Computer Vision Homework # 2: SIFT Implementation

Jiazheng Li[1]
[1] Beijing Institute of Technology

Abstract

This report addresses the challenge of keypoint detection and feature match with SIFT. In this study, I have employed SIFT to address this issue and compare with SIFT from Opencv. The code and dataset are available at https://github.com/foreverlasting1202/CV-hw-2.

1. Introduction

The Scale-Invariant Feature Transform (SIFT) algorithm was introduced by David G. Lowe in 2004 [3]. It was developed to address the challenges of feature detection and matching in images with variations in scale, rotation, and lighting conditions.

SIFT is widely used in computer vision and image processing for a variety of applications:

1. Object Recognition: SIFT features enable robust object recognition, even when objects undergo changes in scale, rotation, or perspective.
2. Image Stitching: SIFT is used to align and stitch together multiple images to create panoramas and mosaics.
3. 3D Reconstruction: It aids in the creation of 3D models from multiple images by matching keypoints across different views.
4. Augmented Reality: SIFT is used for tracking and registering objects in real-time augmented reality applications.
5. Image Retrieval: SIFT features are employed in content-based image retrieval systems for finding similar images in large databases.
6. Video Analysis: SIFT is used in video tracking and scene recognition, contributing to applications in surveillance and video analysis.

In this article, I have employed SIFT for keypoint detection and feature match, and conducted a comparative analysis with Opencv's SIFT.

2. Related work

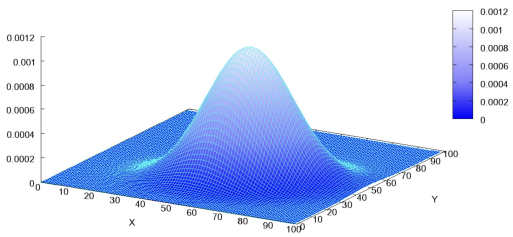
SIFT. The Scale-Invariant Feature Transform (SIFT) algorithm has garnered significant attention and application in image matching and feature point detection since its inception. SIFT algorithm detects stable features in images concerning scale, rotation, and affine transformations, providing an effective means for image recognition and matching [3]. Despite being considered an excellent algorithm for extracting image feature points, SIFT faces challenges due to high computational complexity and memory requirements, limiting its real-time performance [4]. To address these issues, researchers have conducted various studies and optimizations of the SIFT algorithm.

In the context of feature matching, researchers have explored SIFT matching algorithms by using the Euclidean distance as a measure of similarity between keypoints and employing RANSAC to study their theoretical underpinnings [1]. Additionally, studies have shown that the SIFT algorithm maintains high matching performance under various image distortions, such as rotation, scaling, and motion distortions [2]. Compared to other local feature descriptors, SIFT and its variants like PCA-SIFT and GLOH exhibit higher matching accuracy and descriptor distinctiveness under various image transformation conditions. Particularly, in cases of image rotation and scale changes, SIFT-based descriptors outperform other descriptors [5].

Recent research has proposed new methods to enhance the performance of the SIFT algorithm. For instance, using irregular histogram grids instead of the traditional 4x4 histogram grid significantly improves descriptor stability under scale changes. Another enhancement is the SIFT-Rank descriptor, which improves the performance of SIFT descriptors in affine feature matching by sorting each histogram bin [5]. These studies not only delve into the SIFT algorithm and its improvements but also provide valuable insights and references for the further application of SIFT in the field of computer vision.

3. Method

The main idea of SIFT is to focus on finding key points in an image for matching. It assumes that scale and rotation-



0.00000067	0.00002292	**0.00019117**	0.00038771	**0.00019117**	0.00002292	0.00000067
0.00002292	0.00078633	0.00655965	0.01330373	0.00655965	0.00078633	0.00002292
0.00019117	0.00655965	0.05472157	0.11098164	0.05472157	0.00655965	**0.00019117**
0.00038771	0.01330373	0.11098164	**0.22508352**	0.11098164	0.01330373	0.00038771
0.00019117	0.00655965	0.05472157	0.11098164	0.05472157	0.00655965	**0.00019117**
0.00002292	0.00078633	0.00655965	0.01330373	0.00655965	0.00078633	0.00002292
0.00000067	0.00002292	**0.00019117**	0.00038771	**0.00019117**	0.00002292	0.00000067

Figure 1. **Gaussian Function and Kernel**

invariant points may be key points and employs a suitable method to identify and modify these key points. Ultimately, it achieves a description of these key points, which is used for feature matching.

