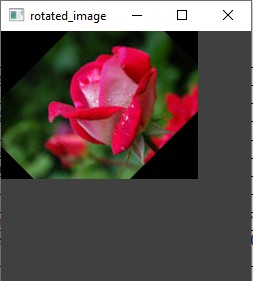
**rotated\_image**

**scaled\_image**



**sheared\_image**





**Code (Reflection):**

import cv2

# Load the image

image = cv2.imread("moon.jpg")

# Reflection w.r.t. X-axis (vertical flip) reflected\_x = cv2.flip(image, 0)

# Reflection w.r.t. Y-axis (horizontal flip) reflected\_y = cv2.flip(image, 1)

# Reflection w.r.t. Origin (both axes flip) reflected\_origin = cv2.flip(image, -1)

# Display the results cv2.imshow("Original Image",image)

cv2.imshow("Reflection wrt X",reflected\_x) cv2.imshow("Reflection wrt Y",reflected\_y) cv2.imshow("Reflection wrt Origin",reflected\_origin)

#Resize Images cv2.resizeWindow('Original Image',350,350)

cv2.resizeWindow('Reflection wrt X',350,350) cv2.resizeWindow('Reflection wrt Y',350,350) cv2.resizeWindow('Reflection wrt Origin',350,350)

cv2.waitKey(0) cv2.destroyAllWindows()

**Output:**

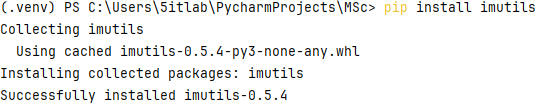
 

**Part B**

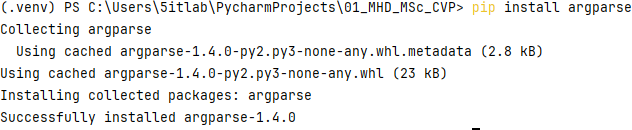
**Aim: Perform Image Stitching Steps:**

**Packages Used: imutils, cv2 Command:**

**pip install imutils**

****

**pip install argparse**

****

**Images Used:**

P1>>building1.jpg, building2.jpg, building3.jpg, building4.jpg, building5.jpg P2>>p21.png, p22.png, p23.png

P3>>001.jpg, 002.jpg, 003.jpg, 004.jpg, 005.jpg **Code:**

from imutils import paths import imutils

import cv2 import argparse

imagePaths = sorted(list(paths.list\_images('P1'))) #'P..' is folder containing images images=[]

# loop over the image paths, load each one, and add them to our

# images to stitch list

for imagePath in imagePaths:

image = cv2.imread(imagePath) images.append(image)

# initialize OpenCV's image stitcher object and then perform the image # stitching

print("[INFO] stitching images...")

#stitcher = cv2.Stitcher\_create(cv2.Stitcher\_PANORAMA)

stitcher = cv2.createStitcher() if imutils.is\_cv3() else cv2.Stitcher\_create() (status, stitched) = stitcher.stitch(images)

# if the status is '0', then OpenCV successfully performed image # stitching

if status == 0:

# write the output stitched image to disk cv2.imwrite("P1Stitched.jpg", stitched)

# display the output stitched image to our screen

cv2.imshow("Stitched", stitched) cv2.waitKey(0)

else:

print("[INFO] image stitching failed ({})".format(status))

**Output: P1Stitched.jpg**



**P2Stitched.jpg**

**P3Stitched.jpg**



**Note:** Output image is also stored in the folder



**Part C**

**Aim: Perform Camera Calibration**

**Steps:**

**Packages Used: numpy, cv2, os, glob**

**Images Used:**

p3.jpg p32.jpg p33.jpg

**Code:**

import cv2

import numpy as np import os

import glob

# Define the dimensions of checkerboard CHECKERBOARD = (6, 9)

# stop the iteration when specified # accuracy, epsilon, is reached or

# specified number of iterations are completed.

criteria = (cv2.TERM\_CRITERIA\_EPS + cv2.TERM\_CRITERIA\_MAX\_ITER, 30, 0.001)

# Vector for 3D points threedpoints = []

# Vector for 2D points

twodpoints = []

# 3D points real world coordinates

objectp3d = np.zeros((1, CHECKERBOARD[0] \* CHECKERBOARD[1], 3), np.float32) objectp3d[0, :, :2] = np.mgrid[0:CHECKERBOARD[0], 0:CHECKERBOARD[1]].T.reshape(-1, 2)

prev\_img\_shape = None

# Extracting path of individual image stored # in a given directory. Since no path is

# specified, it will take current directory # jpg files alone

images = glob.glob('\*.jpg')

for filename in images:

image = cv2.imread(filename)

grayColor = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# Find the chess board corners

# If desired number of corners are # found in the image then ret = true

ret, corners = cv2.findChessboardCorners( grayColor, CHECKERBOARD, cv2.CALIB\_CB\_ADAPTIVE\_THRESH

+ cv2.CALIB\_CB\_FAST\_CHECK + cv2.CALIB\_CB\_NORMALIZE\_IMAGE)

# If desired number of corners can be detected then, # refine the pixel coordinates and display

# them on the images of checker board if ret == True:

threedpoints.append(objectp3d)

# Refining pixel coordinates # for given 2d points. corners2 = cv2.cornerSubPix(

grayColor, corners, (11, 11), (-1, -1), criteria)

twodpoints.append(corners2)

corners2, ret)

# Draw and display the corners

image = cv2.drawChessboardCorners(image, CHECKERBOARD,

cv2.imshow('img', image) cv2.waitKey(0)

cv2.destroyAllWindows() h, w = image.shape[:2]

# Perform camera calibration by

# passing the value of above found out 3D points (threedpoints) # and its corresponding pixel coordinates of the

# detected corners (twodpoints)

ret, matrix, distortion, r\_vecs, t\_vecs = cv2.calibrateCamera( threedpoints, twodpoints, grayColor.shape[::-1], None, None)

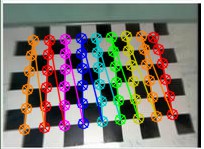
# Displaying required output print(" Camera matrix:") print(matrix)

print("\n Distortion coefficient:") print(distortion)

print("\n Rotation Vectors:") print(r\_vecs)

print("\n Translation Vectors:") print(t\_vecs)

**Output:**

**Case – I (p3.jpg):**

|  |  |  |  |
| --- | --- | --- | --- |
| **Camera matrix:** | |  | |
| [[ 58.17561698 | | 0. | 102.67813943] |
| [ 0. | 60.45272444 | | 93.55060487] |
| [ 0. | 0. 1. | | ]] |

