Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Computer Vision
Autumn 2023

Feature Extraction and Matching

Homework-1

October 10, 2023

Alpay Ozkan

aoezkan@student.ethz.ch

By submitting this work, I verify that it is my own. That is both implementation and report are my own work.

Contents

1 Feature Detection: Harris Corner Detection

1.1 Implementation

First I started with calculating image gradients: I_x, I_y which can be calculated easily with convolutions with the corresponding kernels as:

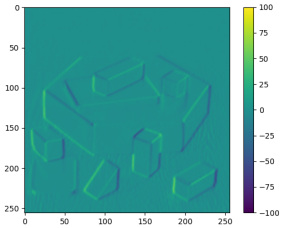
$$I_x = \begin{bmatrix} 0 & 0 & 0 \\ -0.5 & 0 & +0.5 \\ 0 & 0 & 0 \end{bmatrix} \quad I_y = \begin{bmatrix} 0 & -0.5 & 0 \\ 0 & 0 & 0 \\ 0 & +0.5 & 0 \end{bmatrix}$$

$\pm$ direction is relative and not important as we will calculate determinant and trace of the corner score (signal) matrix which is defined as: $M = \begin{bmatrix} I_x^2 & I_x * I_y \\ I_y * I_x & I_y^2 \end{bmatrix}$

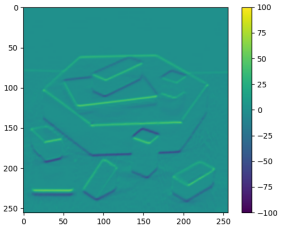
After obtaining I_x I_y, I calculated each element of the matrix by elementwise matrix multiplication. I obtained 4 separate matrices of size (h,w) where h is height and w is the width of the given image. As suggested in the paper and in the lecture, I have applied gaussian convolution with size (3,3) and given sigma value to smoothen the gradient and mitigate the noise. As clear from Derivative Plots in Figure 1, we are able to capture changes in x and y directions. Also, taking the derivative products intensifies vertical or horizontal signal values. Also, the smoothened derivatives are depicted in Figure 2.

The harris corner signal should be calculated for each pixel within the image resolution, hence we will have a matrix of dimension (h,w), same dimensionality as the image. I concatenated derivative matrices as (2,2) and obtained a matrix of dimension (2,2,h,w). In order to calculate the determinant and the trace, I made use of numpy's internal matrix operators: np.linalg.det and .trace but before that I have transposed my matrix to (h,w,2,2) for broadcasting. Trace and determinant calculations for each (2,2) matrix for each pixel yields a matrix of (h,w) and harris score matrix is calculated as $R = det - k * (tr * *2)$ with elementwise operations which of dimension (h,w). The harris corner signal value is depticted in Figure 3(a).

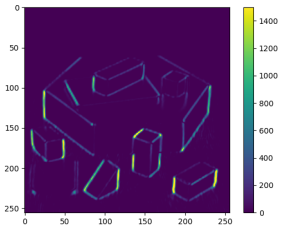
Then I applied non-maximum suppression to filter out low signals if below threshold and also if dominated by a neighboring signal value. I used maximum_filter for dominant signal selection for a window size which I chose as (3,3) and a scalar comparison for thresholding with the given value. Then I selected the non-zero remaining signals after the masking. I had to change the order of the (x,y) after np.where as the coordinates did not match the rest of the pipeline. The harris corner signal value after non-maximum suppression is depticted in Figure 3(b).



