ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

LAB 02 - Feature extraction and matching

Computer Vision Lab Report

Federica Bruni 23-947-773

October 10, 2023

Department of Computer Science, ETH Zürich

Contents

Chapter 1

Introduction

Feature extraction and matching are fundamental tasks in computer vision that have a wide range of practical applications.

Feature extraction involves identifying and describing distinct points or regions within an image, which are crucial for subsequent analysis.

Feature matching aims to establish correspondences between these distinctive points across different images, enabling tasks such as object recognition, image stitching, and 3D reconstruction. In this lab report, I experimented different methodologies and implementations of feature extraction and matching.

1.1 Task 1: Feature extraction

For the task of feature extraction, the Harris corner detection method was adopted.

Harris corner detection identifies distinctive points within an image, known as corners or key interest points. These corners possess unique characteristics, making them invariant to changes in translation, rotation, and illumination.

This lab report explores the implementation and utilization of the Harris corner detection algorithm, examining its effectiveness in feature extraction from images and its relevance in practical computer vision applications.

1.2 Task 2: Description & Matching

In the context of the feature matching task, an exploration has been conducted involving three distinct methods: one-way nearest neighbors matching, mutual nearest neighbors matching, and the ratio test matching technique.

- The one-way nearest neighbors matching method involves identifying the nearest match for each feature in one image within another image.
- Conversely, mutual nearest neighbors matching imposes the condition that both matched features must consider each other as their closest neighbor, thereby enhancing the overall robustness of the matching process.
- Furthermore, the ratio test matching method is investigated, which assesses the similarity between the best match and the second-best match for a given feature, yielding a more dependable measure of correspondence.

Chapter 2

Methods

In this section, a comprehensive description of the methodology employed in the laboratory investigation is presented.

2.1 Harris Corner Detection

In simple terms, here's how the Harris Corner Detector algorithm works:

1. Identifying Intense Changes: The algorithm checks small image sections (windows) to see which ones show significant changes in intensity when moved in both horizontal (X) and vertical (Y) directions. These changes are like the slopes or gradients in those areas.
2. Scoring the Windows: For each of these responsive windows, the algorithm calculates a score R. This score helps determine how corner-like a window is.
3. Selecting Important Corners: By applying a certain threshold and non-maximum suppression to the score important corners are selected and marked.

2.1.1 step 1 : Identifying Intense Changes

The change function $E(u, v)$ is defined as the summation of squared differences (SSD) across all pixels within the window, where u and v represent the x and y coordinates of each pixel, and I denotes the pixel's intensity value. Features within the image consist of those pixels for which $E(u, v)$ exceeds a certain threshold.

$$\mathrm{E}(u,v) = \sum_{x,y} \underbrace{w(x,y)}_{\text{window function}} \; [\underbrace{\mathrm{I}(x+u, y+v)}_{\text{shifted intensity}} - \underbrace{\mathrm{I}(x,y)}_{\text{intensity}}]^2$$

To spot corners effectively, the objective is to maximize $E(u, v)$. This entails the maximization of the second term. Applying Taylor Expansion to the above equation and using some mathematical steps, the following final equation is reached:

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

with

$$M = \sum w(x,y) \begin{bmatrix} {I_x}^2 & I_x I_y \\ I_x I_y & {I_y}^2 \end{bmatrix}$$

2.1.2 step 2 : Scoring the windows

The second step entailed scoring the windows using the Harris response, which is defined as:

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

2.1.3 step 3 : Selecting Important Corners

Subsequently, in the process of selecting significant corners, two criteria were employed: firstly, the R value needed to surpass a specific threshold, and secondly, a non-maximum suppression method was applied. This method involves examining the computed R values and retaining only the local maxima, effectively reducing redundancy and ensuring that distinct corners are chosen.

2.2 One-way nearest neighbors matching

One-way nearest neighbors matching is an algorithm used to establish associations between distinctive features in two separate images.

This is the basic idea behind the One-way nearest neighbors matching algorithm:

1. Feature Detection and Description: The algorithm begins by identifying distinctive features (corners). Each detected feature is then described through a numerical representation called a feature descriptor. These descriptors encode information about the local region surrounding each feature.
2. Descriptor Comparison: The algorithm computes a similarity score, sum of squared distances, between the descriptor of the feature in the first image and the descriptors of all features in the second image. This quantifies how similar the feature descriptors are.

