CSc 8830: Computer Vision

Homework Assignment 2

Submission in Classroom:

For Part A,

convert your problem solving by hand into a digital format (typed or scanned only. You can use camera scanner apps) and embedded/appended into the final PDF documentation. Camera images of paper worksheets will NOT be accepted

For Part B submit a MATLAB Live script (.mlx file) and also convert the .mlx file to PDF and append to PDF from Part A.

The MATLAB Live Script document must contain all the solutions, including graphs. The file must be saved as ".mlx" format. See here for live scripts: https://www.mathworks.com/help/matlab/matlab_prog/create-live-scripts.html

For Part C, manage all your code in a github repo for each assignment. Provide a link to the repo in the PDF document for Part A. Create a working demonstration of your application and record a screen-recording or a properly captured footage of the working system. Upload the video in the Google classroom submission.

Hardware: Unless otherwise specified, use the OAK-D Lite camera provided to you.
Software: Either of the following will work: Use MATLAB R2018b or later version as installed in your machine (installation instructions already provided) **OR** Use MATLAB Online (https://www.mathworks.com/products/matlab-online.html).

For OAK-D you can implement your solutions in either Python or C/C++: https://docs.luxonis.com/en/latest/

PART A: Theory

1. Pick a region of interest in the image making sure there is an EDGE in that region. Pick a 5 x 5 image patch in that region that constitutes the edge. Perform the steps of CANNY EDGE DETECTION manually and note the pixels that correspond to the EDGE.

```
def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.299, 0.587, 0.144])

img = mpimg.imread('/content/55parkplace_team2_1.png')

gray = rgb2gray(img)

plt.imshow(gray, cmap = plt.get_cmap('gray'))

plt.savefig('greyscaleimg.png')
plt.show()
```

```python
from PIL import Image

img = Image.open('/content/55parkplace_team2_1.png')
imgGray = img.convert('L')
imgGray.save('test_grayimg.png')
```

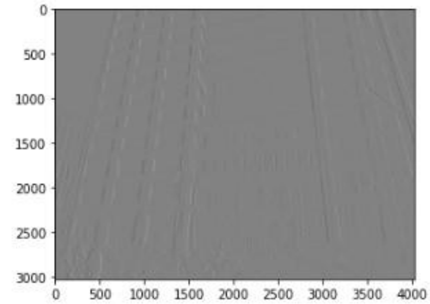
```python
#Import the image
img = cv2.imread('/content/test_grayimg.png', 1)
# define a function for box filter
def box_kernel(size):
  k = np.ones((size,size),np.float32)/(size**2)
  return k
# Basically, the smallest the kernel, the less visible is the blur. In our example, we will use a 5 by 5 kernel.
size=5
box_filter_img = cv2.filter2D(img,-1,box_kernel(size))
```

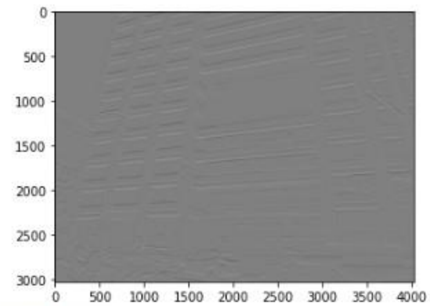
+ Code   + Text

```python
# Define a function for box filter
def gaussian_kernel(size, sigma=1):
  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
size=5
# Apply the gaussian blur
gaussian_filter_img = cv2.filter2D(img,-1,
                                   gaussian_kernel(size, sigma=1))
```

```python
def SobelFilter(img, direction):
    if(direction == 'x'):
        Gx = np.array([[-1,0,+1], [-2,0,+2],  [-1,0,+1]])
        Res = ndimage.convolve(img, Gx)
        #Res = ndimage.convolve(img, Gx, mode='constant', cval=0.0)
    if(direction == 'y'):
        Gy = np.array([[-1,-2,-1], [0,0,0], [+1,+2,+1]])
        Res = ndimage.convolve(img, Gy)
        #Res = ndimage.convolve(img, Gy, mode='constant', cval=0.0)

    return Res
def Normalize(img):
    #img = np.multiply(img, 255 / np.max(img))
    img = img/np.max(img)
    return img
```

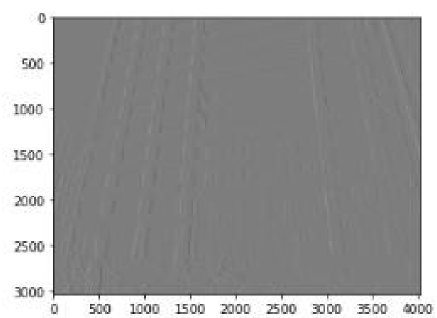
```python
gx = SobelFilter(img_guassian_filter, 'x')
gx = Normalize(gx)
plt.imshow(gx, cmap = plt.get_cmap('gray'))
plt.show()
gy = SobelFilter(img_guassian_filter, 'y')
gy = Normalize(gy)
plt.imshow(gy, cmap = plt.get_cmap('gray'))
plt.show()
type(gx)
gx.shape
```

