Practical No: 1

A. Aim: Perform Geometric Transformation

Tool Used: MATLAB

Objectives:

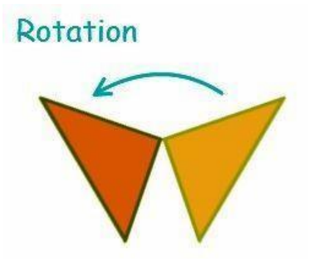
1. To understand the basic concept of geometric transformation.
2. To use MATLAB as a tool for performing basic image operations.

Theory:

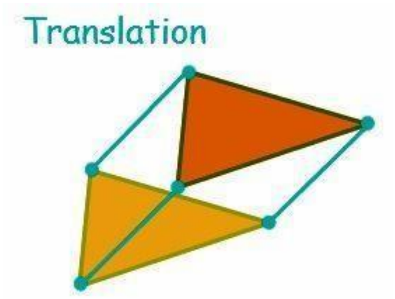
I. **Geometric Transformation** : An object with a geometric structure on its own or another set is bijective in the geometric transformation. The transformation of a shape modifies its appearance. Subsequently, the form might resemble or be consistent with its counterpart. A shift in an object's appearance is the true definition of transformations. Transformations can be broadly classified into four categories.

a) Rotation
b) Translation
c) Scaling
d) Reflection

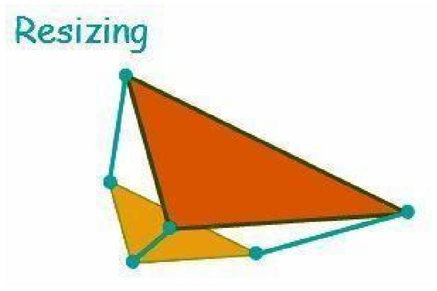
Rotation: This type of transformation has an object about a fixed point without changing its size or shape.



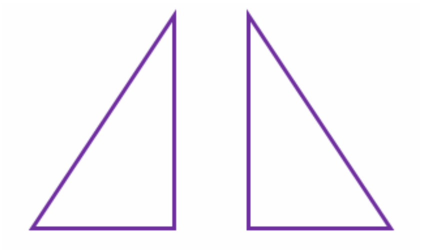
Translation: This type of translation is defined as moving the object in space by keeping its size, shape or orientation constant.



Scaling: This type of translation expands or contracts the object by keeping its orientation or shape the same. This is also known as resizing.



Reflection: This type of translation is called reflection because it flips the object across a line by keeping its shape or size constant.

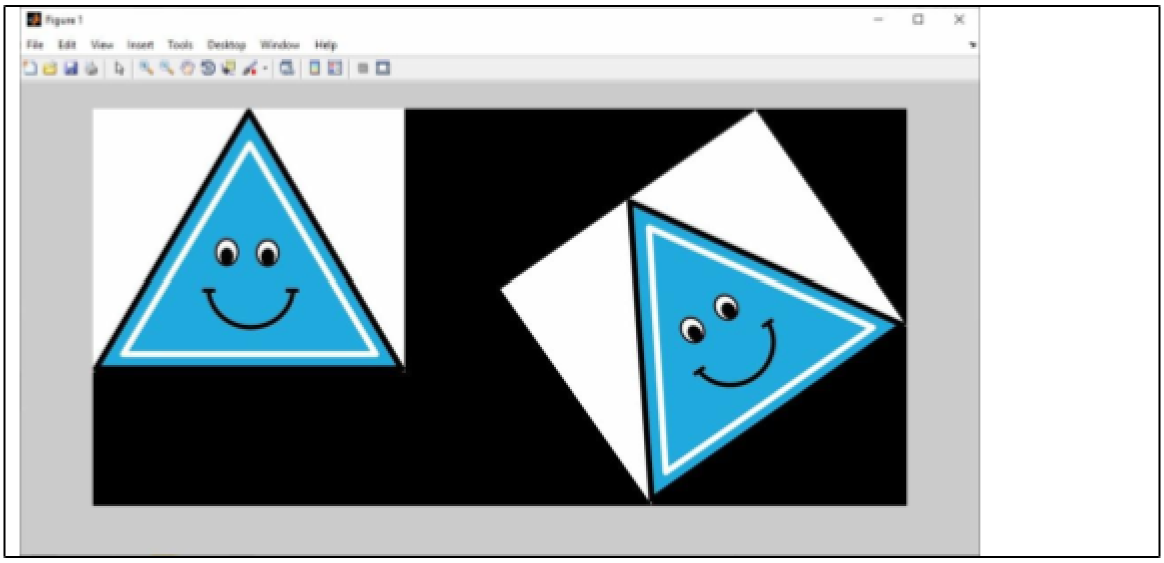


MATLAB Code:

1. Rotation:

```
%Read an image into the workspace.
I = imread('E:\Ms IT_I\Computer Vision\traingle.jpg');

%Rotate the image 35 degrees counterclockwise using bilinear
interpolation.
J = imrotate(I,35,'bilinear');

%Display the original image and the rotated image. By default, the
output image is large enough to include the entire original image.
Pixels that fall outside the boundaries of the original image are set
to 0 and appear as a black background in the output image. figure
imshowpair(I,J,'montage')
```

