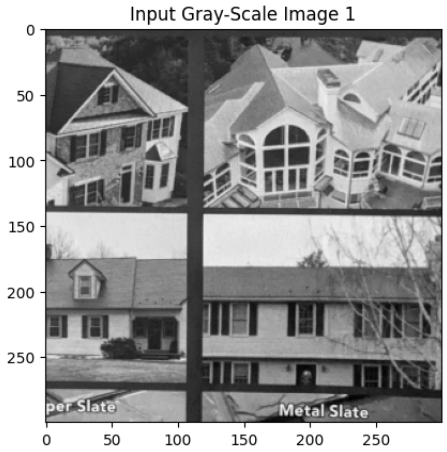
ESE 568 Project 3 documentation

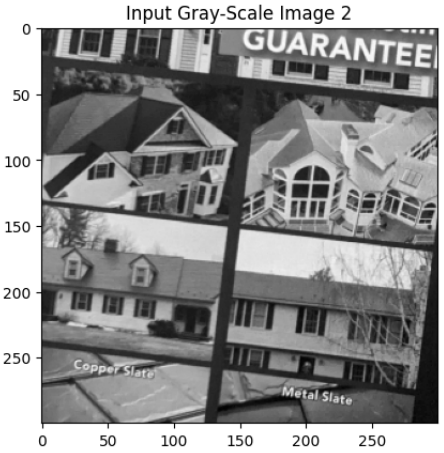
Taharina Tasnim (115613595)

October 25, 2023

Part 1: Image Filtering/Convolution

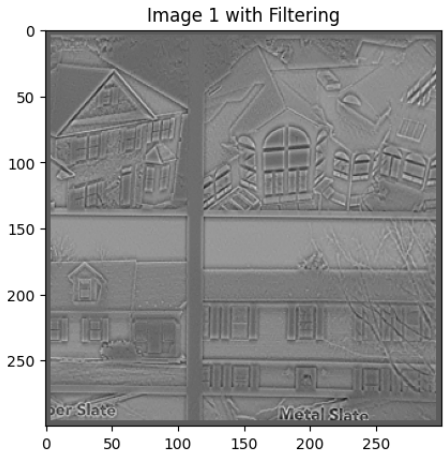
Input Images

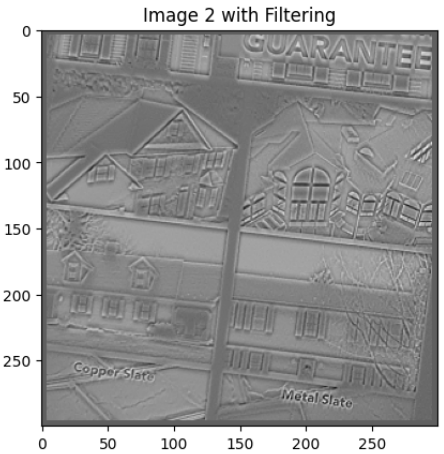




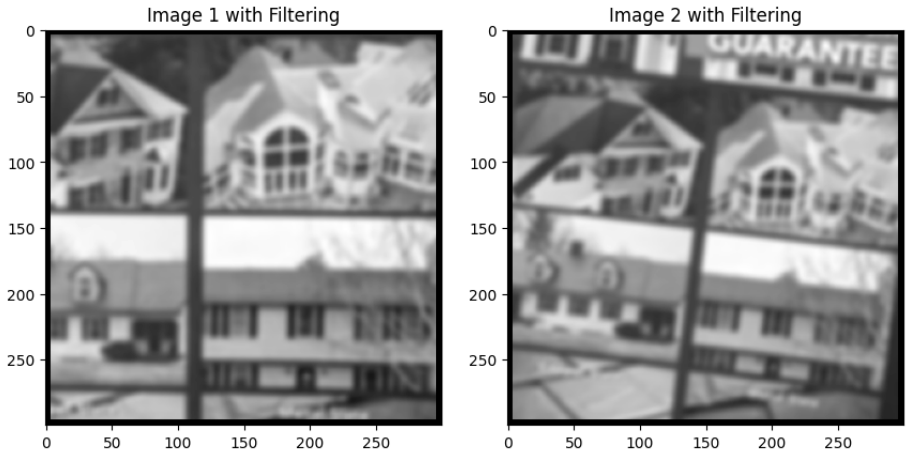
Output after filtering with reading coefficients from file:

```
0,0,0,-0,-1,0,0,0,0
0,0,0,-0,-1,0,0,0,0
0,0,0,0,-1,0,0,0,0
0,0,0,-1,-2,-1,0,0,0
0,0,-1,-2,16,-2,-1,0,0
0,0,0,-1,-2,-1,0,0,0
0,0,0,0,-1,0,0,0,0
0,0,0,0,-1,0,0,0,0
0,0,0,0,-1,0,0,0,0
```