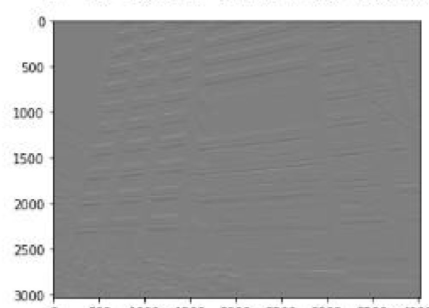




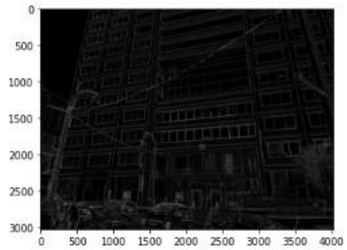

```
(3024, 4032)
```

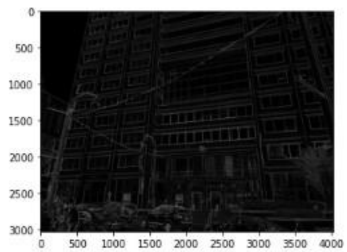
```
[ ] dx = ndimage.sobel(img_guassian_filter, axis=1) # horizontal derivative
    plt.imshow(dx, cmap = plt.get_cmap('gray'))
    plt.show()
    dy = ndimage.sobel(img_guassian_filter, axis=0) # vertical derivative
    plt.imshow(dy, cmap = plt.get_cmap('gray'))
    plt.show()
    type(dx)
    dx.shape
    dy.shape
```



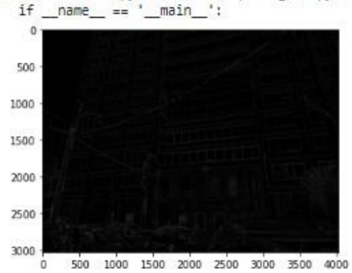


```
Mag = np.hypot(gx,gy)
Mag.shape
plt.imshow(Mag, cmap = plt.get_cmap('gray'))
plt.show()
mag = np.hypot(dx,dy)
mag.shape
plt.imshow(mag, cmap = plt.get_cmap('gray'))
plt.show()
```



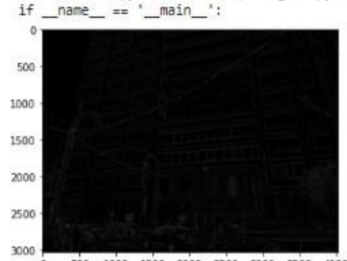


```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: RuntimeWarning: invalid value encountered in true_divide
  if __name__ == '__main__':
```



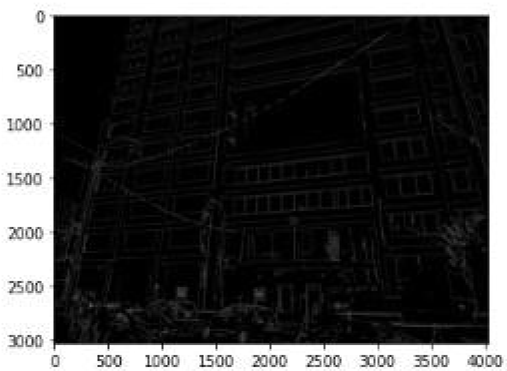
```
nms = NonMaxSupWithInterpol(mag, gradient, dx, dy)
nms = Normalize(nms)
plt.imshow(nms, cmap = plt.get_cmap('gray'))
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: RuntimeWarning: invalid value encountered in true_divide
  if __name__ == '__main__':
```



```python
# Double threshold Hysterisis
# Note that I have used a very slow iterative approach for ease of understanding, a faster implementation using recursion can be done instead
# This recursive approach would recurse through every strong edge and find all connected weak edges
def DoThreshHyst(img):
    highThresholdRatio =0.32
    lowThresholdRatio = 0.30
    GSup = np.copy(img)
    h = int(GSup.shape[0])
    w = int(GSup.shape[1])
    highThreshold = np.max(GSup) * highThresholdRatio
    lowThreshold = highThreshold * lowThresholdRatio
    x = 0.1
    oldx=0

    # The while loop is used so that the loop will keep executing till the number of strong edges do not change, i.e all weak edges connected to strong edges have been found
#     while(oldx != x):
#         oldx = x
    for i in range(1,h-1):
        for j in range(1,w-1):
            if(GSup[i,j] > highThreshold):
                GSup[i,j] = 1
            elif(GSup[i,j] < lowThreshold):
                GSup[i,j] = 0
            else:
                if((GSup[i-1,j-1] > highThreshold) or
                    (GSup[i-1,j] > highThreshold) or
                    (GSup[i-1,j+1] > highThreshold) or
                    (GSup[i,j-1] > highThreshold) or
                    (GSup[i,j+1] > highThreshold) or
                    (GSup[i+1,j-1] > highThreshold) or
                    (GSup[i+1,j] > highThreshold) or
                    (GSup[i+1,j+1] > highThreshold)):
                    GSup[i,j] = 1
#         x = np.sum(GSup == 1)

    GSup = (GSup == 1) * GSup # This is done to remove/clean all the weak edges which are not connected to strong edges

    return GSup
```

```python
Final_Image = DoThreshHyst(NMS)
plt.imshow(Final_Image, cmap = plt.get_cmap('gray'))
plt.show()
```



2. Pick a region of interest in the image making sure there is a CORNER in that region. Pick a 5 x 5 image patch in that region that constitutes the corner. Perform the steps of HARRIS CORNER DETECTION manually and note the pixels that correspond to the CORNER.

```python
#grey sclae conversion

def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.299, 0.587, 0.144])

img = mpimg.imread('/content/55parkplace_team2_1.png')

gray = rgb2gray(img)

plt.imshow(gray, cmap = plt.get_cmap('gray'))

plt.savefig('greyscaleimg.png')
plt.show()
```

[ ] from PIL import Image

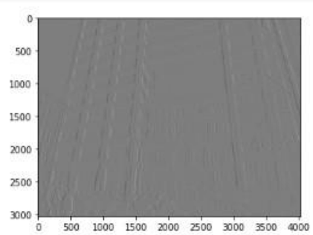
```python
from PIL import Image

img = Image.open('/content/55parkplace_team2_1.png')
imgGray = img.convert('L')
imgGray.save('test_grayimg.png')
```