**Distortion coefficient:**

[[ 0.02229387 -0.02371451 0.00047977 -0.00194139 0.00608793]]

**Rotation Vectors:**

(array([[-0.13607382], [ 0.08651059],

[ 1.49228225]]),)

**Translation Vectors:**

(array([[ 3.51290647],

[-3.70110923],

[ 3.98457147]]),)

**Case – II (p32.jpg):**

**Camera matrix:**

[[ 8.8650501 0. 75.00007197]

[ 0. 8.86817689 99.50002698]

[ 0. 0. 1. ]]

**Distortion coefficient:**

[[ 1.32152374e-04 -2.46703289e-06 -5.41268873e-05 -2.85386206e-05 1.27621105e-08]]

**Rotation Vectors:** (array([[-0.03259909], [-0.00766796],

[ 3.07373454]]),)

**Translation Vectors:** (array([[2.9027577 ], [3.72600396],

[0.60404124]]),)

**Case – III (p33.jpg):**



**Camera matrix:**

[[ 28.92668812 0. 57.87918422]

[ 0. 30.55971048 103.85696559]

[ 0. 0. 1. ]]

**Distortion coefficient:**

[[ 3.21879713e-03 -5.56108458e-04 1.09806782e-04 1.98033459e-04 2.63324031e-05]]

**Rotation Vectors:** (array([[-0.01773896], [ 0.07318732],

[-0.05676499]]),)

**Translation Vectors:** (array([[-1.47326778], [-4.01274967],

[ 2.08423295]]),

**Practical No. 2**

**Part A.1 -**

**Aim - Perform Face Detection Packaged Used - CV2, Matplotlib**

**Files Used -** input1.jpg, input2.jpg, input3.jpg **Command -** Terminal

* pip install opencv-python (cv2)
* Pip install matplotlib (Matplotlib)

**Code** - import cv2

import matplotlib.pyplot as plt img = cv2.imread('input4.jpeg')

gray\_image = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

face\_classifier = cv2.CascadeClassifier(

cv2.data.haarcascades + "haarcascade\_frontalface\_default.xml") face = face\_classifier.detectMultiScale(

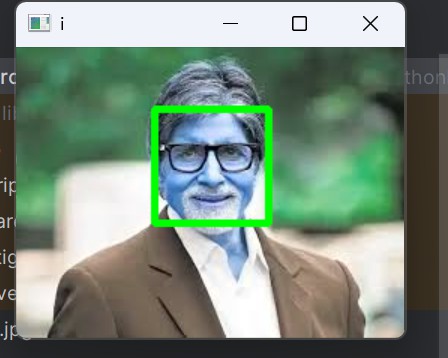
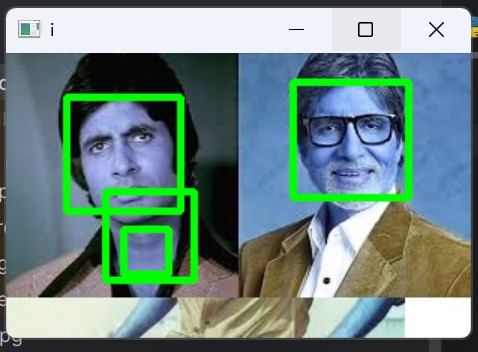
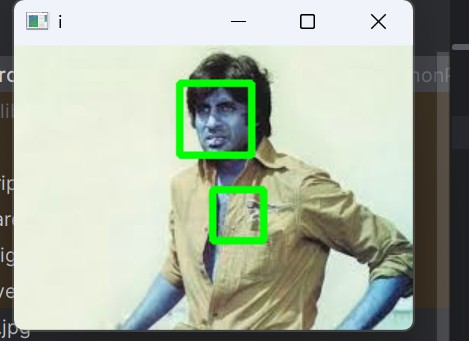
gray\_image, scaleFactor=1.01, minNeighbors=2, minSize=(5, 5))

for (x, y, w, h) in face:

cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 4) img\_rgb = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB) cv2.imshow('i',img\_rgb)

#plt.axis('off') cv2.waitKey(0)

**Output -**

****

**Part A.2 -**

**Aim - Perform Object Detection**

**Image Used -** stop.png, stop1.jpeg, stop2.jpeg **Files used -** stop\_data.xml

**Steps -**

* **Open pycharm**
* **Create Project / Open Project**
* **Upload the images on the project**
* **Upload stop\_data.xml in the project**
* **Start Execution**

**Code -** import cv2

from matplotlib import pyplot as plt

# Opening image

img = cv2.imread("stop.png")

# OpenCV opens images as BRG so it must be converted to RGB and grayscale version

img\_gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

img\_rgb = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB) # Use minSize because for not

# bothering with extra-small

# dots that would look like STOP signs

stop\_data = cv2.CascadeClassifier('stop\_data.xml')

found = stop\_data.detectMultiScale(img\_gray, minSize =(20, 20)) # Don't do anything if there's

# no sign

amount\_found = len(found) if amount\_found != 0:

# There may be more than one

# sign in the image

for (x, y, width, height) in found:

# We draw a green rectangle around # every recognized sign

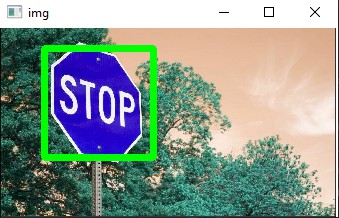
cv2.rectangle(img\_rgb, (x, y),

(x + height, y + width), (0, 255, 0), 5)

# Creates the environment of # the picture and shows it cv2.imshow('img',img\_rgb) plt.show()

cv2.waitKey(0) **Output -**

****



**Part A.3 -**

**Aim - Perform Pedestrian detection**

**Packages Used – imutils Command -** Terminal

* pip install imutils

**Image Used -** input1.png, people1.jpeg, people2.jpeg **Code -**

import cv2 import imutils

# Initializing the HOG person

# detector

hog = cv2.HOGDescriptor() hog.setSVMDetector(cv2.HOGDescriptor\_getDefaultPeopleDetector()) # Reading the Image

image = cv2.imread('people2.jpeg')

# Resizing the Image

image = imutils.resize(image,width=min(400, image.shape[1]))

# Detecting all the regions in the

# Image that has a pedestrians inside it (regions, \_) = hog.detectMultiScale(image,

winStride=(4, 4),

padding=(10, 10), scale=1.05)

print(regions)

