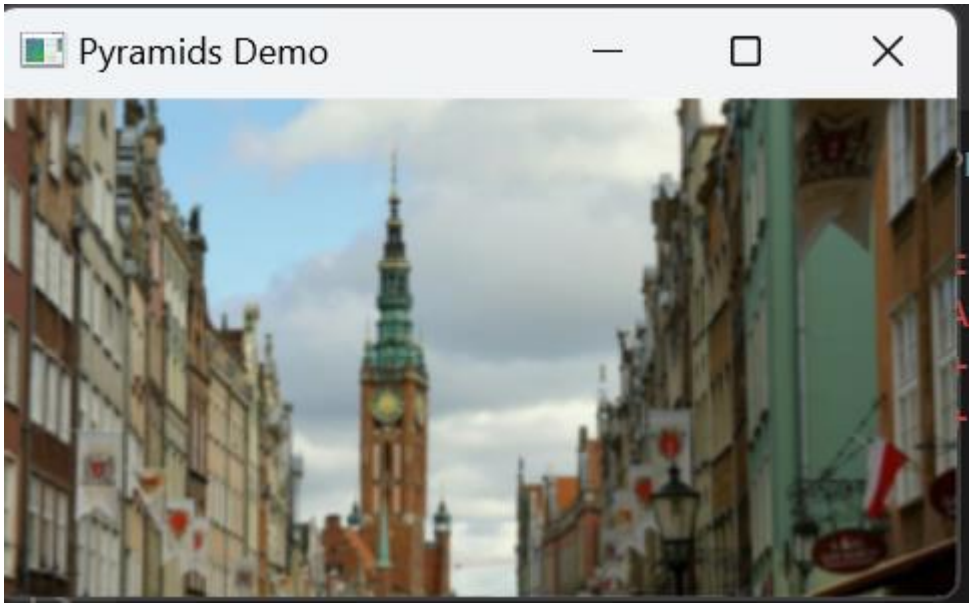
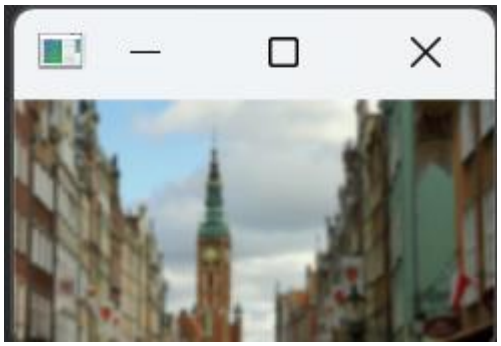
1) Image pyramids

Code

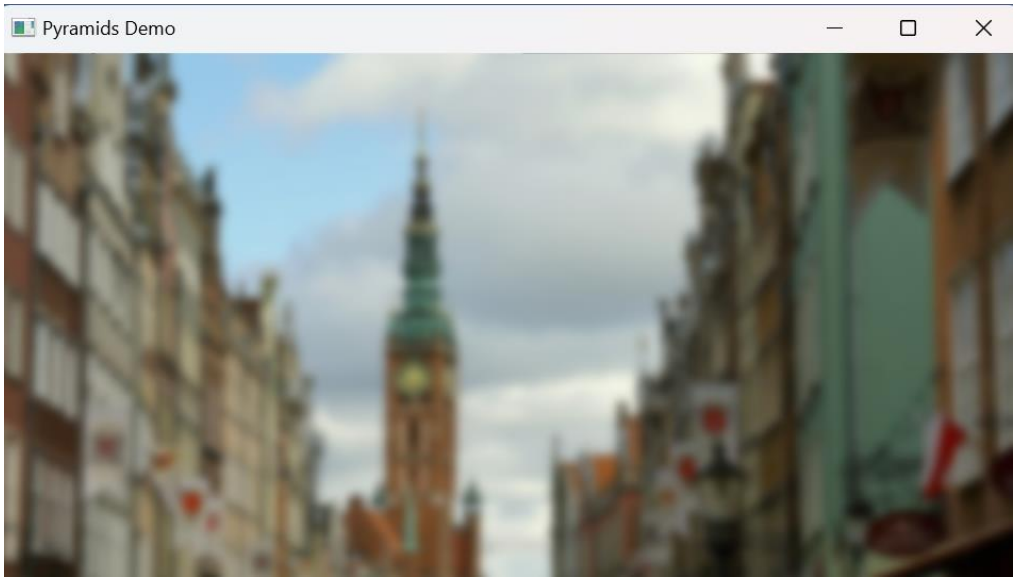
```
import sys
import cv2 as cv

def main(argv):
    print("""
    Zoom In-Out demo
    ------------------
    * [i] -> Zoom [i]n
    * [o] -> Zoom [o]ut
    * [ESC] -> Close program
    """)

    filename = argv[0] if len(argv) > 0 else 'chicky_512.png'
    # Load the image
    src = cv.imread(cv.samples.findFile(filename))
    # Check if image is loaded fine
    if src is None:
        print ('Error opening image!')
        print ('Usage: pyramids.py [image_name -- default
../data/chicky_512.png] \n')
        return -1

    while 1:
        rows, cols, _channels = map(int, src.shape)

        cv.imshow('Pyramids Demo', src)

        k = cv.waitKey(0)
        if k == 27:
            break

        elif chr(k) == 'i':
            src = cv.pyrUp(src, dstsize=(2 * cols, 2 * rows))
            print ('** Zoom In: Image x 2')

        elif chr(k) == 'o':
            src = cv.pyrDown(src, dstsize=(cols // 2, rows // 2))
            print ('** Zoom Out: Image / 2')

    cv.destroyAllWindows()
    return 0

if __name__ == "__main__":
    main(sys.argv[1:])
```