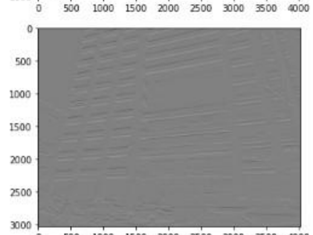
```python
#smoothing

img=imageio.imread('/content/test_grayimg.png')
img=img.astype('int32')
plt.imshow(img,cmap=plt.get_cmap('gray'))
plt.show()
```

```
#Applying gaussain filter

img_gaussian = ndimage.gaussian_filter(img, sigma= 1.4)
plt.imshow(img_gaussian ,cmap=plt.get_cmap('gray'))
plt.show()
```


```
#Applying Sobel filter
```

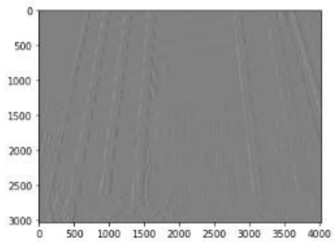
```
gx = SobelFilter(img_gaussian, 'x')
gx = Normalize(gx)
plt.imshow(gx, cmap = plt.get_cmap('gray'))
plt.show()
gy = SobelFilter(img_gaussian, 'y')
gy = Normalize(gy)
plt.imshow(gy, cmap = plt.get_cmap('gray'))
plt.show()
type(gx)
gx.shape
```

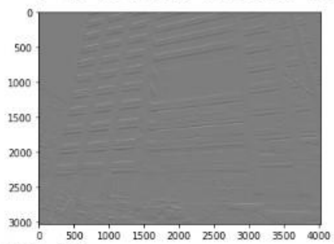





```
(3024, 4032)
```

```python
dx = ndimage.sobel(img_gaussian, axis=1) # horizontal derivative
plt.imshow(dx, cmap = plt.get_cmap('gray'))
plt.show()
dy = ndimage.sobel(img_gaussian, axis=0) # vertical derivative
plt.imshow(dy, cmap = plt.get_cmap('gray'))
plt.show()
type(dx)
dx.shape
dy.shape
```





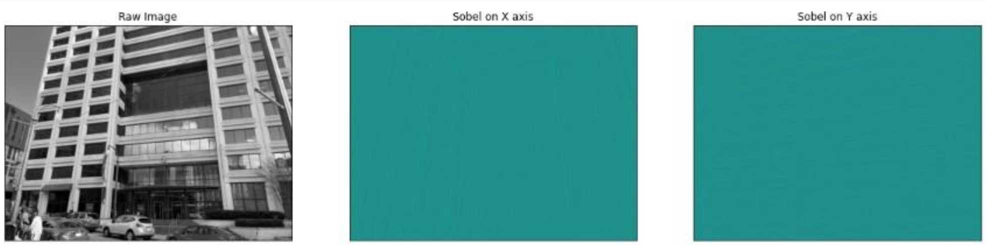

```
(3024, 4032)
```

```python
def convolve(img, kernel):
    """
    Convolve function for odd dimensions.
    IT CONVOLVES IMAGES
    """
    if kernel.shape[0] % 2 != 1 or kernel.shape[1] % 2 != 1:
        raise ValueError("Only odd dimensions on filter supported")

    img_height = img.shape[0]
    img_width = img.shape[1]
    pad_height = kernel.shape[0] // 2
    pad_width = kernel.shape[1] // 2

    pad = ((pad_height, pad_height), (pad_height, pad_width))
    g = np.empty(img.shape, dtype=np.float64)
    img = np.pad(img, pad, mode='constant', constant_values=0)
    # Do convolution
    for i in np.arange(pad_height, img_height+pad_height):
        for j in np.arange(pad_width, img_width+pad_width):
          roi = img[i - pad_height:i + pad_height +
                    1, j - pad_width:j + pad_width + 1]
          g[i - pad_height, j - pad_width] = (roi*kernel).sum()

    if (g.dtype == np.float64):
        kernel = kernel / 255.0
        kernel = (kernel*255).astype(np.uint8)
    else:
        g = g + abs(np.amin(g))
        g = g / np.amax(g)
        g = (g*255.0)
    return g
```

```python
def harris(img, threshold=0.5):

    img_cpy = img.copy() # copying image

    img1_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # grayscaling (0-1)

    dx = convolve(img1_gray, SOBEL_X) # convolving with sobel filter on X-axis
    dy = convolve(img1_gray, SOBEL_Y) # convolving with sobel filter on Y-axis
    # square of derivatives
    dx2 = np.square(dx)
    dy2 = np.square(dy)
    dxdy = dx*dy #cross filtering
    # gauss filter for all directions (x,y,cross axis)
    g_dx2 = convolve(dx2, GAUSS)
    g_dy2 = convolve(dy2, GAUSS)
    g_dxdy = convolve(dxdy, GAUSS)
    # Harris Function
    harris = g_dx2*g_dy2 - np.square(g_dxdy) - 0.12*np.square(g_dx2 + g_dy2) # r(harris) = det - k*(trace**2)
    # Normalizing inside (0-1)
    cv2.normalize(harris, harris, 0, 1, cv2.NORM_MINMAX)

    # find all points above threshold (nonmax supression line)
    loc = np.where(harris >= threshold)
    # drawing filtered points
    for pt in zip(*loc[::-1]):
        cv2.circle(img_cpy, pt, 3, (0, 0, 255), -1)

    return img_cpy,g_dx2,g_dy2,dx,dy,loc
```

