# INDEX

|  |  |  |  |
| --- | --- | --- | --- |
| **Practical Units.** | **Name of Practical** | **Date** | **Sign** |
| 1 | 1. Perform Geometric transformations 2. Perform Image Stitching 3. Perform Camera Calibration |  |  |
| 2 | 1. Face detection 2. Object detection 3. Pedestrian detection 4. Face recognition 5. Construct 3D model from images 6. Implement object detection and tracking from video |  |  |
| 3 | 1. Perform Feature extraction using RANSAC 2. Perform Colorization |  |  |
| 4 | 1. Perform Text detection and recognition 2. Perform Image matting   and Composting |  |  |

## Practical 1

##### Perform Geometric Transformations

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):

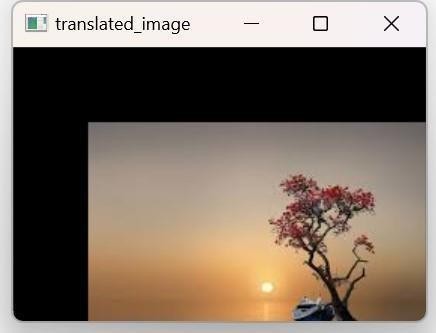
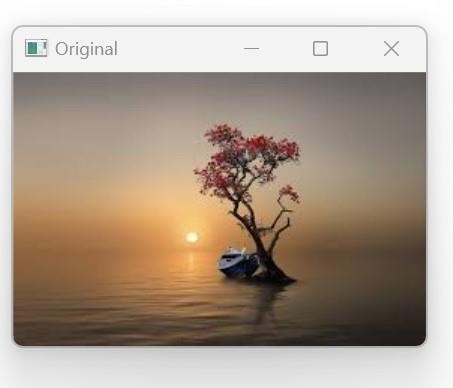
M = np.float32([[1, shear\_x, 0], [shear\_y, 1, 0]]) sheared\_image = cv2.warpAffine(image, M, (image.shape[1],

image.shape[0])) return sheared\_image

image = cv2.imread('img.jpg') 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) 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) **Output:**



****

##### Perform Image Stitching

from imutils import paths import numpy as np import argparse

import imutils import cv2

imagePaths = sorted(list(paths.list\_images('p2'))) #p2 is folder containing images

images=[]

for imagePath in imagePaths:

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

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

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

if status == 0:

# write the output stitched image to disk cv2.imwrite("p2output.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:**

[INFO] stitching images…



##### Perform Camera Caliberation

import cv2

import numpy as np import os

import glob chb = (6, 9)

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

threedpoints = [] twodpoints = []

objectp3d = np.zeros((1, chb[0] \* chb[1], 3), np.float32) objectp3d[0, :, :2] = np.mgrid[0:chb[0], 0:chb[1]].T.reshape(-1, 2) prev\_img\_shape = None

images = glob.glob('\*.jpg') for filename in images:

image = cv2.imread(filename)

grayColor = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY) ret, corners = cv2.findChessboardCorners(grayColor, chb,

cv2.CALIB\_CB\_ADAPTIVE\_THRESH

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

if ret == True:

threedpoints.append(objectp3d) corners2 = cv2.cornerSubPix(

grayColor, corners, (11, 11), (-1, -1), criteria) twodpoints.append(corners2)

image = cv2.drawChessboardCorners(image, chb, corners2, ret) cv2.imshow('img', image)

cv2.waitKey(0) cv2.destroyAllWindows() h, w = image.shape[:2]

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

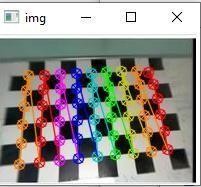
print(" Camera matrix:") print(matrix)

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

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

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

**Output:**



Camera matrix:

[[40.8618761 0. 83.80597971]

[ 0. 41.29134045 94.37718569]

[ 0. 0. 1. ]]

Distortion coefficient:

[[ 0.00051989 -0.00105068 -0.00512204 -0.00037702 0.00015534]]