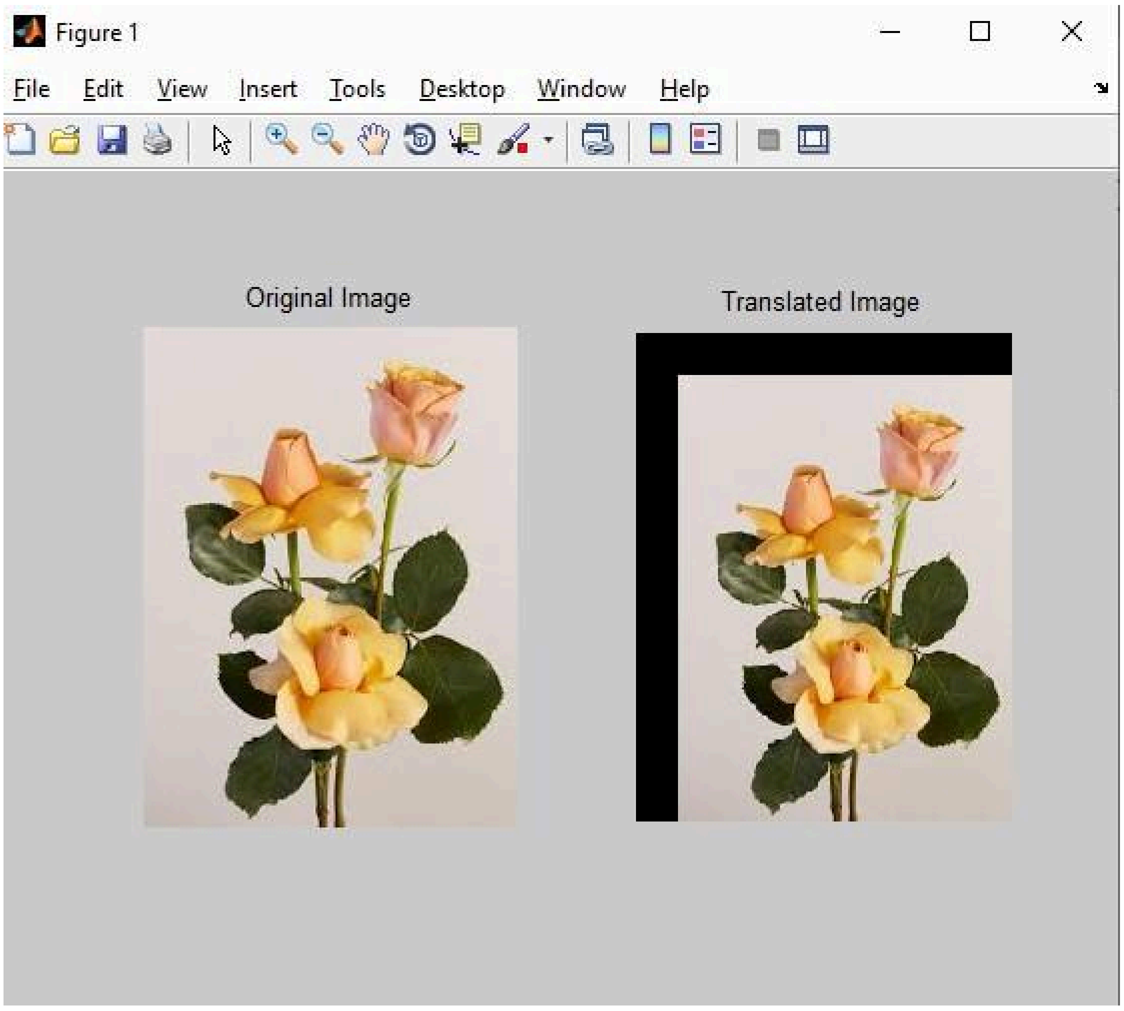
Output:



2. Translation:

```
I = imread('rose.jpg'); shiftX = 25; % shift
columns shiftY = 25; % shift rows
subplot(1,2,
);
imshow(I); title('Original Image');



% Assigning empty matrix for result, expected to be shiftX-1 larger in rows and shiftY-1
larger in columns
nI = uint8( zeros(size(I,1)+shiftY-1, size(I,2)+shiftX-1, size(I,3)));

% Translate
nI(shiftY:end, shiftX:end, :) = I;
subplot(1,2,2);
imshow(nI);
title('Translated Image');
```

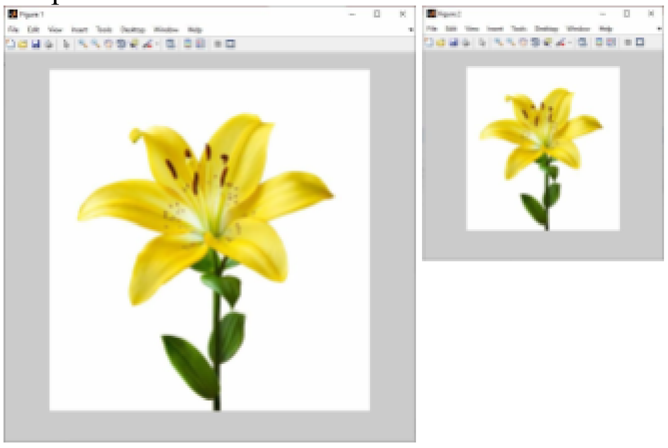
output:



3. Scaling:

```
    I = imread('E:\Ms IT_I\Computer Vision\lilly.jpg');
    J = imresize(I, 0.5);
    figure, imshow(I), figure, imshow(J)
```