[ ]
```python
import numpy as np
import cv2
import matplotlib.pyplot as plt
```

```python
#display raw & filtered images
plt.figure(figsize=(20, 20))
plt.subplot(131), plt.imshow(img)
plt.title("Raw Image"), plt.xticks([]), plt.yticks([])
plt.subplot(132), plt.imshow(dx)
plt.title("Sobel on X axis"), plt.xticks([]), plt.yticks([])
plt.subplot(133), plt.imshow(dy)
plt.title("Sobel on Y axis"), plt.xticks([]), plt.yticks([])
plt.show()
```



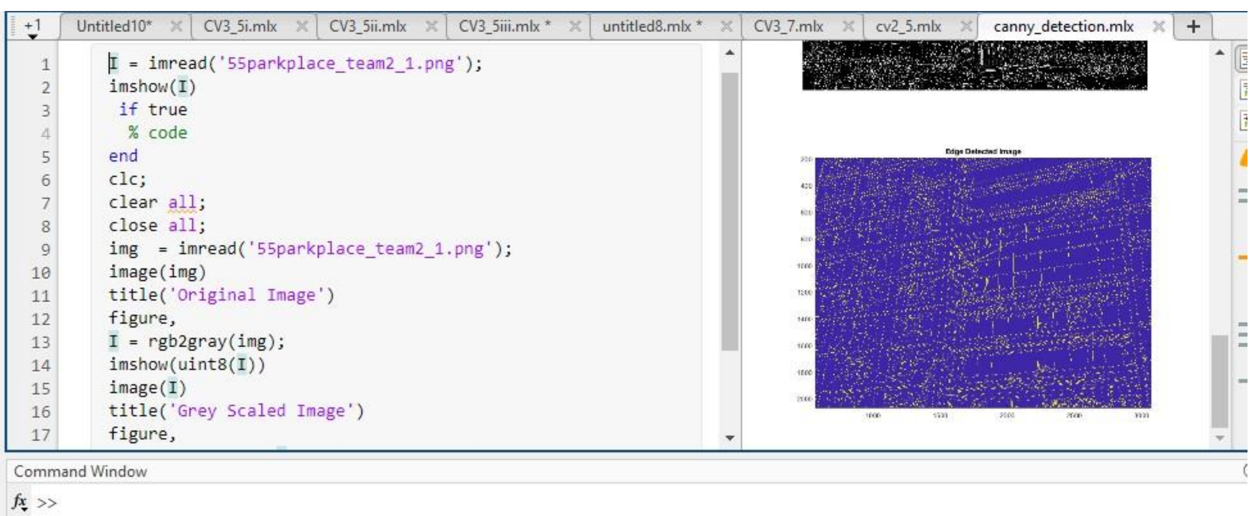
[ ]
```python
plt.figure(figsize=(20, 20))
```

```python
plt.figure(figsize=(20, 20))
plt.subplot(131), plt.imshow(img)
plt.title("Raw Image"), plt.xticks([]), plt.yticks([])
plt.subplot(132), plt.imshow(g_dx2)
plt.title("Gauss on X axis"), plt.xticks([]), plt.yticks([])
plt.subplot(133), plt.imshow(g_dy2)
plt.title("Gauss on Y axis"), plt.xticks([]), plt.yticks([])
plt.show()
```

[ ]
```python
plt.figure(figsize=(20, 20))
```

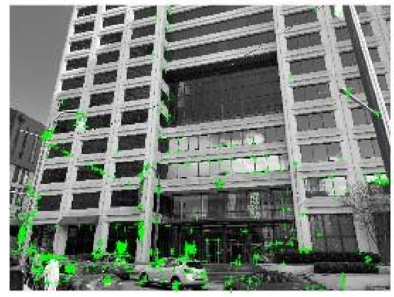
PART B: MATLAB Prototyping

3. Compare the outcome of problem (1) with MATLAB's Canny edge detection function.



4. Compare the outcome of problem (2) with MATLAB's Harris corner detection function.

```
I = imread('55parkplace_team2_1.png');
HI = rgb2gray(I);
corners = detectHarrisFeatures(HI);
imshow(HI);
hold on;
plot(corners.selectStrongest(1000));
```



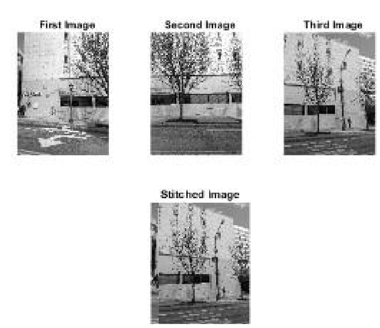
5. Implement the image stitching application in MATLAB (not necessary to be real-time). Test your application for any FIVE of a set of 3 image-set available in the gsu_building_database. That is, your stitching application should stitch 3 images. You must test the performance of your application for FIVE such sets.