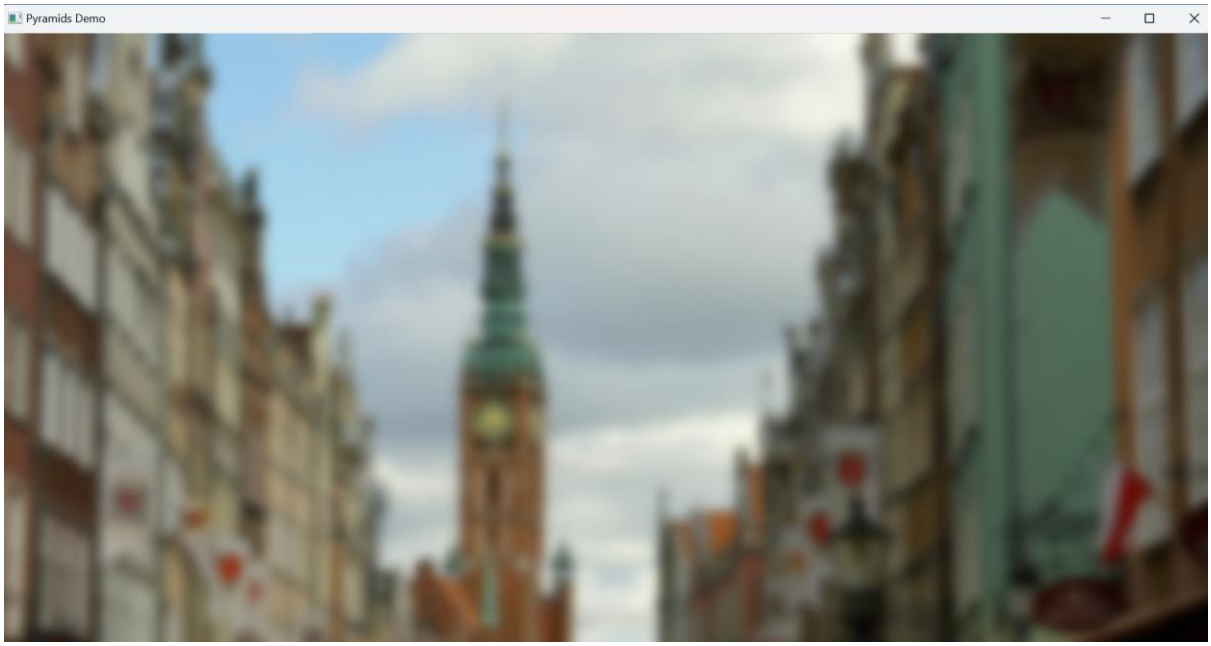
Downsampling





Upsampling





2) Canny filter

To add Gaussian noise

Code

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

def add_gaussian_noise(image, mean=0, sigma=50):
    row, col, ch = image.shape
    gauss = np.random.normal(mean, sigma, (row, col, ch))
    noisy_image = image + gauss.astype(np.uint8)
    return noisy_image
```