3. Selecting the Nearest Neighbor: Among all the computed similarity scores, the algorithm identifies the feature in the second image that has the closest descriptor to the feature in the first image. This feature in the second image becomes the "nearest neighbor" of the feature in the first image.
4. Establishing Correspondence: The algorithm establishes a correspondence between the feature in the first image and its nearest neighbor in the second image. This correspondence indicates that these two features likely represent the same point or object in the real world.

2.3 Mutual nearest neighbors matching

Mutual nearest neighbors matching is an algorithm used to establish associations between distinctive features in two separate images.

In simple terms, here's how the Mutual nearest neighbors matching algorithm works:

1. Feature Detection and Description: The algorithm begins by identifying distinctive features (corners). Each detected feature is then described through a numerical representation called a feature descriptor. These descriptors encode information about the local region surrounding each feature.
2. Descriptor Comparison: The algorithm computes a similarity score, sum of squared distances, between the descriptor of the feature in the first image and the descriptors of all features in the second image. This quantifies how similar the feature descriptors are.
3. Selecting the Nearest Neighbors: The algorithm identifies the feature in the second image that has the closest descriptor to the feature in the first image. Simultaneously, it also finds the nearest neighbor of the feature in the first image within the second image.
4. Mutual Verification: To establish a correspondence, both features must recognize each other as the nearest neighbor. In other words, if Feature A in the first image is the nearest neighbor of Feature B in the second image and vice versa, they are considered mutual nearest neighbors and form a valid correspondence.
5. Establishing Correspondence: The algorithm establishes a correspondence between the feature in the first image and its nearest neighbor in the second image. This correspondence indicates that these two features likely represent the same point or object in the real world.

2.4 Ratio test matching

Ratio test matching is an algorithm used to establish associations between distinctive features in two separate images.

This is the key idea behind the Ratio test matching algorithm:

1. Feature Detection and Description: The algorithm begins by identifying distinctive features (corners). Each detected feature is then described through a numerical representation called a feature descriptor. These descriptors encode information about the local region surrounding each feature.
2. Descriptor Comparison: The algorithm computes a similarity score, sum of squared distances, between the descriptor of the feature in the first image and the descriptors of all features in the second image. This quantifies how similar the feature descriptors are.
3. Ranking Matches: For each feature in the first image, the algorithm ranks the potential matches in the second image based on their similarity scores. The potential matches are ordered from best to worst.
4. Ratio Test: The algorithm introduces a threshold. It compares the similarity score of the best match (closest neighbor) to the score of the second-best match (next closest neighbor).
5. Matching Decision: If the similarity score of the best match is significantly better than the score of the second-best match by a margin exceeding the set threshold, the algorithm accepts the best match as the correct correspondence. However, if the scores are very close, indicating uncertainty or ambiguity, the match is rejected, and that feature is not considered a valid correspondence.
6. Establishing Correspondence: The algorithm establishes a correspondence between the feature in the first image and its corresponding in the second image. This correspondence indicates that these two features likely represent the same point or object in the real world.

Chapter 3

Implementation

In this section, the code that implements the methods described is provided. The implementation was executed using Python, with a specific focus on the utilization of the NumPy, SciPy, and OpenCV libraries.

3.1 Harris Corner Detection

The code implementing the Harris Corner Detection algorithm (extract_harris.py) can be broken down into several key steps:

3.1.1 Identifying Intense Changes

1. Image Preprocessing

   This step normalizes pixel values of the input image to the range [0, 1]. Normalization is a common practice in computer vision to ensure consistent data handling.

   ```
   img = img.astype(float) / 255.0
   ```

2. Gradient Computation

   This section computes the image gradients in the x and y directions using convolution. The x and y kernels represent Sobel operators for gradient computation. Ix and Iy are the resulting gradient images.