https://drive.google.com/drive/folders/1cgVYdrzn9yUpYYi14mgvNyQUv8Ym5gui?usp=sharing

```
clear all;
close all;
F = imread("C:\Users\Sahithi Kolla\Downloads\GSU_Building_Dat
S = imread("C:\Users\Sahithi Kolla\Downloads\GSU_Building_Dat
V = imread("C:\Users\Sahithi Kolla\Downloads\GSU_Building_Dat

%Converting color images to Grayscale
F = im2double(rgb2gray(F));
S = im2double(rgb2gray(S));
V = im2double(rgb2gray(V));
[rows cols] = size(F);
Tmp = [];
Tmp1 = [];
temp = 0;
% Saving the patch(rows x 5 columns) of second(S) & third(V)
% S1 & V1 resp for future use.
for i = 1:rows
    for j = 1:5
        S1(i,j) = S(i,j);
        V1(i,j) = V(i,j);
    end
end
% Performing Correlation i.e. Comparing the (rows x 5 column)
% first image with patch of second image i.e. S1 saved earlie
for k = 0:cols-5 % (cols - 5) prevents j from going beyond bc
    for j = 1:5
```

```
    temp = 0;
end
[Min_value, Index] = max(Tmp);% Gets the Index with maximum v
% Determining the number of columns of new image. Rows remair
n_cols = Index + cols - 1;
Opimg = [];
for i = 1:rows
    for j = 1:Index-1
        Opimg(i,j) = F(i,j);% First image is pasted till Inde
    end
    for k = Index:n_cols
        Opimg(i,k) = S(i,k-Index+1);%Second image is pasted a
    end
end
[r_Opimg c_Opimg] = size(Opimg);
% Performing Correlation i.e. Comparing the (rows x 5 column)
% second image with patch of third image i.e. V1 saved earlie
for k = 0:c_Opimg-5% to prevent j to go beyond boundaries.
    for j = 1:5
        Opimg1(:,j) = Opimg(:,k+j);% Forming patch of rows x
    end
    temp = corr2(Opimg1,V1);% comparing the patches using cor
    Tmp1 = [Tmp1 temp]; % Tmp keeps growing, forming a matrix
    temp = 0;
```

```
end
% Determining the size of third image for future use.
[r_V, c_V] = size(V);
[Min_value, Index] = max(Tmp1);
% Determining new column for final stitched image.
% Rows remain the same.
n_cols = Index + c_V - 1;
Opimg1 = [];
for i = 1:rows
    for j = 1:Index-1
        Opimg1(i,j) = Opimg(i,j);% Previous stitched image is
    end
    for k = Index:n_cols
        Opimg1(i,k) = V(i,k-Index+1);%Third image is pasted a
    end
end
% Determining the size of Final Stitched image.
[r_Opimg c_Opimg] = size(Opimg1);
figure,
subplot(2,3,1);
imshow(F);axis ([1 c_Opimg 1 r_Opimg])
title('First Image');
subplot(2,3,2);
imshow(S);axis ([1 c_Opimg 1 r_Opimg])
title('Second Image');
```



```
Opimg1 = [];
for i = 1:rows
    for j = 1:Index-1
        Opimg1(i,j) = Opimg(i,j);% Previous stitched image is
    end
    for k = Index:n_cols
        Opimg1(i,k) = V(i,k-Index+1);%Third image is pasted a
    end
end
% Determining the size of Final Stitched image.
[r_Opimg c_Opimg] = size(Opimg1);
figure,
subplot(2,3,1);
imshow(F);axis ([1 c_Opimg 1 r_Opimg])
title('First Image');
subplot(2,3,2);
imshow(S);axis ([1 c_Opimg 1 r_Opimg])
title('Second Image');
subplot(2,3,3);
imshow(V);axis ([1 c_Opimg 1 r_Opimg])
title('Third Image');
subplot(2,3,[4 5 6]);% Final Stitched image should get most c
imshow(Opimg1);axis ([1 c_Opimg 1 r_Opimg])
title('Stitched Image');
% End of Code
```