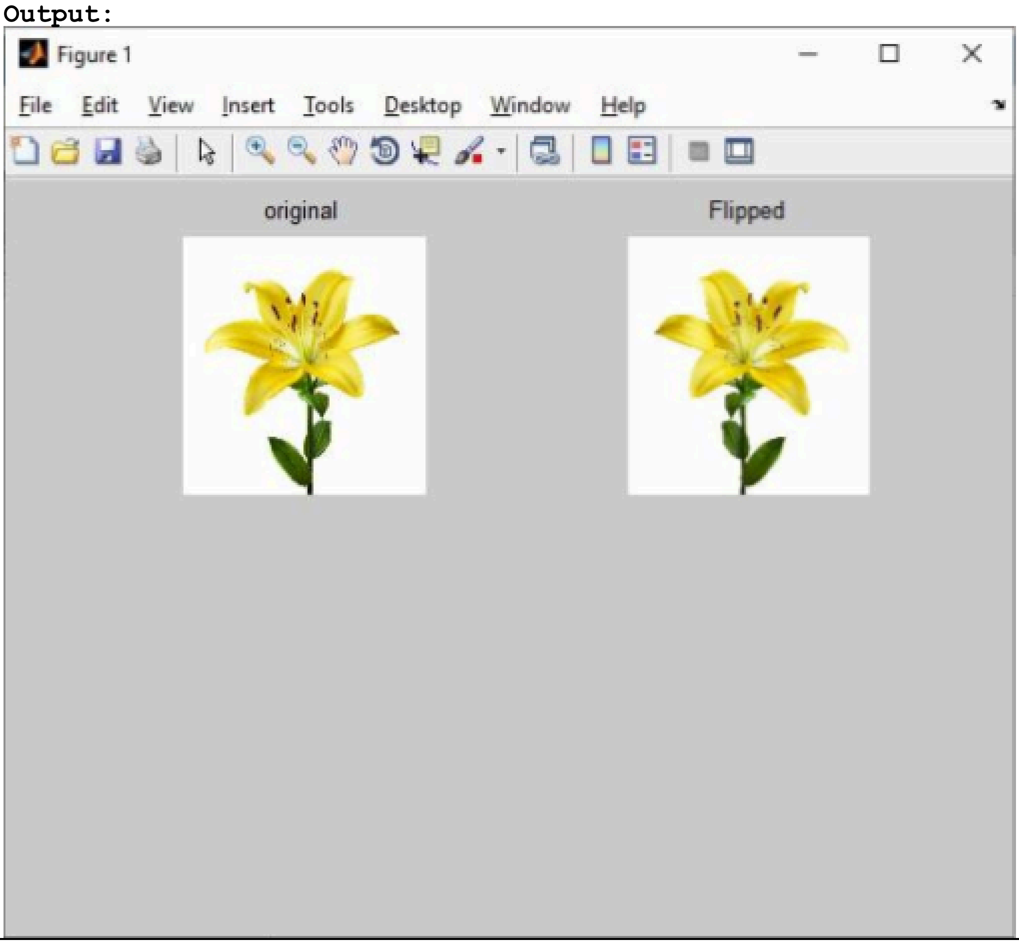
Output:



4. Reflection

```
% Read the target image file
img = imread('E:\Ms IT_I\Computer Vision\rose.jpg');
subplot(2,2,1)
imshow(img)
title ('original')

Ir=flipdim(I,2)

subplot(2,2,2)
imshow(Ir)
title ('Flipped')
```

Output:



B: Image Stitching

Tools Used:MATLAB

Objective:

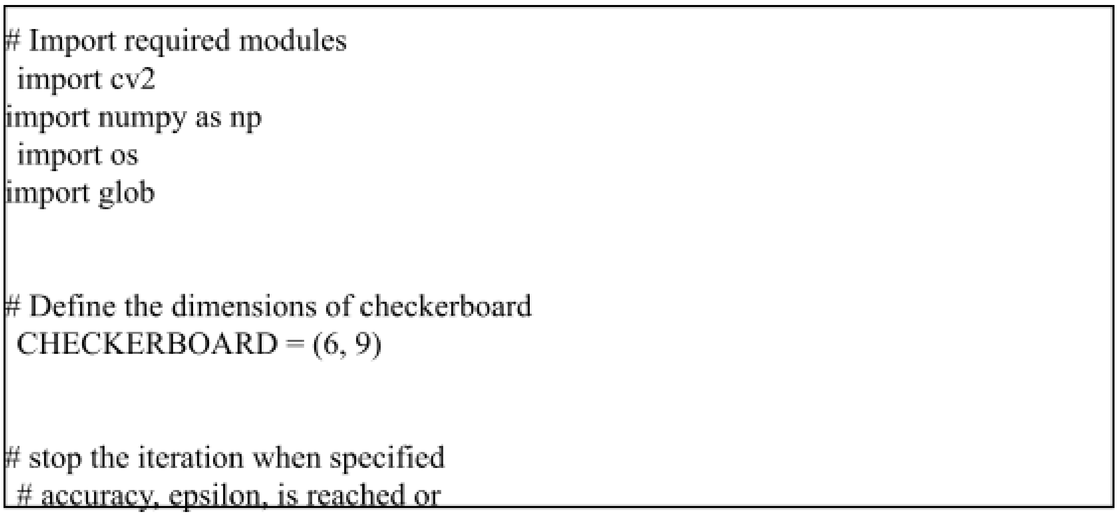
1. To understand the basic concept of geometric transformation.
2. To use MATLAB as a tool for performing basic image operations.

Procedure:

```
clc
close all
clear all

a= imread('E:\Ms IT_I\Computer Vision\1p.jpg'); %read the left part of the
image

b=imread('E:\Ms IT_I\Computer Vision\1q.jpg'); %read the right part of the
image


sa= size(a); %get the size of the left image
sb = size(b);%get the size of the left image

b= imresize(b,[sa(1) sa(2)]); %now resize 'b' as per the size of 'a' in order to get
perfect sized image
c= [a b]; % club the image a & b into another new image after making their size
same

%note the missing semicolon in the above line inside bracket
imshow(c) % show the image
```

C: Camera Calibration:

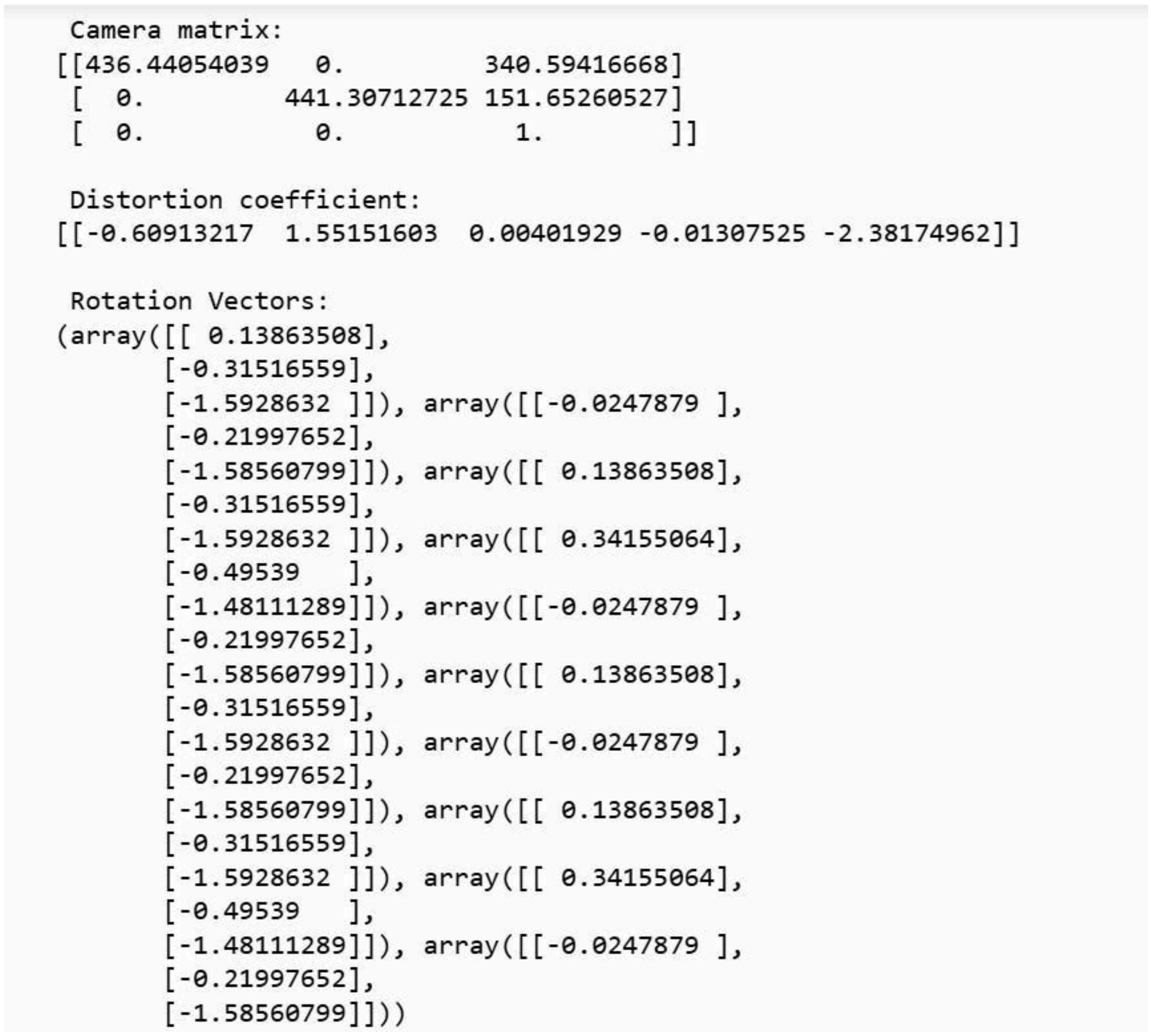
Tool used: Jupyter Notebook
Objectives:



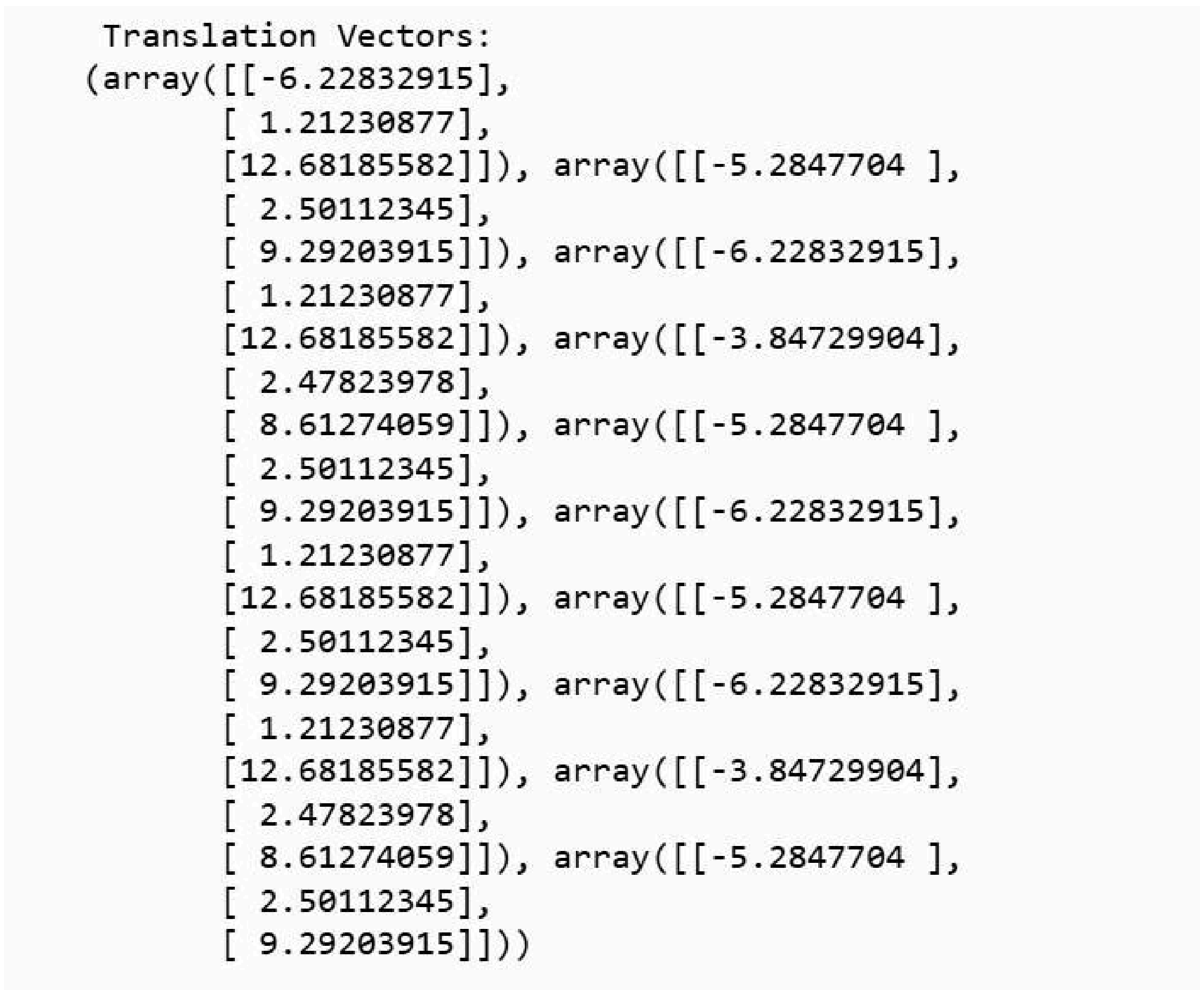
```
# Import required modules
 import cv2
import numpy as np
 import os
import glob


# 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:CHECK RBOARD[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)

   # 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)
```

```
      # Draw and display the corners
      image = cv2.drawChessboardCorners(image,
                                 CHECKERBOARD,
                                 corners2, ret)

   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, g ayColor.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)
```



```
 Camera matrix:
[[436.44054039   0.         340.59416668]
 [  0.         441.30712725 151.65260527]
 [  0.           0.           1.        ]]

 Distortion coefficient:
[[-0.60913217  1.55151603  0.00401929 -0.01307525 -2.38174962]]

 Rotation Vectors:
(array([[ 0.13863508],
       [-0.31516559],
       [-1.5928632 ]]), array([[-0.0247879 ],
       [-0.21997652],
       [-1.58560799]]), array([[ 0.13863508],
       [-0.31516559],
       [-1.5928632 ]]), array([[ 0.34155064],
       [-0.49539   ],
       [-1.48111289]]), array([[-0.0247879 ],
       [-0.21997652],
       [-1.58560799]]), array([[ 0.13863508],
       [-0.31516559],
       [-1.5928632 ]]), array([[-0.0247879 ],
       [-0.21997652],
       [-1.58560799]]), array([[ 0.13863508],
       [-0.31516559],
       [-1.5928632 ]]), array([[ 0.34155064],
       [-0.49539   ],
       [-1.48111289]]), array([[-0.0247879 ],
       [-0.21997652],
       [-1.58560799]]))
```



```
 Translation Vectors:
(array([[-6.22832915],
       [ 1.21230877],
       [12.68185582]]), array([[-5.2847704 ],
       [ 2.50112345],
       [ 9.29203915]]), array([[-6.22832915],
       [ 1.21230877],
       [12.68185582]]), array([[-3.84729904],
       [ 2.47823978],
       [ 8.61274059]]), array([[-5.2847704 ],
       [ 2.50112345],
       [ 9.29203915]]), array([[-6.22832915],
       [ 1.21230877],
       [12.68185582]]), array([[-5.2847704 ],
       [ 2.50112345],
       [ 9.29203915]]), array([[-6.22832915],
       [ 1.21230877],
       [12.68185582]]), array([[-3.84729904],
       [ 2.47823978],
       [ 8.61274059]]), array([[-5.2847704 ],
       [ 2.50112345],
       [ 9.29203915]]))
```

Practical No: 2

A. Face Detection:

In this practical session we will learn how to detect face, using the MATLAB built-in class and function. Based on Viola-Jones face detection algorithm, the computer vision system toolbox contains vision.CascadeObjectDetector System object which detects objects based on above mentioned algorithm.

Tool Used: MATLAB

Objectives:

1. To learn Matlab Function to detect faces
2. To acquire knowledge about computer vision using Matlab

Explanation of commands :

1. **Line 8: " hold on"** - Matlab's 'hold' command determines whether the newly created graphic object will be added to the existing graph or will it replace the existing objects in our graph. The command 'hold on' is used to retain our current plot & its axes properties in order to add subsequent graphic commands to our existing graph.

2. **Line 9: "i = 1:size(BB,1)"** – size(obj,dim) returns the length of the dimension specified by the scalar dim. For example, size(obj,1) returns the number of rows.

Procedure:

1.Clear the console :

```
clear all
clc
```

2. Detect objects(Face) using Viola-Jones Algorithm:

```
FDetect = vision.CascadeObjectDetector;
```

3. Read Input Image :

```
I = imread('HarryPotter.jpg');
```

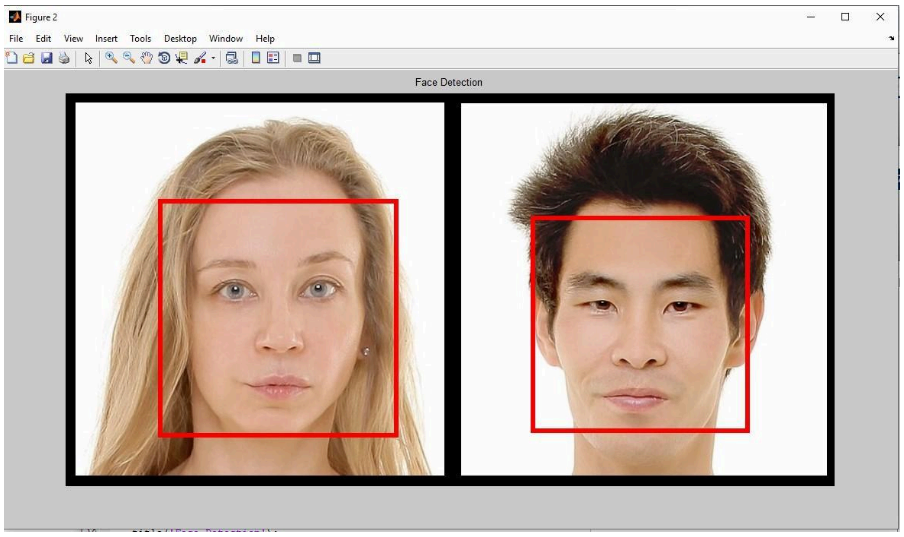
4. Returns Bounding Box values based on number of objects:

```
BB = step(FDetect,I);
```

5. Plot the rectangles around the detected faces:

```
figure,
imshow(I);
hold on
for i = 1:size(BB,1)
      rectangle('Position',BB(i,:),'LineWidth',5,'LineStyle','-
','EdgeColor','r');
end
title('Face Detection');
hold off;
```

Output:



B Object Detection

C Pedestrian Detection

D Face Recognition:

Note: keep video file and matlab code in the same directory

Tool Used: MATLAB

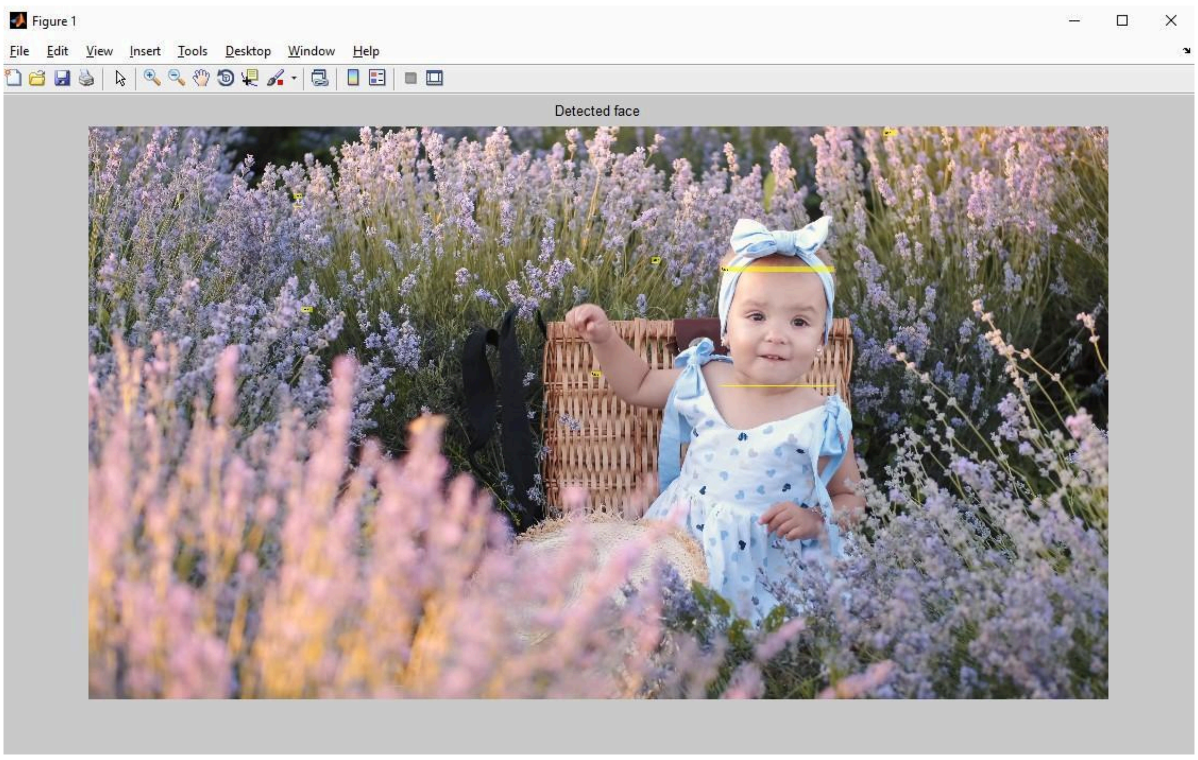
Objectives:

3. To learn Matlab Function to recognise faces
4. To acquire knowledge about computer vision using Matlab

Procedure:

```
% Create a cascade
detector object.
faceDetector =
vision.CascadeObjectDetect
or();
% Read a video frame and run
the detector. videoFileReader =
vision.VideoFileReader(baby.avi
'); videoFrame  =
step(videoFileReader);
bbox          = step(faceDetector,
videoFrame);
% Draw the returned bounding box
around the detected face.

videoOut = insertObjectAnnotation(videoFrame,'rectangle',bbox,'Face');
figure, imshow(videoOut), title('Detected face');
```

Output:



Practical No 3

A:Perform Feature extraction using RANSAC

B:Perform Colorization

Practical No 4

A: Perform Text Detection and recognition

Tool: Jupyter Notebook

Objectives:

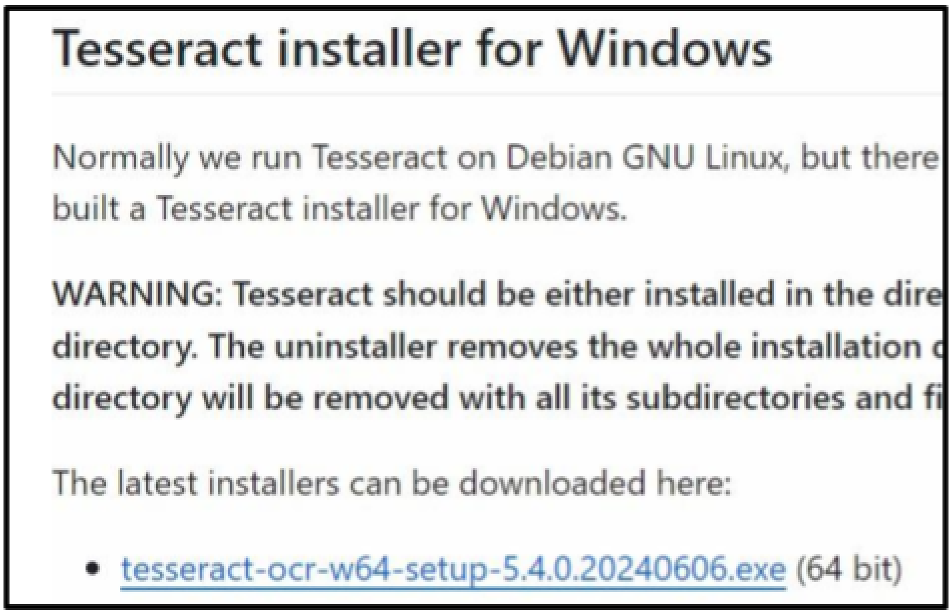
1. **To learn python libraries**

2. **To acquire the knowledge about "pytesseract"**
3. **To detect and recognise text from**

image Procedure:

Step 1: Download and Install "pytesseract.exe" on computer

[Note: you can download application using this link "https://github.com/UB-Mannheim/tesseract/wiki")



Tesseract installer for Windows

Normally we run Tesseract on Debian GNU Linux, but there
built a Tesseract installer for Windows.

WARNING: Tesseract should be either installed in the dire
directory. The uninstaller removes the whole installation d
directory will be removed with all its subdirectories and fi

The latest installers can be downloaded here:

- tesseract-ocr-w64-setup-5.4.0.20240606.exe (64 bit)

(Location after installation: C:\Program Files\Tesseract-OCR)

Step 2: Install below packages

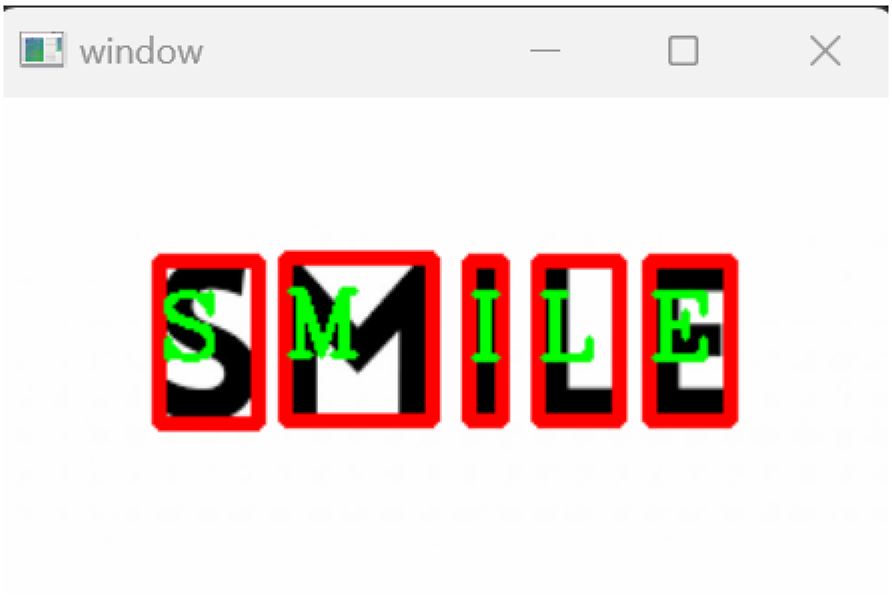
```
pip install pytesseract pip
install opencv-python
```

Step 3:Write below script

```
#import necessary packages

import cv2
import pytesseract as pytesseract

#locate the installed tesseract application
pytesseract.pytesseracprintt.tesseract_cmd =r'C:/Program Files/Tesseract-
OCR/tesseract.exe'

#read image
img=cv2.imread("text_img.png")

#read height and width of image
height,width,c=img.shape

#extract the words from image
words_in_image=pytesseract.image_to_string(img)
print(words_in_image)

#draw boxes around the words
letter_boxes=pytesseract.image_to_boxes(img)

#fix the boxes size
for box in letter_boxes.splitlines():
    box=box.split()
    x,y,w,h=int(box[1]),int(box[2]),int(box[3]),int(box[4])
    cv2.rectangle(img,(x,height-y),(w,height-h),(0,0,255),3)
    cv2.putText(img,box[0], (x,height-
h+32),cv2.FONT_HERSHEY_COMPLEX,1,(0,255,0),2) #recognize display
letters in green colour

#display image
```

```
cv2.imshow("window",img)
cv2.waitKey(0)
```

output:



B: Perform Image Matting and Composition

Image Matting is the process of extracting the foreground object from an image, while preserving fine details such as hair or fur. It aims to create a new image with a transparent background, allowing the isolated object to be placed on a different background or composited with other images seamlessly.

Image Composition refers to the process of combining multiple images into a single image, often to create a new scene or visual effect. This can include placing objects from one image into another, adjusting lighting and color to match the new environment, and ensuring that the final composition looks realistic and visually appealing.