Rotation Vectors:

(array([[-0.09670096], [ 0.06446519],

[ 1.50927077]]), array([[-0.15910316],

[-0.0178901 ],

[ 3.07539003]]), array([[-0.03866621],

[ 0.09626499],

[-0.07417175]]))

Translation Vectors:

(array([[ 4.65717678],

[-3.68545603],

[ 2.75350073]]), array([[2.34304791],

[4.05578132],

[2.76539468]]), array([[-3.12854143],

[-3.38972333],

[ 2.83149017]]))

## Practical 2

##### Face Detection

import cv2

img = cv2.imread('input1.jpg')

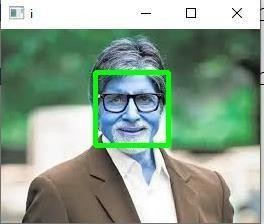
gray\_image = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY) face\_classifier = cv2.CascadeClassifier(

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

gray\_image, scaleFactor=1.1, minNeighbors=5, minSize=(40, 40)) 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)

**Output:**

****

Changing another image : img = cv2.imread('input3.jpg')



##### Object detection

import cv2

# Opening image

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

img\_gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY) img\_rgb = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB) stop\_data = cv2.CascadeClassifier('stop\_data.xml')

found = stop\_data.detectMultiScale(img\_gray,minSize =(20, 20)) amount\_found = len(found)

if amount\_found != 0:

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

cv2.rectangle(img\_rgb, (x, y),(x + height, y + width),(0, 255, 0), 5) cv2.imshow('imgobject',img\_rgb)

**Output:**

****

##### Pedestrain Detection

import cv2 import imutils

hog = cv2.HOGDescriptor() hog.setSVMDetector(cv2.HOGDescriptor\_getDefaultPeopleDetector()) image = cv2.imread('pedestrian1.png')

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

(regions, \_) = hog.detectMultiScale(image,winStride=(4, 4),padding=(4, 4),scale=1.05)

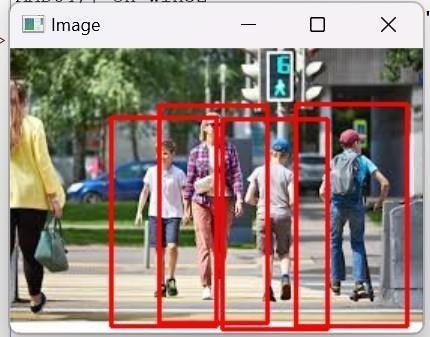
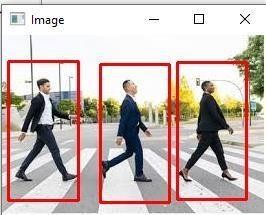
for (x, y, w, h) in regions:

cv2.rectangle(image,(x, y),(x + w, y + h),(0, 0, 255), 2) cv2.imshow("Image", image)

cv2.waitKey(0)

cv2.destroyAllWindows()

**Output :**

****

##### Face Recognition

Write command in Command Prompt:

pip install dlib-19.24.99-cp312-cp312-win\_amd64.whl (need the file to be downloaded)

pip install cmake

pip install face\_recognition pip install setuptools

Coding:

import cv2

import face\_recognition

imgmain = face\_recognition.load\_image\_file('random11.jpg') imgmain = cv2.cvtColor(imgmain, cv2.COLOR\_BGR2RGB) imgTest = face\_recognition.load\_image\_file('random12.jpg') imgTest = cv2.cvtColor(imgTest, 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:**

****

Changes

imgmain = face\_recognition.load\_image\_file('random11.jpg') imgmain = cv2.cvtColor(imgmain, cv2.COLOR\_BGR2RGB) imgTest = face\_recognition.load\_image\_file('random2.jpg') imgTest = cv2.cvtColor(imgTest, cv2.COLOR\_BGR2RGB)