```
# Load image
image = cv2.imread('/content/city.jpg')

# Add Gaussian noise
noisy_image = add_gaussian_noise(image)

# Display original and noisy images
cv2_imshow(image)
cv2_imshow(noisy_image)
#cv2.imshow('Noisy Image', noisy_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Applying canny filter

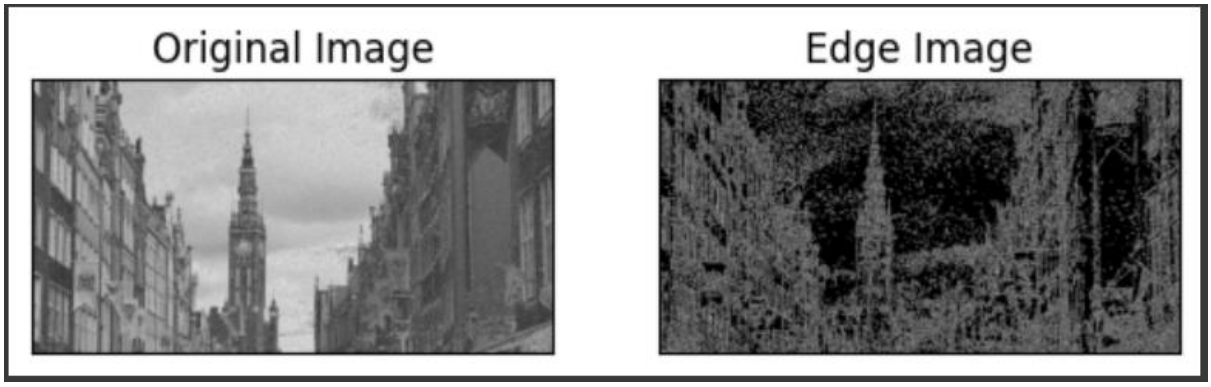
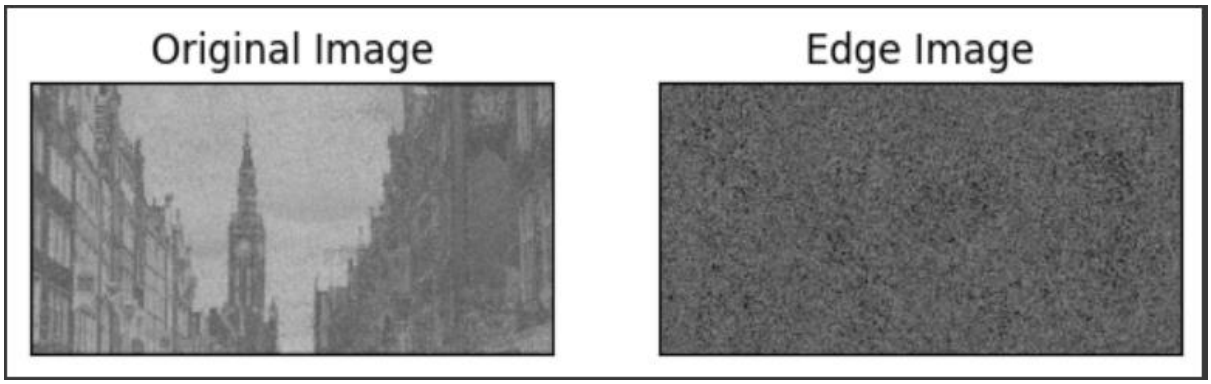


