COMPUTER VISION
SAQIB UR REHMAN
BCSF20A510

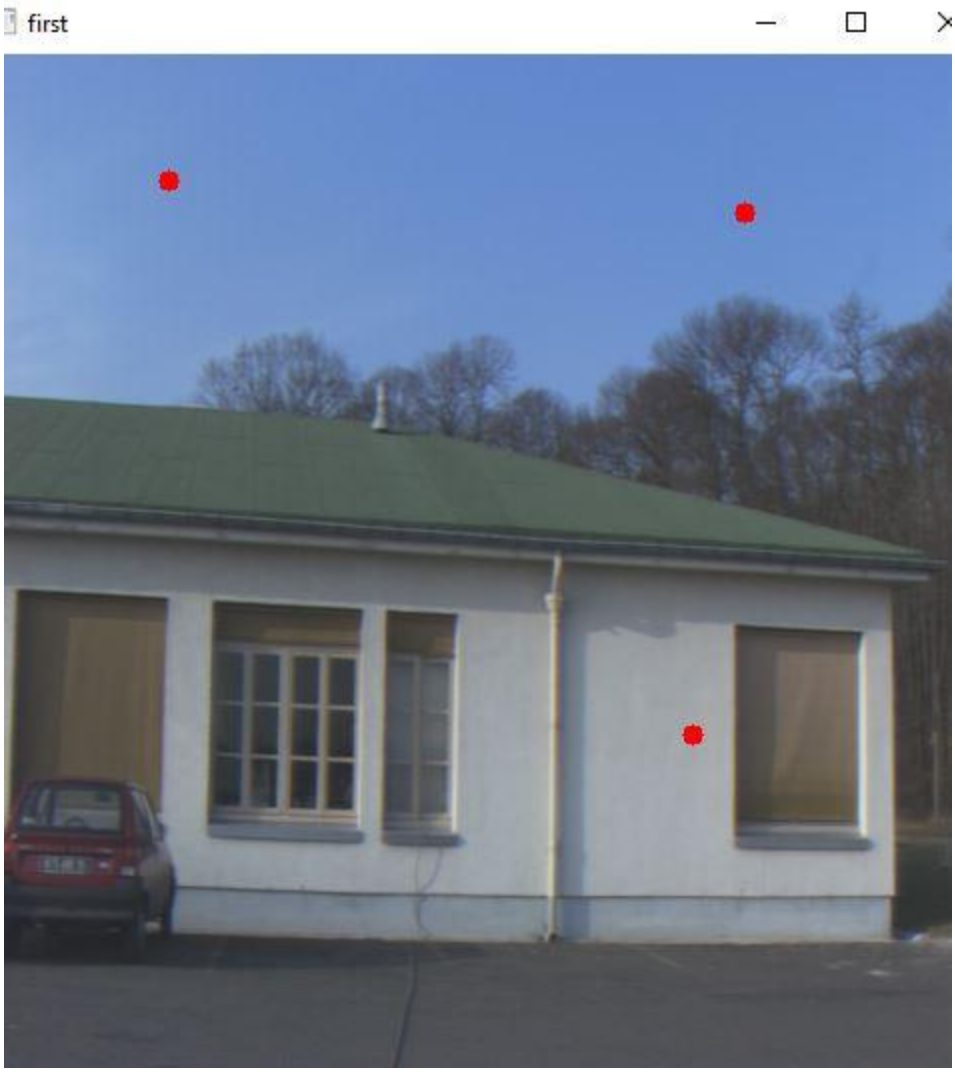
ASSIGNMENT NO.3

QUESTION NO 1:
(a)

```
import cv2
import numpy as np

# Load the images
img1 = cv2.imread("Image-1.png")
img2 = cv2.imread("Image-2.png")

# Function to get the points
def get_points(image, title):
    points = []
    print("Click on 4 points in the", title, "image")
    cv2.imshow(title, image)
    cv2.setMouseCallback(title, on_mouse, (image, title, points))
    while len(points) < 4:
        cv2.waitKey(100)
    cv2.destroyAllWindows()
    return points

def on_mouse(event, x, y, flags, param):
    if event == cv2.EVENT_LBUTTONDOWN:
        image, title, points = param
        print(title, "point recorded:", x, y)
        cv2.circle(image, (x, y), 5, (0, 0, 255), -1)
        cv2.imshow(title, image)
        points.append([x, y])

# Get the 4 point correspondences
pts1 = np.float32(get_points(img1, "first"))
```

```
pts2 = np.float32(get_points(img2, "second"))