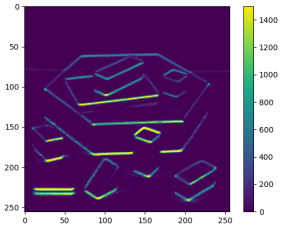
((a)) I_x



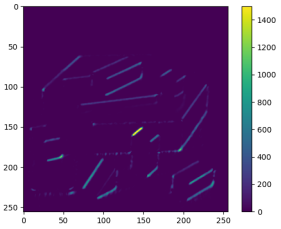
((b)) I_y



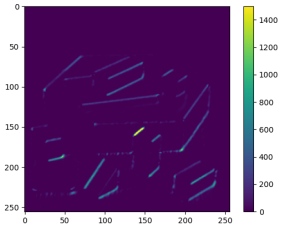
((c)) I_{xx}



((d)) I_{yy}

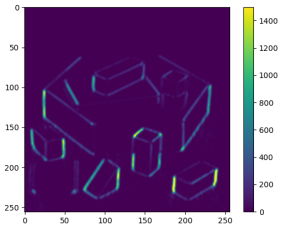


((e)) I_{xy}

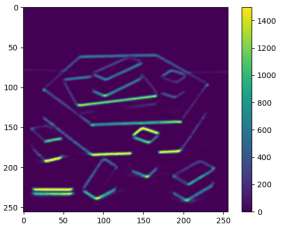


((f)) I_{yx}

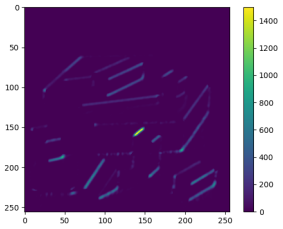
Figure 1: Image Derivatives & Products



((a)) I_{xxs}

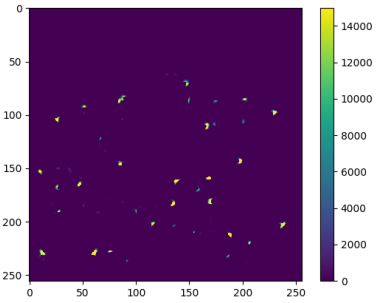


((b)) I_{yys}

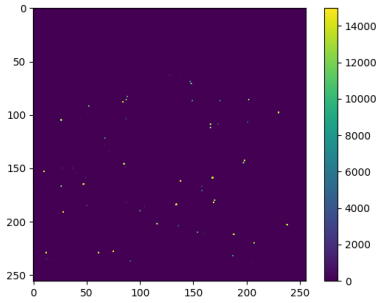


((c)) I_{xys}

Figure 2: Smoothened Derivative Products



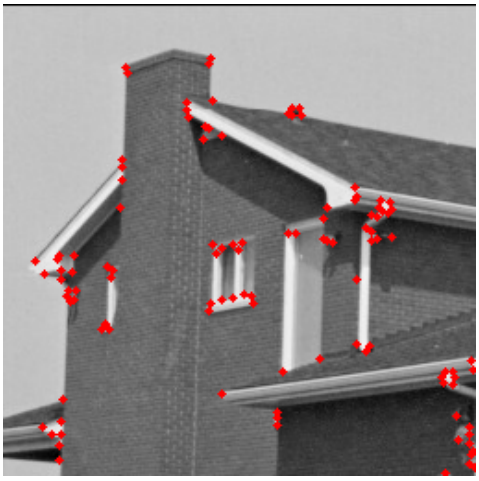
((a)) Before Non-Max Sup



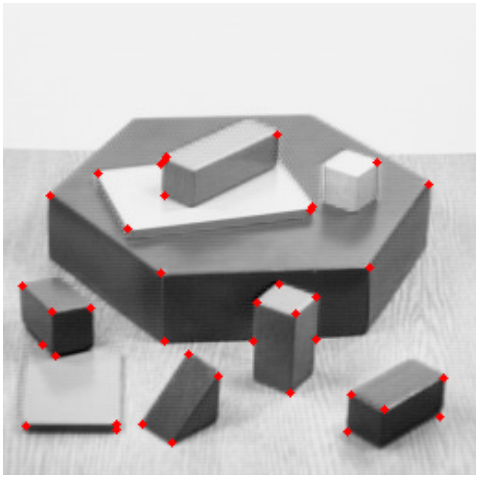
((b)) After Non-Max Sup

Figure 3: Harris Corner Signal Value

1.2 Experiments



((a)) *House*



((b)) *Blocks*

Figure 4: Harris Corner Detection

I have tested my harris corner implementation on two given images. My algorithm has achieved to detect several keypoints as in Figure 4. In Figure 4(a), window corners, pipe corners, rectangular corners like chimney and roof are clearly captured by the algorithm. Interestingly one point near the vertical pipe along the edge does not make much sense as it is not a corner but it is most probably due to the high pixel contrast difference between white pipe and dark bricks. In Figure 4(b), keypoints are even more obvious and they are all localised with the corners of the geometric prisms except for one outlier on the left rectangle there is a keypoint along the edge which I also believe might be due to high contrast difference between dark object and white ground. Also, I observe some missed keypoints on top of the hexagon as there is lower contrast and background is also low resulting in low corner score signals compared to other parts of the image.

1.3 Challenges

During my experiments, I faced some challenges to capture keypoints. The first problem was that I was not able to get any keypoints which happened to be due to too much gaussian convolution. I was applying gaussian convolution both for the image, image gradients, and the squared gradient terms of the matrix which decreased the signal strength due to too much smoothening. Then I applied gaussian convolution to squared gradients term directly and only which kept sufficient signal strength for the detection.