   ```
   x = 1/2*np.array([1,0,-1]).reshape(1,3)
   y = 1/2*np.array([1,0,-1]).reshape(3,1)
   Ix = signal.convolve2d(img, x, mode='same',
      boundary='symm')
   Iy = signal.convolve2d(img, y, mode='same',
      boundary='symm')
   ```

3. Gradient Smoothing

   In this step, the computed gradient images are smoothed using Gaussian blurring. This smoothing reduces noise in the gradients and prepares them for the subsequent computation of the Harris response.

```
Ixx_blr = cv2.GaussianBlur(Ix ** 2, (0, 0), sigma
    , borderType=cv2.BORDER_REPLICATE)
Iyy_blr = cv2.GaussianBlur(Iy ** 2, (0, 0), sigma
    , borderType=cv2.BORDER_REPLICATE)
Ixy_blr = cv2.GaussianBlur(Ix * Iy, (0, 0), sigma
    , borderType=cv2.BORDER_REPLICATE)
```

3.1.2 Harris Corner Response Calculation

The core of the Harris Corner Detection algorithm is the calculation of the Harris response function C. It involves computing the determinant and trace of the structure tensor of image gradients. The k variable represents a weighting coefficient that affects the Harris response.

```
det_M = Ixx_blr * Iyy_blr - (Ixy_blr**2)
trace_M = Ixx_blr + Iyy_blr
C = det_M - k * (trace_M**2)
```

3.1.3 Corner Detection with Thresholding and Non-Maximum Suppression

This final step involves thresholding the computed Harris response to identify potential corner locations. Additionally, non-maximum suppression is applied to retain only local maxima as corner candidates. The resulting corner coordinates are returned in the corners variable.

```
mask = C > thresh
local_maxima = ndimage.maximum_filter(C, size=(3, 3),
    mode='constant') == C
corners_y, corners_x = np.where(mask & local_maxima)
corners = np.stack((corners_x, corners_y), axis=-1)
return corners, C
```

3.2 Matching

The following code is the implementation of descriptor matching task contained in the match_descriptors.py file.

3.2.1 Data Preparation

1. Dimension Check

   This initial step ensures that the dimensions of the input descriptors desc1 and desc2 are compatible. Matching descriptors require that the feature vectors have the same number of elements.

   ```
   assert desc1.shape[1] == desc2.shape[1]
   ```

2. Distance Computation

   In this section, the squared distances between descriptors are computed using a function manually implemented in the same file. This metric quantifies the dissimilarity between descriptors and is commonly used in descriptor matching.

   ```
   distances = ssd(desc1, desc2)
   ```

3. Initialization

   Here, the number of descriptors in each set (q1 and q2) is determined, and an empty matches array is initialized to store the matched keypoints.

   ```
   q1, q2 = desc1.shape[0], desc2.shape[0]
   matches = None
   ```

3.2.2 One-way Matching

In one-way matching, the goal is to identify the nearest neighbor in image 2 for each keypoint in image 1.

This method involves the following steps:

1. Nearest Neighbor Computation

   For each keypoint in image 1, it computes the index of the nearest neighbor in image 2 based on the calculated distances between descriptors.

   ```
   match_indices = np.argmin(distances, axis=1)
   ```

2. Creating Matched Pairs

   The matched pairs of keypoints are stored in the matches array, where each row represents a match. The first column contains the indices of keypoints in image 1, and the second column contains the indices of their correspondin in image 2.

   ```
   matches = np.column_stack((np.arange(q1),
      match_indices))
   ```

3.2.3 Mutual Matching

Mutual matching seeks to find keypoints that are nearest neighbors of each other in both images, ensuring mutual agreement.

This method involves the following steps:

1. Nearest Neighbor Computation

   In this step, the code computes the nearest neighbor for each keypoint in image 1 (match_idxs) and creates an array indices representing keypoint indices.

   ```
   indices = np.arange(q1)
   match_idxs = np.argmin(distances, axis=1)
   ```

2. Mutual Agreement Check

   To ensure mutual agreement, the code verifies that the keypoints matched in the previous step also agree that they are each other's closest neighbors. The mask variable stores a Boolean array indicating which matches meet this condition.

   ```
   mutual_match_idxs = np.argmin(distances[:,
      match_idxs], axis=0)
   mask = mutual_match_idxs == indices
   ```

3. Creating Matched Pairs

   Finally, the code creates the matches array, which contains the indices of matched keypoints in both images that mutually agree on their nearest neighbor status.