# Calculate the projective transformation matrix
M = cv2.getPerspectiveTransform(pts1, pts2)
# Apply the perspective transformation to the first image
rows, cols, channels = img1.shape
#forward mapping
dst = cv2.warpPerspective(img1, M, (cols, rows))
# Save the output image
cv2.imwrite("output.jpg", dst)
cv2.imshow("Recovered Image", dst)
cv2.waitKey(0)
cv2.destroyAllWindows()

# Apply the backward transformation to the second image
r, c, ch = img2.shape
#Backward Mapping
output_21 = cv2.warpPerspective(img2, np.linalg.inv(M), (cols, rows))
# Save the output image
cv2.imwrite("output_21.jpg", output_21)
cv2.imshow("Backward Mapped Image", output_21)
cv2.waitKey(0)
```

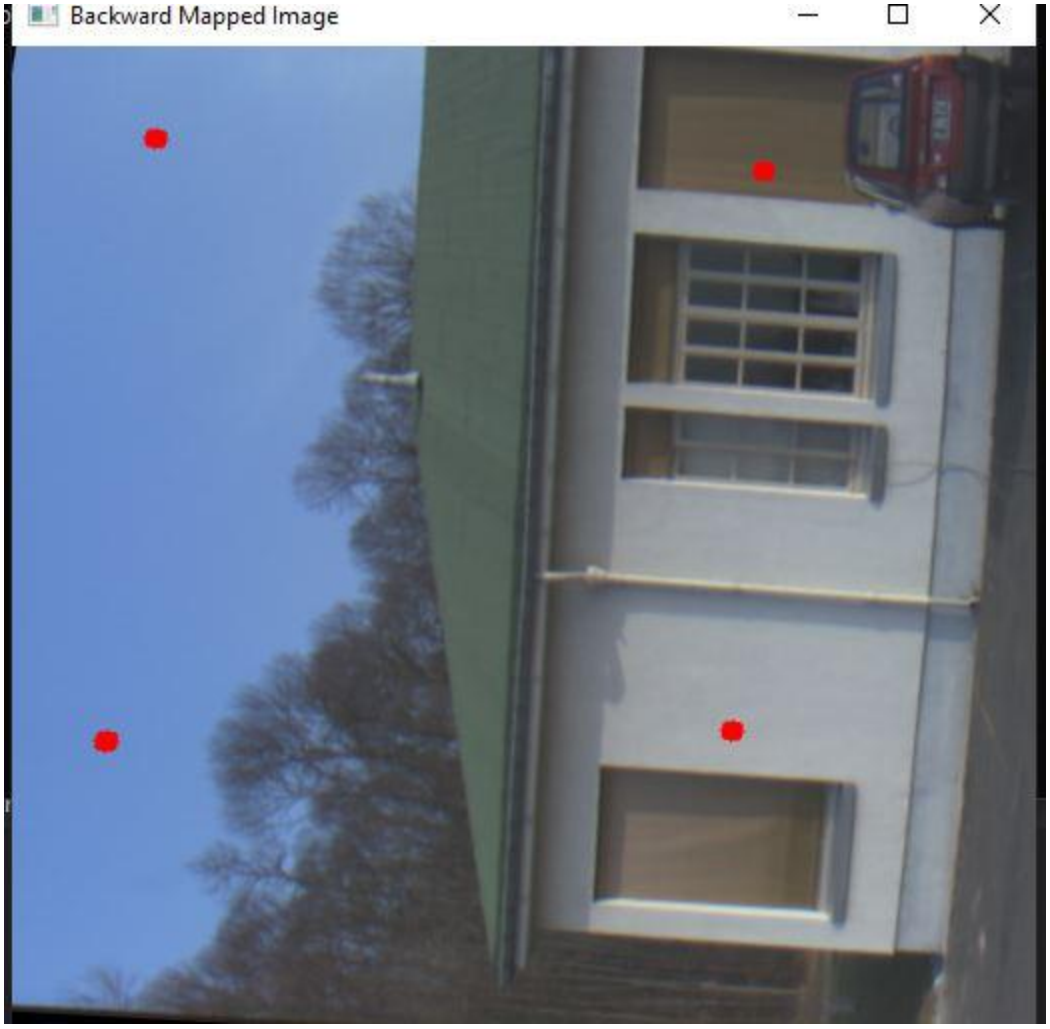
(b)


first


second

The problem with forward mapping is that there might remain some locations in the output image unassigned.As a result , some parts of the original image may not have a corresponding pixel in the transformed image.This is called forward mapping problem.