Image with more gaussian noise

After applying canny filter



Canny filter code

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('/content/gaussian2.png', cv.IMREAD_GRAYSCALE)
assert img is not None, "file could not be read, check with
os.path.exists()"
edges = cv.Canny(img,100,200)
plt.subplot(121),plt.imshow(img,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])
plt.show()
```

3) Hough transform

Code

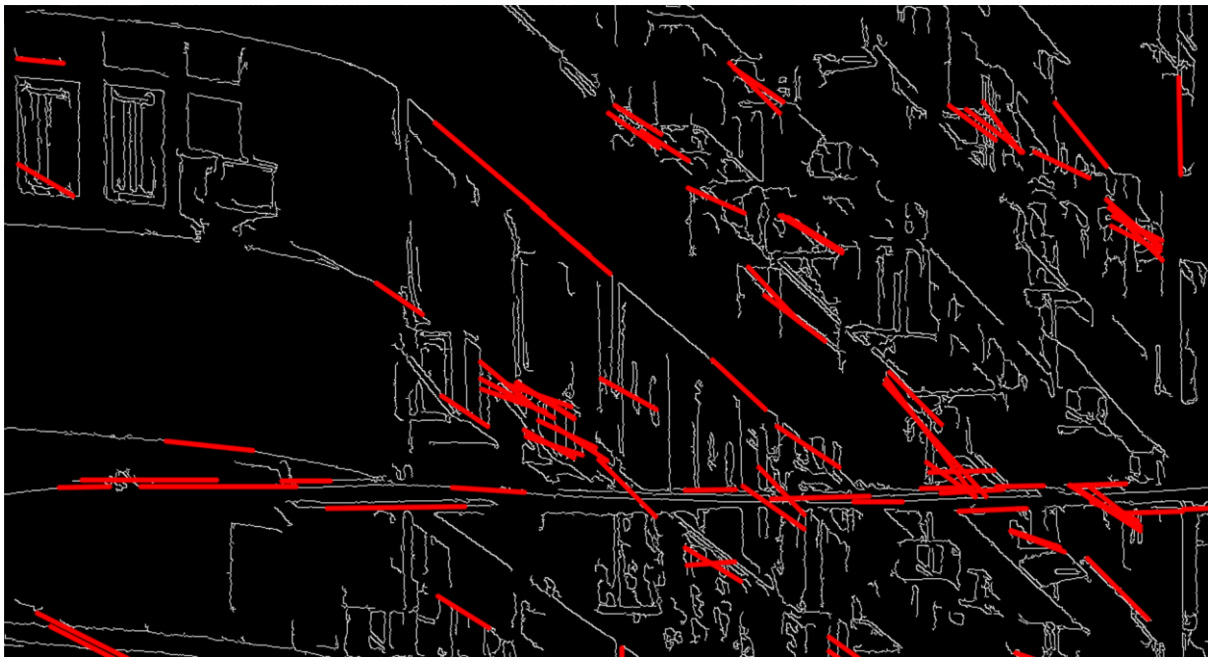
```
"""
@file hough_lines.py
@brief This program demonstrates line finding with the Hough transform
"""
import sys
import math
import cv2 as cv
import numpy as np
def main(argv):

 default_file = 'sudoku.png'
 filename = argv[0] if len(argv) > 0 else default_file
 # Loads an image
 src = cv.imread(cv.samples.findFile(filename), cv.IMREAD_GRAYSCALE)
 # Check if image is loaded fine
 if src is None:
    print ('Error opening image!')
    print ('Usage: hough_lines.py [image_name -- default ' + default_file + ']
\n')
    return -1


 dst = cv.Canny(src, 50, 200, None, 3)

 # Copy edges to the images that will display the results in BGR
 cdst = cv.cvtColor(dst, cv.COLOR_GRAY2BGR)
 cdstP = np.copy(cdst)

 lines = cv.HoughLines(dst, 1, np.pi / 180, 150, None, 0, 0)

 if lines is not None:
    for i in range(0, len(lines)):
        rho = lines[i][0][0]
        theta = lines[i][0][1]
        a = math.cos(theta)
        b = math.sin(theta)
        x0 = a * rho
        y0 = b * rho
        pt1 = (int(x0 + 1000*(-b)), int(y0 + 1000*(a)))
        pt2 = (int(x0 - 1000*(-b)), int(y0 - 1000*(a)))
        cv.line(cdst, pt1, pt2, (0,0,255), 3, cv.LINE_AA)


 linesP = cv.HoughLinesP(dst, 1, np.pi / 180, 50, None, 50, 10)
```

```
 if linesP is not None:
    for i in range(0, len(linesP)):
        l = linesP[i][0]
        cv.line(cdstP, (l[0], l[1]), (l[2], l[3]), (0,0,255), 3, cv.LINE_AA)

 cv.imshow("Source", src)
 cv.imshow("Detected Lines (in red) - Standard Hough Line Transform", cdst)
 cv.imshow("Detected Lines (in red) - Probabilistic Line Transform", cdstP)