   ```
   matches = np.column_stack((indices[mask],
      match_idxs[mask]))
   ```

3.2.4 Ratio Matching

Ratio matching aims to find valid matches by comparing the ratio of distances between the best and second-best matches.

This method involves several steps:

1. Distance Sorting

   The code sorts the indices of descriptors in image 2 based on their distances to descriptors in image 1.

   ```
   sorted_indices = np.argsort(distances, axis=1)
   ```

2. Distance Ratio Calculation

   For each keypoint in image 1 the code computes the ratio of distances between the best and second-best matches.

```
best_match_indices = sorted_indices[:, 0]
second_best_match_indices = sorted_indices[:, 1]
ratios = distances[np.arange(q1),
   best_match_indices] / distances[np.arange(q1),
    second_best_match_indices]
```

3. Valid Match Identification

   The code identifies valid matches by checking if the computed distance ratios are below a certain threshold.

```
matching_indices = np.where(ratios < ratio_thresh
   )[0]
```

4. Creating Matched Pairs

   Finally, the code creates the matches array, containing the indices of matched keypoints in both images that meet the distance ratio criteria.

```
matches = np.column_stack((np.arange(q1)[
   matching_indices], best_match_indices[
   matching_indices]))
```

Chapter 4

Results

In the upcoming Results section, the outcomes of the experiments will be presented and analyzed.

4.1 Harris Corner Detection

Following the implementation of the Harris detector algorithm, an exploration of various parameter combinations was initiated to identify the most effective configuration.

4.1.1 Default values

[thresh = 1e-5, sigma = 1.0, k = 0.05]

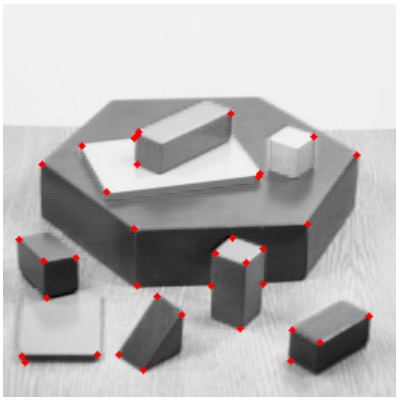

Figure 4.1: Blocks image.

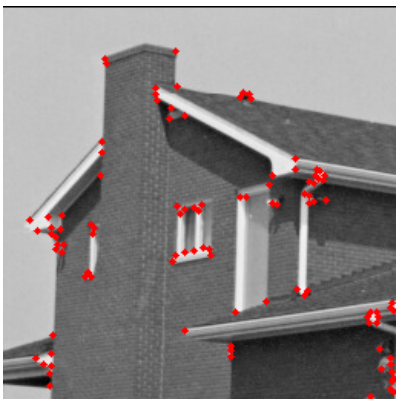

Figure 4.2: House image.

The default values yield satisfactory results, although, especially in the image of the blocks, the outcome can sometimes be somewhat approximate. For

instance, the light cube on the right side of the image is identified by just a single point, making the identification somewhat weak.

4.1.2 Lower thresh value

[thresh = 1e-6, sigma = 1.0, k = 0.05]

The idea was, therefore, to attempt lowering the threshold to capture more points.

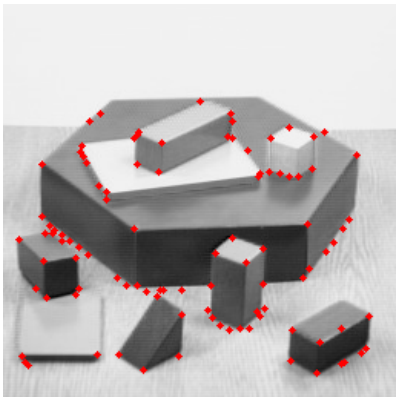

Figure 4.3: Blocks image.

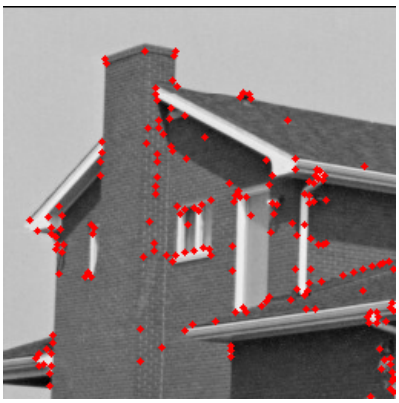

Figure 4.4: House image.