3.1. Gaussian Blur

Gaussian blur is a method for creating scale space. It can be proven that the Gaussian convolution kernel is the only transform kernel for achieving scale invariance, and it is the only linear kernel.

The primary idea behind Gaussian blur is to convolve the image with a selected convolution kernel, which is determined using the Gaussian distribution function.

The N-dimensional Gaussian distribution function is given by:

$$G(x, y, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}^N} e^{-(x^2+y^2)/(2\sigma^2)}$$

Here, σ represents the standard deviation of the Gaussian distribution. A larger σ value results in a more blurred image.

In two-dimensional convolution, the Gaussian distribution is a radially symmetric normal distribution spreading from the center in all directions, and it is convolved with every position in the original image. This kind of blurring process better preserves the effects of edges.

Taking precision and other factors into consideration, in practical applications, when computing the discrete approximation of the Gaussian function, pixels beyond approximately 3σ distance can be considered negligible, and their calculations can be disregarded. Typically, image processing programs only need to compute a $(6\sigma + 1) \times (6\sigma + 1)$ matrix.

3.2. Gaussian Pyramid

Difference of Gaussian function. In order to find extremal points in scale space, we need to use the Laplacian operator to compute derivatives of the Gaussian function. However,

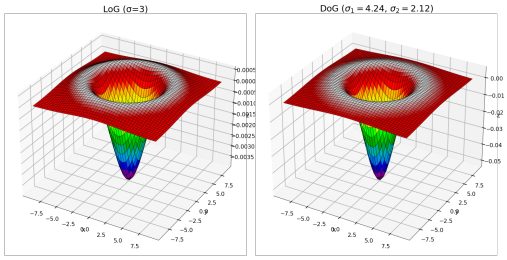


Figure 2. **Laplacian operator and Gaussian difference function**

in practice, calculating the Laplacian operator itself can be quite cumbersome. Yet, through mathematical derivation, we have found that it can be transformed into a simple difference operator.

We have that,

$$\frac{\partial G}{\partial \sigma} = \frac{-2\sigma^2 + x^2 + y^2}{2\pi\sigma^5} e^{-(x^2+y^2)/2\sigma^2}$$

$$\nabla^2 G = \frac{\partial^2 G}{\partial x^2} + \frac{\partial^2 G}{\partial y^2} = \frac{-2\sigma^2 + x^2 + y^2}{2\pi\sigma^6} e^{-(x^2+y^2)/2\sigma^2}$$

$$\implies \frac{\partial G}{\partial \sigma} = \sigma\nabla^2.$$

Otherwise,

$$\frac{\partial G}{\partial \sigma} = \lim_{\Delta\sigma \to 0} \frac{G(x, y, \sigma + \Delta\sigma) - G(x, y, \sigma)}{(\sigma + \Delta) - \sigma}$$
$$\approx \frac{G(x, y, k\sigma) - G(x, y, \sigma)}{k\sigma - \sigma},$$

which implies that

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k - 1)\sigma^2\nabla^2 G.$$

So, we can use the Gaussian difference function to replace the scale-normalized Laplacian of Gaussian function.

It is worth noting that differentiation is performed to find extremal points, and, in practice, the term $k-1$ does not actually affect the positions of these extremal points. Therefore, in concrete implementations, we often ignore the $k-1$ term.

Gaussian Pyramid. Based on the definition of the Gaussian difference function, we essentially only need to define the Gaussian pyramid. The Gaussian difference pyramid can be obtained by taking the difference between adjacent layers of the Gaussian pyramid.

The definition of the Gaussian pyramid here differs slightly. In order to better represent the scale space, this Gaussian pyramid employs a hierarchical structure, with downsampling between levels of the pyramid. Within each level, different values of σ are used for blurring to create distinct layers.