Output after filtering with a Gaussian having the spread parameter sigma = M/4.0



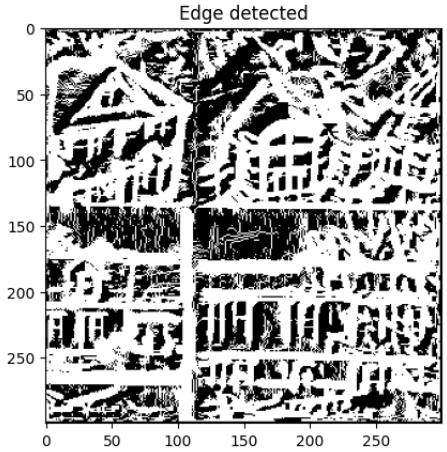
Code:

```
Part 1: Image filtering/convolution
# read grayscale image
f = io.imread('/content/drive/MyDrive/Colab
Notebooks/project3/images/pic1grey300.jpg')
f2nd = io.imread('/content/drive/MyDrive/Colab
Notebooks/project3/images/pic2grey300.jpg')
N, _ = f.shape

plt.imshow(f, cmap=plt.cm.gray)
plt.title('Input Gray-Scale Image 1')
plt.show()
plt.imshow(f2nd, cmap=plt.cm.gray)
plt.title('Input Gray-Scale Image 2')
plt.show()

# filter
M = 9
isReadingFromFile = input("Want to read the filter coefficients of from file?
(y/n): ").lower() == 'y'

if isReadingFromFile == True:
    # read from file
    g = np.loadtxt('/content/drive/MyDrive/Colab
Notebooks/project3/filter_g9x9.txt', dtype='i', delimiter=',')
else:
    # create a Gaussian filter
    sigma = M / 4.0
    g = np.zeros((M, M), dtype=np.float)
```

```
    center = (M-1) / 2
    for i in range(M):
        for j in range(M):
            x, y = i - center, j - center
            g[i, j] = np.exp(-(x ** 2 + y ** 2) / (2 * sigma ** 2)) / (2 *
np.pi * sigma ** 2)

# normalize all the filter coefficients
g = g / np.sum(g)

# convolution of image and filters
hg = np.zeros(f.shape, dtype = np.float)
hg2nd = np.zeros(f.shape, dtype = np.float)

for i in range((M-1)//2,N-(M-1)//2):
    for j in range((M-1)//2,N-(M-1)//2):
        for k in range(M):
            for l in range(M):
                hg[i,j] += g[k,l]*f[i-k+(M-1)//2,j-l+(M-1)//2]
                hg2nd[i,j] += g[k,l]*f2nd[i-k+(M-1)//2,j-l+(M-1)//2]


# Normalized output images
hg = ((hg - np.min(hg)) / (np.max(hg) - np.min(hg)) * 255).astype(np.uint8)
hg2nd = ((hg2nd - np.min(hg2nd)) / (np.max(hg2nd) - np.min(hg2nd)) *
255).astype(np.uint8)

plt.imshow(hg, cmap=plt.cm.gray)
plt.title('Image 1 with Filtering')
plt.show()
plt.imshow(hg2nd, cmap=plt.cm.gray)
plt.title('Image 2 with Filtering')
plt.show()
```