 cv.waitKey()
 return 0

if __name__ == "__main__":
 main(sys.argv[1:])
```

Image 1

Converted source

Probabilistic line transform

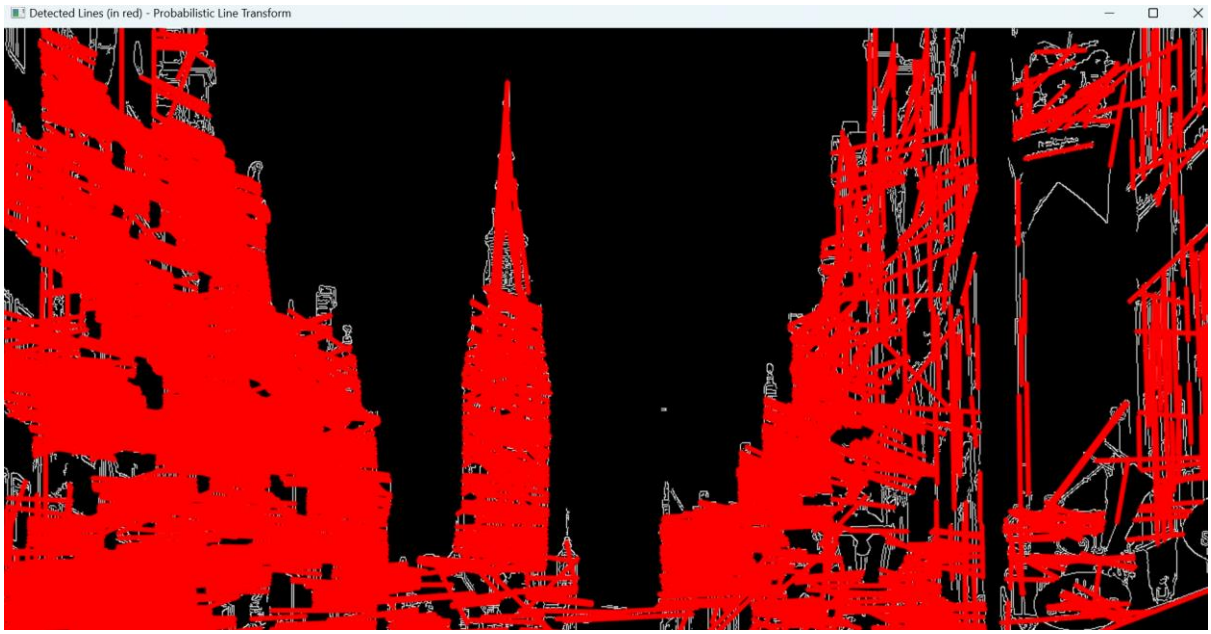