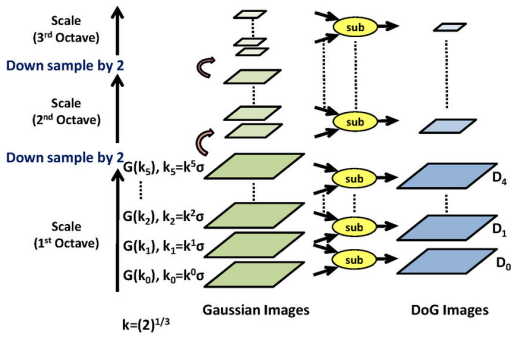


Figure 3. **Gaussian difference pyramid**

Downsampling, in general, involves removing every other row and column from an image to create a new image. To better accommodate downsampling, we will compute scales as follows:

Let there be a total of O levels, each with S layers, where (o, s) represents the scale standard for a particular layer, and σ_0 is the initial blur scale. We have $\sigma = \sigma_0 2^{o+\frac{s}{S}}$, and setting $k = 2^{1/S}$, we get $\sigma = \sigma_0 2^o k^s$.

So, for each layer in the Gaussian pyramid, $L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$, where $I(x, y)$ is the initial image corresponding to the respective level. On the other hand, each layer in the Gaussian difference pyramid is given by $D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma)$.

The initial blur scale is typically experimentally chosen as 1.6 or $\sqrt{1.6^2 - 0.5^2}$. The total number of levels, O, is typically chosen as $\log_2(\min(M, N)) - t$, where M and N represent the size of the image, and t is the logarithm of the smallest dimension of the top-layer image. The number of layers per level, S, is generally set to 3.

3.3. Local Extrema Detection

Local extremal points refer to the points with the most extreme values within their neighborhoods. Generally, in the Gaussian difference pyramid, the neighborhood refers to the eight points around the point in the same layer and the eighteen points from the adjacent two layers. This is why each level in the Gaussian difference pyramid requires $S + 2$ layers. We only detect points from the second layer to the second-to-last layer in each level, ensuring that each level has S layers.

In practical implementation, to reduce the influence of noise, a threshold T is set to be $T = 0.02/S$, and only absolute values greater than T are considered.

However, the extremal points obtained in this way are often undesirable. For instance, they might include points in a white background and are sensitive to edges. To avoid these low-quality extremal points, some corrections are needed.

3.4. Accurate Keypoint Localization

Sub-pixel Interpolation. It's important to note that all the operations we perform are in discrete space, so the extremal points we find are actually discrete-space extremal points, not true extrema in the continuous domain. To obtain smoother extremal points, we opt to use **Taylor expansion** for interpolation, a technique known as sub-pixel interpolation.

We have that,

$$D(\mathbf{x}) = D + \frac{\partial D^T}{\partial \mathbf{x}}\mathbf{x} + \frac{1}{2}\mathbf{x}^T \frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x}$$

Here, $\mathbf{x} = (x, y, s)^T$.

Take the derivative of it, we can obtain $\hat{\mathbf{x}} = -\frac{\partial^2 D}{\partial \mathbf{x}^2}^{-1} \frac{\partial D}{\partial \mathbf{x}}$ and real result is $D(\hat{\mathbf{x}}) = D + \frac{1}{2}\frac{\partial D}{\partial \mathbf{x}}^T \hat{\mathbf{x}}$.

In the actual implementation, to ensure the appropriateness of pixel values, a uniform scaling of 255 times is applied. Furthermore, since x, y and s are integers, the (x, y, s) components of $\hat{\mathbf{x}}$ undergo rounding. Noise reduction is still carried out, which means points with absolute values less than $\frac{0.04}{S}$ will be removed. Derivatives are computed using the finite difference method.

Eliminating Edge Responses. The Hessian matrix is a square matrix composed of the second partial derivatives of a multivariate function, describing the local curvature of the function. Before understanding how the Hessian matrix eliminates edge effects, we need to know what edge effects mean.

Characteristics at the edges include the fact that when the gradient in one direction is larger, the gradient in the other direction becomes smaller. This aligns with the ratio of the two eigenvalues of the Hessian matrix. If we denote the Hessian matrix as H and the eigenvalues as α and β, representing the gradients in the x and y directions, the ratio of eigenvalues is given by $r = \frac{\alpha}{\beta}$. We have $\mathrm{Tr}(H) = \alpha + \beta$ and $\det(H) = \alpha\beta$, which leads to the equation $\frac{\mathrm{Tr}(H)^2}{\det(H)} = \frac{(r+1)^2}{r}$.