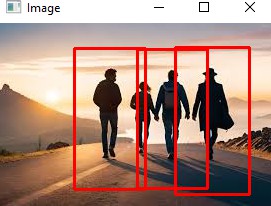
# Drawing the regions in the Image for (x, y, w, h) in regions:

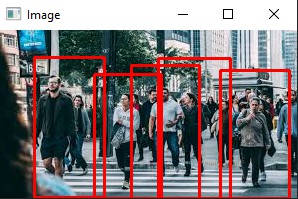
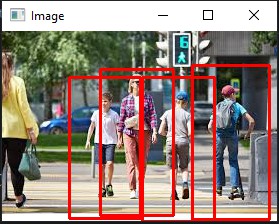
cv2.rectangle(image, (x, y),

(x + w, y + h), (0, 0, 255), 2)

# Showing the output Image cv2.imshow("Image", image) cv2.waitKey(0) cv2.destroyAllWindows()

**Output -**

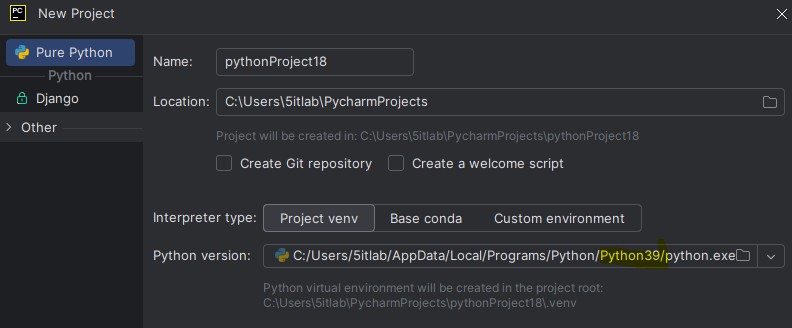
****



**Part A.4 -**

**Aim - Perform Face Recognition**

**Packages Used -** cv2, cmake, face\_recognition **Instruction**:

1. Install Python 3.9>>Set Python path on python 3.9
2. Copy **dlib-19.23.0-cp39-cp39-win\_amd64.whl** file in 3 locations
   * Pycharm Project
   * C:\Users\5itlab\PycharmProjects\pythonProject17\.venv\Scripts
   * C:\Users\5itlab\AppData\Local\Programs\Python\Python39\Scripts
3. Install all the packages in Pycharm Projects pip install cmake

pip install dlib-19.23.0-cp39-cp39-win\_amd64.whl

pip install opencv-python ( version 3.9) pip install face\_recognition

pip uninstall pillow pip install pillow pip uninstall numpy

pip install numpy==1.26.4

1. Install the packages in (PYCHARM LINK)

C:\Users\5itlab\PycharmProjects\pythonProject17\.venv\Scripts pip install cmake

pip install dlib-19.23.0-cp39-cp39-win\_amd64.whl pip install opencv-python ( version 3.9)

pip install face\_recognition

1. Install the packages in (PROGRAM LINK)

C:\Users\5itlab\AppData\Local\Programs\Python\Python39\Scripts pip install cmake

pip install dlib-19.23.0-cp39-cp39-win\_amd64.whl pip install opencv-python ( version 3.9)

pip install face\_recognition

**Files Used: nana.jpg, non.jgp, dlib-19.23.0-cp39-cp39-win\_amd64.whl**

**Code:**

import cv2

import face\_recognition

imgmain1 = face\_recognition.load\_image\_file('nana.jpg') imgmain = cv2.cvtColor(imgmain1, cv2.COLOR\_BGR2RGB) imgTest1 = face\_recognition.load\_image\_file('non.jpg') imgTest = cv2.cvtColor(imgTest1, cv2.COLOR\_BGR2RGB)

faceLoc = face\_recognition.face\_locations(imgmain)[0] encodeElon = face\_recognition.face\_encodings(imgmain)[0]

cv2.rectangle(imgmain, (faceLoc[3], faceLoc[0]), (faceLoc[1], faceLoc[2]), (255, 0, 255), 2)

faceLocTest = face\_recognition.face\_locations(imgTest)[0] encodeTest = face\_recognition.face\_encodings(imgTest)[0]

cv2.rectangle(imgTest, (faceLocTest[3], faceLocTest[0]), (faceLocTest[1],

faceLocTest[2]), (255, 0, 255), 2)

results = face\_recognition.compare\_faces([encodeElon], encodeTest) faceDis = face\_recognition.face\_distance([encodeElon], encodeTest) print(results, faceDis)

cv2.putText(imgTest, f'{results} {round(faceDis[0], 2)}', (20, 20),

cv2.FONT\_HERSHEY\_COMPLEX, 1, (0, 0, 255), 2)

cv2.imshow('Main Image', imgmain) cv2.imshow('Test Image', imgTest) cv2.waitKey(0)

**Output:**



**Case 2:**

**Code:**

import cv2

import face\_recognition

imgmain1 = face\_recognition.load\_image\_file('jackiedada.jpg') imgmain = cv2.cvtColor(imgmain1, cv2.COLOR\_BGR2RGB) imgTest1 = face\_recognition.load\_image\_file('nanap.jpg') imgTest = cv2.cvtColor(imgTest1, cv2.COLOR\_BGR2RGB) faceLoc = face\_recognition.face\_locations(imgmain)[0] encodeElon = face\_recognition.face\_encodings(imgmain)[0]

cv2.rectangle(imgmain, (faceLoc[3], faceLoc[0]), (faceLoc[1], faceLoc[2]), (255, 0, 255), 2)

faceLocTest = face\_recognition.face\_locations(imgTest)[0] encodeTest = face\_recognition.face\_encodings(imgTest)[0]

cv2.rectangle(imgTest, (faceLocTest[3], faceLocTest[0]), (faceLocTest[1], faceLocTest[2]), (255, 0, 255), 2)

results = face\_recognition.compare\_faces([encodeElon], encodeTest) faceDis = face\_recognition.face\_distance([encodeElon], encodeTest) print(results, faceDis)

cv2.putText(imgTest, f'{results} {round(faceDis[0], 2)}', (20, 20),

cv2.FONT\_HERSHEY\_COMPLEX, 1, (0, 0, 255), 2)

cv2.imshow('Main Image', imgmain) cv2.imshow('Test Image', imgTest) cv2.waitKey(0)

**Output:**



**Part B -**

**Aim - Perform Construct 3D model from images**

**Packages used - Pillow Command -**

* C:\Users\5itlab\PycharmProjects\CVP\venu\Scripts\activate (To Set the

given script path in the terminal.