Standard hough transform

Image 2

Converted source

Probabilistic line transform

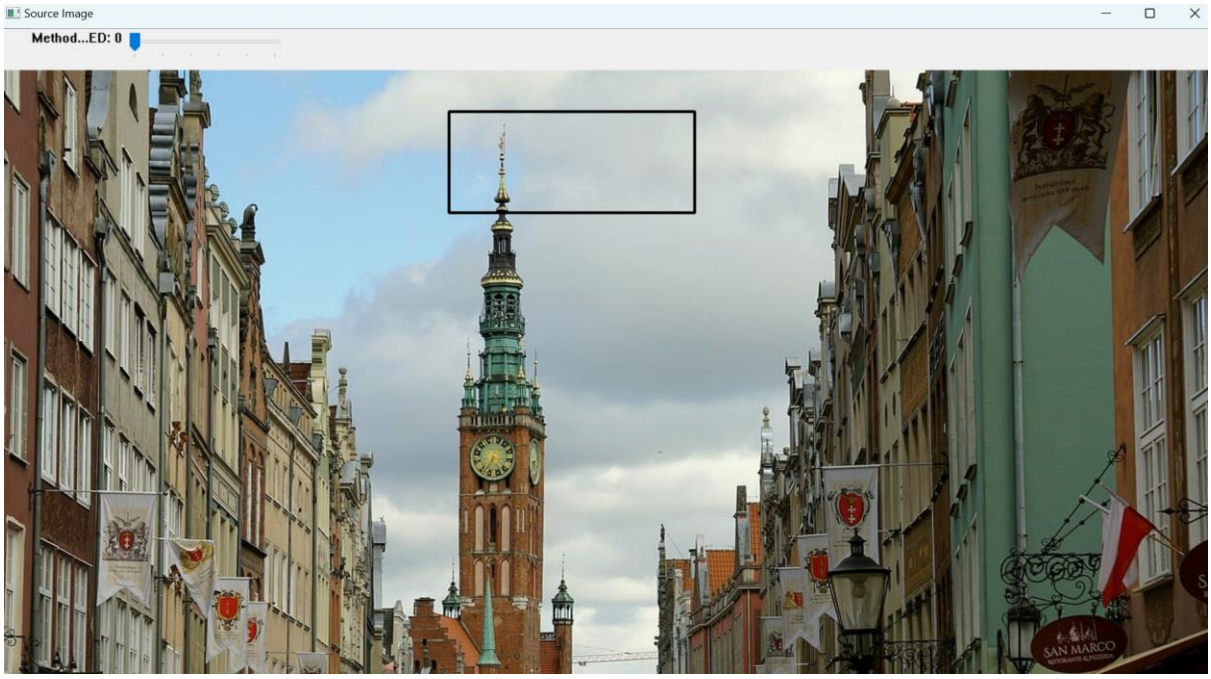


Standard hough transform

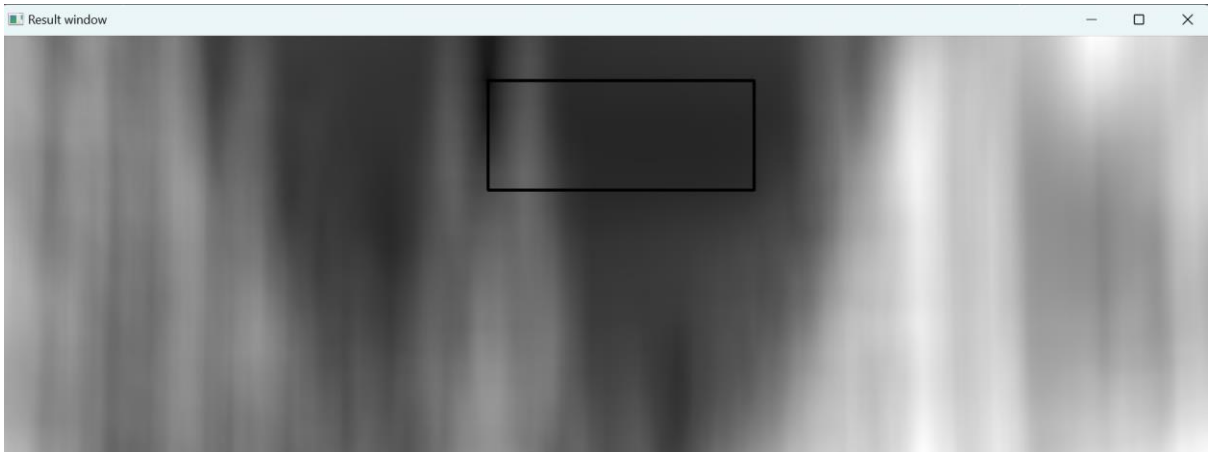
