ASSIGNMENT -2:

1)

```
import numpy as np
from scipy import ndimage
import cv2
from PIL import Image
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
```

```
def gaussian_kernel(size, sigma=10):
    size = int(size) // 2
    x, y = np.mgrid[-size:size+1, -size:size+1]
    normal = 1 / (2.0 * np.pi * sigma**2)
    g =  np.exp(-((x**2 + y**2) / (2.0*sigma**2))) * normal
    return g
```

```
def sobel_filters(img):
    Kx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], np.float32)
    Ky = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]], np.float32)

    Ix = ndimage.filters.convolve(img, Kx)
    Iy = ndimage.filters.convolve(img, Ky)

    G = np.hypot(Ix, Iy)
    G = G / G.max() * 255
    theta = np.arctan2(Iy, Ix)

    return (G, theta)
```


```
def visualize(img,dst):
    plt.subplot(121),plt.imshow(img),plt.title('Original')
    plt.xticks([]), plt.yticks([])
    plt.subplot(122),plt.imshow(dst,cmap="gray"),plt.title('Blurred')
    plt.xticks([]), plt.yticks([])
    plt.show()
```

Canny Detector

```
def Canny_detector(img):
    weak_th = None
    strong_th = None
    # conversion of image to grayscale
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Noise reduction step
    g=gaussian_kernel(5,5)

    img= cv2.filter2D(src=img, kernel=g, ddepth=19)

    mag,ang=sobel_filters(img)

    # setting the minimum and maximum thresholds
    # for double thresholding
    mag_max = np.max(mag)
    if not weak_th:weak_th = mag_max * 0.1
    if not strong_th:strong_th = mag_max * 0.5

    # getting the dimensions of the input image
    height, width = img.shape

    # Looping through every pixel of the grayscale
    # image
    for i_x in range(width):
        for i_y in range(height):

            grad_ang = ang[i_y, i_x]
            grad_ang = abs(grad_ang-180) if abs(grad_ang)>180 else abs(grad_ang)

            # selecting the neighbours of the target pixel
```

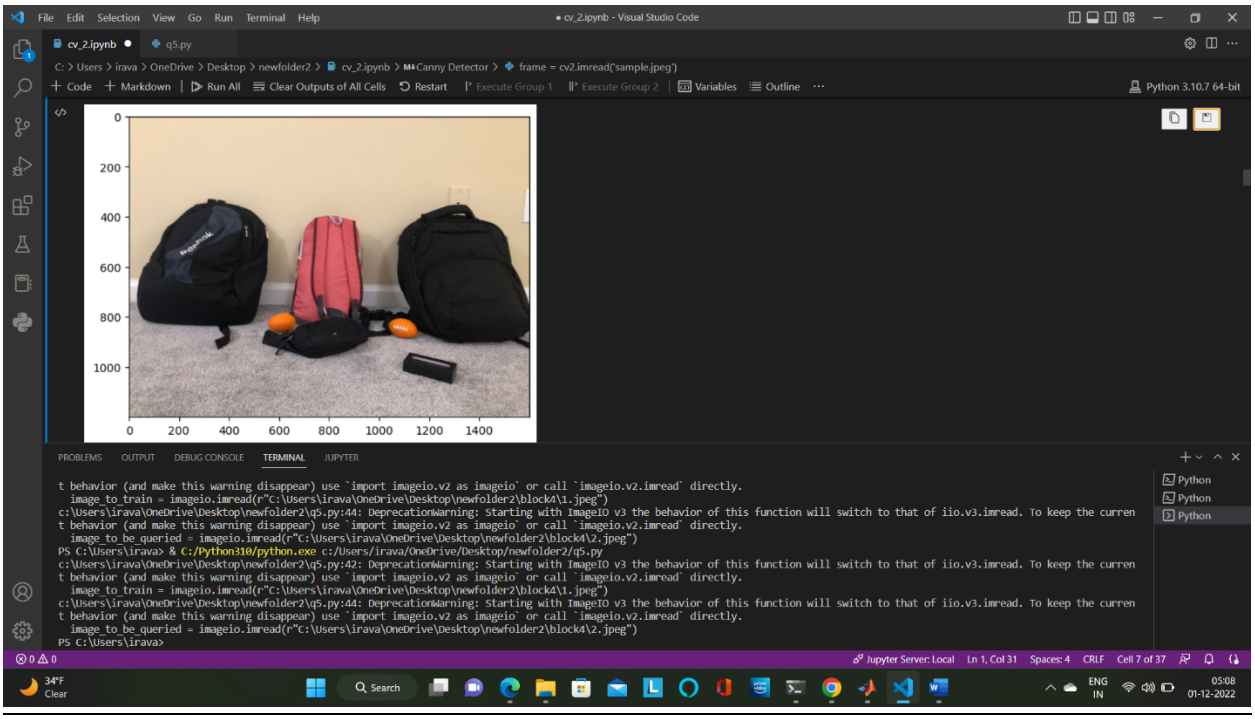
```
            # according to the gradient direction
            # In the x axis direction
            if grad_ang<= 22.5:
                neighb_1_x, neighb_1_y = i_x-1, i_y
                neighb_2_x, neighb_2_y = i_x + 1, i_y

            # top right (diagonal-1) direction
            elif grad_ang>22.5 and grad_ang<=(22.5 + 45):
                neighb_1_x, neighb_1_y = i_x-1, i_y-1
                neighb_2_x, neighb_2_y = i_x + 1, i_y + 1

            # In y-axis direction
            elif grad_ang>(22.5 + 45) and grad_ang<=(22.5 + 90):
                neighb_1_x, neighb_1_y = i_x, i_y-1
                neighb_2_x, neighb_2_y = i_x, i_y + 1

            # top left (diagonal-2) direction
            elif grad_ang>(22.5 + 90) and grad_ang<=(22.5 + 135):
                neighb_1_x, neighb_1_y = i_x-1, i_y + 1
                neighb_2_x, neighb_2_y = i_x + 1, i_y-1

            # Now it restarts the cycle
            elif grad_ang>(22.5 + 135) and grad_ang<=(22.5 + 180):
                neighb_1_x, neighb_1_y = i_x-1, i_y
                neighb_2_x, neighb_2_y = i_x + 1, i_y

            # Non-maximum suppression step
            if width>neighb_1_x>= 0 and height>neighb_1_y>= 0:
                if mag[i_y, i_x]<mag[neighb_1_y, neighb_1_x]:
                    mag[i_y, i_x]= 0
                    continue

            if width>neighb_2_x>= 0 and height>neighb_2_y>= 0:
                if mag[i_y, i_x]<mag[neighb_2_y, neighb_2_x]:
                    mag[i_y, i_x]= 0
```