Part 2: Edge detection

Output of Input image 1 after detecting the edge



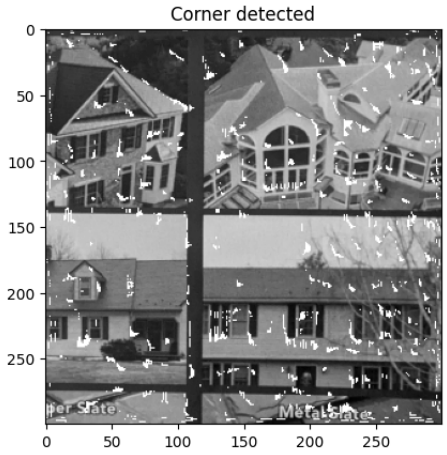
Code:

```
Part 2: Edge detection
# Create 1-D separable gaussian filter (Separable Filter)

def covWithSeparableGauss(_M, _sigma, _f):
    _center = (_M-1) / 2
    _Gauss = np.zeros((_M,1), dtype = np.float)

    for i in range(_M):
        _x = i - _center
        _Gauss[i] = np.exp(-_x**2 / (2 * _sigma**2)) / (np.sqrt(2 * np.pi) *
_sigma)

    _Gauss = _Gauss / np.sum(_Gauss)
    _smoothedf = _f.copy()

    # Smoothing along rows
    for i in range(N):
        for j in range((_M-1)//2,N-(_M-1)//2):
            _summ = 0
            for k in range(_M):
                _summ += _Gauss[k]*_f[i,j-k+(_M-1)//2]
            _smoothedf[i,j] = _summ

    # Smoothing along columns
    for j in range(N):
        for i in range((_M-1)//2,N-(_M-1)//2):
            _summ = 0
            for k in range(_M):
```

```
                _summ += _Gauss[k]*_smoothedf[i-k+(_M-1)//2,j]
            _smoothedf[i,j] = _summ

    return _smoothedf

M = 9
sigma = 2.0
smoothed_f = covWithSeparableGauss(M, sigma, f)
plt.imshow(smoothed_f, cmap=plt.cm.gray)
plt.title('Gaussian Smoothed Image for Edge Detection')
plt.show()


# compute edge
Gx = np.zeros(f.shape, dtype = np.float)
Gy = np.zeros(f.shape, dtype = np.float)
Gm = np.zeros(f.shape, dtype = np.float)
edgeDetectedImage = np.zeros(f.shape, dtype = np.float)
Threshold = 40

for i in range(N-1):
    for j in range(N-1):
        Gx[i,j] = smoothed_f[i,j+1] - smoothed_f[i,j]
        Gy[i,j] = smoothed_f[i+1,j] - smoothed_f[i,j]
        Gm[i,j] = np.sqrt(Gx[i,j]**2 + Gy[i,j]**2)
        if Gm[i,j] > Threshold:
            edgeDetectedImage[i,j] = 255
        else:
            edgeDetectedImage[i,j] = 0

plt.imshow(edgeDetectedImage, cmap=plt.cm.gray)
plt.title('Edge detected')
plt.show()
```

Part 3a: Corner detection

Output of Input image 1 after detecting corner



Code:

```
Part 3a: Corner detection
window_size = 11
window_sigma = 5.5

# step 1
M = 9
sigma = 2.0
smoothed_f = covWithSeparableGauss(M, sigma, f)

# step 2
Ix = np.zeros(f.shape, dtype = np.float)
Iy = np.zeros(f.shape, dtype = np.float)
IxIx = np.zeros(f.shape, dtype = np.float)
IyIy = np.zeros(f.shape, dtype = np.float)
IxIy = np.zeros(f.shape, dtype = np.float)

for i in range(N-1):
    for j in range(N-1):
        Ix[i,j] = (smoothed_f[i,j+1] - smoothed_f[i,j]) / 10
        Iy[i,j] = (smoothed_f[i+1,j] - smoothed_f[i,j]) / 10
        # step 3
        IxIx[i,j] = Ix[i,j]**2
        IyIy[i,j] = Ix[i,j]**2
        IxIy[i,j] = Iy[i,j] * Iy[i,j]

# step 4
smoothed_IxIx = covWithSeparableGauss(window_size, window_sigma, IxIx)
```

```
smoothed_IyIy = covWithSeparableGauss(window_size, window_sigma, IyIy)
smoothed_IxIy = covWithSeparableGauss(window_size, window_sigma, IxIy)

# step 5 and 6
R = np.zeros(f.shape, dtype = np.float)

for i in range(N):
    for j in range(N):
        Mat = np.array([[smoothed_IxIx[i,j], smoothed_IxIy[i,j]],
                        [smoothed_IxIy[i,j], smoothed_IyIy[i,j]]])
        R[i,j] = np.linalg.det(Mat) - 0.04 * np.trace(Mat)**2

# step 7
Threshold = 30000
corner_f = np.zeros(f.shape, dtype = np.float)  # to keep the corner index

for i in range(N):
    for j in range(N):
        if R[i,j] > Threshold:
            corner_f[i,j] = R[i,j]

for i in range(1, N-1):
    for j in range(1, N-1):
        if corner_f[i,j] != 0:
            if corner_f[i,j] > corner_f[i-1, j] and corner_f[i,j] >
corner_f[i+1, j] and corner_f[i,j] > corner_f[i, j-1] and corner_f[i,j] >
corner_f[i, j+1] and corner_f[i,j] > corner_f[i-1, j-1] and corner_f[i,j] >
corner_f[i-1, j+1] and corner_f[i,j] > corner_f[i+1, j-1] and corner_f[i,j] >
corner_f[i+1, j+1]:
                corner_f[i,j] = 1
            else:
                corner_f[i,j] = 0

# Non-Maxima Suppression
harrisf = f.copy()
for i in range(N):
    for j in range(N):
        if corner_f[i,j] == 1:
            harrisf[i,j] = 255

plt.imshow(harrisf, cmap=plt.cm.gray)
plt.title('Corner detected')
plt.show()
```