##### Construct 3D model from images

from PIL import Image import numpy as np

def shift\_image(img, depth\_img, shift\_amount=10): img = img.convert("RGBA")

data = np.array(img)

depth\_img = depth\_img.convert("L") depth\_data = np.array(depth\_img)

deltas = ((depth\_data / 255.0) \* float(shift\_amount)).astype(int) shifted\_data = np.zeros\_like(data)

height, width, \_ = data.shape for y, row in enumerate(deltas):

for x, dx in enumerate(row):

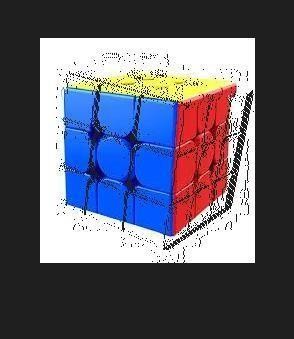
if x + dx < width and x + dx >= 0: shifted\_data[y, x + dx] = data[y, x]

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:**

****

##### Object detection from video

import cv2

face\_classifier = cv2.CascadeClassifier(

cv2.data.haarcascades + "haarcascade\_frontalface\_default.xml") video\_capture = cv2.VideoCapture('a.mp4')

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+100, y+h+100), (0,0,255), 4) return faces

while True:

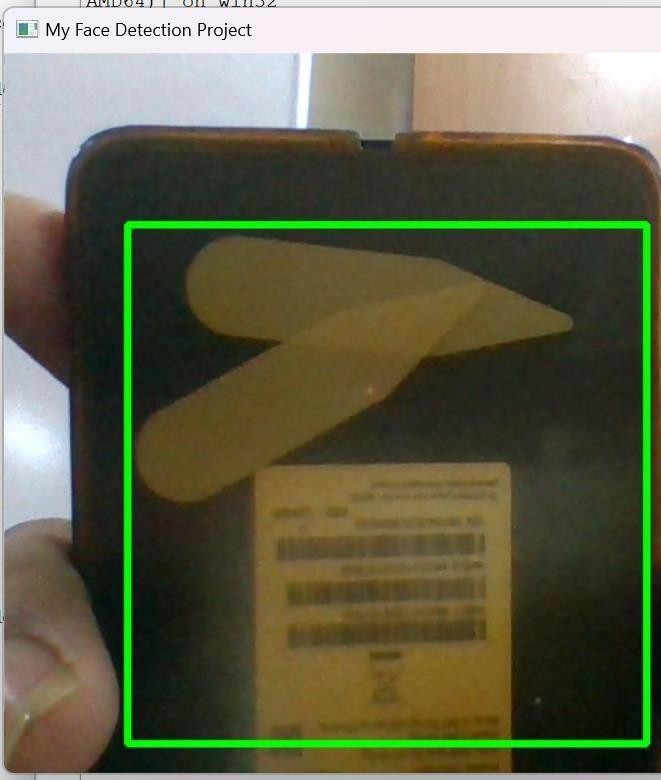
result, video\_frame = video\_capture.read() if result is False:

break

faces = detect\_bounding\_box(video\_frame) cv2.imshow("My Object Detection Project", video\_frame) if cv2.waitKey(1) & 0xFF == ord("q"):

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

**Output:**

****

## Practical 3

##### RANSAC Implementation

**linearleastsqaure.py**

import numpy as np

class LinearLeastSqaureModel: 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) return model

**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)

while num\_iterations > iterations\_done: np.random.shuffle(total\_data) sample\_data = total\_data[:num\_sample, :]

estimated\_model = self.curve\_fitting\_model.fit(sample\_data[:,:-1],

sample\_data[:, -1:]) y\_cap = A.dot(estimated\_model)

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

inlier\_count = np.count\_nonzero(err < threshold) 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\_inlier\_count: ', max\_inlier\_count) return best\_model

**Modelfitting.py:**

**Ransac Implementation Coding:**

import numpy as np import math

import pandas as pd

from matplotlib import pyplot as plt from ransac import RansacModel

from linearleastsquare import LinearLeastSqaureModel def fit\_curve(data):

x\_values = np.array(data['x']) y\_values = np.array(data['y']) x\_sq = np.power(x\_values, 2)