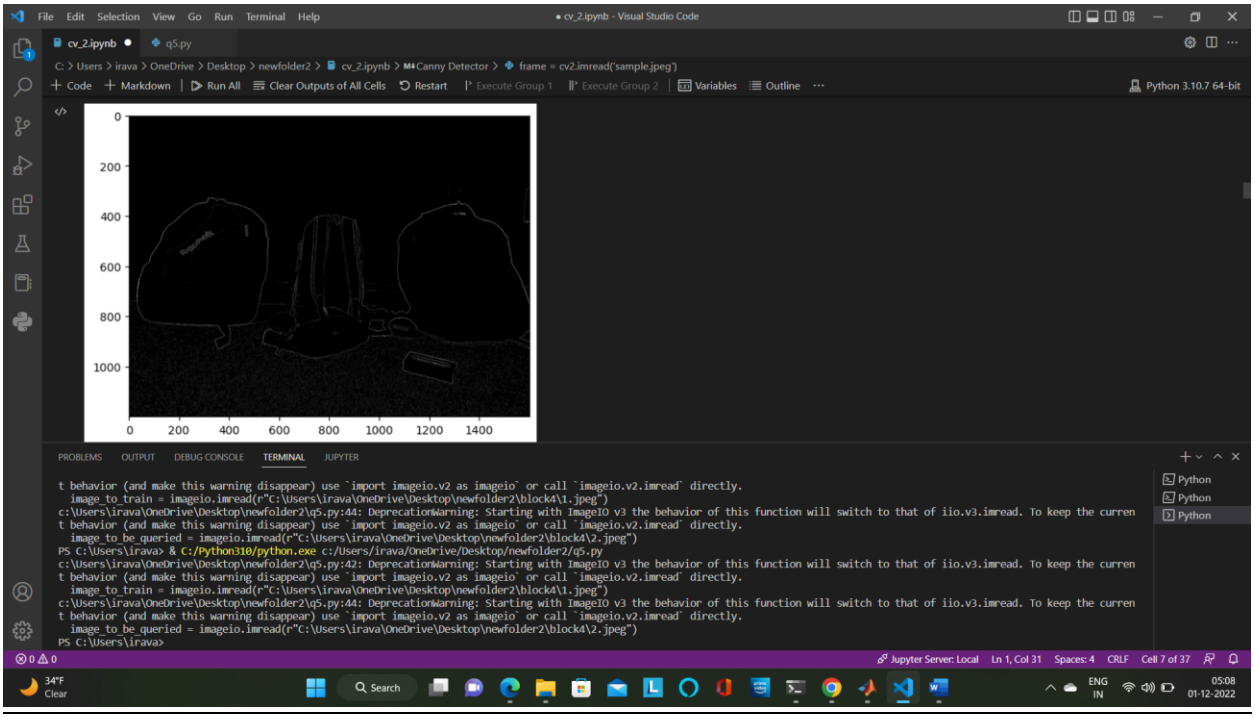
```
    weak_ids = np.zeros_like(img)
    strong_ids = np.zeros_like(img)
    ids = np.zeros_like(img)
    # double thresholding step
    for i_x in range(width):
        for i_y in range(height):

            grad_mag = mag[i_y, i_x]

            if grad_mag<weak_th:
                mag[i_y, i_x]= 0
            elif strong_th>grad_mag>= weak_th:
                ids[i_y, i_x]= 1
            else:
                ids[i_y, i_x]= 2


    # finally returning the magnitude of
    # gradients of edges
    return mag
```

[5]

```
frame = cv2.imread('sample.jpeg')
canny_img = Canny_detector(frame)

plt.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
```

[6]


File Edit Selection View Go Run Terminal Help
• cv_2.ipynb - Visual Studio Code
cv_2.ipynb
q5.py
C: > Users > irava > OneDrive > Desktop > newfolder2 > cv_2.ipynb > Canny Detector > frame = cv2.imread('sample.jpeg')
Code Markdown Run All Clear Outputs of All Cells Restart Execute Group 1 Execute Group 2 Variables Outline
Python 3.10.7 64-bit
0 200 400 600 800 1000
0 200 400 600 800 1000 1200 1400
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER
t behavior (and make this warning disappear) use `import imageio.v2 as imageio` or call `imageio.v2.imread` directly.
image_to_train = imageio.imread(r"C:\Users\irava\OneDrive\Desktop\newfolder2\block4\1.jpeg")
c:\Users\irava\OneDrive\Desktop\newfolder2\q5.py:44: DeprecationWarning: Starting with ImageIO v3 the behavior of this function will switch to that of iio.v3.imread. To keep the curren
t behavior (and make this warning disappear) use `import imageio.v2 as imageio` or call `imageio.v2.imread` directly.
image_to_be_queried = imageio.imread(r"C:\Users\irava\OneDrive\Desktop\newfolder2\block4\2.jpeg")
PS C:\Users\irava> & C:/Python310/python.exe c:/Users/irava/OneDrive/Desktop/newfolder2/q5.py
c:\Users\irava\OneDrive\Desktop\newfolder2\q5.py:42: DeprecationWarning: Starting with ImageIO v3 the behavior of this function will switch to that of iio.v3.imread. To keep the curren
t behavior (and make this warning disappear) use `import imageio.v2 as imageio` or call `imageio.v2.imread` directly.
image_to_train = imageio.imread(r"C:\Users\irava\OneDrive\Desktop\newfolder2\block4\1.jpeg")
c:\Users\irava\OneDrive\Desktop\newfolder2\q5.py:44: DeprecationWarning: Starting with ImageIO v3 the behavior of this function will switch to that of iio.v3.imread. To keep the curren
t behavior (and make this warning disappear) use `import imageio.v2 as imageio` or call `imageio.v2.imread` directly.
image_to_be_queried = imageio.imread(r"C:\Users\irava\OneDrive\Desktop\newfolder2\block4\2.jpeg")
PS C:\Users\irava>
Python
Python
Python
Jupyter Server: Local Ln 1, Col 31 Spaces: 4 CRLF Cell 7 of 37
34°F Clear
Search
ENG IN
05:08 01-12-2022