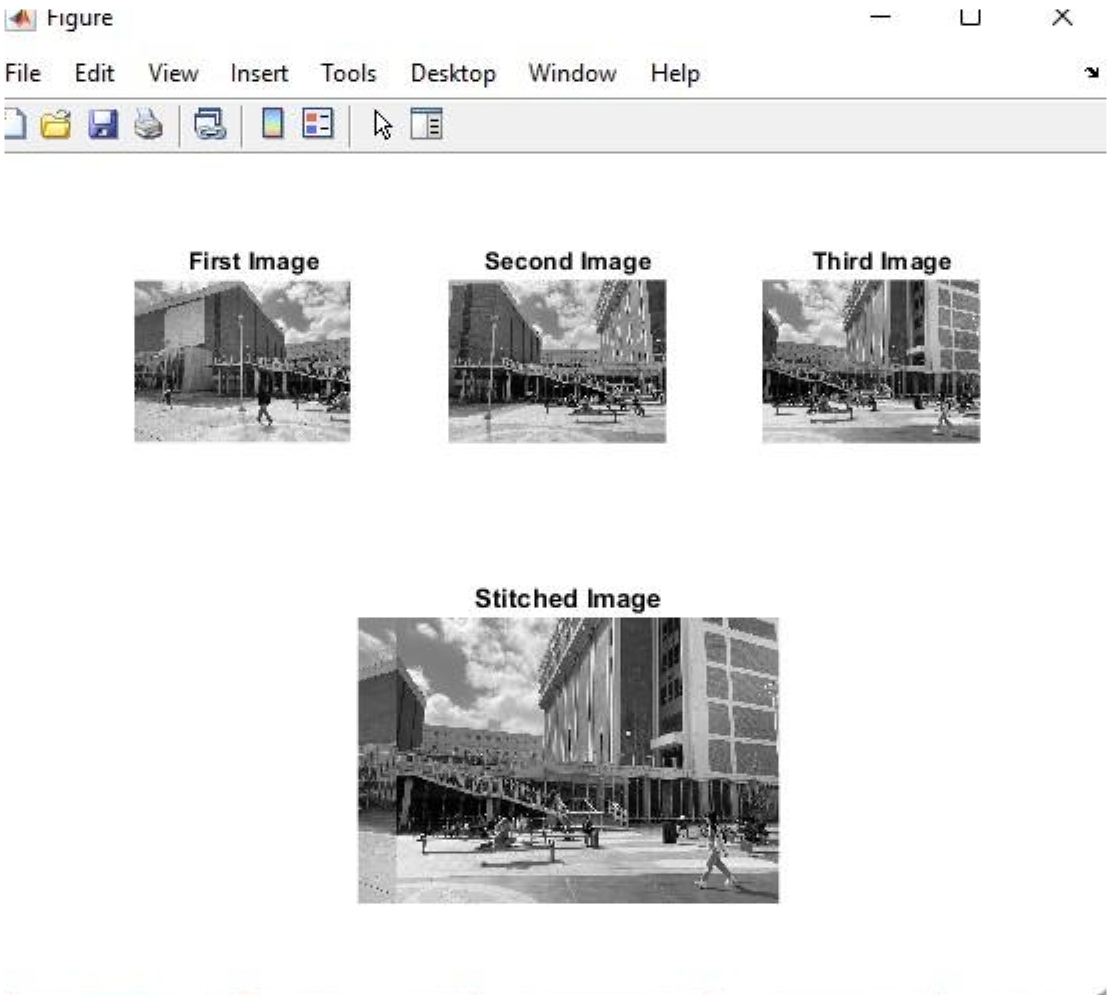
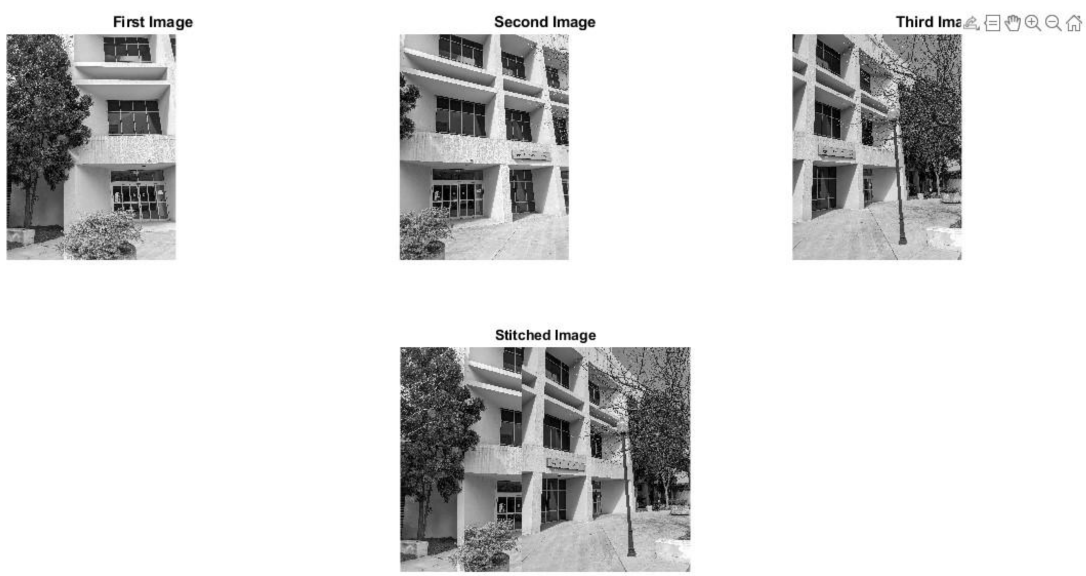


Figure
File
Edit
View
Insert
Tools
Desktop
Window
Help
First Image
Second Image
Third Image
Stitched Image


First Image
Second Image
Third Ima
Stitched Image

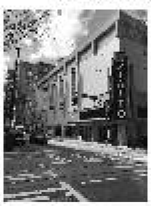
First Image

Second Image

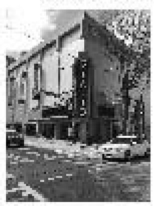
Third Image

Stitched Image

First Image

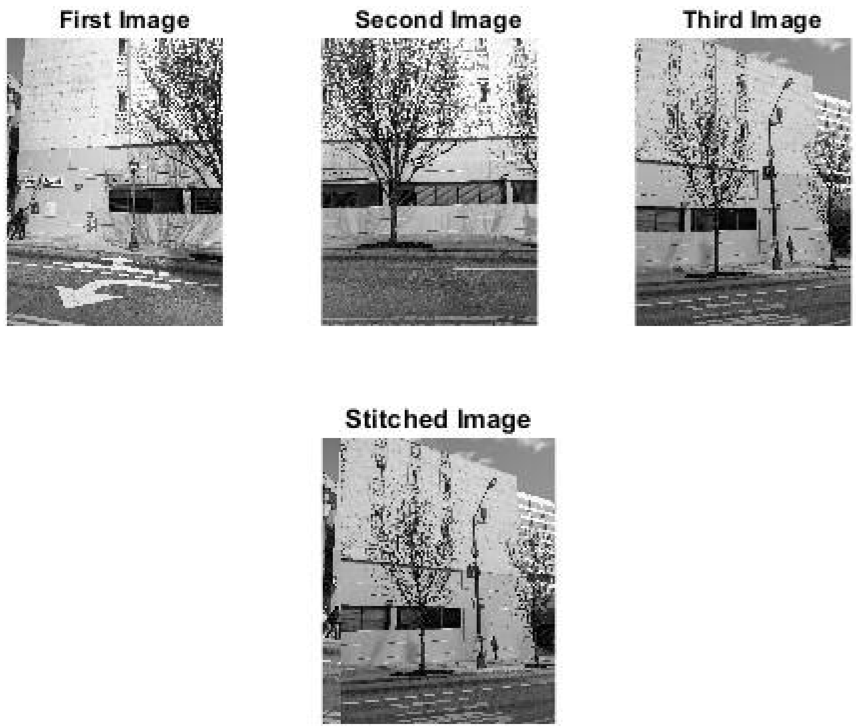


Second Image



Third Image

Stitched Image



PART C: Application development

6. Implement an application that will compute and display the INTEGRAL image feed along with the stereo and RGB feed. You **cannot** use a built-in function such as