As anticipated, lowering the threshold resulted in the identification of a greater number of points. Taking the example of the cube once more, it is now distinctly characterized. Nevertheless, a notable amount of noise is present, particularly discernible in the image of the house.

4.1.3 Lower thresh value with bigger sigma

[thresh = 1e-6, sigma = 2.0, k = 0.05]

To mitigate the noise, a potential solution involves increasing the sigma value. Consequently, the next experiment was conducted by manteining a low threshold while elevating the sigma value.

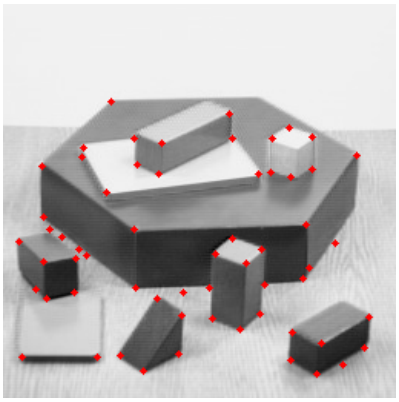

Figure 4.5: Blocks image.

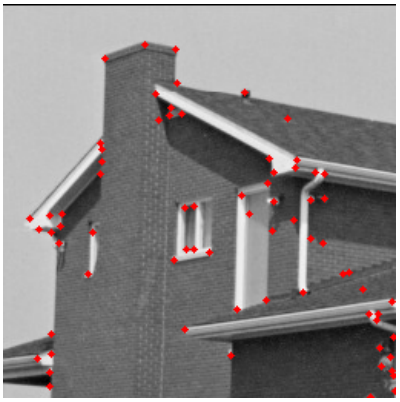

Figure 4.6: House image.

The obtained solution yields excellent results. Currently, the cube is represented by 6 key points, and, in general, all shapes exhibit enhanced characterization. Furthermore, there is an improved outcome for the image of the house.

4.1.4 Lower thresh value, bigger sigma and constant k

[thresh = 1e-6, sigma = 2.0, k = 0.06]

In the end, for further result refinement, I modified the value of k by increasing it, effectively reducing the remaining subtle noise.

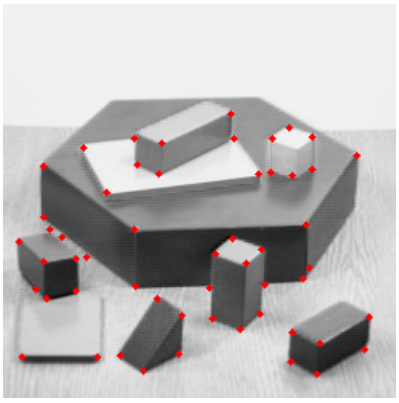

Figure 4.7: Blocks image.

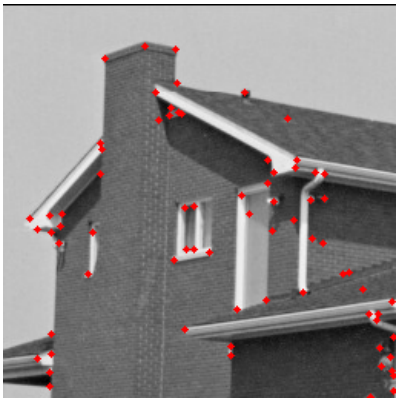

Figure 4.8: House image.

I find this result quite promising. It identifies a slightly lower number of keypoints compared to the previous version but also exhibits reduced noise. It's worth noting that, depending on the specific application, one can fine-tune the value of k to retain either more or fewer keypoints and allow for more or less noise.