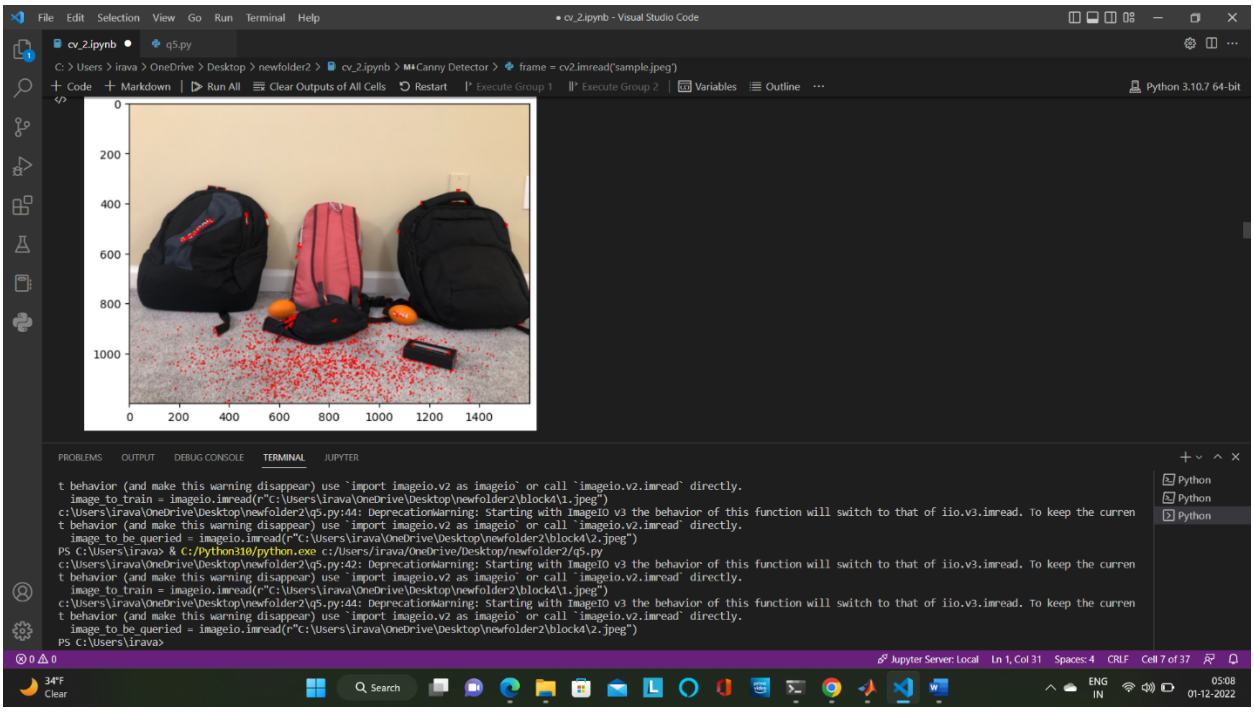

File Edit Selection View Go Run Terminal Help
• cv_2.ipynb - Visual Studio Code
cv_2.ipynb
q5.py
C: > Users > irava > OneDrive > Desktop > newfolder2 > cv_2.ipynb > Canny Detector > frame = cv2.imread('sample.jpeg')
Code Markdown Run All Clear Outputs of All Cells Restart Execute Group 1 Execute Group 2 Variables Outline
Python 3.10.7 64-bit
0 200 400 600 800 1000
0 200 400 600 800 1000 1200 1400
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER
t behavior (and make this warning disappear) use `import imageio.v2 as imageio` or call `imageio.v2.imread` directly.
image_to_train = imageio.imread(r"C:\Users\irava\OneDrive\Desktop\newfolder2\block4\1.jpeg")
c:\Users\irava\OneDrive\Desktop\newfolder2\q5.py:44: DeprecationWarning: Starting with ImageIO v3 the behavior of this function will switch to that of iio.v3.imread. To keep the curren
t behavior (and make this warning disappear) use `import imageio.v2 as imageio` or call `imageio.v2.imread` directly.
image_to_be_queried = imageio.imread(r"C:\Users\irava\OneDrive\Desktop\newfolder2\block4\2.jpeg")
PS C:\Users\irava> & C:/Python310/python.exe c:/Users/irava/OneDrive/Desktop/newfolder2/q5.py
c:\Users\irava\OneDrive\Desktop\newfolder2\q5.py:42: DeprecationWarning: Starting with ImageIO v3 the behavior of this function will switch to that of iio.v3.imread. To keep the curren
t behavior (and make this warning disappear) use `import imageio.v2 as imageio` or call `imageio.v2.imread` directly.
image_to_train = imageio.imread(r"C:\Users\irava\OneDrive\Desktop\newfolder2\block4\1.jpeg")
c:\Users\irava\OneDrive\Desktop\newfolder2\q5.py:44: DeprecationWarning: Starting with ImageIO v3 the behavior of this function will switch to that of iio.v3.imread. To keep the curren
t behavior (and make this warning disappear) use `import imageio.v2 as imageio` or call `imageio.v2.imread` directly.
image_to_be_queried = imageio.imread(r"C:\Users\irava\OneDrive\Desktop\newfolder2\block4\2.jpeg")
PS C:\Users\irava>
Python
Python
Python
Jupyter Server: Local Ln 1, Col 31 Spaces: 4 CRLF Cell 7 of 37
34°F Clear
Search
ENG IN
05:08 01-12-2022

Harris Edge Detection

```
    image = frame
    operatedImage = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    operatedImage = np.float32(operatedImage)
    dest = cv2.cornerHarris(operatedImage, 2, 3, 0.07)
    dest = cv2.dilate(dest,gaussian_kernel(5,5))
    image[dest > 0.01 * dest.max()]=[0, 0, 255]
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    if cv2.waitKey(0) & 0xff == 27:
        cv2.destroyAllWindows()
    image
```

Pyth

```
Output exceeds the size limit. Open the full output data in a text editor
array([[[192, 223, 248],
        [191, 222, 247],
        [191, 222, 247],
        ...,
        [165, 205, 230],
        [165, 205, 230],
        [165, 205, 230]],

       [[191, 222, 247],
        [191, 222, 247],
        [191, 222, 247],
        ...,
        [165, 205, 230],
        [165, 205, 230],
        [165, 205, 230]],

       [[190, 221, 246],
        [190, 221, 246],
        [190, 221, 246],
        ...,
        [166, 206, 231],
        [166, 206, 231],
        [166, 206, 231]],

       ...,
...
        [178, 185, 202],
        ...,
        [164, 174, 184],
        [160, 170, 180],
        [150, 160, 170]]], dtype=uint8)
```




```
canny_img
```

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]], dtype=float32)
```

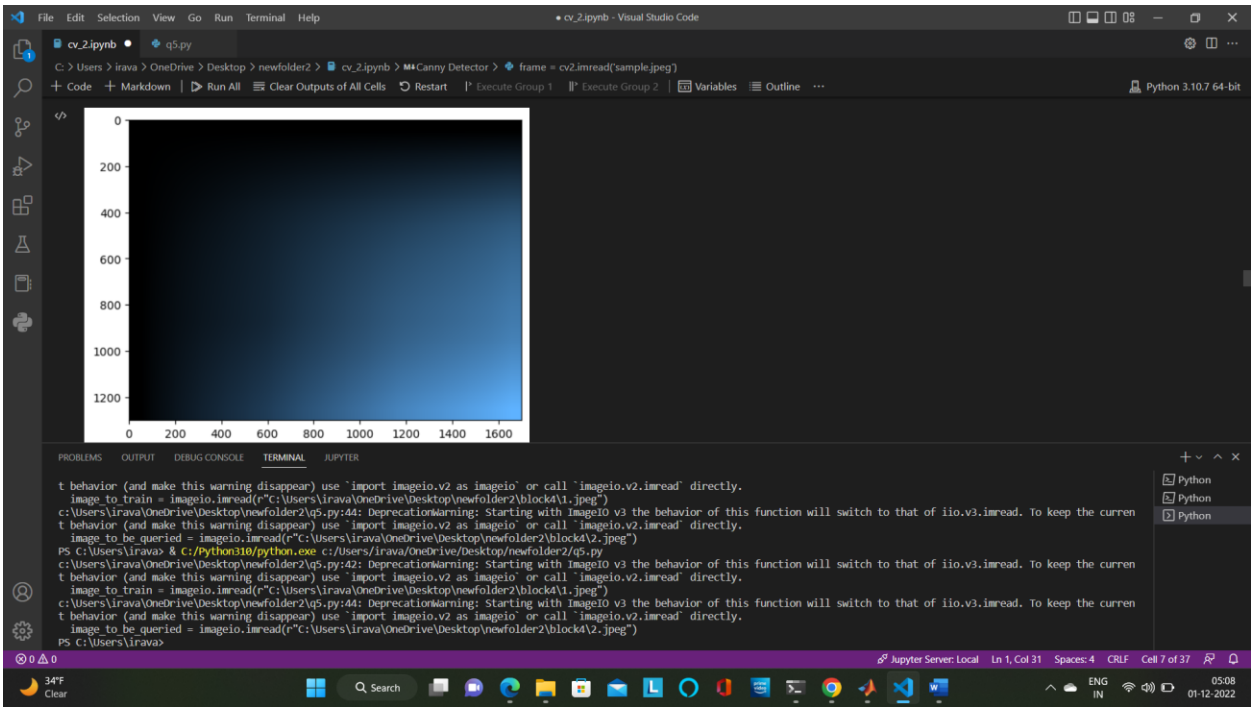
3)Integral Image