A = np.stack((x\_sq, x\_values, np.ones((len(x\_values)), dtype = int)), axis = 1) threshold = np.std(y\_values)/2

linear\_ls\_model = LinearLeastSqaureModel() linear\_ls\_model\_estimate = linear\_ls\_model.fit(A, y\_values) linear\_model\_y = A.dot(linear\_ls\_model\_estimate) 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 ':

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:**

# inliers: 117

# prob\_outlier: 0.532

# s: 1

# n: 27.700875935823316

# max\_inlier\_count: 117

# inliers: 175

# prob\_outlier: 0.30000000000000004

# s: 2

# n: 7.131485905524426

# max\_inlier\_count: 175

# inliers: 76

# prob\_outlier: 0.696

# s: 3

# n: 105.1257167640639

# max\_inlier\_count: 175

# inliers: 134

# prob\_outlier: 0.46399999999999997

# s: 4

# n: 17.91439390915469

# max\_inlier\_count: 175

# inliers: 187

# prob\_outlier: 0.252

# s: 5

# n: 5.525552495791251

# max\_inlier\_count: 187

# inliers: 68

# prob\_outlier: 0.728

# s: 6

# n: 147.36332180032895

# max\_inlier\_count: 187

# inliers: 64

# prob\_outlier: 0.744

# s: 7

# n: 177.05746802114584

# max\_inlier\_count: 187

# inliers: 87

# prob\_outlier: 0.652

# s: 8

# n: 69.57430607360442

# max\_inlier\_count: 187

# inliers: 72

# prob\_outlier: 0.712

# s: 9

# n: 123.90418270340892

# max\_inlier\_count: 187

# inliers: 145

# prob\_outlier: 0.42000000000000004

# s: 10

# n: 13.801901471212751

# max\_inlier\_count: 187

# inliers: 58

# prob\_outlier: 0.768

# s: 11

# n: 238.4038555094377

# max\_inlier\_count: 187

# inliers: 118

# prob\_outlier: 0.528

# s: 12

# n: 26.963389844145

# max\_inlier\_count: 187