(c)



Question No 2:

```
import numpy as np
import cv2
import glob

# Method implements the Harris Corner Detection algorithm
def CornerDetection(image):

    # The two Sobel operators - for x and y direction
    # X and Y derivative of image using Sobel operator
    ImgX = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)
    ImgY = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)
    ImgX_2 = np.square(ImgX)
```

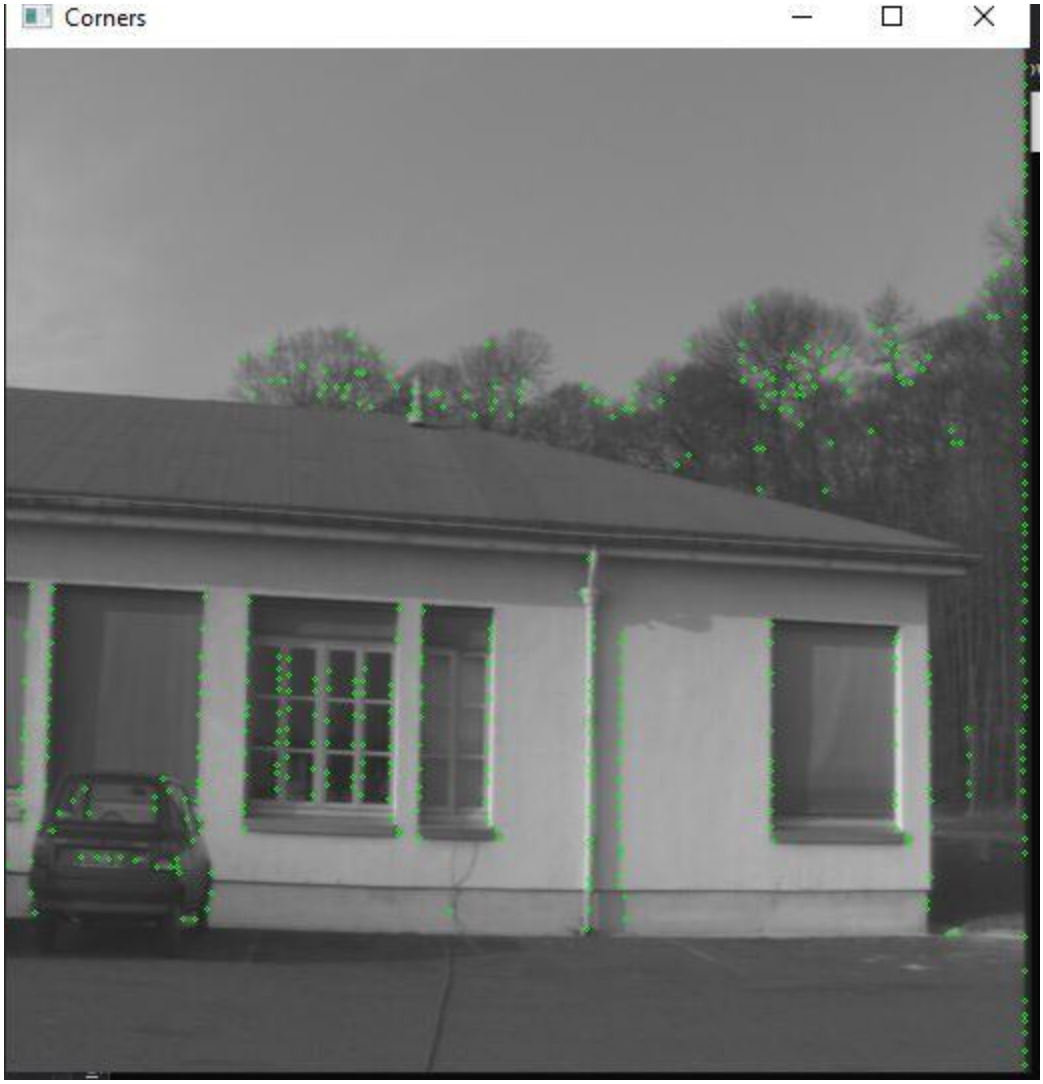
```
    ImgY_2 = np.square(ImgY)

    ImgXY = np.multiply(ImgX, ImgY)
    ImgYX = np.multiply(ImgY, ImgX)

    #Use Gaussian Blur
    Sigma = 1.4
    kernelsize = (3, 3)
    ImgX_2 = cv2.GaussianBlur(ImgX_2, kernelsize, Sigma)
    ImgY_2 = cv2.GaussianBlur(ImgY_2, kernelsize, Sigma)
    ImgXY = cv2.GaussianBlur(ImgXY, kernelsize, Sigma)
    ImgYX = cv2.GaussianBlur(ImgYX, kernelsize, Sigma)

    # options
    algorithms = ['Harris', 'Shi-Tomasi',
    'Rohr', 'Triggs', 'Brown , Szeliski, and Winder']
    print('Available algorithms:')