* Pip install pillow (Pillow)

**Image Used -** cube1.jpeg, cube2.jpeg **Code -**

from PIL import Image # pip install Pillow import numpy as np

def shift\_image(img, depth\_img, shift\_amount=10):

# Ensure base image has alpha img = img.convert("RGBA") data = np.array(img)

# Ensure depth image is grayscale (for single value) depth\_img = depth\_img.convert("L")

depth\_data = np.array(depth\_img)

deltas = ((depth\_data / 255.0) \* float(shift\_amount)).astype(int) # This creates the transparent resulting image.

# For now, we're dealing with pixel data. shifted\_data = np.zeros\_like(data) height, width, \_ = data.shape

for y, row in enumerate(deltas): for x, dx in enumerate(row):

if x + dx < width and x + dx >= 0: shifted\_data[y, x + dx] = data[y, x]

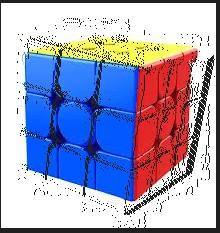
# Convert the pixel data to an image.

shifted\_image = Image.fromarray(shifted\_data.astype(np.uint8)) return shifted\_image

img = Image.open("cube1.jpeg") depth\_img = Image.open("cube2.jpeg")

shifted\_img = shift\_image(img, depth\_img, shift\_amount=10) shifted\_img.show()

**Output –**

****

**Part C**

**Aim -** Implement object detection and tracking from video **Files used -** v2.mp4, v3.mp4

**Code -** import cv2

# Load the pre-trained face detection model face\_classifier = cv2.CascadeClassifier(

cv2.data.haarcascades + "haarcascade\_frontalface\_default.xml"

)

# Replace webcam capture with video file capture video\_capture = cv2.VideoCapture("v4.mp4")

# Function to detect faces and draw bounding boxes

def detect\_bounding\_box(vid):

gray\_image = cv2.cvtColor(vid, cv2.COLOR\_BGR2GRAY) # Convert to grayscale

faces = face\_classifier.detectMultiScale(gray\_image, 1.1, 5, minSize=(40, 40))

for (x, y, w, h) in faces:

cv2.rectangle(vid, (x, y), (x + w, y + h), (0, 255, 0), 4) # Draw rectangle return faces

# Process video frames while True:

result, video\_frame = video\_capture.read() # Read frames from the video file if result is False: # terminate the loop if the frame is not read successfully

break

detect\_bounding\_box(video\_frame) # Detect faces and draw bounding boxes cv2.imshow(

"Face Detection in Video", video\_frame

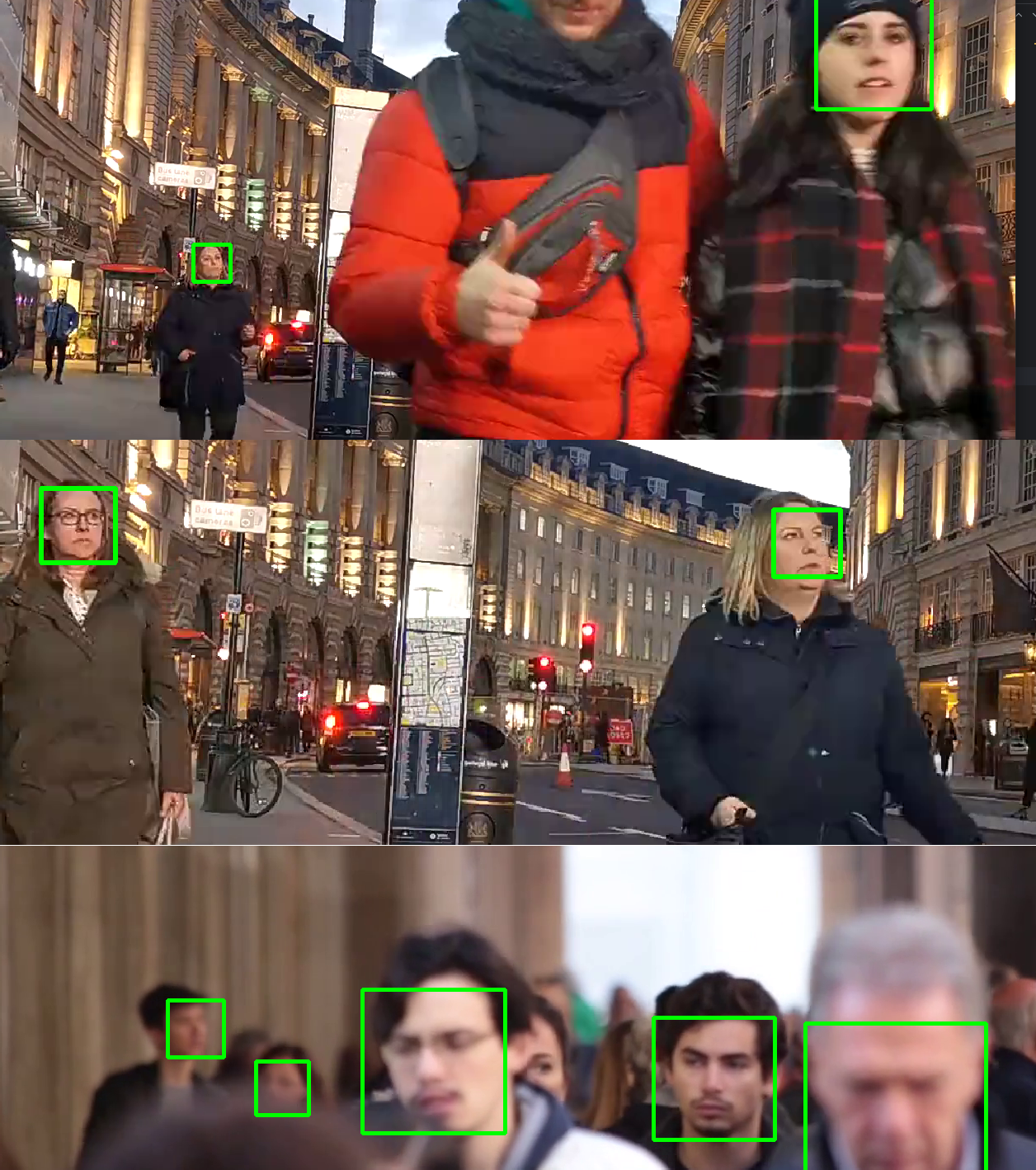
) # Display the processed frame in a window

if cv2.waitKey(1) & 0xFF == ord("q"): # Exit when 'q' is pressed break

# Release video capture and close display window video\_capture.release()

cv2.destroyAllWindows()