Part 3b: Local feature descriptor

Output of Normalised gradient direction histogram

```
pixel at (i,j) = ( 114 297 ) has histogram [ 0. 0. 0. 0. 172.87426218 2.30217289 25.8 0. ]
pixel at (i,j) = ( 115 3 ) has histogram [ 0. 0. 0. 0. 152.5 72.05425046 0.1 0. ]
pixel at (i,j) = ( 115 53 ) has histogram [ 0. 0. 0. 0. 96.9601365 86.82228402 25.3 0. ]
pixel at (i,j) = ( 115 72 ) has histogram [ 0. 0. 0. 0. 25.72380288 1.02333455 0.4 0. ]
pixel at (i,j) = ( 115 149 ) has histogram [ 0. 0. 0. 0. 184.21015261 70.50372711 0. 0. ]
...
pixel at (i,j) = ( 149 250 ) has histogram [0. 0. 0.2 0.2236068 0.4 0. 0. 0. ]
pixel at (i,j) = ( 149 259 ) has histogram [ 0. 0. 0. 0. 102.00019608 36.06244584 25.6 0. ]
pixel at (i,j) = ( 149 274 ) has histogram [ 0. 0. 0. 0. 225.20460238 0. 0. 0. ]
pixel at (i,j) = ( 149 275 ) has histogram [ 0. 0. 0. 0. 150.40279839 25.50019608 0.2 0. ]
...
pixel at (i,j) = ( 149 294 ) has histogram [ 0. 0. 0. 0. 152.65894029 1.78218697 0. 0. ]
pixel at (i,j) = ( 149 297 ) has histogram [ 0. 0. 0. 0. 214.36911177 0. 0. 0. ]
...
pixel at (i,j) = ( 150 233 ) has histogram [ 0. 0. 0. 0. 101.90196153 0.2236068 0.6 0. ]
pixel at (i,j) = ( 150 234 ) has histogram [ 0. 0. 0. 0. 25.50019608 0.78929222 0.6 0. ]
pixel at (i,j) = ( 150 245 ) has histogram [ 0. 0. 0. 0. 178.30078508 71.91310749 0. 0. ]
pixel at (i,j) = ( 150 249 ) has histogram [ 0. 0. 25.8 36.06244584 51. 0. 0. 0. ]
pixel at (i,j) = ( 150 259 ) has histogram [ 0. 0. 0. 0. 101.90039293 36.06244584 25.6 0. ]
...
pixel at (i,j) = ( 192 62 ) has histogram [ 0. 0. 0. 0. 127.2 0. 0.2 0. ]
pixel at (i,j) = ( 192 63 ) has histogram [ 0. 0. 0. 0. 203.1 0. 0.1 0. ]
pixel at (i,j) = ( 192 64 ) has histogram [ 0. 0. 0. 0. 202.6 35.85094141 0. 0. ]
pixel at (i,j) = ( 192 65 ) has histogram [ 0. 0. 0. 0. 177. 71.70188282 0. 0. ]
pixel at (i,j) = ( 192 145 ) has histogram [ 0. 0. 0. 0. 153.00058823 0. 0.2 0. ]
...
pixel at (i,j) = ( 298 289 ) has histogram [ 0. 0. 0. 0. 152.30433764 0. 0. 0. ]
pixel at (i,j) = ( 298 290 ) has histogram [ 0. 0. 0. 0. 126.90650281 0. 25.5 0. ]
pixel at (i,j) = ( 298 291 ) has histogram [ 0. 0. 0. 0. 101.60806507 25.30019763 25.5 0. ]
pixel at (i,j) = ( 298 292 ) has histogram [ 0. 0. 0. 0. 50.90628647 50.40199039 25.7 0. ]
pixel at (i,j) = ( 298 293 ) has histogram [ 0. 0. 0. 25.81936483 75.20279683 0.2 0. 0. ]
```