In the specific implementation, a value of $r = 10$ is set, and all points for which $\frac{\mathrm{Tr}(H)^2}{\det(H)} > \frac{(r+1)^2}{r}$ are removed.

3.5. Orientation Assignment

To maintain rotational invariance when describing, it is necessary to assign a reference direction to each keypoint. This is a requirement for local feature-based image processing. For keypoints detected in the Gaussian difference pyramid, we collect the gradient and orientation distribution within a window of size $3 \times 1.5 \times \sigma$ in the corresponding Gaussian pyramid image. The magnitude $m(x, y)$ and ori-

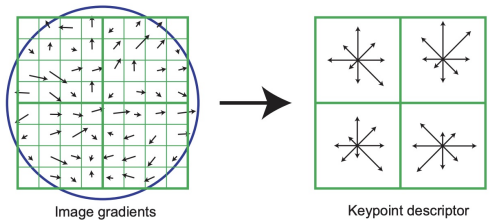


Figure 4. **Keypoint Descriptor**

entation of the gradient $\theta(x, y)$ are as follows:

$$
\begin{aligned}
\mathrm{d}L_x &= L(x, y+1) - L(x, y-1) \\
\mathrm{d}L_y &= (L(x+1, y) - L(x-1, y) \\
p(x, y) &= \exp(-(x^2 + y^2)/(2 \times (1.5\sigma)^2)) \\
m(x, y) &= \sqrt{(\mathrm{d}L_x)^2 + (\mathrm{d}L_y)^2} \times p(\Delta x, \Delta y) \\
\theta(x, y) &= \tan^{-1}(\frac{\mathrm{d}L_y}{\mathrm{d}L_x})
\end{aligned}
$$

After computing the gradients in the keypoints and their neighborhoods, we tally the sum of gradient directions and the corresponding gradient magnitudes. The peak direction with the highest sum is chosen as the primary orientation. To enhance the robustness of matching, we retain any direction with a peak greater than 80% of the primary orientation peak as a secondary orientation for that keypoint. In practice, this is achieved by duplicating the keypoint multiple times.

In the specific programming implementation, we divide the 360 degrees into 36 segments, each spanning 10 degrees, to calculate the sum of magnitudes.

3.6. Keypoint Descriptor

Once we have the information regarding magnitudes and orientations, we need to create appropriate descriptors for the keypoints. These descriptors should exhibit rotation and illumination invariance.

Similarly, we gather information from the neighborhood. At this stage, the orientation plays a useful role as it allows us to rotate the entire neighborhood to a common reference direction, often chosen as 0 degrees.

The size of the neighborhood itself is typically $3\sigma(d + 1) \times 3\sigma(d + 1)$, and after rotation, it becomes the diagonal of this neighborhood, which is

$$
\frac{3\sigma(d+1)\sqrt{2}}{2} \times \frac{3\sigma(d+1)\sqrt{2}}{2}.
$$

.

Next, we partition the neighborhood into 4×4 small squares, with each small square recording the sum of gradient magnitudes for various keypoint orientations.

Typically, we evenly divide the 360 degrees into 8 angle ranges. The center of each square represents that square, but not all neighboring pixels falling within the square are exactly at its center. Therefore, we need to further process the gradient magnitudes based on their contribution to the center's position. This involves weighting the gradient magnitudes within the square according to the distance of each pixel relative to the center of the square. Generally, we use a simple coordinate transformation as follows:

$$
\mathbf{x} = \frac{1}{3\sigma}\mathbf{x} + \frac{d}{2} - 0.5
$$

Here, $\mathbf{x} = \binom{x}{y}$.

After this, we apply trilinear interpolation to distribute the contribution of each pixel to the eight corner points of the square. The contributions are typically weighted accordingly to represent the corresponding volumes.

Now, we have obtained the descriptor for the feature points. To make it less sensitive to lighting variations, we perform vector normalization on the descriptor set. This involves normalizing once and then re-normalizing all values by taking the minimum between each value and 0.2 before re-normalization.