```
def integral_image(image, *, dtype=None):
    if dtype is None and image.real.dtype.kind == 'f':
        dtype = np.promote_types(image.dtype, np.float64)

    S = image
    for i in range(image.ndim):
        S = S.cumsum(axis=i, dtype=dtype)
    return S


def integrate(ii, start, end):
    start = np.atleast_2d(np.array(start))
    end = np.atleast_2d(np.array(end))
    rows = start.shape[0]

    total_shape = ii.shape
    total_shape = np.tile(total_shape, [rows, 1])

    start_negatives = start < 0
    end_negatives = end < 0
    start = (start + total_shape) * start_negatives + \
            start * ~(start_negatives)
    end = (end + total_shape) * end_negatives + \
           end * ~(end_negatives)

    if np.any((end - start) < 0):
        raise IndexError('end coordinates must be greater or equal to start')

    S = np.zeros(rows)
    bit_perm = 2 ** ii.ndim
    width = len(bin(bit_perm - 1)[2:])
    for i in range(bit_perm):
        binary = bin(i)[2:].zfill(width)
        bool_mask = [bit == '1' for bit in binary]
```

```
        bool_mask = [bit == '1' for bit in binary]

        sign = (-1)**sum(bool_mask)

        bad = [np.any(((start[r] - 1) * bool_mask) < 0)
               for r in range(rows)]

        corner_points = (end * (np.invert(bool_mask))) + \
                        ((start - 1) * bool_mask)

        S += [sign * ii[tuple(corner_points[r])] if(not bad[r]) else 0
              for r in range(rows)]
    return S
```

```
frame = cv2.imread('sample.jpeg')
frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
frame= cv2.copyMakeBorder(frame, 50, 50, 50, 50, cv2.BORDER_CONSTANT, (0,0,0))
frame=integral_image(frame)
frame = frame/np.amax(frame)
frame = np.clip(frame, 0,255)
plt.imshow(frame)
```

Integral Image



2)Stitching

Image Stitching

```
class Image_Stitching():
    def __init__(self) :
        self.ratio=0.85
        self.min_match=10
        self.sift=cv2.SIFT_create()
        self.smoothing_window_size=800

    def registration(self,img1,img2):
        kp1, des1 = self.sift.detectAndCompute(img1, None)
        kp2, des2 = self.sift.detectAndCompute(img2, None)
        matcher = cv2.BFMatcher()
        raw_matches = matcher.knnMatch(des1, des2, k=2)
        good_points = []
        good_matches=[]
        for m1, m2 in raw_matches:
            if m1.distance < self.ratio * m2.distance:
                good_points.append((m1.trainIdx, m1.queryIdx))
                good_matches.append([m1])
        img3 = cv2.drawMatchesKnn(img1, kp1, img2, kp2, good_matches, None, flags=2)
        cv2.imwrite('matching.jpg', img3)
        if len(good_points) > self.min_match:
            image1_kp = np.float32(
                [kp1[i].pt for (_, i) in good_points])
            image2_kp = np.float32(
                [kp2[i].pt for (i, _) in good_points])
            H, status = cv2.findHomography(image2_kp, image1_kp, cv2.RANSAC,5.0)
        return H

    def create_mask(self,img1,img2,version):
        height_img1 = img1.shape[0]
```

```
    def create_mask(self,img1,img2,version):
        height_img1 = img1.shape[0]
        width_img1 = img1.shape[1]
        width_img2 = img2.shape[1]
        height_panorama = height_img1
        width_panorama = width_img1 +width_img2
        offset = int(self.smoothing_window_size / 2)
        barrier = img1.shape[1] - int(self.smoothing_window_size / 2)
        mask = np.zeros((height_panorama, width_panorama))
        if version== 'left_image':
            mask[:, barrier - offset:barrier + offset ] = np.tile(np.linspace(1, 0, 2 * offset ).T, (height_panorama, 1))
            mask[:, :barrier - offset] = 1
        else:
            mask[:, barrier - offset :barrier + offset ] = np.tile(np.linspace(0, 1, 2 * offset ).T, (height_panorama, 1))
            mask[:, barrier + offset:] = 1
        return cv2.merge([mask, mask, mask])

    def blending(self,img1,img2):
        H = self.registration(img1,img2)
        height_img1 = img1.shape[0]
        width_img1 = img1.shape[1]
        width_img2 = img2.shape[1]
        height_panorama = height_img1
        width_panorama = width_img1 +width_img2

        panorama1 = np.zeros((height_panorama, width_panorama, 3))
        mask1 = self.create_mask(img1,img2,version='left_image')
        panorama1[0:img1.shape[0], 0:img1.shape[1], :] = img1
        panorama1 *= mask1
        mask2 = self.create_mask(img1,img2,version='right_image')
        panorama2 = cv2.warpPerspective(img2, H, (width_panorama, height_panorama))*mask2
        result=panorama1+panorama2

        rows, cols = np.where(result[:, :, 0] != 0)
        min_row, max_row = min(rows), max(rows) + 1
        min_col, max_col = min(cols), max(cols) + 1
```

Building 1 Stitching-

```
        panorama2 = cv2.warpPerspective(img2, H, (width_panorama, height_panorama))*mask2
        result=panorama1+panorama2

        rows, cols = np.where(result[:, :, 0] != 0)
        min_row, max_row = min(rows), max(rows) + 1
        min_col, max_col = min(cols), max(cols) + 1
        final_result = result[min_row:max_row, min_col:max_col, :]
        return final_result
```

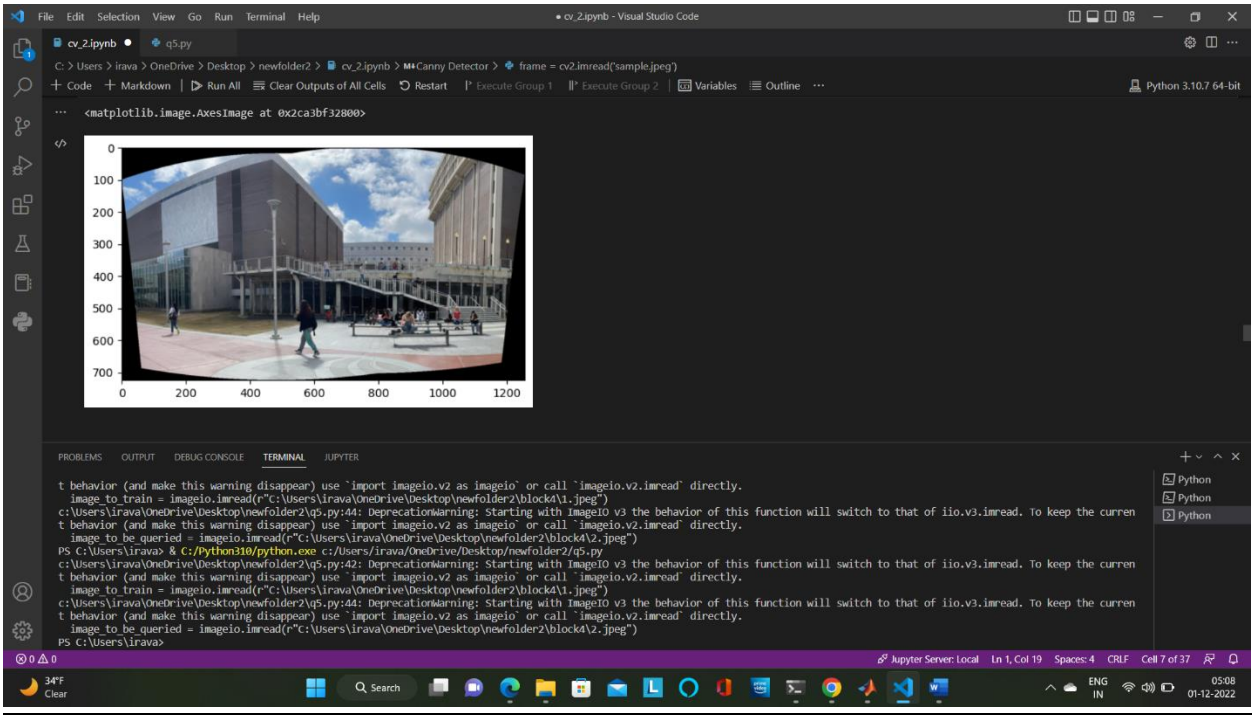
```
d='block1'
img1=cv2.cvtColor(cv2.imread(d+'/1.jpeg'), cv2.COLOR_BGR2RGB)
img2=cv2.cvtColor(cv2.imread(d+'/2.jpeg'), cv2.COLOR_BGR2RGB)
img3=cv2.cvtColor(cv2.imread(d+'/3.jpeg'), cv2.COLOR_BGR2RGB)
```