“output = integral_image(input)”

```python
def myIntegralImage(img):
    temp = np.pad(img, 1, 'constant', constant_values=0).astype(np.longlong)
    # print(img.dtype)
    # print(temp.dtype)
    for j in range(1, temp.shape[0] - 1):
        for i in range(1, temp.shape[1] - 1):
            temp[j, i] = temp[j - 1, i] + temp[j, i - 1] + temp[j, i] - temp[j - 1, i - 1]

    temp = (temp - np.min(temp)) / (np.max(temp - np.min(temp))) * 255
    temp = temp.astype(img.dtype)
    return temp


# Create pipeline
pipeline = dai.Pipeline()

# Define sources and outputs
camRgb = pipeline.create(dai.node.ColorCamera)
xoutRgb = pipeline.create(dai.node.XLinkOut)

monoLeft = pipeline.create(dai.node.MonoCamera)
monoRight = pipeline.create(dai.node.MonoCamera)
xoutLeft = pipeline.create(dai.node.XLinkOut)
xoutRight = pipeline.create(dai.node.XLinkOut)

xoutLeft.setStreamName('left')
xoutRight.setStreamName('right')
xoutRgb.setStreamName("rgb")


# Properties
monoLeft.setBoardSocket(dai.CameraBoardSocket.LEFT)
monoLeft.setResolution(dai.MonoCameraProperties.SensorResolution.THE_480_P)
monoRight.setBoardSocket(dai.CameraBoardSocket.RIGHT)
monoRight.setResolution(dai.MonoCameraProperties.SensorResolution.THE_480_P)

camRgb.setPreviewSize(300, 300)
camRgb.setInterleaved(False)
camRgb.setColorOrder(dai.ColorCameraProperties.ColorOrder.RGB)

# Linking
monoRight.out.link(xoutRight.input)
monoLeft.out.link(xoutLeft.input)
camRgb.preview.link(xoutRgb.input)

# Connect to device and start pipeline
with dai.Device(pipeline) as device:
    print('Connected cameras: ', device.getConnectedCameras())

    # Output queues will be used to get the grayscale frames from the outputs defined above
    qLeft = device.getOutputQueue(name="left", maxSize=4, blocking=False)
    qRight = device.getOutputQueue(name="right", maxSize=4, blocking=False)
    qRgb = device.getOutputQueue(name="rgb", maxSize=4, blocking=False)
    while True:
        # Instead of get (blocking), we use tryGet (non-blocking) which will return the available data or None otherwise
        inLeft = qLeft.tryGet()
        inRight = qRight.tryGet()
        # print(inLeft.getCvFrame().shape)
        if inLeft is not None:
            iL = inLeft.getCvFrame()
            mii = myIntegralImage(iL)
            # print(mii,np.sum(iL))
            # assert np.sum(iL) == mii[-2,-2]
            # print(iL.shape)
            cv2.imshow("left", iL)
            cv2.imshow("integral image", myIntegralImage(iL))
```

```
# Connect to device and start pipeline
with dai.Device(pipeline) as device:
    print('Connected cameras: ', device.getConnectedCameras())

    # Output queues will be used to get the grayscale frames from the outputs defined above
    qLeft = device.getOutputQueue(name="left", maxSize=4, blocking=False)
    qRight = device.getOutputQueue(name="right", maxSize=4, blocking=False)
    qRgb = device.getOutputQueue(name="rgb", maxSize=4, blocking=False)
    while True:
        # Instead of get (blocking), we use tryGet (non-blocking) which will return the available data or None otherwise
        inLeft = qLeft.tryGet()
        inRight = qRight.tryGet()
        # print(inLeft.getCvFrame().shape)
        if inLeft is not None:
            iL = inLeft.getCvFrame()
            mii = myIntegralImage(iL)
            # print(mii,np.sum(iL))
            # assert np.sum(iL) == mii[-2,-2]
            # print(iL.shape)
            cv2.imshow("left", iL)
            cv2.imshow("integral image", myIntegralImage(iL))

        if inRight is not None:
            cv2.imshow("right", inRight.getCvFrame())

        inRgb = qRgb.get()  # blocking call, will wait until a new data has arrived

        # Retrieve 'bgr' (opencv format) frame
        cv2.imshow("rgb", inRgb.getCvFrame())

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

7. Implement the image stitching, for at least 1 pair of images. This should function in real-time. You **can** use any type of features. You **can** use built-in libraries/tools provided by the DepthAI API.

If available, you can also simply call any built-in function “image_stitch(image1, image1)”. **However,** in that case, you need to show a 180 or 360degree panoramic output.

```
import numpy as np
import cv2
import glob
import imutils
from google.colab.patches import cv2_imshow
```

```
[ ] image_paths = glob.glob('/content/unstiched_images/*.jpeg')
    images = []
```

```
[ ] !curl -o logo.png https://colab.research.google.com/img/colab_favicon_256px.png
    import cv2