**Output -**

****

**Case 2 :** Implement object detection and tracking from Camera Input.

**Files Used:** haarcascade\_frontalface\_default.xml - location **(C:\Users\5itlab\PycharmProjects\pythonProject17\.venv\Lib\site-packages\cv 2\data)**

**Code:**

import cv2 face\_classifier =

cv2.CascadeClassifier(cv2.data.haarcascades+"haarcascade\_frontalface\_defa

ult.xml")

video\_capture = cv2.VideoCapture(0)

def detect\_bounding\_box(vid):

gray\_image = cv2.cvtColor(vid, cv2.COLOR\_BGR2GRAY)

faces = face\_classifier.detectMultiScale(gray\_image, 1.1, 5, minSize=(40, 40)) for (x, y, w, h) in faces:

cv2.rectangle(vid, (x, y), (x + w, y + h), (0, 255, 0), 4) return faces

while True:

result, video\_frame = video\_capture.read() # read frames from the video if result is False:

break # terminate the loop if the frame is not read successfully faces = detect\_bounding\_box(

video\_frame

) # apply the function we created to the video frame cv2.imshow(

"My Face Detection Project", video\_frame

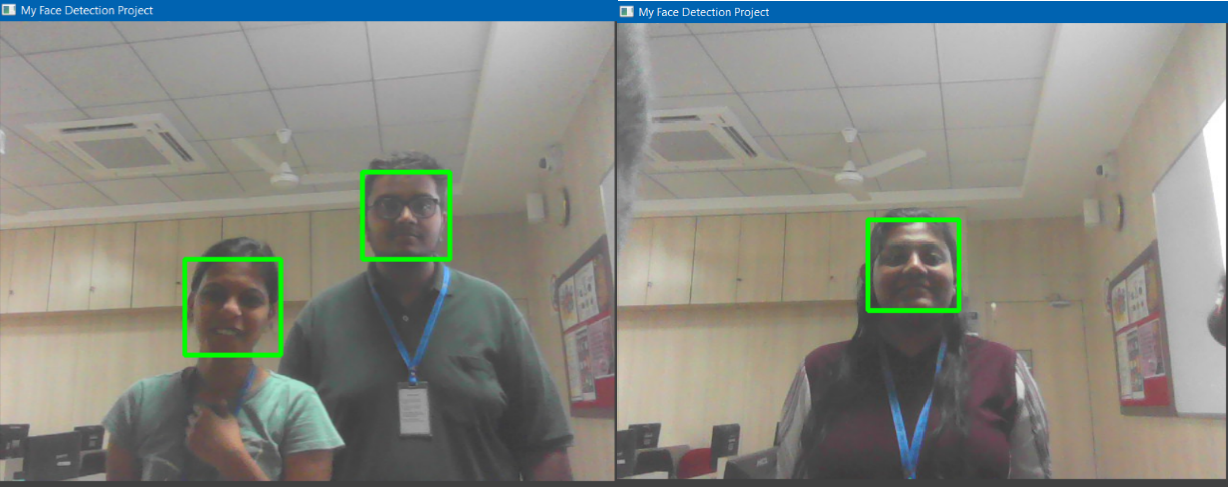
) # display the processed frame in a window named "My Face Detection Project"

if cv2.waitKey(1) & 0xFF == ord("q"):

break

video\_capture.release() cv2.destroyAllWindows()

**Output:**



**Practical No. 3**

**Part A**

**Aim: Perform Feature extraction using RANSAC**

**Steps:**

**Packages Used: numpy, math, pandas, matplotlib**

**Command:**

**pip install numpy pip install pandas pip install matplotlib**

**Files Used:**

**data\_1.csv, data\_2.csv**

**ransac.py**

import numpy as np import math

class RansacModel:

def \_\_init\_\_(self, curve\_fitting\_model): self.curve\_fitting\_model = curve\_fitting\_model

def fit(self, A, Y, num\_sample, threshold): num\_iterations = math.inf iterations\_done = 0

num\_sample = 3

max\_inlier\_count = 0 best\_model = None

prob\_outlier = 0.5

desired\_prob = 0.95

total\_data = np.column\_stack((A, Y)) # [A | Y] data\_size = len(total\_data)

# Adaptively determining the number of iterations while num\_iterations > iterations\_done:

# Shuffle the rows and take the first 'num\_sample' rows as sample data np.random.shuffle(total\_data)

sample\_data = total\_data[:num\_sample, :]

estimated\_model = self.curve\_fitting\_model.fit(sample\_data[:, :-1], sample\_data[:, -1:]) # [a b c]

# Count the inliers within the threshold y\_cap = A.dot(estimated\_model)

err = np.abs(Y - y\_cap.T)

inlier\_count = np.count\_nonzero(err<threshold)

# check for the best model

if inlier\_count > max\_inlier\_count: max\_inlier\_count = inlier\_count best\_model = estimated\_model

prob\_outlier = 1 - inlier\_count / data\_size print('# inliers:', inlier\_count)

print('# prob\_outlier:', prob\_outlier)

num\_iterations = math.log(1 - desired\_prob) / math.log(1 - (1 - prob\_outlier) \*\* num\_sample)

iterations\_done = iterations\_done + 1

print('# s:', iterations\_done) print('# n:', num\_iterations)

print('# max\_inliner\_count: ', max\_inlier\_count) return best\_model

**linearleastsquare.py** import numpy as np

class LinearLeastSquareModel: def fit(self, A, Y):

A\_transpose = A.transpose() ATA = A\_transpose.dot(A) ATY = A\_transpose.dot(Y)

model = (np.linalg.inv(ATA)).dot(ATY)

# For a linear eq. AP = Y to solve a least sqaure problem, P = (inverse(A'A))(A'Y) return model

**modelfitting.py** import numpy as np import math

import pandas as pd

from matplotlib import pyplot as plt from ransac import RansacModel

from linearleastsquare import LinearLeastSquareModel # from<filename>import<classname>

def fit\_curve(data):

x\_values = np.array(data['x']) y\_values = np.array(data['y'])

x\_sq = np.power(x\_values, 2) ## A = [x^2 x 1]

A = np.stack((x\_sq, x\_values, np.ones((len(x\_values)), dtype=int)), axis=1)

threshold = np.std(

y\_values) / 2 # this can be tuned to sd/3 or sd/5 for various curves and better consistent results as a result of random sampling