4) Templates

Source image

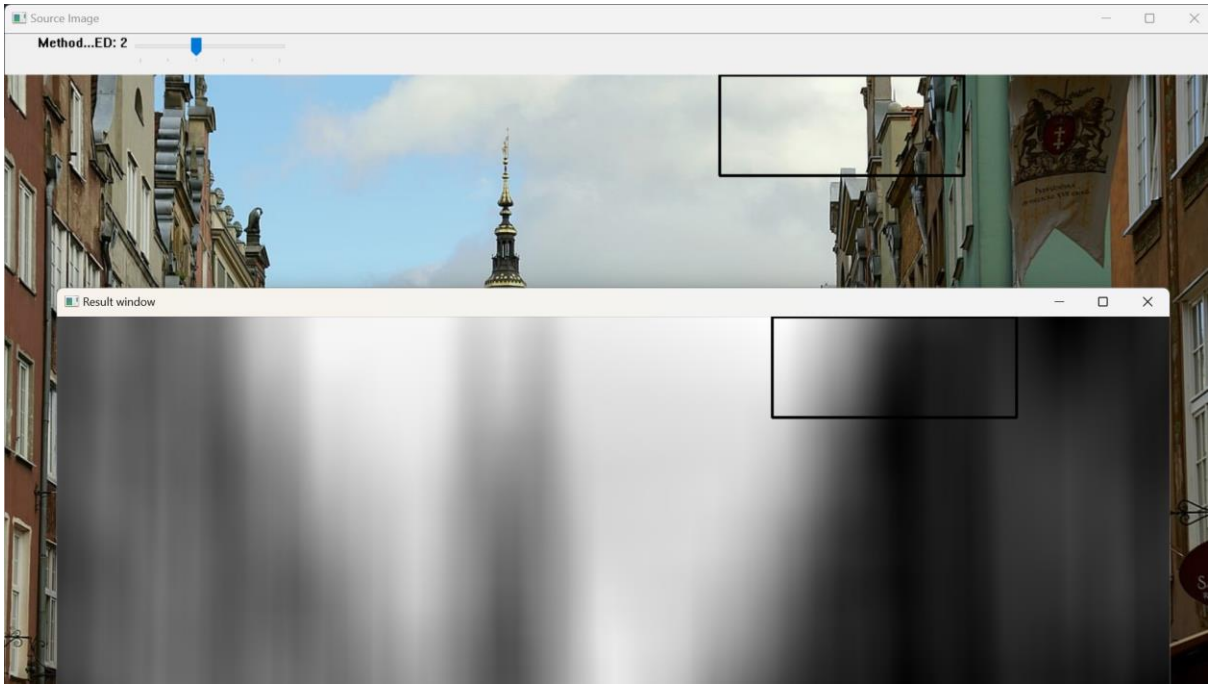
Template



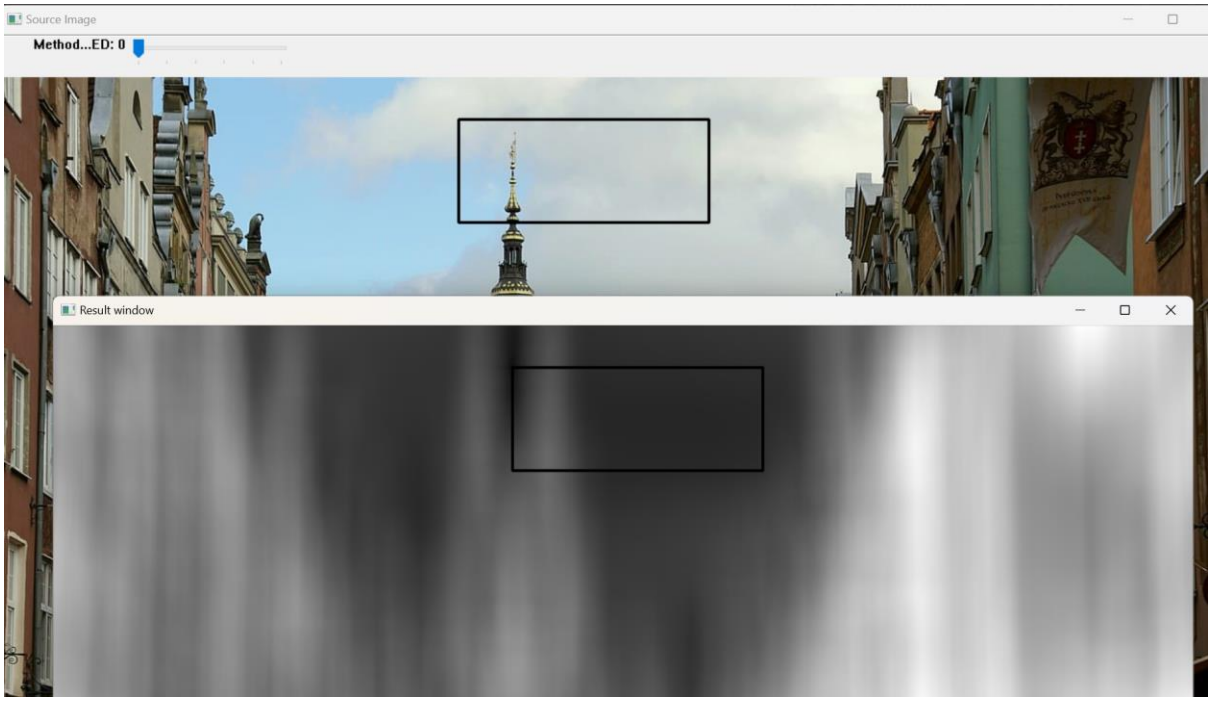
Result



Method: TM_ CCORR



Method : SQDIFF



Code

