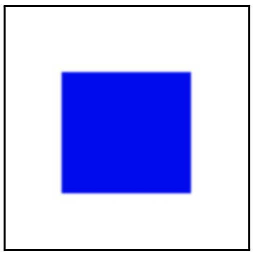
K. N. TOOSI UNIVERSITY OF TECHNOLOGY

Lab Instructions - session 9

Corner Detection

Corner Detection



We want to find the corners in the following image. Harris corner detector gives a ***score*** for each pixel telling how similar the local structure is to a corner. The following code tries to count the number of corners in the image. But there is a problem with the code.

File: **detect_corners.py**

```
import cv2
import numpy as np

I = cv2.imread('square.jpg')
G = cv2.cvtColor(I,cv2.COLOR_BGR2GRAY)

G = np.float32(G)
window_size = 2
soble_kernel_size  = 3 # kernel size for gradients
alpha = 0.04
H = cv2.cornerHarris(G,window_size,soble_kernel_size,alpha)

# normalize C so that the maximum value is 1
H = H / H.max()

# C[i,j] == 255 if H[i,j] > 0.01, and C[i,j] == 0 otherwise
C = np.uint8(H > 0.005) * 255

## connected components
# nc,CC = cv2.connectedComponents(C);

# to count the number of corners we count the number
# of nonzero elements of C (wrong way to count corners!)
n = np.count_nonzero(C)

# Show corners as red pixels in the original image
I[C != 0] = [0,0,255]

cv2.imshow('corners',C)
cv2.waitKey(0) # press any key

font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(I,'There are %d corners!'%n,(20,40), font, 1,(0,0,255),2)
cv2.imshow('corners',I)
cv2.waitKey(0) # press any key

cv2.destroyAllWindows()
```

- Why does the method not work for finding the number of corners? Zoom the image next to the corners to see why.
- Uncomment the line **`nc,CC = cv2.connectedComponents(C)`** to find the connected components of C. Using that fix the number of corners. Notice that background is counted as a separate connected component, thus, the number of connected components will be equal to **`nc-1`**.
- Non-maximum suppression is an alternative to connected components for counting the corners. Think about its advantages and disadvantages.

Corner Detection

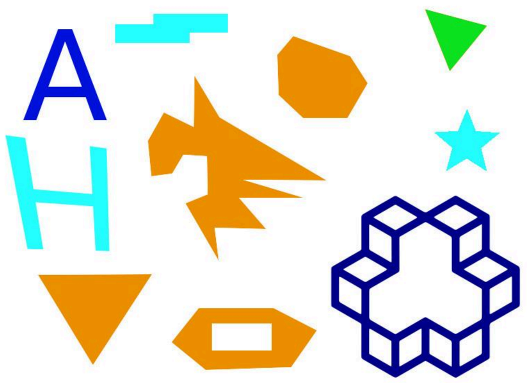
The following code loops through a bunch of images and finds the locations with large Harris scores. It then performs connected components analysis on thresholded Harris scores and computes the centre of each connected component as the corner location. Next, it refines the corner locations using the **`cv2.cornerSubPix`** function.

File: **test_corner.py**

```python
import cv2
import numpy as np
import glob

fnames = glob.glob('*.jpg')
for filename in fnames:
    I = cv2.imread(filename)
    G = cv2.cvtColor(I,cv2.COLOR_BGR2GRAY)
    G = np.float32(G)

    window_size = 3
    soble_kernel_size  = 3 # kernel size for gradients
    alpha = 0.04
    H = cv2.cornerHarris(G,window_size,soble_kernel_size,alpha)
    H = H / H.max()

    C = np.uint8(H > 0.01) * 255
    J = I.copy()
    J[C != 0] = [0,0,255]
    cv2.imshow('corners',J)
    if cv2.waitKey(0) & 0xFF == ord('q'):
        break

    # plot centroids of connected components as corner locations
    nC, CC, stats, centroids = cv2.connectedComponentsWithStats(C)

    J = I.copy()
    for i in range(1,nC):
        cv2.circle(J,(int(centroids[i,0]),int(centroids[i,1])),3,(0,0,255))
    cv2.imshow('corners',J)
    if cv2.waitKey(0) & 0xFF == ord('q'):
        break
    # fine-tune corner locations
    criteria=(cv2.TERM_CRITERIA_EPS+cv2.TERM_CRITERIA_MAX_ITER,100,0.001)
    corners=cv2.cornerSubPix(G,np.float32(centroids),(5,5),(-1,-1),criteria)
    J = I.copy()
    for i in range(1,nC):
        cv2.circle(J,(int(corners[i,0]),int(corners[i,1])), 3, (0,0,255))
    cv2.imshow('corners',J)
    if cv2.waitKey(0) & 0xFF == ord('q'):
        break
```

- There seems to be a lot of corners in highly textured areas. Why?
- In some images some of the corners have not been found. Can you guess why? Zooming in might help. Change the parameter window_size and see the effect on such corners.

