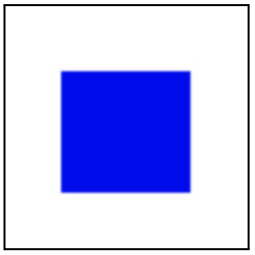
Lab Instructions - session 9

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()
```

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$$

Here, I_x and I_y are image derivatives in x and y directions respectively. (Can be easily found out using cv2.Sobel())

$$R = \det(M) - k(\mathrm{trace}(M))^2$$

this will determine if a window can contain a corner or not.

where

- $\det(M) = \lambda_1 \lambda_2$
- $\mathrm{trace}(M) = \lambda_1 + \lambda_2$
- λ_1 and λ_2 are the eigen values of M

- 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.

Advantages:
Precision
Noise Reduction
Performance

Identifying and labeling connected components can be computationally more intensive than NMS

Disadvantages Parameter Sensitivity: The performance of NMS depends on the choice of parameters such as the suppression window size.

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**

```
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
```

The Harris corner detection algorithm calculates the gradient in both the x and y directions for each pixel. In areas with high texture, there are many changes in intensity, leading to high gradient values. This results in high Harris scores, indicating corners.

Thresholding: If the threshold value for Harris scores (H > 0.01) is too high, some genuine corners may not have scores high enough to be considered corners.

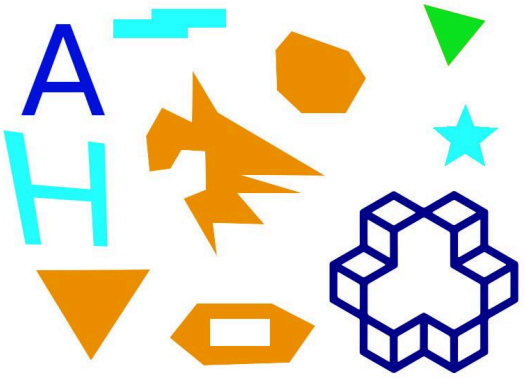
Window Size: The window size used in the Harris corner detector can affect the sensitivity to corners. A too-small window might not capture enough neighborhood information, while a too-large window might average out important variations.

```
# 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.

Image Noise:
Noisy images can cause the algorithm to either miss genuine corners or detect too many false positives.

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: Find polygons using non maximum suppression

Like the above task, you need to find the number and location of the corners. For this task, first, you should separate each polygon from the main image using **`cv2.connectedComponents`**, then calculate each pixel's Harris score and threshold the scores. The difference is in the third step that you should use **`non-maximum suppression`** to find the number and the exact location of corners instead of applying **`cv2.connectedComponents`**. To do this step, you need to compare the Harris score of each pixel with the Harris score of 8 neighboring pixels.

References

- OpenCV-Python Tutorials - Harris Corner Detection