```
stitcher = cv2.Stitcher_create()
(status, stitched) = stitcher.stitch([img1,img2,img3])
print(status)
```

```
0
```

```
plt.imshow(stitched)
```

```
<matplotlib.image.AxesImage at 0x1cd337d2200>
```


File Edit Selection View Go Run Terminal Help
• cv_2.ipynb - Visual Studio Code
cv_2.ipynb
q5.py
C: > Users > irava > OneDrive > Desktop > newfolder2 > cv_2.ipynb > Canny Detector > frame = cv2.imread('sample.jpeg')
Code Markdown Run All Clear Outputs of All Cells Restart Execute Group 1 Execute Group 2 Variables Outline
Python 3.10.7 64-bit
<matplotlib.image.AxesImage at 0x2ca3bf32800>
0 100 200 300 400 500 600 700
0 200 400 600 800 1000 1200
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER
t behavior (and make this warning disappear) use `import imageio.v2 as imageio` or call `imageio.v2.imread` directly.
image_to_train = imageio.imread(r"C:\Users\irava\OneDrive\Desktop\newfolder2\block4\1.jpeg")
c:\Users\irava\OneDrive\Desktop\newfolder2\q5.py:44: DeprecationWarning: Starting with ImageIO v3 the behavior of this function will switch to that of iio.v3.imread. To keep the curren
t behavior (and make this warning disappear) use `import imageio.v2 as imageio` or call `imageio.v2.imread` directly.
image_to_be_queried = imageio.imread(r"C:\Users\irava\OneDrive\Desktop\newfolder2\block4\2.jpeg")
PS C:\Users\irava> & C:/Python310/python.exe c:/Users/irava/OneDrive/Desktop/newfolder2/q5.py
c:\Users\irava\OneDrive\Desktop\newfolder2\q5.py:42: DeprecationWarning: Starting with ImageIO v3 the behavior of this function will switch to that of iio.v3.imread. To keep the curren
t behavior (and make this warning disappear) use `import imageio.v2 as imageio` or call `imageio.v2.imread` directly.
image_to_train = imageio.imread(r"C:\Users\irava\OneDrive\Desktop\newfolder2\block4\1.jpeg")
c:\Users\irava\OneDrive\Desktop\newfolder2\q5.py:44: DeprecationWarning: Starting with ImageIO v3 the behavior of this function will switch to that of iio.v3.imread. To keep the curren
t behavior (and make this warning disappear) use `import imageio.v2 as imageio` or call `imageio.v2.imread` directly.
image_to_be_queried = imageio.imread(r"C:\Users\irava\OneDrive\Desktop\newfolder2\block4\2.jpeg")
PS C:\Users\irava>
Python
Python
Python
Jupyter Server: Local Ln 1, Col 19 Spaces: 4 CRLF Cell 7 of 37
34°F Clear
Search
ENG IN
05:08 01-12-2022

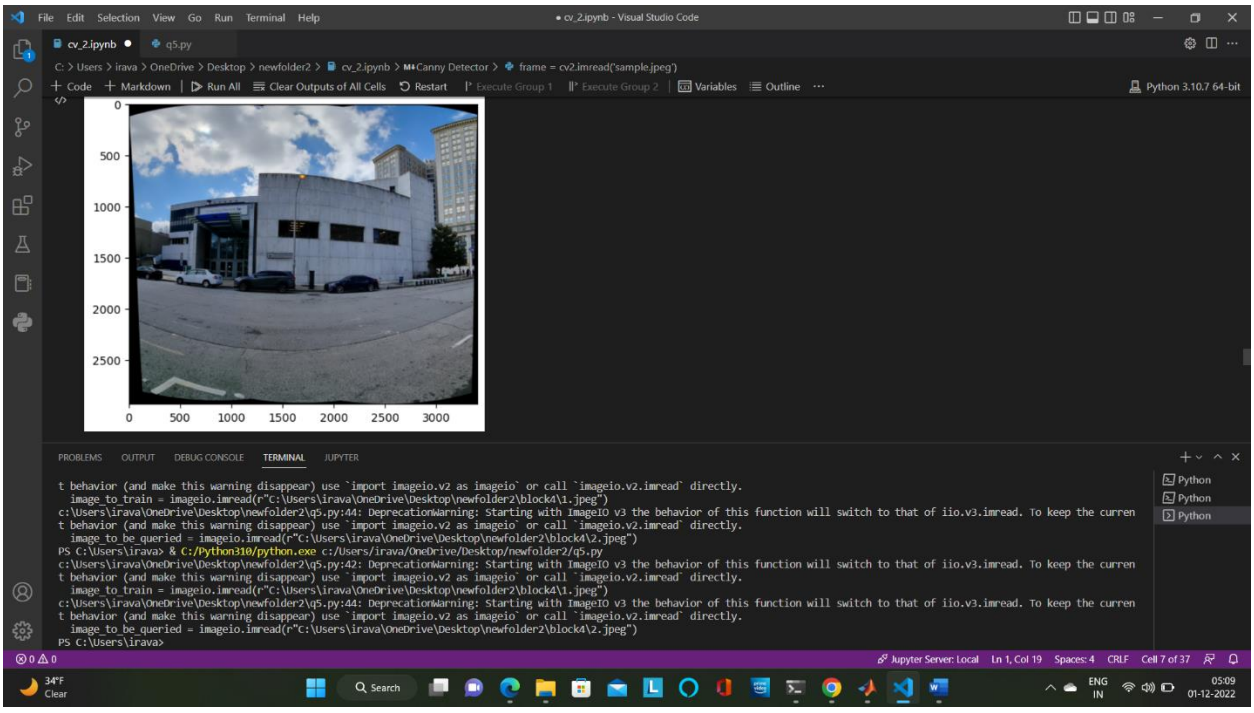
Building 2 stitching