3.7. Feature Match

Feature matching naturally involves considering the nearest points. However, in reality, the keypoints in two images may not necessarily correspond one-to-one, which means not every point will have a matching point.

Considering this, a preliminary check is typically performed. If the ratio of the distance to the nearest point compared to the distance to the second nearest point is less than 0.8, it is considered a match; otherwise, it is not.

4. Implementation details

I have implemented the SIFT algorithm using NumPy and conducted experimental comparisons with OpenCV's SIFT algorithm and perform codes on macOS Ventura 13.3.1, M1 pro, 16GB with PyCharm.

5. Experiments

5.1. Datasets, metrics

Datasets. We use an image "thumb" along with its 180-degree flip as the dataset.

Metrics. For simplicity, we used visual measurements.

5.2. Experiment Result

Keypoint Detection

By comparing Figure 6 and Figure 7, it can be observed that the self-implemented SIFT tends to detect keypoints in

Figure 5. **Dataset**

Figure 6. **Keypoint Detection by myself**

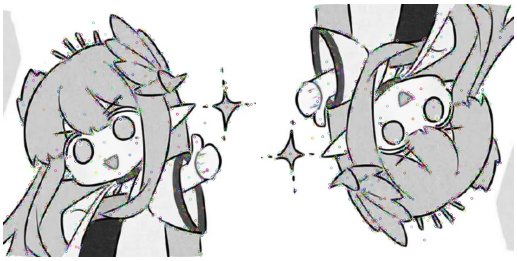


Figure 7. **Keypoint Detection by Opencv**

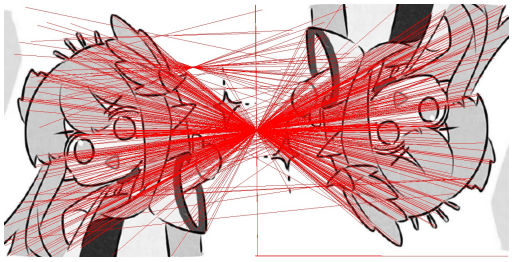


Figure 8. **Feature Match by myself**

background and edge regions, suggesting that the threshold may not be set high enough for keypoint detection in such areas.

Feature Match

By comparing Figure 8 and Figure 9, it can be observed that due to previous issues with keypoint detection, there have been some problems with feature matching. However, for keypoints that are detected correctly, the self-implemented SIFT's performance is now quite close to that of OpenCV. Overall, the results are reasonably good.

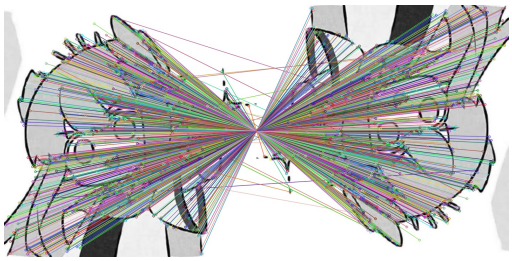


Figure 9. **Feature Match by Opencv**

6. Conclusion

In this paper, we have introduced the fundamental principles of the SIFT algorithm and provided a manual implementation of the algorithm, comparing it to OpenCV's built-in implementation.

Limitations. However, it has been observed that OpenCV's performance is slightly better. Upon reading the original paper [3], it is suspected that this difference may be due to the manually implemented thresholds not being entirely suitable and the absence of gradient direction fitting with a parabola.

References

[1] Feng Guo, Jie Yang, Yilei Chen, and Bao Yao. Research on image detection and matching based on sift features. In *2018 3rd International Conference on Control and Robotics Engineering (ICCRE)*, pages 130–134, 2018. 1

[2] Ebrahim Karami, Mohamed Shehata, and Andrew Smith. Image identification using sift algorithm: Performance analysis against different image deformations, 2018. 1

[3] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 2004. 1, 5

[4] TH Tsai, RZ Wang, and NC Tung. Hardware architecture design for real-time sift extraction with reduced memory usage. *Multimed Tools Appl*, 2023. 1

[5] Wikipedia contributors. Scale-invariant feature transform — Wikipedia, the free encyclopedia, 2023. [Online; accessed 25-October-2023]. 1