# inliers: 63

# prob\_outlier: 0.748

# s: 13

# n: 185.69618036762273

# max\_inlier\_count: 187

# inliers: 168

# prob\_outlier: 0.32799999999999996

# s: 14

# n: 8.283822964377974

# max\_inlier\_count: 187

# inliers: 204

# prob\_outlier: 0.18400000000000005

# s: 1

# n: 3.821999420219228

# max\_inlier\_count: 204

# inliers: 115

# prob\_outlier: 0.54

# s: 2

# n: 29.25380161992817

# max\_inlier\_count: 204

# inliers: 143

# prob\_outlier: 0.42800000000000005

# s: 3

# n: 14.457625659547233

# max\_inlier\_count: 204

# inliers: 91

# prob\_outlier: 0.636

# s: 4

# n: 60.60513156707005

# max\_inlier\_count: 204

# inliers: 120

# prob\_outlier: 0.52

# s: 5

# n: 25.561028720085947

# max\_inlier\_count: 204

# inliers: 184

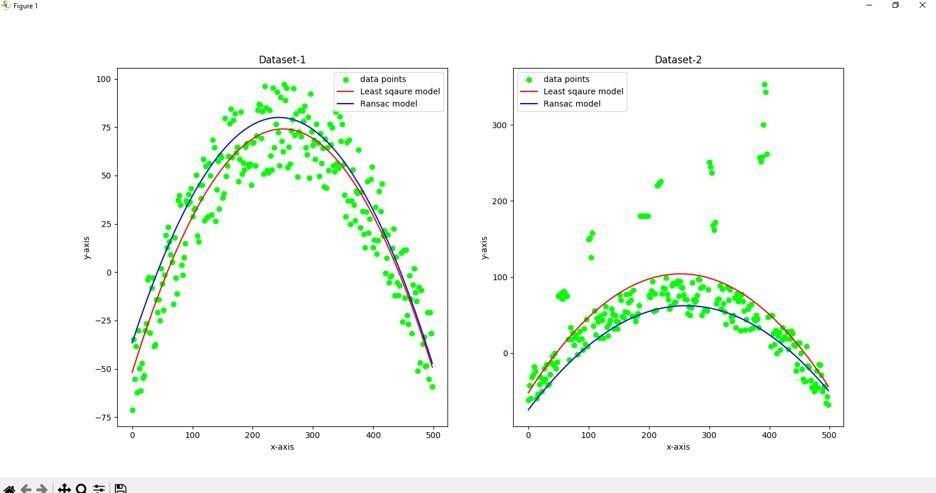
# prob\_outlier: 0.264

# s: 6

# n: 5.889670195389181

# max\_inlier\_count: 204

**Graph:**

****

##### Colorization

import argparse

import matplotlib.pyplot as plt

from colorizers import \* #local package c1 = eccv16(pretrained=True).eval()

c2 = siggraph17(pretrained=True).eval() img = load\_img('testbw1.jpg')

(orig,rs) = preprocess\_img(img, HW=(256,256))

img\_bw = postprocess\_tens(orig, torch.cat((0\*orig,0\*orig),dim=1)) img1 = postprocess\_tens(orig, c1(rs).cpu())

img2 = postprocess\_tens(orig, c2(rs).cpu()) plt.imsave('s1.png', img1) plt.imsave('s2.png', img2) plt.figure(figsize=(12,8))

plt.subplot(2,2,1) plt.imshow(img) plt.title('Original Image') plt.axis('off') plt.subplot(2,2,2) plt.imshow(img\_bw)

plt.title('Input Black and White') plt.axis('off')

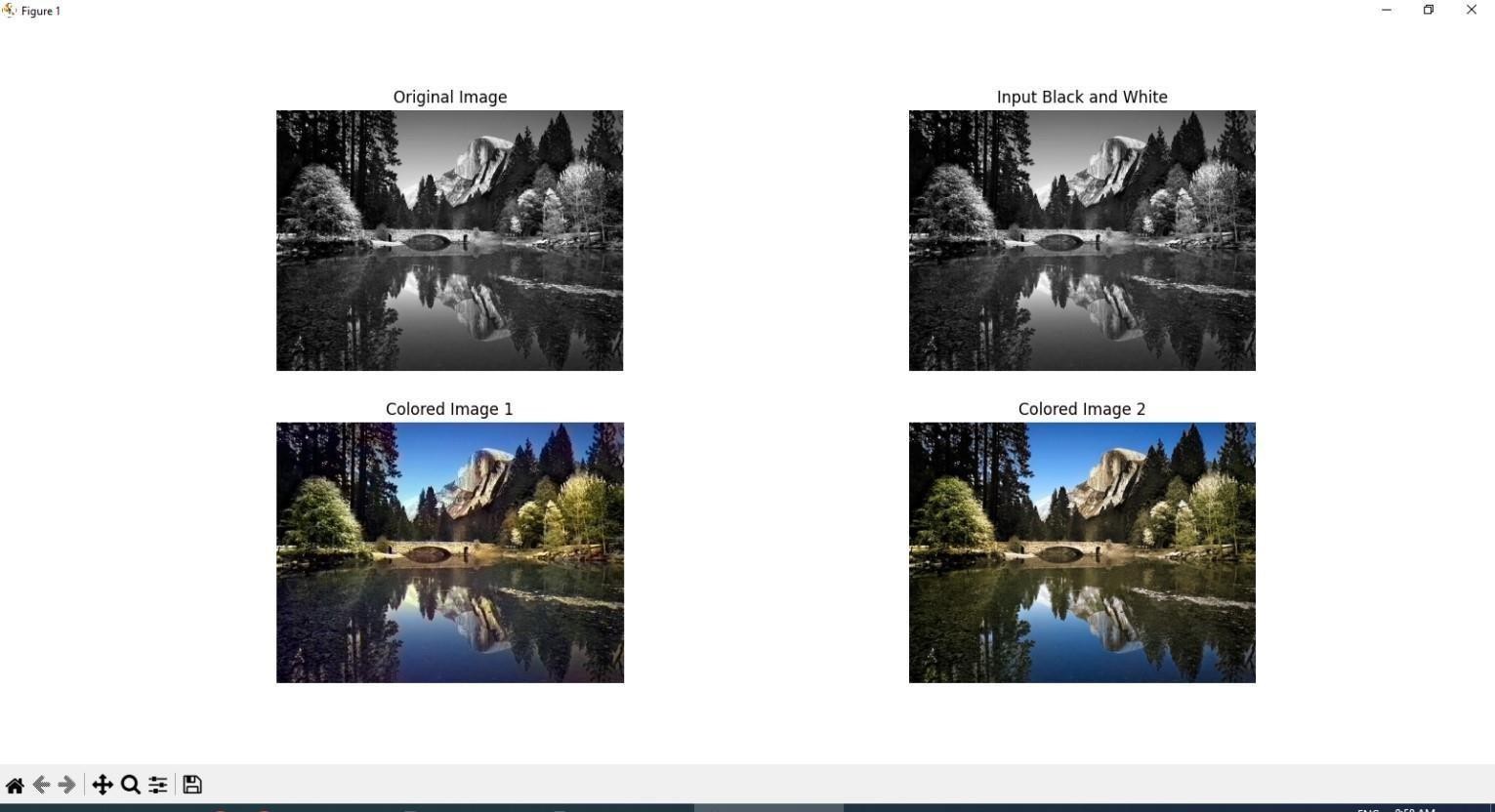
plt.subplot(2,2,3)