```
d='block2'
img1=cv2.cvtColor(cv2.imread(d+'/1.jpg'), cv2.COLOR_BGR2RGB)
img2=cv2.cvtColor(cv2.imread(d+'/2.jpg'), cv2.COLOR_BGR2RGB)
img3=cv2.cvtColor(cv2.imread(d+'/3.jpg'), cv2.COLOR_BGR2RGB)
```

```
stitcher = cv2.Stitcher_create()
(status, stitched) = stitcher.stitch([img1,img2,img3])
print(status)
```

```
0
```

```
plt.imshow(stitched)
```



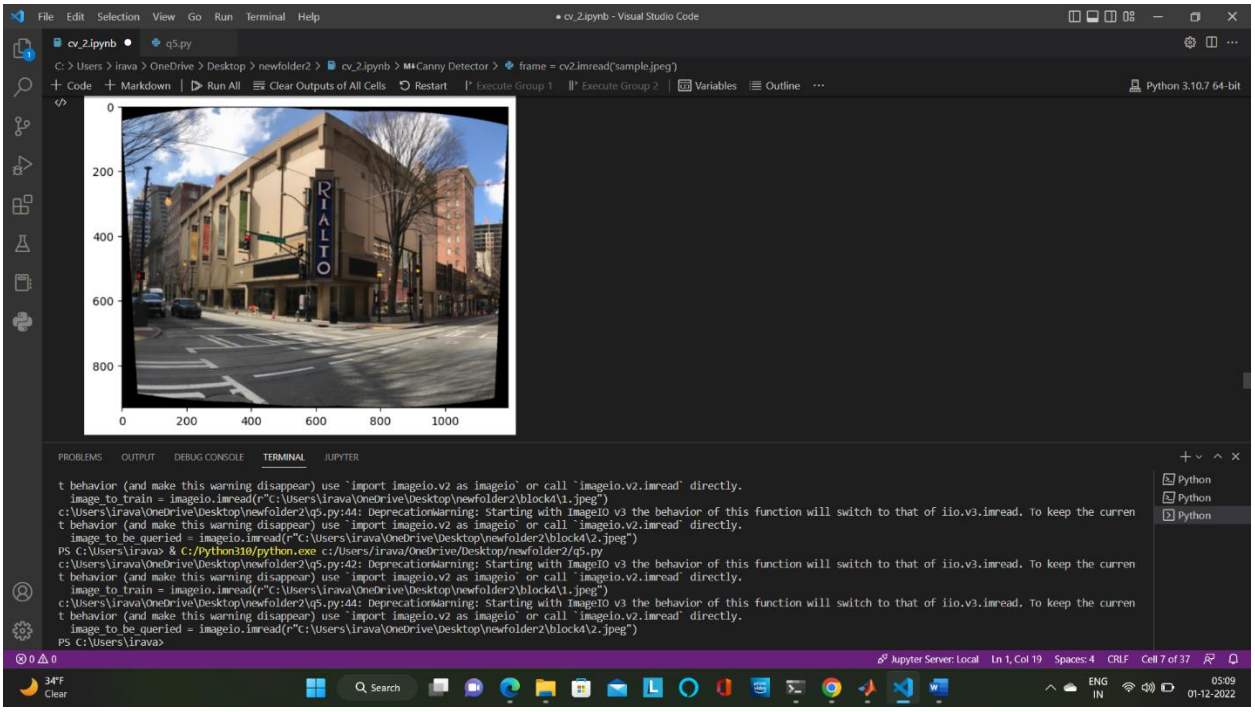
Building 3 stitching

```
d='block3'
img1=cv2.cvtColor(cv2.imread(d+'/1.jpeg'), cv2.COLOR_BGR2RGB)
img2=cv2.cvtColor(cv2.imread(d+'/2.jpeg'), cv2.COLOR_BGR2RGB)
img3=cv2.cvtColor(cv2.imread(d+'/3.jpeg'), cv2.COLOR_BGR2RGB)
```

```
stitcher = cv2.Stitcher_create()
(status, stitched) = stitcher.stitch([img1,img2,img3])
print(status)
```

```
0
```

```
plt.imshow(stitched)
```



Building 4 stitching

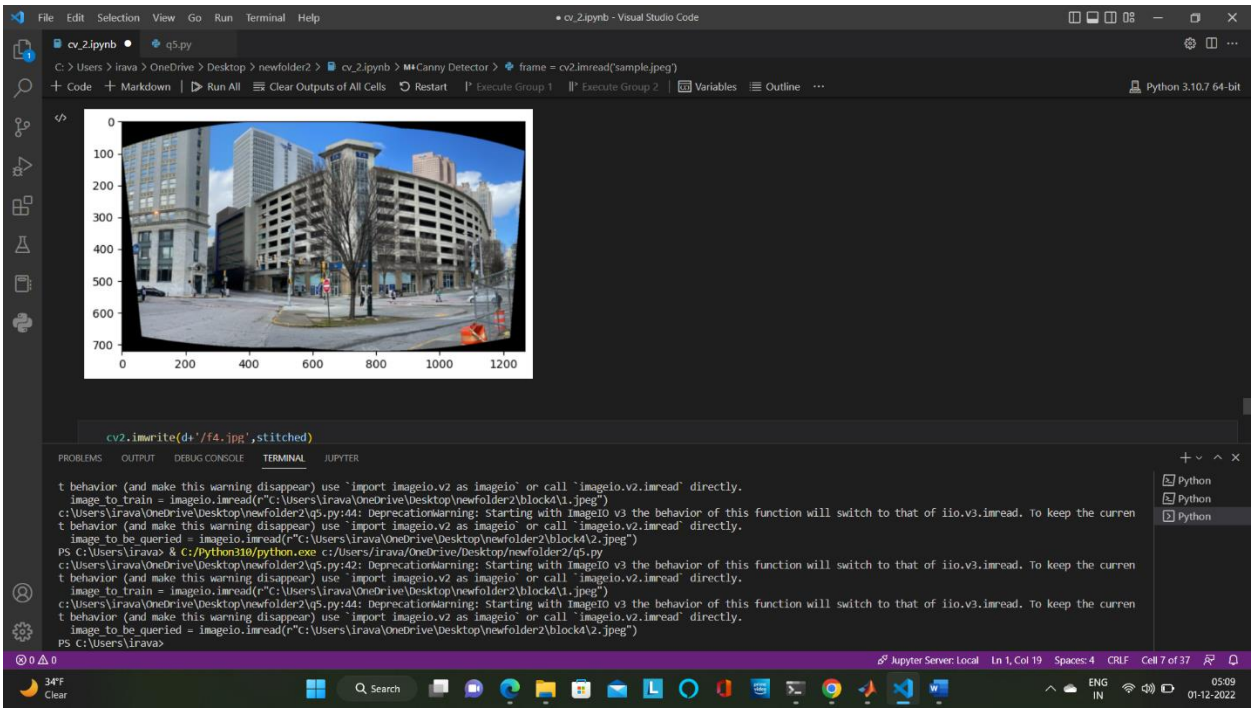
```
d='block4'
img1=cv2.cvtColor(cv2.imread(d+'/1.jpeg'), cv2.COLOR_BGR2RGB)
img2=cv2.cvtColor(cv2.imread(d+'/2.jpeg'), cv2.COLOR_BGR2RGB)
img3=cv2.cvtColor(cv2.imread(d+'/3.jpeg'), cv2.COLOR_BGR2RGB)
```

```
stitcher = cv2.Stitcher_create()
(status, stitched) = stitcher.stitch([img1,img2,img3])
print(status)
```

