Assignment2

PartA:

1.Canny edge Detection

```
#canny edge detector
#assignment2
import numpy as np
import os
import cv2
import matplotlib.pyplot as plt
# defining the canny detector function
# here weak_th and strong_th are thresholds for
# double thresholding step
def Canny_detector(img, weak_th = None, strong_th = None):
        # conversion of image to grayscale
        img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
       # Noise reduction step
        img = cv2.GaussianBlur(img, (5, 5), 1.4)
        # Calculating the gradients
        gx = cv2.Sobel(np.float32(img), cv2.CV_64F, 1, 0, 3)
        gy = cv2.Sobel(np.float32(img), cv2.CV_64F, 0, 1, 3)
        # Conversion of Cartesian coordinates to polar
        mag, ang = cv2.cartToPolar(gx, gy, angleInDegrees = True)
        # 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

   return mag
```

```
frame = cv2.imread('woman_pic.png')
canny_img = Canny_detector(frame)
# Displaying the input and output image
plt.figure()
f, plots = plt.subplots(1,2)
plots[0].imshow(frame)
plots[1].imshow(canny_img)
```

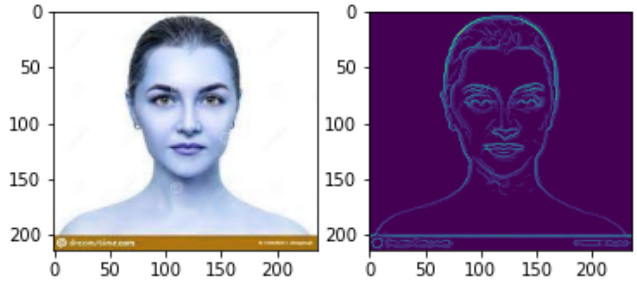
The original image which is feeded into the program is given below.

- Original image

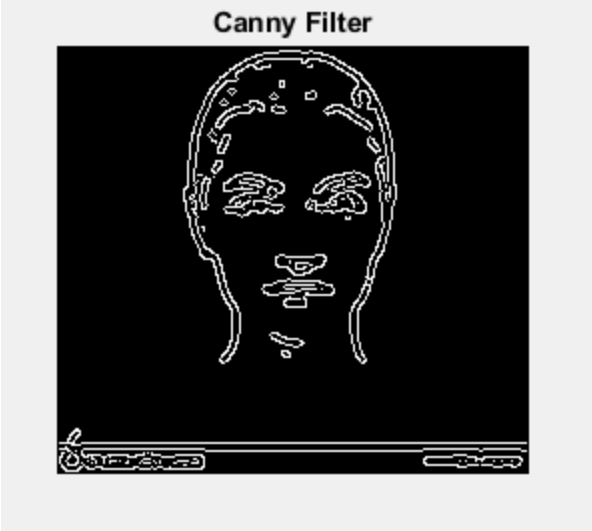
The edges are detected in the figure and it is giving the output as follows in python:

```
<matplotlib.image.AxesImage at 0x2033eee7a60>

<Figure size 432x288 with 0 Axes>
```



In matlab , by using the canny filter the output we get as follows:



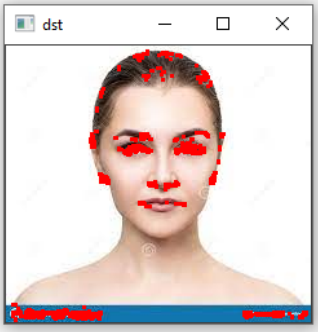
---

2. Harris corner detection python code:

After applying the python script of the harris corner detection i got the image as follows:
Here I have applied the harris corner detection python code to the woman_pic as shown, i got the following result as given:

```
In [*]: #harris corner
        #assign2

        import numpy as np
        import cv2 as cv
        filename = 'WOMAN_PIC.jpg'
        img = cv.imread(filename)
        gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
        gray = np.float32(gray)
        dst = cv.cornerHarris(gray,2,3,0.04)
        #result is dilated for marking the corners, not important
        dst = cv.dilate(dst,None)
        # Threshold for an optimal value, it may vary depending on the image.
        img[dst>0.01*dst.max()]=[0,0,255]
        cv.imshow('dst',img)
        if cv.waitKey(0) & 0xff == 27:
            cv.destroyAllWindows()
```



I applied the harris corner detection in matlab , the following is the output we got
Here it is detecting the corners in the image same as the python script as well:

Output of matlab Harris corner detection:

```
% Load images.
buildingDir = fullfile('data2_stitch');
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', 2000);

    % Compute T(n) * T(n-1) * ... * T(1)
    tforms(n).T = tforms(n).T * tforms(n-1).T;
end
```

```
Warning: Maximum number of trials reached. Consider increasing the maximum distance or decreasing the
desired confidence.
```

```
% 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);
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)
```

```
I1 = imread('WOMAN_PIC.jpg');
%I=rgb2gray(I1)
corners = detectHarrisFeatures(I);
imshow(I); hold on;
plot(corners.selectStrongest(50));
```

```
I = imread('WOMAN_PIC.jpg');
I = rgb2gray(I);
figure
imshow(I)
```


```
Iblur = imgaussfilt(I);
BW2 = edge(Iblur,'sobel');
BW2= edge(BW2,'canny');
imshow(BW2)
title('Canny Filter')
```