    for i, algorithm in enumerate(algorithms):
        print(f'{i+1}. {algorithm}')
    choice = int(input('Enter your choice (1-5): '))

    alpha = 0.06
    R = np.zeros((w, h), np.float32)
    # For every pixel find the corner strength
    for row in range(w):
        for col in range(h):
            M_bar = np.array([[ImgX_2[row][col], ImgXY[row][col]],
[ImgYX[row][col], ImgY_2[row][col]]])
            if choice==1:
                R[row][col] = np.linalg.det(M_bar) - (alpha *
np.square(np.trace(M_bar)))
            elif choice==2:
                eigen_vals = np.linalg.eigvals(M_bar)
                temp1 = eigen_vals[0]
                R[row,col] = temp1
```

```
            elif choice==3:
                det = np.linalg.det(M_bar)
                temp = det
                R[row,col] = temp
            elif choice==4:
                eigen_vals = np.linalg.eigvals(M_bar)
                lambda_1 = eigen_vals[0]
                lambda_2 = eigen_vals[1]
                temp = (lambda_1-alpha*lambda_2)
                R[row,col] = temp
            elif choice == 5:
                det = np.linalg.det(M_bar)
                tr = np.square(np.trace(M_bar))
                temp = det/tr
                R[row,col] = temp
    return R


#### Main Program ####
firstimagename = cv2.imread("Image-1.png")

# Get the first image
firstimage = cv2.cvtColor(firstimagename, cv2.COLOR_BGR2GRAY)
w, h = firstimage.shape
# Convert image to color to draw colored circles on it
bgr = cv2.cvtColor(firstimage, cv2.COLOR_GRAY2RGB)
# Corner detection
R = CornerDetection(firstimage)
# Empirical Parameter
# This parameter will need tuning based on the use-case
CornerStrengthThreshold = np.percentile(R,95)
# Plot detected corners on image
radius = 1
color = (0, 255, 0)  # Green
thickness = 1
```

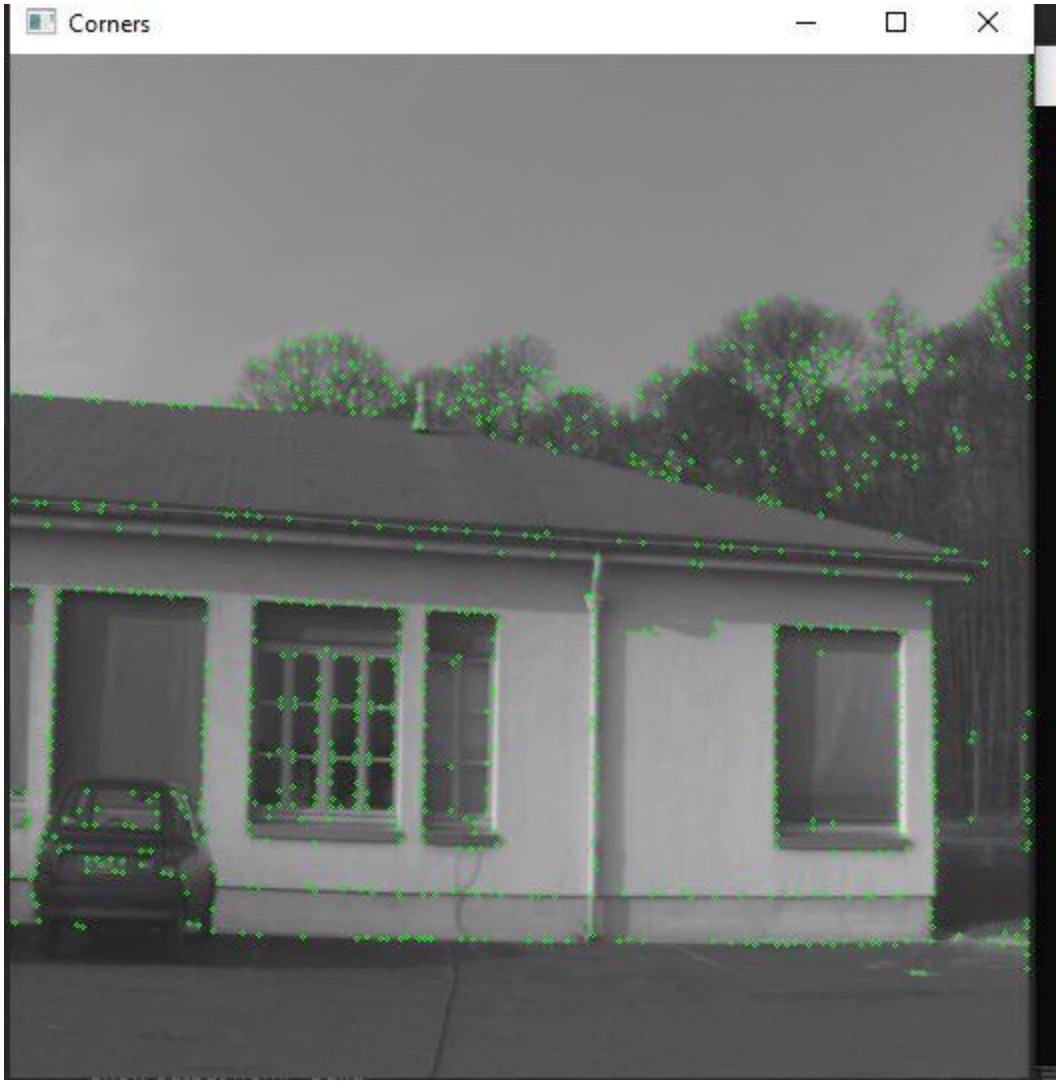
```
PointList = []
# Look for Corner strengths above the threshold
for row in range(w):
    for col in range(h):
        if R[row][col] > CornerStrengthThreshold:
            # print(R[row][col])
            max = R[row][col]
            # Local non-maxima suppression
            skip = False
            for nrow in range(5):
                for ncol in range(5):
                    if row + nrow - 2 < w and col + ncol - 2 < h:
                        if R[row + nrow - 2][col + ncol - 2] > max:
                            skip = True
                            break
            if not skip:
                # Point is expressed in x, y which is col, row
                cv2.circle(bgr, (col, row), radius, color, thickness)
                PointList.append((row, col))

# Display image indicating corners and save it
cv2.imshow("Corners", bgr)
outname = "Output_" + str(CornerStrengthThreshold) + ".png"
cv2.imwrite(outname, bgr)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Results:

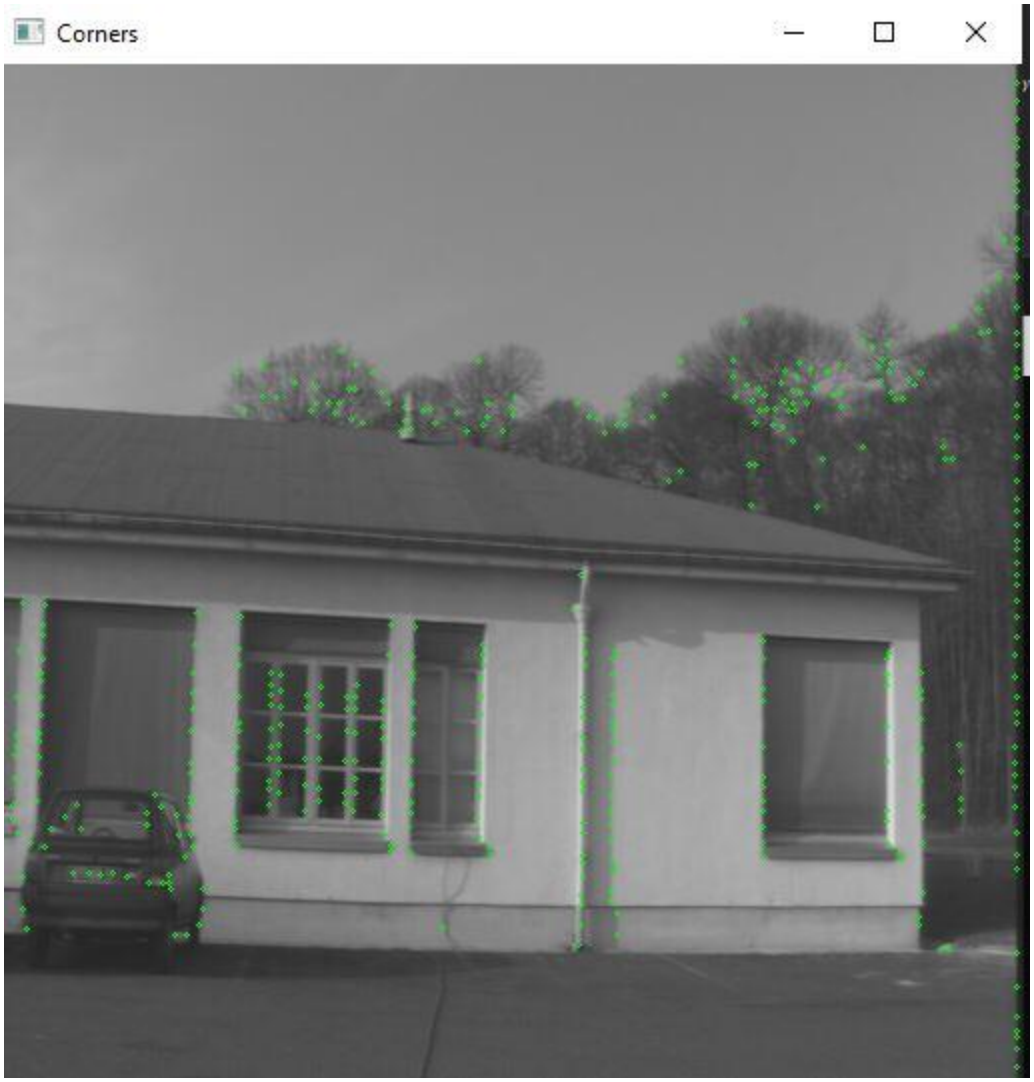
Harris Detector



Shi_Tomassi Detector



Rohr Detector



Triggs Detector

<----------—----------->