```
from __future__ import print_function
import sys
import cv2 as cv
use_mask = False
img = None
templ = None
mask = None
image_window = "Source Image"
```

```
result_window = "Result window"
match_method = 0
max_Trackbar = 5
def main(argv):
 if (len(sys.argv) < 3):
    print('Not enough parameters')
    print('Usage:\nmatch_template_demo.py <image_name> <template_name>
[<mask_name>]')
    return -1

 global img
 global templ
 img = cv.imread(sys.argv[1], cv.IMREAD_COLOR)
 templ = cv.imread(sys.argv[2], cv.IMREAD_COLOR)
 if (len(sys.argv) > 3):
    global use_mask
    use_mask = True
    global mask
    mask = cv.imread( sys.argv[3], cv.IMREAD_COLOR )
 if ((img is None) or (templ is None) or (use_mask and (mask is None))):
    print('Can\'t read one of the images')
    return -1


 cv.namedWindow( image_window, cv.WINDOW_AUTOSIZE )
 cv.namedWindow( result_window, cv.WINDOW_AUTOSIZE )


 trackbar_label = 'Method: \n 0: SQDIFF \n 1: SQDIFF NORMED \n 2: TM CCORR \n
3: TM CCORR NORMED \n 4: TM COEFF \n 5: TM COEFF NORMED'
 cv.createTrackbar( trackbar_label, image_window, match_method, max_Trackbar,
MatchingMethod )

 MatchingMethod(match_method)

 cv.waitKey(0)
 return 0

def MatchingMethod(param):
 global match_method
 match_method = param

 img_display = img.copy()

 method_accepts_mask = (cv.TM_SQDIFF == match_method or match_method ==
cv.TM_CCORR_NORMED)
 if (use_mask and method_accepts_mask):
    result = cv.matchTemplate(img, templ, match_method, None, mask)
```

```
 else:
    result = cv.matchTemplate(img, templ, match_method)


    cv.normalize( result, result, 0, 1, cv.NORM_MINMAX, -1 )

    _minVal, _maxVal, minLoc, maxLoc = cv.minMaxLoc(result, None)


 if (match_method == cv.TM_SQDIFF or match_method == cv.TM_SQDIFF_NORMED):
    matchLoc = minLoc
 else:
    matchLoc = maxLoc


 cv.rectangle(img_display, matchLoc, (matchLoc[0] + templ.shape[0],
matchLoc[1] + templ.shape[1]), (0,0,0), 2, 8, 0 )
 cv.rectangle(result, matchLoc, (matchLoc[0] + templ.shape[0], matchLoc[1] +
templ.shape[1]), (0,0,0), 2, 8, 0 )
 cv.imshow(image_window, img_display)
 cv.imshow(result_window, result)

 pass
if __name__ == "__main__":
 main(sys.argv[1:])
```

5) Contour detection

Code

```
import numpy as np
import cv2 as cv
im = cv.imread('/content/city.jpg')
assert im is not None, "file could not be read, check with
os.path.exists()"
imgray = cv.cvtColor(im, cv.COLOR_BGR2GRAY)
ret, thresh = cv.threshold(imgray, 127, 255, 0)
contours, hierarchy = cv.findContours(thresh, cv.RETR_TREE,
cv.CHAIN_APPROX_SIMPLE)
cnt = contours[4]
cv.drawContours(im, [cnt], 0, (0,255,0), 3)
```

Output

```
array([[[ 40,  58,  89],
    [ 41,  60,  87],
    [ 45,  68,  90],
    ...,
    [ 74,  83,  86],
    [ 88,  99, 103],
    [ 51,  62,  66]],
```

```
[[ 42,  60,  91],
 [ 43,  62,  89],
 [ 43,  66,  88],
 ...,
 [ 75,  84,  87],
 [ 89, 100, 104],
 [ 52,  63,  67]],

[[ 43,  61,  92],
 [ 44,  63,  90],
 [ 37,  60,  82],
 ...,
 [ 76,  85,  88],
 [ 89, 100, 104],
 [ 54,  65,  69]],

...,

[[ 11,  34,  56],
 [  9,  32,  54],
 [ 13,  31,  54],
 ...,
 [  0,   0,   1],
 [  0,   0,   1],
 [  0,   0,   1]],

[[ 29,  53,  73],
 [ 25,  49,  69],
 [ 28,  47,  68],
 ...,
 [  0,   0,   0],
 [  0,   0,   0],
 [  1,   1,   1]],

[[ 49,  73,  93],
 [ 48,  72,  92],
 [ 58,  77,  98],
 ...,
 [  0,   0,   0],
 [  1,   1,   1],
 [  2,   2,   2]]], dtype=uint8)
```