```
0
```

```
plt.imshow(stitched)
```

```
<matplotlib.image.AxesImage at 0x1cd3384e830>
```



Building 5 Stitching

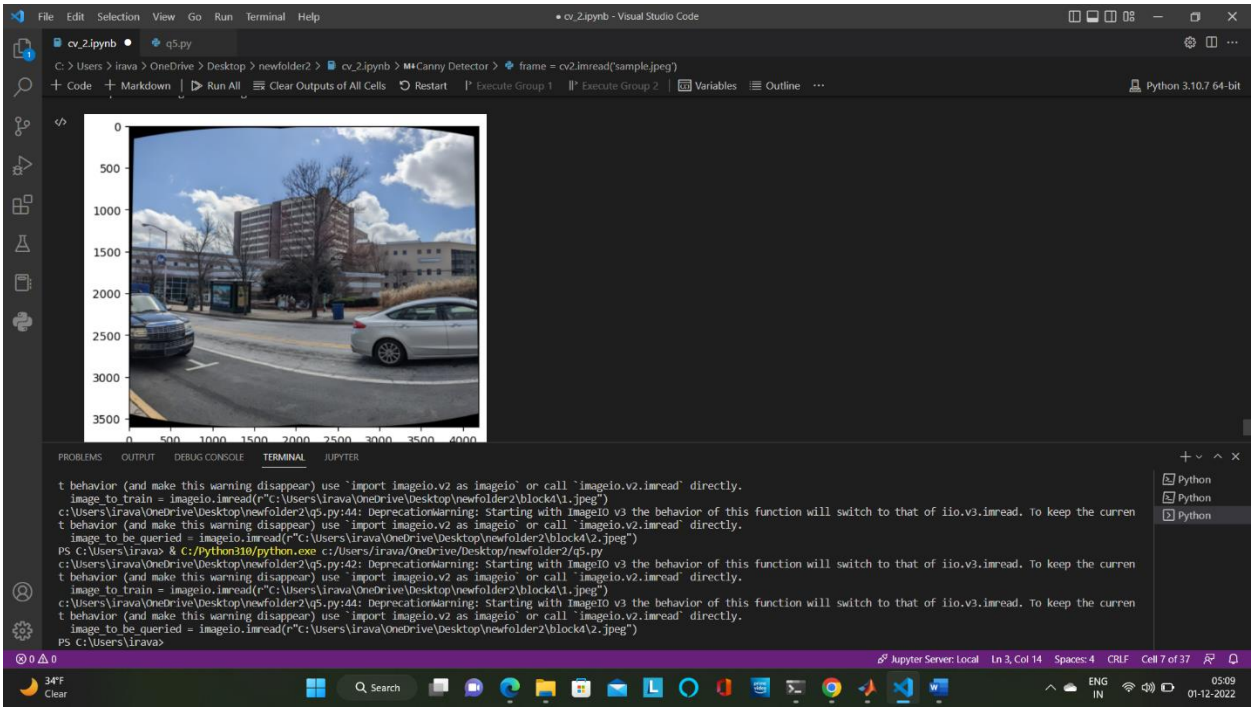
```
d='block5'
img1=cv2.cvtColor(cv2.imread(d+'/1.jpg'), cv2.COLOR_BGR2RGB)
img2=cv2.cvtColor(cv2.imread(d+'/2.jpg'), cv2.COLOR_BGR2RGB)
img3=cv2.cvtColor(cv2.imread(d+'/3.jpg'), cv2.COLOR_BGR2RGB)
```

```
stitcher = cv2.Stitcher_create()
(status, stitched) = stitcher.stitch([img1,img2,img3])
print(status)
```

```
0
```

```
plt.imshow(stitched)
```

```
<matplotlib.image.AxesImage at 0x1cd338d59c0>
```



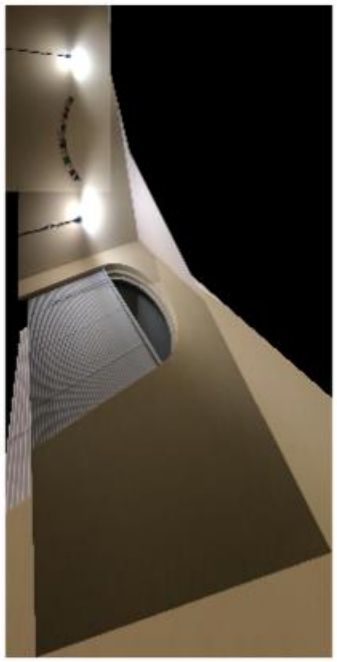
4)

```
% Load images.
buildingDir = fullfile('C:\Users\irava\OneDrive\Desktop\newfolder2\q4images');
buildingScene = imageDatastore(buildingDir);

% Display images to be stitched.
montage(buildingScene.Files)
% Read the first image from the image set.
I = readimage(buildingScene,1);

% Initialize features for I(1)
grayImage = im2gray(I);
points = detectSURFFeatures(grayImage);
[features, points] = extractFeatures(grayImage,points);


numImages = numel(buildingScene.Files);
tforms(numImages) = projective2d(eye(3));

imageSize = zeros(numImages,2);

% Iterate over remaining image pairs
for n = 2:numImages

    % Store points and features for I(n-1).
    pointsPrevious = points;
    featuresPrevious = features;

    % Read I(n).
    I = readimage(buildingScene, n);
```

```
    % Convert image to grayscale.
    grayImage = im2gray(I);

    % Save image size.
    imageSize(n,:) = size(grayImage);

    % Detect and extract SURF features for I(n).
    points = detectSURFFeatures(grayImage);
    [features, points] = extractFeatures(grayImage, points);

    % Find correspondences between I(n) and I(n-1).
    indexPairs = matchFeatures(features, featuresPrevious, 'Unique', true);

    matchedPoints = points(indexPairs(:,1), :);
    matchedPointsPrev = pointsPrevious(indexPairs(:,2), :);

    % Estimate the transformation between I(n) and I(n-1).
    tforms(n) = estimateGeometricTransform2D(matchedPoints, matchedPointsPrev,.
        'projective', 'Confidence', 99.9, 'MaxNumTrials', 2000);

    % Compute T(n) * T(n-1) * ... * T(1)
    tforms(n).T = tforms(n).T * tforms(n-1).T;
end
% Compute the output limits for each transform.
for i = 1:numel(tforms)
    [xlim(i,:), ylim(i,:)] = outputLimits(tforms(i), [1 imageSize(i,2)], [1 ima
end
avgXLim = mean(xlim, 2);
[~,idx] = sort(avgXLim);
```

```
[~,idx] = sort(avgXLim);
centerIdx = floor((numel(tforms)+1)/2);
centerImageIdx = idx(centerIdx);
Tinv = invert(tforms(centerImageIdx));
for i = 1:numel(tforms)
    tforms(i).T = tforms(i).T * Tinv.T;
end
for i = 1:numel(tforms)
    [xlim(i,:), ylim(i,:)] = outputLimits(tforms(i), [1 imageSize(i,2)], [1 ima
end

maxImageSize = max(imageSize);

% Find the minimum and maximum output limits.
xMin = min([1; xlim(:)]);
xMax = max([maxImageSize(2); xlim(:)]);

yMin = min([1; ylim(:)]);
yMax = max([maxImageSize(1); ylim(:)]);

% Width and height of panorama.
width  = round(xMax - xMin);
height = round(yMax - yMin);