2 Feature Matching

2.1 Implementation

In this part, I implemented SSD feature distance function and three different feature matching algorithms for the keypoints. SSD implementation is straightforward: given two feature descriptors of size F1=(q1, d) and F2=(q2, d), we need to calculate (q1, q2) matrix with each cell having squared error difference for each feature vector between $F1_i$ and $F2_j$. I reshaped my tensors in order to get the right broadcasting in numpy. So I reshaped F1 => (q1, 1, d) and F2 = (q2, d) (same shape). $F1 - F2$ gives a difference matrix of (q1, q2, d). Then we take the square to calculate

squared error for each feature dimension along d. Finally, we sum along feature dimension to get the ssd matrix of dimension (q1, q2) in which each cell denotes the similarity value for each keypoint pair from descriptor-1 (F1) and descriptor-2 (F2).

For the keypoint matching algorithm, we are asked to implement 3 different ways to match the keypoints: one-way, mutual, and ratio. The matching makes use of the **ssd** calculation and the (q1, q2) distance matrix.

One-way nearest neighbor feature matching is based on matching the keypoints of the first image to the nearest keypoints in the second image resulting in (q1,2) matches: q1 many and 2 indices in F1 and F2 correspondingly. In the implementation, I just took argmin accross q2 th channel of (q1,q2) ssd matrix.

Mutual nearest neighbor feature matching is one-way feature matching and another one-way feature matching with the order of images reversed. Then the indices from these 2 one-way feature matches are kept if they are captured from both ways. In the implementation, I applied one-way matching with different order of the descriptors and just selected matches if they exist in both of the ways.

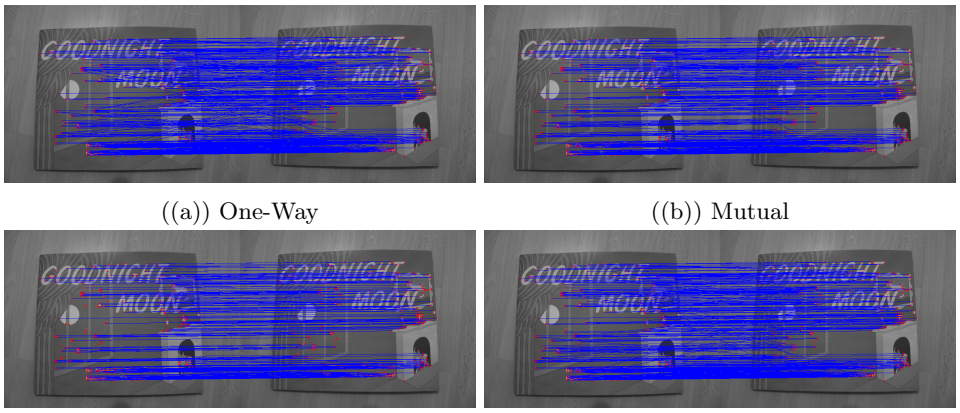
Ratio based nearest neighbor feature matching is the ratio of one-way best match and one-way second best match that is thresholded by a value ie. if this ratio is smaller than the threshold the match is accepted. This puts a more strict condition for the cases where threshold is between (0,1). In the implementation, I applied one-way match to find the best matches and used np.partition to get second best matches. Then took the ratio and thresholded the matches simply.

2.2 Experiments

The experiment is conducted on a pair of book image with different slight orientations. Harris corner keypoints are obtained and then a descriptor function fuses information about the keypoint coordinates and harris signal value to a 81-dim feature vector. That is followed by ssd calculation and matching algorithms. Harris keypoint detections in both images are depicted in Figure 5. The matching results areshown in Figure 6 as expected must be mostly as straight as possible with few exceptions (tilted lines). One-way, mutual, and ratio:0.5 have 428, 332, and 242 feature matches respectively. The matching condition gets more stricter from one-way to ratio matching and as expected we have less matches. Ratio:0.9 is less stricter than ratio:0.5 and have more matches and lines depicted as a result.

((a)) Book IMG-1 ((b)) Book IMG-2

Figure 5: Harris Corner Detection: Books



((a)) One-Way ((b)) Mutual

((c)) Ratio: 0.5 ((d)) Ratio: 0.9

Figure 6: Feature Matching: Books

2.3 Challenges

I faced some difficulties and bugs during my implementation. The most critical mistake I did was during the reshaping operation of my tensors which I needed for broadcasting to calculate ssd matrix.

```
diff = desc1.reshape(dw1, 1, dh1) - desc2.reshape(dw2, dh2, 1) # wrong, dimension order violated
diff = desc1.reshape(dh1, 1, dw1) - desc2 # alternative solution no permutation needed
```

(dw2, dh2, 1) reshaping violates the order in the dimension, it just transforms the dimension without swapping dimensions as we want. So the solution would be with tranpose operation or alternatively without any extra reshaping as illustrated in the code above. With the wrong method I could not match the best pair of keypoints resultin in many cross lines rather than straight.