plt.imshow(img1) plt.title('Colored Image 1') plt.axis('off') plt.subplot(2,2,4)

plt.imshow(img2) plt.title('Colored Image 2') plt.axis('off')

plt.show()

**Output:**

****

## Practical 4

##### Text Detection And Recognition from Image

import cv2

import pytesseract

pytesseract.pytesseract.tesseract\_cmd = r'C:\Program Files\Tesseract- OCR\tesseract.exe'

img = cv2.imread("imgtest1.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, hierarchy = 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:**

****

1. **Perform Image matting and Composting**

from pymatting import \* import numpy as np import cv2

scale = 1.0

image = load\_image("4b1.png", "RGB", scale, "box") trimap = load\_image("4b2.png", "GRAY", scale, "nearest") alpha = estimate\_alpha\_cf(image, trimap) new\_background = np.zeros(image.shape) new\_background[:, :] = [0.5, 0.5, 0.5]

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

save\_image("output1.png", alpha) save\_image("output2.png", foreground) save\_image("output3.png", background) cutout = stack\_images(foreground, alpha) save\_image("output4.png", cutout)

cutout = cv2.imread('output4.png', cv2.IMREAD\_UNCHANGED) new\_background = cv2.imread('4b3.png')

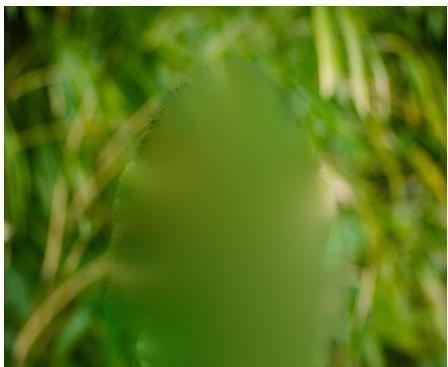
foreground = cutout[:, :, :3] am = cutout[:, :, 3] / 255.0

newbg\_resized = cv2.resize(new\_background, (foreground.shape[1], foreground.shape[0]))

composite = np.zeros\_like(foreground, dtype=np.uint8) for c in range(3):

composite[:, :, c]=foreground[:,:,c]\*am+newbg\_resized[:,:,c]\*(1-am) cv2.imwrite('composite\_image.png', composite)

**Output:**



****