PART A: Theory

1.

Please refer to document HW2_PartA_CANNY.pdf

2.

Please refer to document HW2_PartA_HARRIS.pdf

PART B: MATLAB Prototyping

3.

```
%Read 5 by 5 image
Image = imread('./edge.jpg')
```

Image = 5×5 uint8 matrix

```
    28    39    32    87   199
    30    39    32    89   202
    32    39    32    93   206
    32    40    34    94   206
    28    40    35    92   202
```

```
%Magnifying image and displaying because of the small size
figure; imshow(Image, InitialMagnification=5000);
title('Input Image')
%compute canny edge detection
canny_edge_op = edge(Image, 'canny')
```

canny_edge_op = 5×5 logical array

```
   0   0   0   0   0
   0   0   0   1   0
   0   0   0   1   0
   0   0   0   1   0
   0   0   0   0   0
```

```
%Displaying the detected edge on the image
imshow(canny_edge_op, InitialMagnification=5000)
title('Canny Edge Detected')
```

4.

```
%Read 5 by 5 image
%Harris Corner Detection
Image = imread('./corner.jpg')
```

Image = 5×5 uint8 matrix

```
   255   245   255     0     3
   231   240   255     0    14
   255   255   239     8     0
     0    12    14     0     3
```

```
   27    14     3     7    10
```

```
imshow(Image, InitialMagnification=5000)
title('Input Image')
corners = detectHarrisFeatures(Image,"FilterSize",3)
```

corners =

cornerPoints with properties:

Location: [3.0040 3.0244]

Metric: 0.1409

Count: 1

```
imshow(Image, InitialMagnification=5000);
hold on;
plot(corners.selectStrongest(2))
title('Corner Detected')
```

5.

```
% Load images.
buildingDir = fullfile("./bookstore");
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);

% Initialize all the transforms to the identity matrix. Note that the
% projective transform is used here because the building images are fairly
% close to the camera. Had the scene been captured from a further distance,
% an affine transform would suffice.
numImages = numel(buildingScene.Files);
tforms(numImages) = projective2d(eye(3));

% Initialize variable to hold image sizes.
```

```
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', 10000);

    % 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
imageSize(i,1)]);
end
avgXLim = mean(xlim, 2);
[~,idx] = sort(avgXLim);
centerIdx = floor((numel(tforms)+1)/2);
centerImageIdx = idx(centerIdx);
%Finally, apply the center image's inverse transform to all the others.
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
imageSize(i,1)]);
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');

% 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',
panoramaView);

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

```
figure
imshow(panorama);
title('Stitched Image')
```

PART C: Application development

6.

```
import cv2
import depthai as dai
import numpy as np

def getFrame(queue):
    #Get frame from the queue
    frame = queue.get()
    #convert the frmae to OpenCV format and return
    return frame.getCvFrame()

def getMonoCamera(pipeline, isLeft):
    #configure mono camera
    mono = pipeline.createMonoCamera()

    #set the Camera Resolution
    mono.setResolution(dai.MonoCameraProperties.SensorResolution.THE_400_P)
    if isLeft:
        mono.setBoardSocket(dai.CameraBoardSocket.LEFT)
    else:
        mono.setBoardSocket(dai.CameraBoardSocket.RIGHT)
    return mono

if __name__ == '__main__':

    #define a pipeline
    pipeline = dai.Pipeline()

    #set up left and right cameras
    monoLeft = getMonoCamera(pipeline, isLeft = True)
    monoRight = getMonoCamera(pipeline, isLeft = False)
    camRgb = pipeline.create(dai.node.ColorCamera)

    #set the camera resolution
    camRgb.setBoardSocket(dai.CameraBoardSocket.RGB)

camRgb.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
    camRgb.setVideoSize(1920, 1080)
    camRgb.setVideoSize(720, 640)
    #set output Xlink for left camera
    xoutLeft = pipeline.createXLinkOut()
    xoutLeft.setStreamName("left")