Tools: Jupyter Notebook

Programming Language: Python

Objective:

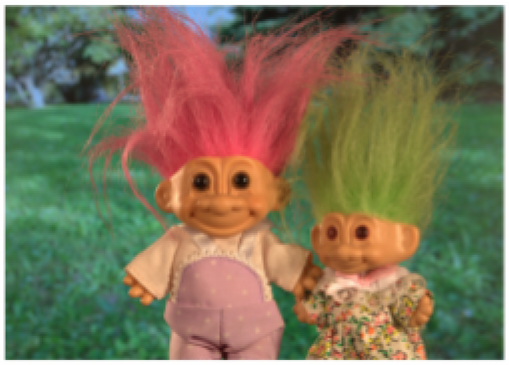
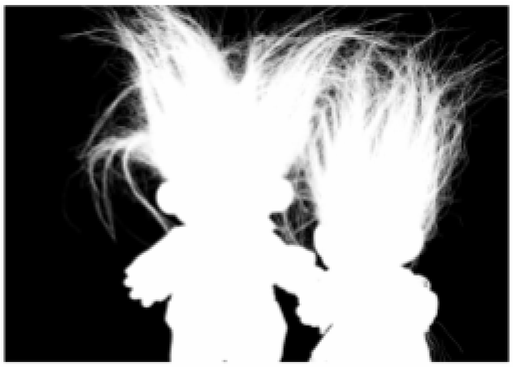
1. To learn the concept of image matting

2. To learn the concept of image composition

Procedure:

```
import cv2

import matplotlib.pyplot as plt

import numpy as np



I = cv2.imread('GT04.png')

I = cv2.cvtColor(I, cv2.COLOR_BGR2RGB)/255                    # convert to RGB and
normalize

plt.imshow(I), plt.axis('off')                         # plot I

plt.show();

alpha_ex = cv2.imread('GT04_alpha.png', cv2.IMREAD_GRAYSCALE)/255 # load exact
alpha and normalize

plt.imshow(alpha_ex, cmap='gray'), plt.axis('off')          # plot alpha

plt.show();
```

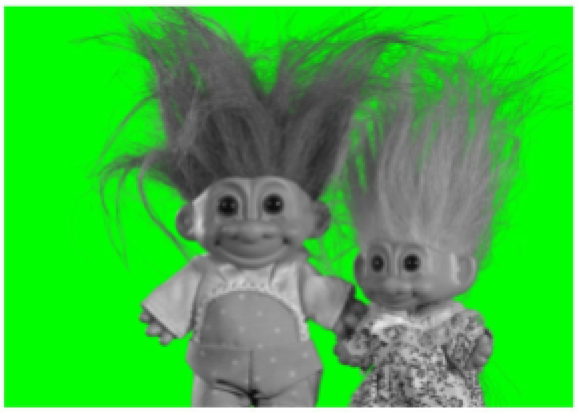
O/P:

#create a gray version of the image with a green background as follows:

```
n_rows = I.shape[0]                          # number of rows in I

n_cols = I.shape[1]                        # number of columns in I

n_pixels = n_rows * n_cols                  # number of pixels in I

I = np.reshape(I, (n_pixels, 3))

I[:, 0] = I[:, 1]                      # red = green

I[:, 2] = I[:, 1]                      # blue = green

alpha_ex = np.reshape(alpha_ex, (n_pixels, 1))

G_B = 1                                # green screen value

I = alpha_ex * I + (1 - alpha_ex) * [0, G_B, 0] # replace background with green screen


I = np.reshape(I, (n_rows, n_cols, 3))

plt.imshow(I), plt.axis('off')                # plot I

plt.show();
```

```
O/P:
```



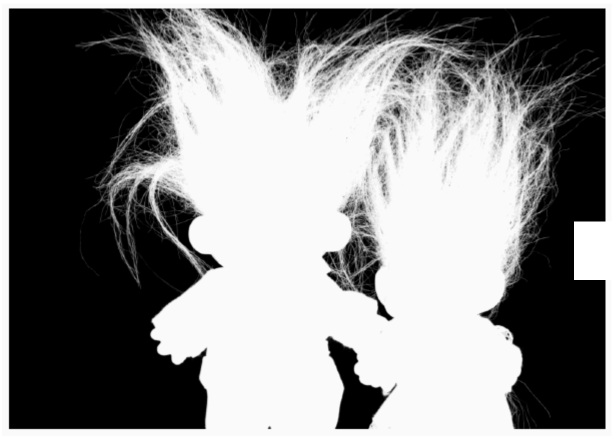
#extract the RGB components of *I*:

```
R_I = I[:, :, 0] # red component of I

G_I = I[:, :, 1] # green component of I
```

```
B_I = I[:, :, 2] # blue component of I
```

compute the matte α:

```
alpha = (R_I - (G_I - G_B))/G_B                  # compute alpha with formula
plt.imshow(alpha, cmap='gray'), plt.axis('off') # plot alpha
plt.show();
```



#The relative error between our matte and the "exact" one is around machine precision:

```
alpha = np.reshape(alpha, (n_pixels, 1))
error = np.linalg.norm(alpha - alpha_ex)/np.linalg.norm(alpha_ex)
print(f'Error (alpha):  {error:.2e}')
```

```
Error (alpha): 3.63e-17
```

#We're going to replace the green screen with the following image of Toronto's skyline:

```
K = cv2.imread('toronto.jpg')                 # load image (BGR format)
K = cv2.cvtColor(K, cv2.COLOR_BGR2RGB)/255   # convert to RGB and normalize
K = K[:n_rows, :n_cols, :]                   # adjust size of K to size of I
plt.imshow(K), plt.axis('off')               # plot K
plt.show();
```

```
K = np.reshape(K, (n_pixels, 3))
```

```
#Let's now replace the green screen with the image K:

R_I = np.reshape(R_I, (n_pixels, 1))

J = np.tile(R_I, 3) + (1 - alpha) * K   # new image J with different background K

J = np.reshape(J, (n_rows, n_cols, 3))

plt.imshow(J), plt.axis('off')          # plot

J plt.show();
```

MSC IT SEM II - C