4.2 One-way nearest neighbors matching

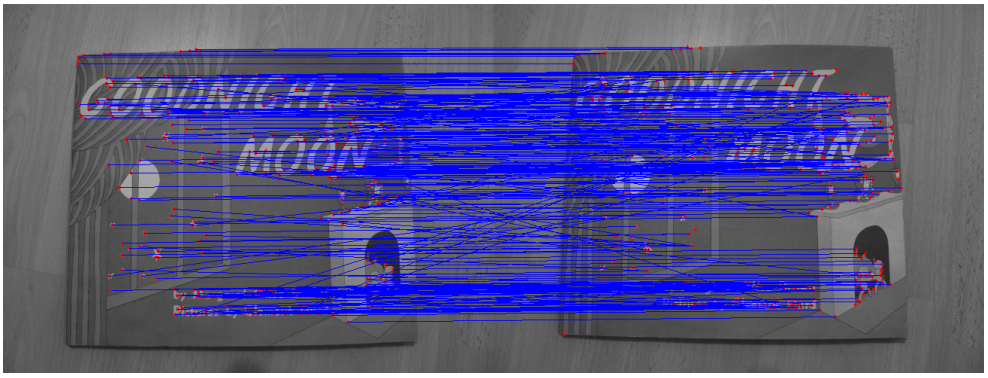

Figure 4.9: One-way nearest neighbors matching.

4.3 Mutual nearest neighbors matching

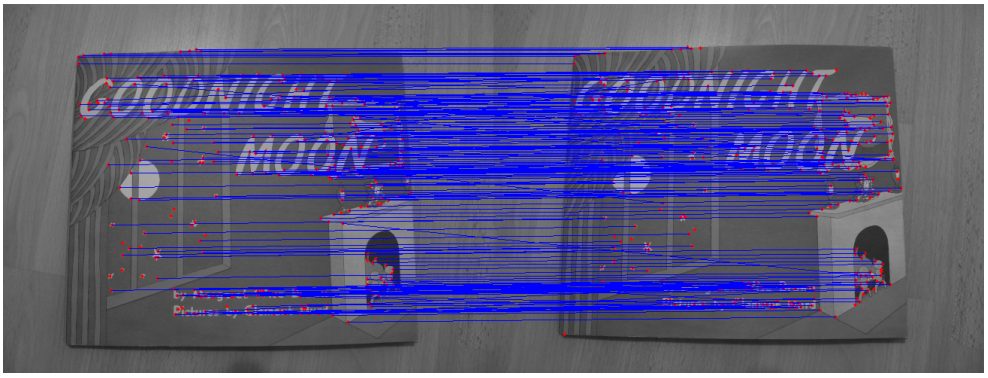

Figure 4.10: Mutual nearest neighbors matching.

4.4 Ratio test matching

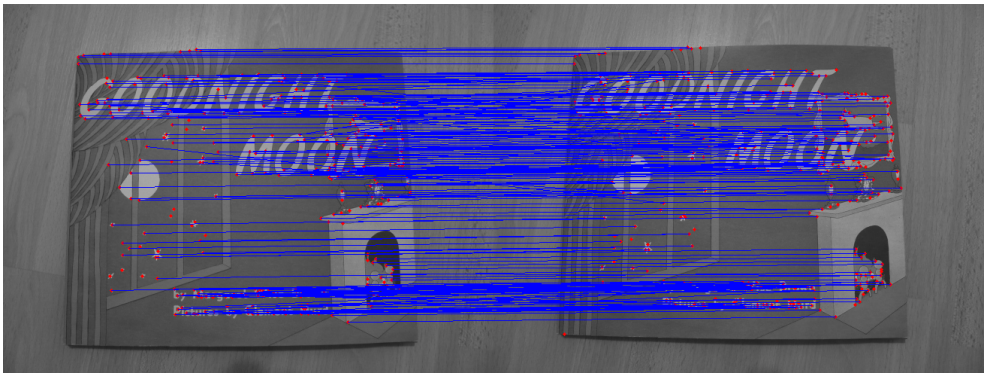

Figure 4.11: Ratio test matching.

Chapter 5

Conclusions

Concluding the Lab 02 on feature extraction and matching, the feature extraction algorithm using the Harris detector was implemented successfully. Parameters were fine-tuned to find the optimal combination, resulting in good image outcomes.

Additionally, various matching methods have been implemented: one-way nearest neighbor matching, mutual nearest neighbor matching and ratio test matching.

The results demonstrate successful matching, with the ratio test method achieving the best performance.