    #set output XLink for color camera
    xoutRgb = pipeline.create(dai.node.XLinkOut)
    xoutRgb.setStreamName("color")

    #set output Xlink for right camera
    xoutRight = pipeline.createXLinkOut()
    xoutRight.setStreamName("right")
```

```
    #Attach cameras to output XLink
    monoLeft.out.link(xoutLeft.input)
    monoRight.out.link(xoutRight.input)
    camRgb.video.link(xoutRgb.input)
    xoutRgb.input.setBlocking(False)
    xoutRgb.input.setQueueSize(1)
    #pipeline is defined, now we can connect to the device
    with dai.Device(pipeline) as device:

        #get the output queues.
        leftQueue = device.getOutputQueue(name = 'left', maxSize=1)
        rightQueue = device.getOutputQueue(name = 'right', maxSize = 1)
        rgb = device.getOutputQueue(name="color", maxSize=1)

        #Set the display window name

        #Variable used to toggle between side by side view and one frame view
        sidebySide = True
        while True:
            #get left Frame
            leftFrame = getFrame(leftQueue)
            #get right frame
            rightFrame = getFrame(rightQueue)

            if sidebySide:
                #show side by side view
                imOut = np.hstack((leftFrame, rightFrame))

            else:
                imOut = np.uint8(leftFrame/2 + rightFrame/2)
            #Display the output image
            cv2.imshow("Stereo Pair", imOut)

            color = getFrame(rgb)
            color = np.array(color)
            output_manual = np.cumsum(color, axis=1).cumsum(axis=0)
            output_manual = cv2.normalize(output_manual, None, 255,0,
cv2.NORM_MINMAX, cv2.CV_8UC1)
            #Display the output image
            # cv2.imshow("leftFrame", leftFrame)
            # cv2.imshow("rightFrame", rightFrame)
            cv2.imshow("Color", color)
            cv2.imshow("Integral Image", output_manual)
            #cv2.imshow("color", rgb)
            #check for keyboard input
            key = cv2.waitKey(1)
            if key == ord('q'):
                break # quit when q is pressed
            elif key == ord('t'):
                sidebySide = not sidebySide
```

7.

```
import cv2
import depthai as dai
```

```python
#create a pipeline
pipeline = dai.Pipeline()

#define the source and output
camRgb = pipeline.create(dai.node.ColorCamera)
xoutVideo = pipeline.create(dai.node.XLinkOut)

xoutVideo.setStreamName("video")

#properties

camRgb.setBoardSocket(dai.CameraBoardSocket.RGB)
camRgb.setResolution(dai.ColorCameraProperties.SensorResolution.THE_1080_P)
camRgb.setVideoSize(1920, 1080)
camRgb.setVideoSize(1280, 720)

xoutVideo.input.setBlocking(False)
xoutVideo.input.setQueueSize(1)

#Linking
camRgb.video.link(xoutVideo.input)

with dai.Device(pipeline) as device:

    video = device.getOutputQueue(name="video", maxSize=1, blocking=False)
    img_count=0
    frames = []
    while True:
        videoIn = video.get()
        output = videoIn.getCvFrame()
        #Get BGR from NV12 encoded video frame to show with opencv
        cv2.imshow("video", output)


        if cv2.waitKey(1) == ord('c'):
            frames.append(output)
            img_count +=1
            print(len(frames))
        if cv2.waitKey(1) == ord('p'):
            print('Creating Panaroma')
            if img_count < 2:
                print('click more pictures for a panaroma')
            else:
                stitcher=cv2.Stitcher.create()
                code,panaroma =stitcher.stitch(frames)
                if code != cv2.STITCHER_OK:
                    print("stitching not successful")
                else:
                    print('Your Panorama is ready!!!')
                    cv2.imshow('final result',panaroma)
                    cv2.imwrite('panaroma.jpg', panaroma)

        if cv2.waitKey(1) == ord('q'):
            break
```

Link to the github repository:

https://github.com/annieee6446/CSC-8830-Computer-Vision-HW2