Task 1: Find polygons



You need to find all the polygons in the following image, and for each polygon detect the number and location of vertices (corners). Complete the file **task1.py**. Notice that you need to apply **`cv2.connectedComponents`** twice: once for separating each shape, and once for detecting the corners of each shape. Change the parameter window_size (and possibly other parameters) in the Harris corner detector until you get the correct result.

File: **task1.py**

```
import cv2
import numpy as np

I = cv2.imread('polygons.jpg')
G = cv2.cvtColor(I,cv2.COLOR_BGR2GRAY)
ret, T = cv2.threshold(G,220,255,cv2.THRESH_BINARY_INV)
nc1,CC1 = cv2.connectedComponents(T)

for k in range(1,nc1):
    Ck = np.zeros(T.shape, dtype=np.float32)
    Ck[CC1 == k] = 1;
    Ck = cv2.GaussianBlur(Ck,(5,5),0)
    Ck = cv2.cvtColor(Ck,cv2.COLOR_GRAY2BGR)
    # Now, apply corner detection on Ck

    font = cv2.FONT_HERSHEY_SIMPLEX
    cv2.putText(Ck,'There are %d vertices!'%(100),(20,30), font, 1,(0,0,255),1)

    cv2.imshow('corners',Ck)
    cv2.waitKey(0) # press any key
```

Task 2: Refine Corner Detection with Non-Maximum Suppression

In this task, you will refine the results of the Harris corner detector using **Non-Maximum Suppression (NMS)**. The goal is to remove repeated and closely packed corner responses and keep only the most significant ones. You will implement a basic version of **NMS** that checks whether a pixel is the local maximum in a window around it.

To complete this task, work inside the file `task2.py`. You will first apply Harris corner detection to a grayscale image, threshold the corner strength, and then apply your own **NMS** function to extract well-localized corners. Finally, visualize the refined corners by drawing red dots on the image and count the number of final corners found.

File: **task2.py**

```
import cv2
import numpy as np

def non_max_suppression(H, window_size=3):

    suppressed = np.zeros_like(H)
    offset = window_size // 2

    # TODO: Loop over every pixel (except borders) and suppress
non-maximums

    # for i in range(offset, H.shape[0] - offset):
    #     for j in range( ?? ):
    #         window = H[i-offset:i+offset+1, j-offset:j+offset+1]
    #         # TODO: Keep only local maximum
    #         if H[i, j] == ?? :
    #             suppressed[i, j] = H[i, j]

    # return suppressed

# Load image
I = cv2.imread('square.jpg')
G = cv2.cvtColor(I, cv2.COLOR_BGR2GRAY)
G = np.float32(G)
```

```
# Harris parameters
block_size = 2
sobel_ksize = 3
alpha = 0.04

# Compute Harris response
H = cv2.cornerHarris(G, block_size, sobel_ksize, alpha)
H = H / H.max()  # normalize


#Apply Non-Maximum Suppression
# TODO: Call your non_max_suppression function with appropriate window
size
# H_nms = ...

# Threshold the result
# TODO: Threshold to get binary corner map (uint8)
# corner_map = ...


#Color corners (1px)
# I[ ?? ] = [0, 0, 255]


# Find connected components and their centroids
# TODO: Use connectedComponentsWithStats
# num_labels, labels, stats, centroids = ...


# TODO: Loop through all detected components (excluding background)
# for i in range(1, num_labels):
#     x, y = ...
#     cv2.circle(I, (x, y), 3, (0, 255, 0), -1)
#     print(f"Corner {i}: (x={x}, y={y})")

# Show and annotate
font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(I, f'Total Corners: {num_labels - 1}', (20, 40), font, 2, (0,
0, 255), 2)

cv2.imshow('Corners with NMS', I)
cv2.waitKey(0)
cv2.destroyAllWindows()# press any key
```

- Why do we need **non-maximum suppression** in corner detection? What problem does it solve?

- What happens if the **NMS** window size is too small? What about too large? Try with different sizes and explain the effect.

- Compare the result of using `np.count_nonzero()` vs. `connectedComponentsWithStats()` for counting corners.

References

- OpenCV-Python Tutorials - Harris Corner Detection