# Instantiating the linear least square model linear\_ls\_model = LinearLeastSquareModel() linear\_ls\_model\_estimate = linear\_ls\_model.fit(A, y\_values) linear\_model\_y = A.dot(linear\_ls\_model\_estimate)

# Instantiating the ransac model

ransac\_model = RansacModel(linear\_ls\_model) ransac\_model\_estimate = ransac\_model.fit(A, y\_values, 3, threshold) ransac\_model\_y = A.dot(ransac\_model\_estimate)

return linear\_model\_y, ransac\_model\_y if \_\_name\_\_ == '\_\_main\_\_':

# reading the values

df1 = pd.read\_csv('data\_1.csv') df2 = pd.read\_csv('data\_2.csv')

ls\_model\_y1, ransac\_model\_y1 = fit\_curve(df1) ls\_model\_y2, ransac\_model\_y2 = fit\_curve(df2)

fig, (ax1, ax2) = plt.subplots(1, 2) ax1.set\_title('Dataset-1')

ax1.scatter(df1['x'], df1['y'], marker='o', color=(0, 1, 0), label='data points')

ax1.plot(df1['x'], ls\_model\_y1, color='red', label='Least sqaure model') ax1.plot(df1['x'], ransac\_model\_y1, color='blue', label='Ransac model') ax1.set(xlabel='x-axis', ylabel='y-axis')

ax1.legend()

ax2.set\_title('Dataset-2')

ax2.scatter(df2['x'], df2['y'], marker='o', color=(0, 1, 0), label='data points') ax2.plot(df2['x'], ls\_model\_y2, color='red', label='Least sqaure model') ax2.plot(df2['x'], ransac\_model\_y2, color='blue', label='Ransac model') ax2.set(xlabel='x-axis', ylabel='y-axis')

ax2.legend() plt.show()

**Output:**

****

**Console:**

# inliers: 14

# prob\_outlier: 0.944

# s: 1

# n: 17056.92657043659

# max\_inliner\_count: 14

# inliers: 13

# prob\_outlier: 0.948

# s: 2

# n: 21304.062760691897

# max\_inliner\_count: 14

# inliers: 72

# prob\_outlier: 0.712

# s: 3

# n: 123.90418270340892

# max\_inliner\_count: 72

# inliers: 39

# prob\_outlier: 0.844

# s: 4

# n: 787.5960236371116

# max\_inliner\_count: 72

# inliers: 27

# prob\_outlier: 0.892

# s: 5

# n: 2376.6106833968684

# max\_inliner\_count: 72

# inliers: 157

# prob\_outlier: 0.372

# s: 6

# n: 10.526691399992457

# max\_inliner\_count: 157

# inliers: 130

# prob\_outlier: 0.48

# s: 7

# n: 19.769880355510256

# max\_inliner\_count: 157

# inliers: 188

# prob\_outlier: 0.248

# s: 8

# n: 5.409062282538675

# max\_inliner\_count: 188

# inliers: 187

# prob\_outlier: 0.252

# s: 1

# n: 5.525552495791251

# max\_inliner\_count: 187

# inliers: 55

# prob\_outlier: 0.78

# s: 2

# n: 279.841710984081

# max\_inliner\_count: 187

# inliers: 201

# prob\_outlier: 0.19599999999999995

# s: 3

# n: 4.08481384841806

# max\_inliner\_count: 201

# inliers: 191

# prob\_outlier: 0.236

# s: 4

# n: 5.073307052973716

# max\_inliner\_count: 201

# inliers: 67

# prob\_outlier: 0.732

# s: 5

# n: 154.12918046553153

# max\_inliner\_count: 201

# inliers: 184

# prob\_outlier: 0.264

# s: 6

# n: 5.889670195389181

# max\_inliner\_count: 201

**Part B**

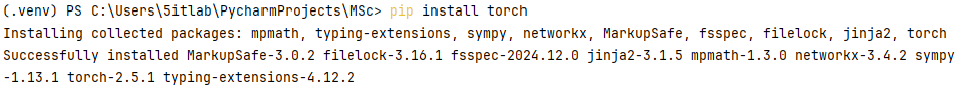
**Aim: Perform Colorization**

**Steps:**

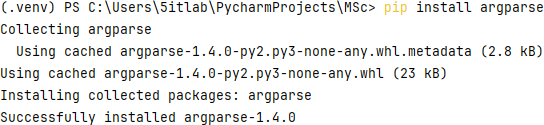
**Packages Used: torch, argparse, ipython, scikit-image, matplotlib, colorizer (local package)**

**Command:**

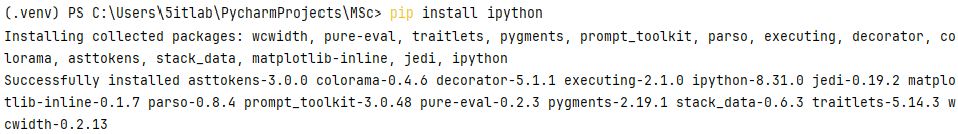
**pip install torch**

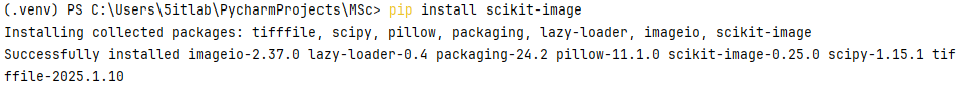
On PyCharm Terminal

**pip install argparse** On PyCharm Terminal



**pip install ipython**

On PyCharm Terminal

**pip install scikit-image** On PyCharm Terminal

**pip install matplotlib** On PyCharm Terminal



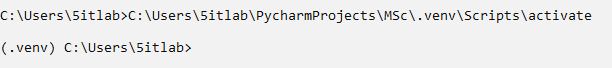


Open the project folder and add the **colorizer** package in the project.