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  4534  100  4534    0     0  70843      0 --:--:-- --:--:-- --:--:-- 70843
```

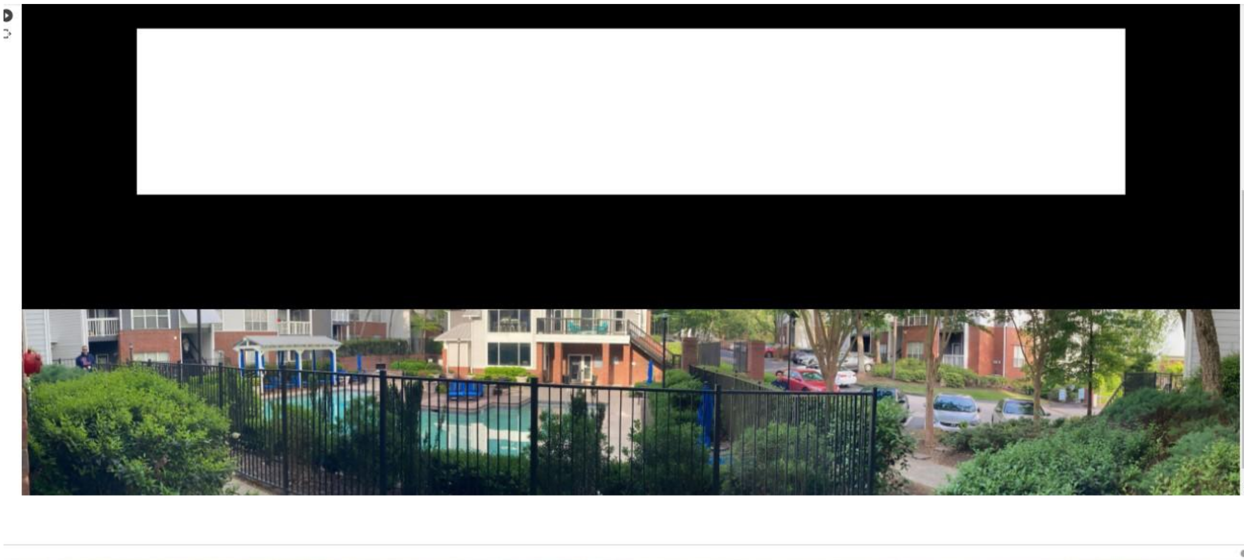
```
[ ] for image in image_paths:
        img = cv2.imread(image)
        images.append(img)
        cv2_imshow(img)
        cv2.waitKey(0)
```

```
[ ] imageStitcher = cv2.Stitcher_create()

    error, stitched_img = imageStitcher.stitch(images)
```

```
[ ]  if not error:

        cv2.imwrite("stitchedOutput.png", stitched_img)
        cv2_imshow(stitched_img)
        cv2.waitKey(0)

        stitched_img = cv2.copyMakeBorder(stitched_img, 10, 10, 10, 10, cv2.BORDER_CONSTANT, (0,0,0))
```


```
if not error:

    cv2.imwrite("stitchedOutput.png", stitched_img)
    cv2_imshow(stitched_img)
```


```
if not error:

    cv2.imwrite("stitchedOutput.png", stitched_img)
    cv2_imshow(stitched_img)
    cv2.waitKey(0)

    stitched_img = cv2.copyMakeBorder(stitched_img, 10, 10, 10, 10, cv2.BORDER_CONSTANT, (0,0,0))

    gray = cv2.cvtColor(stitched_img, cv2.COLOR_BGR2GRAY)
    thresh_img = cv2.threshold(gray, 0, 255 , cv2.THRESH_BINARY)[1]

    cv2_imshow(thresh_img)
    cv2.waitKey(0)

    contours = cv2.findContours(thresh_img.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    contours = imutils.grab_contours(contours)
    areaOI = max(contours, key=cv2.contourArea)

    mask = np.zeros(thresh_img.shape, dtype="uint8")
    x, y, w, h = cv2.boundingRect(areaOI)
    cv2.rectangle(mask, (x,y), (x + w, y + h), 255, -1)

    minRectangle = mask.copy()
    sub = mask.copy()

    while cv2.countNonZero(sub) > 0:
        minRectangle = cv2.erode(minRectangle, None)
        sub = cv2.subtract(minRectangle, thresh_img)


    contours = cv2.findContours(minRectangle.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    contours = imutils.grab_contours(contours)
    areaOI = max(contours, key=cv2.contourArea)

    cv2_imshow(minRectangle)
    cv2.waitKey(0)

    x, y, w, h = cv2.boundingRect(areaOI)

    stitched_img = stitched_img[y:y + h, x:x + w]

    cv2.imwrite("stitchedOutputProcessed.png", stitched_img)
```



Questions:

Capture a 10 sec video footage using a camera of your choice. The footage should be taken with the camera in hand and you need to pan the camera slightly from left-right or right-left during the 10 sec duration.

For all the images, operate at grayscale

1. (25pts) Pick any image frame from the 10 sec video footage. Pick a region of interest in the image making sure there is an EDGE in that region. Pick a 5 x 5 image patch in that region that constitutes the edge. Perform the steps of CANNY EDGE DETECTION manually and note the pixels that correspond to the EDGE. Compare the outcome with MATLAB's Canny edge detection function.

2. (25pts) Pick any image frame from the 10 sec video footage. Pick a region of interest in the image making sure there is a CORNER in that region. Pick a 5 x 5 image patch in that region that constitutes the edge. Perform the steps of HARRIS CORNER DETECTION manually and note the pixels that correspond to the CORNER. Compare the outcome with MATLAB's Harris corner detection function.

3. (50pts) Consider an image pair from your footage where the images are separated by at least 2 seconds. Also ensure there is at least some overlap of scenes in the two images.

a. Pick a pixel (super-pixel patch as discussed in class) on image 1 and a corresponding pixel ((super-pixel patch as discussed in class)) on image 2 (the pixel on image 2 that corresponds to the same object area on image 1). Compute the SIFT feature for each of these 2 pixels manually. Compute the sum of squared difference (SSD) value between the SIFT vector for these two pixels. Verify your result using MATLAB -- The MATLAB code for SIFT feature extraction and matching can be downloaded from here: https://www.cs.ubc.ca/~lowe/keypoints/ (Please first read the ReadMe document in the folder to find instructions to execute the code).
b. Compute the Homography matrix between these two images manually. Verify your result using MATLAB.

You can make assumptions as necessary, however, justify them in your answers/description.