% Initialize the "empty" panorama.
panorama = zeros([height width 3], 'like', I);
blender = vision.AlphaBlender('Operation', 'Binary mask', ...
    'MaskSource', 'Input port');
```

```
% Initialize the "empty" panorama.
panorama = zeros([height width 3], 'like', I);
blender = vision.AlphaBlender('Operation', 'Binary mask', ...
    'MaskSource', 'Input port');

% Create a 2-D spatial reference object defining the size of the panorama.
xLimits = [xMin xMax];
yLimits = [yMin yMax];
panoramaView = imref2d([height width], xLimits, yLimits);

% Create the panorama.
for i = 1:numImages

    I = readimage(buildingScene, i);

    % Transform I into the panorama.
    warpedImage = imwarp(I, tforms(i), 'OutputView', panoramaView);

    % Generate a binary mask.
    mask = imwarp(true(size(I,1),size(I,2)), tforms(i), 'OutputView', panoramaV

    % Overlay the warpedImage onto the panorama.
    panorama = step(blender, panorama, warpedImage, mask);
end

figure
imshow(panorama)
```



5)

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
import imageio
import imutils
cv2.ocl.setUseOpenCL(False)
def Main_Ponts_Func(image):
    descriptor = cv2.ORB_create()
    kps, features = descriptor.detectAndCompute(image, None)
    return (kps, features)
def Match_The_Key_Points_Func(features_train, features_query, ratio):
    bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck = False)
    raw_match = bf.knnMatch(features_train, features_query, 2)
    matches = []
    for m, n in raw_match:
        if m.distance < n.distance * ratio:
            matches.append(m)
    return matches
def Find_Homography(kps_train, kps_query, matches, reprojThresh):
    kpsA = np.float32([kp.pt for kp in kps_train])
    kpsB = np.float32([kp.pt for kp in kps_query])
    if len(matches) > 4:
        ptsA = np.float32([kpsA[m.queryIdx] for m in matches])
        ptsB = np.float32([kpsB[m.trainIdx] for m in matches])
        (Homography, status) = cv2.findHomography(ptsA, ptsB, cv2.RANSAC, reprojThresh)
        return(matches, Homography, status)
    else:
        return None
def Transform_To_Gray_Scale_Func(result):
    gray = cv2.cvtColor(result, cv2.COLOR_BGR2GRAY)
    thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY)[1]
    # To retrieve the contours in the binary image
    cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours(cnts)
    # TO get the maximum contour area
    c = max(cnts, key=cv2.contourArea)
    (x, y, w, h) = cv2.boundingRect(c)
```



```python
    cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours(cnts)
    # TO get the maximum contour area
    c = max(cnts, key=cv2.contourArea)
    (x, y, w, h) = cv2.boundingRect(c)
    result = result[y:y + h, x:x + w]
    return result

#training Img
image_to_train = imageio.imread(r"C:\Users\irava\OneDrive\Desktop\newfolder2\block4\1.jpeg")
#queried Img
image_to_be_queried = imageio.imread(r"C:\Users\irava\OneDrive\Desktop\newfolder2\block4\2.jpeg")
gray_image_to_train = cv2.cvtColor(image_to_train, cv2.COLOR_RGB2GRAY)
gray_image_to_be_queried = cv2.cvtColor(image_to_be_queried, cv2.COLOR_RGB2GRAY)
kps_train, features_train = Main_Ponts_Func(gray_image_to_train) #kps and features of 1
kps_query, features_query = Main_Ponts_Func(gray_image_to_be_queried) #kps and features of 2
matches = Match_The_Key_Points_Func(features_train, features_query, 0.75)
temp_img = cv2.drawMatches(image_to_train, kps_train, image_to_be_queried, kps_query, np.random.choice(matches,100), None, flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POIN
M = Find_Homography(kps_train, kps_query, matches, 4)
matches, homography, status = M
width = image_to_train.shape[1] + image_to_be_queried.shape[1]
height = image_to_train.shape[0] + image_to_be_queried.shape[0]
result = cv2.warpPerspective(image_to_train, homography, (width, height))
result[0:image_to_be_queried.shape[0], 0:image_to_be_queried.shape[1]] = image_to_be_queried
result = Transform_To_Gray_Scale_Func(result)
plt.figure(figsize=(20,10))
plt.imshow(result)
plt.show()
```

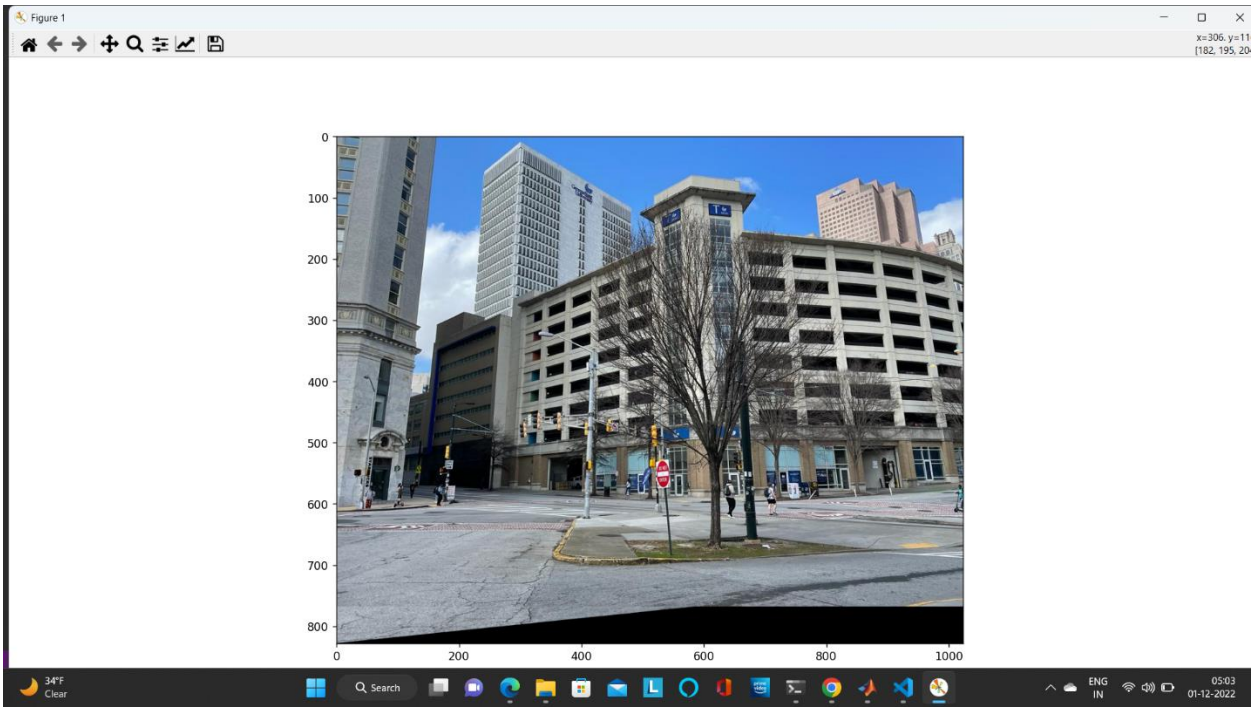


Figure 1
x=306, y=11
[182, 195, 20
0
100
200
300
400
500
600
700
800
0
200
400
600
800
1000
34°F
Clear
Search
ENG
IN
05:03
01-12-2022