**Note:** If error is found in installing packages on both PyCharm Terminal and Command Prompt, execute the following command and then try again to install packages:

**<<Project Path>>\.venv\Scripts\activate**

****

**Images Used:**

imgs>>ansel\_adams.jpg, ansel\_adams2.jpg, ansel\_adams3.jpg, ILSVRC2012\_val\_00041580.JPEG, ILSVRC2012\_val\_00046524.JPEG, ILSVRC2012\_val\_00046834.JPEG

**Code:**

import argparse

import matplotlib.pyplot as plt

from colorizers import \* #local package

# load colorizers

colorizer\_eccv16 = eccv16(pretrained=True).eval() colorizer\_siggraph17 = siggraph17(pretrained=True).eval()

# default size to process images is 256x256

# grab L channel in both original ("orig") and resized ("rs") resolutions img = load\_img('imgs/ansel\_adams.jpg')

(tens\_l\_orig, tens\_l\_rs) = preprocess\_img(img, HW=(256,256))

# colorizer outputs 256x256 ab map

# resize and concatenate to original L channel img\_bw = postprocess\_tens(tens\_l\_orig, torch.cat((0\*tens\_l\_orig,0\*tens\_l\_orig),dim=1))

out\_img\_eccv16 = postprocess\_tens(tens\_l\_orig, colorizer\_eccv16(tens\_l\_rs).cpu())

out\_img\_siggraph17 = postprocess\_tens(tens\_l\_orig, colorizer\_siggraph17(tens\_l\_rs).cpu())

plt.imsave('s\_eccv16.png', out\_img\_eccv16) plt.imsave('s\_siggraph17.png', out\_img\_siggraph17)

plt.figure(figsize=(12,8)) plt.subplot(2,2,1) plt.imshow(img) plt.title('Original') plt.axis('off')

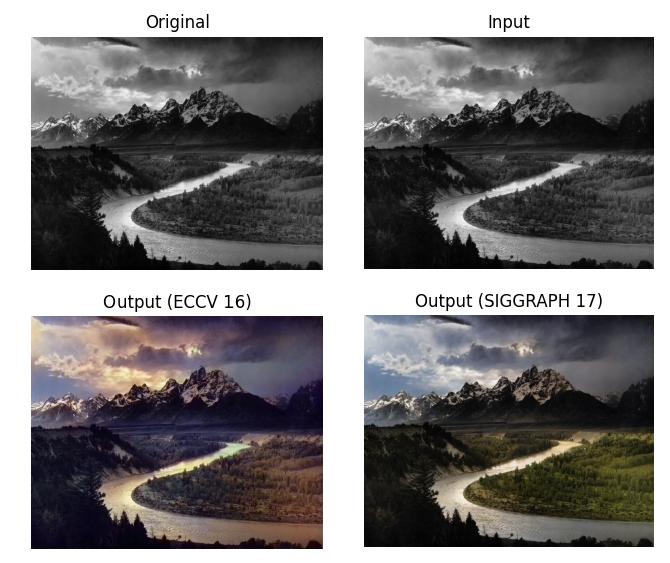
plt.subplot(2,2,2) plt.imshow(img\_bw) plt.title('Input') plt.axis('off')

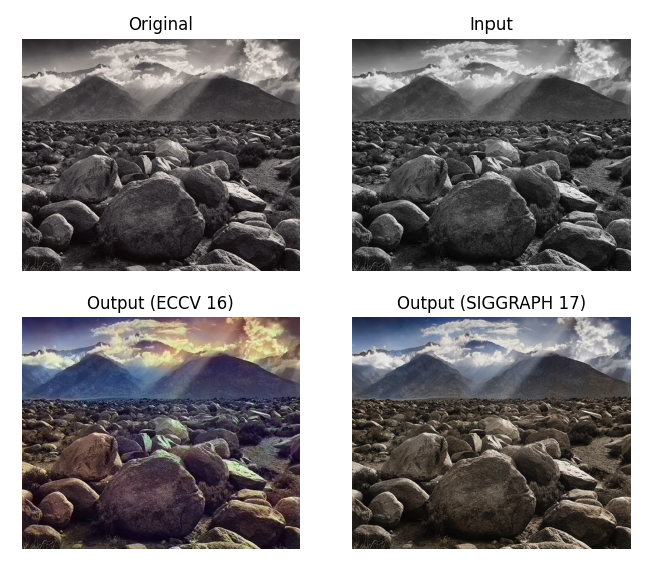
plt.subplot(2,2,3) plt.imshow(out\_img\_eccv16) plt.title('Output (ECCV 16)') plt.axis('off')

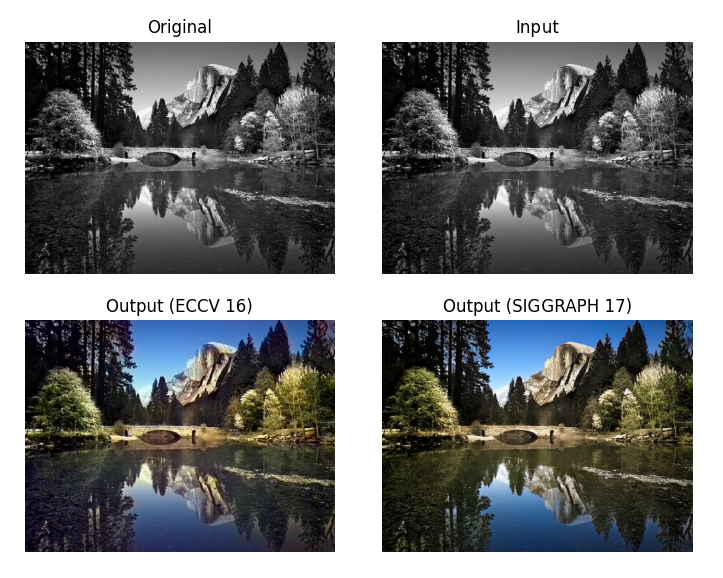
plt.subplot(2,2,4) plt.imshow(out\_img\_siggraph17) plt.title('Output (SIGGRAPH 17)') plt.axis('off')