Code:

```
Part 3b: Local feature descriptor

histBinDirection = [0, 45, 90, 135, 180, 225, 270, 315]
binCnt = len(histBinDirection)

for i in range(N):
    for j in range(N):
        if corner_f[i,j] == 1:
            hist = np.zeros(binCnt)
            histN = np.zeros(binCnt)
            for x in range(-1, 2):
                for y in range(-1, 2):
                    theta = np.degrees(np.arctan2(Iy[i+x,j+y], Ix[i+x,j+y]))
                    if theta < 0:
                        theta += 360
                    magnitude = np.sqrt(Ix[i+x,j+y]**2 + Iy[i+x,j+y]**2)
                    binIdx = int((theta / 45) % binCnt)  # quantized
                    hist[binIdx] += magnitude

            maxbinIdx = np.argmax(hist)
            for x in range(binCnt):
```

```
                histN[(4+x) % binCnt] = hist[(maxbinIdx+x) % binCnt]    #
histBinDirection[4]=180

            np.set_printoptions(suppress=True)
            print('pixel at (i,j) = (', i, j, ') has histogram ', histN)
```

Part 4: Image matching

Output using image 1 and 2:

```
2D affine transformation matrix:
[[ 0.79 -0.09 68.37 ]
 [ 0.08  0.79 31.57 ]]
translation vector [ 68.37181742882609 , 31.567225343034266 ]
rotation angle = 5.890018713868986 deg
scaling factor = ( 0.7977201509778696 , 0.7927805717941157 )
```

Code:

```
Part 4: Image matching
# corner Pixels of image 1
A = np.array([[ 20,  68, 1, 0, 0, 0], [0,0,0,20,68,1],
 [ 40,  79, 1, 0, 0, 0], [0, 0, 0,  40,  79, 1],
 [ 43,  43, 1, 0, 0, 0], [0, 0, 0,  43,  43, 1],
 [ 73,  72, 1, 0, 0, 0], [0, 0, 0,  73,  72, 1],
 [ 83,  66, 1, 0, 0, 0], [0, 0, 0,  83,  66, 1],
 [106,  59, 1, 0, 0, 0], [0, 0, 0, 106,  59, 1],
 [125, 117, 1, 0, 0, 0], [0, 0, 0, 125, 117, 1],
 [155, 133, 1, 0, 0, 0], [0, 0, 0, 155, 133, 1],
 [180,  73, 1, 0, 0, 0], [0, 0, 0, 180,  73, 1],
 [184,  73, 1, 0, 0, 0], [0, 0, 0, 184,  73, 1],
 [199, 103, 1, 0, 0, 0], [0, 0, 0, 199, 103, 1],
 [236,  56, 1, 0, 0, 0], [0, 0, 0, 236,  56, 1],
 [268,  98, 1, 0, 0, 0], [0, 0, 0, 268,  98, 1],
 [  3, 175, 1, 0, 0, 0], [0, 0, 0,   3, 175, 1],
 [ 29, 176, 1, 0, 0, 0], [0, 0, 0,  29, 176, 1],
 [ 17, 235, 1, 0, 0, 0], [0, 0, 0,  17, 235, 1],
 [106, 175, 1, 0, 0, 0], [0, 0, 0, 106, 175, 1],
 [119, 182, 1, 0, 0, 0], [0, 0, 0, 119, 182, 1],
 [138, 233, 1, 0, 0, 0], [0, 0, 0, 138, 233, 1],
 [188, 234, 1, 0, 0, 0], [0, 0, 0, 188, 234, 1],
 [205, 234, 1, 0, 0, 0], [0, 0, 0, 205, 234, 1],
 [298, 276, 1, 0, 0, 0], [0, 0, 0, 298, 276, 1]])

# corner Pixels of image 2
b = np.reshape(np.array([[ 78,  89],
```

```
 [ 93,  97],
 [ 99,  69],
 [123,  94],
 [128,  90],
 [147,  87],
 [158, 134],
 [180, 148],
 [204, 105],
 [207, 104],
 [216, 129],
 [252,  95],
 [271, 129],
 [ 56, 169],
 [ 76, 172],
 [ 60, 218],
 [137, 178],
 [146, 185],
 [157, 226],
 [197, 231],
 [209, 233],
 [283, 275]]), (44, 1))

x = np.linalg.inv(np.transpose(A) @ A) @ np.transpose(A) @ b
x = np.reshape(x, (2,3))
print('2D affine transformation matrix:')
np.set_printoptions(precision=2)
print(x)
print('translation vector [', x[0,2], ',', x[1,2], ']')
print('rotation angle = ', np.degrees(np.arctan2(x[1,0], x[0,0])), 'deg')
print('scaling factor = (', np.sqrt(x[0,0]**2 + x[1,0]**2), ',' ,
np.sqrt(x[0,1]**2 + x[1,1]**2), ')')
```