plt.show() **Output:**

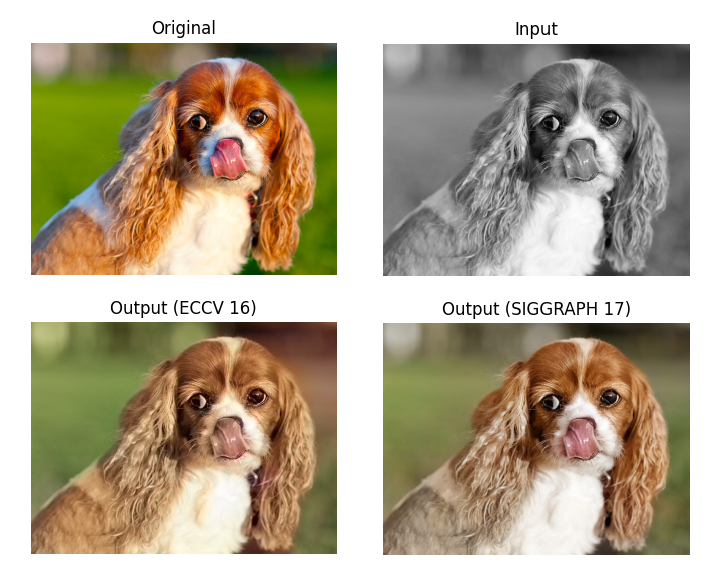
**Case – I (ansel\_adams.jpg)**



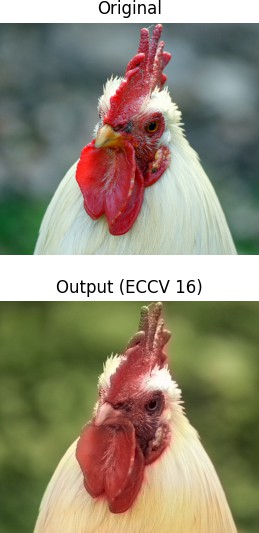
**Case – II (ansel\_adams2.jpg)**

**Case – III (ansel\_adams3.jpg)**

**Case – IV (ILSVRC2012\_val\_00041580.JPEG)**

****

**Case – V (ILSVRC2012\_val\_00046524.JPEG)**

****

**Case – VI (ILSVRC2012\_val\_00046834.JPEG)**

**Practical No. 4**

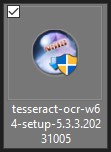
**Part A**

**Aim: Perform Text Detection and Recognition**

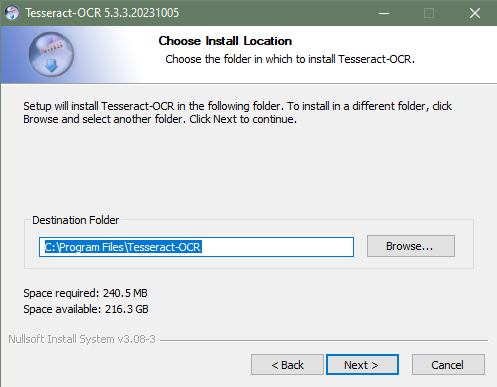
**Steps:**

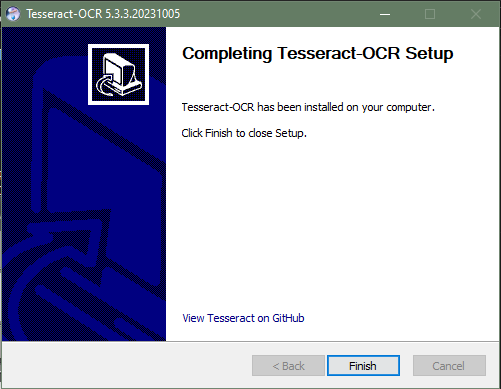
**Packages Used: cv2, pytesseract**

**Pytesseract Installation Guide:** Download the PyTesseract Installer



Check the Install Location&Finish the installation





**Command:**

****

**Images Used:**

img.jpg sample.jpg

**Code:**

import cv2

import pytesseract

# Mention the installed location of Tesseract OCR pytesseract.pytesseract.tesseract\_cmd=r'C:\Program Files\Tesseract-OCR\tesseract.exe'

img = cv2.imread("img.jpg")

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

ret, thresh1 = cv2.threshold(gray, 0, 255, cv2.THRESH\_OTSU | cv2.THRESH\_BINARY\_INV)

rect\_kernel = cv2.getStructuringElement(cv2.MORPH\_RECT,(20, 20)) dilation = cv2.dilate(thresh1, rect\_kernel, iterations=1)

contours, heirarchy = cv2.findContours(dilation, cv2.RETR\_EXTERNAL,

cv2.CHAIN\_APPROX\_NONE)

im2 = img.copy()

for cnt in contours:

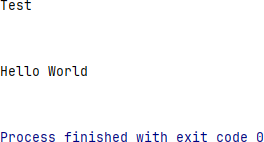
x, y, w, h = cv2.boundingRect(cnt)

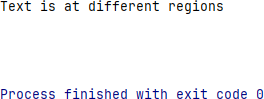
rect = cv2.rectangle(im2, (x, y), (x+w, y+h), (0,255,0), 2) cropped = im2[y: y+h, x: x+w]

text = pytesseract.image\_to\_string(cropped)

print(text) **Output:**

**Case – I (img.jpg):**

****

**Case – II (sample.jpg)**

**Part B**

**Aim: Perform Image Matting and Composting**

**Steps:**

**Packages Used: pymatting, numpy, cv2 Command:**

****

**Images Used:**

in\_lemur.png in\_lemur\_trimap.png in\_beach.png

**Code:**

from pymatting import \* import numpy as np import cv2

scale = 1.0

image = load\_image("in\_lemur.png", "RGB", scale, "box")

trimap = load\_image("in\_lemur\_trimap.png", "GRAY", scale, "box") # MATTING

# Estimate Alpha from Image and Trimap alpha = estimate\_alpha\_cf(image, trimap)

# Make gray background new\_background = np.zeros(image.shape) new\_background[:, :] = [0.5, 0.5, 0.5]

# Estimate foreground from image and alpha

foreground, background = estimate\_foreground\_ml(image, alpha, return\_background=True)

# Save Alpha save\_image("output1\_alpha.png", alpha) # Save Foreground

save\_image("output2\_foreground.png", foreground) # Save Background save\_image("output3\_background.png", background)

# Save Cutout

cutout = stack\_images(foreground, alpha) save\_image("output4\_cutout.png", cutout)

# COMPOSTING

cutout = cv2.imread("output4\_cutout.png", cv2.IMREAD\_UNCHANGED) new\_background = cv2.imread("in\_beach.jpg")

# Split the cutout into foreground and alpha matte foreground = cutout[:, :, :3]

am = cutout[:, :, 3] / 255.0

# Resize the new background to match the size of the foreground newbg\_resized = cv2.resize(new\_background, (foreground.shape[1], foreground.shape[0]))

# Create composite image

composite = np.zeros\_like(foreground, dtype=np.uint8)

# Composite the foreground onto the new background for c in range(3):

composite[:, :, c] = foreground[:, :, c] \* am + newbg\_resized[: , :, c] \* (1-am)

# Save the composite image cv2.imwrite("CompositeImageBeach.png", composite)

**Output: output1\_alpha.png**

**output2\_foreground.png**

**output3\_background.png**

**output4\_cutout.png**



**CompositeImageBeach.png**