

ICE2607 Lab 3: SIFT

SJTU-SEIEEE cny123222

December 12, 2024

1 Experiment Overview (实验概览)

The **Scale-Invariant Feature Transform (SIFT)** algorithm, pioneered by David Lowe, is renowned for its robustness to various image transformations. SIFT features **excel in matching tasks** due to their **invariance to translation, rotation, scaling, and illumination changes**.

The main steps of SIFT algorithm includes scale space extrema detection, keypoint localization, orientation assignment and descriptor generation. Given the complexity of keypoint extraction, our experiment simplifies by adopting Harris corner detection for keypoint identification, and **focuses on the crucial step of descriptor generation**.

In the experiment, a target image is selected for matching against five dataset images, with only one depicting the same object. Through our SIFT algorithm, the target successfully aligns with the correct image. Additionally, several figures, histograms and heat maps are displayed along the process, giving us some intuitive understanding of the algorithm's functionality.

2 Solution Approach (解决思路)

In the initial phase, OpenCV's corner detection function is utilized to extract keypoint locations, simplifying the conventional method. Then for each identified keypoint, our task focuses on computing the primary orientation and generating descriptors.

2.1 Step 1: Orientation Computation

For a keypoint $L(x, y)$, the gradients within its $m \times m$ neighborhood need to be computed. Let $m(x, y)$ represent the magnitudes and $\theta(x, y)$ denote the angles of the gradients at each pixel. The gradient direction is divided into 36 bins covering 360 degrees. Each pixel in the neighborhood contributes a weighted vote to its corresponding bin based on $m(x, y)$. The orientation with the highest accumulated weight is selected as the keypoint's main orientation. The pseudocode below outlines this process:

```
1 def main_orient(img, x, y, m): # x and y are the coordinates of the keypoint
2     magnitudes, angles = calc_gradient(img)
3     neighbor_magnitudes = magnitudes[x-m:x+m, y-m:y+m].flatten()
4     neighbor_angles = angles[x-m:x+m, y-m:y+m].flatten() # angles range from -pi and pi
```

```

5     hist, bin_edges = np.histogram(neighbor_angles, bins=36, range=(-np.pi, np.pi),
6         weights=neighbor_magnitudes)
7     main_dir = (bin_edges[np.argmax(hist)] + bin_edges[np.argmax(hist) + 1]) / 2
8     return main_dir

```

Note that the $m \times m$ neighbourhood should correspond to the 3σ neighbourhood of the Gaussian pyramid as per the original algorithm. However, since we did not employ a pyramid for keypoint extraction, the selection of σ will be discussed later in the subsequent sections.

2.2 Step 2: Descriptor Generation

To generate SIFT descriptors, calculations are performed within a 16×16 region centered around the keypoint in the object coordinate system.

Firstly, all gradient directions computed in the image coordinate system are transformed to the object coordinate system. If the resulting coordinates are not integers, the nearest-neighbor interpolation method is utilized, assigning the value of the closest point in the image coordinate system.

Then the 16×16 region is subdivided into 4×4 blocks, each containing 4×4 pixels. Within each block, the 360-degree range is divided into 8 bins following the method used to determine the main orientation. A histogram of gradient directions is computed within each block, resulting in an 8-dimensional histogram vector for each block. Consequently, each keypoint yields a 128-dimensional SIFT descriptor ($4 \times 4 \times 8$).

Finally, the 128-dimensional SIFT descriptor f_0 is normalized to obtain the ultimate result.

```

1 def generate_descriptor(magnitudes, angles, x, y, main_dir, n): # n represents the size
2     # of the small block (4*4 by default)
3     dst = [] # Array to store the descriptor
4     # Iterate through the 16 4*4 regions
5     for offset_x in range(-2, 2):
6         for offset_y in range(-2, 2):
7             near_magnitudes = []
8             near_angles = []
9             # Within each fixed 4*4 region
10            for i in range(offset_x * n, offset_x * n + n):
11                for j in range(offset_y * n, offset_y * n + n):
12                    # Coordinate transformation
13                    ori_x = x + int(round(i * np.cos(main_dir) - j * np.sin(main_dir)))
14                    ori_y = y + int(round(i * np.sin(main_dir) + j * np.cos(main_dir)))
15                    near_magnitudes.append(magnitudes[ori_x, ori_y])
16                    near_angles.append(angles[ori_x, ori_y]) # Boundary check needed
17                    # Angle transformation
18                    near_angles = np.array(near_angles) - main_dir
19                    near_angles[near_angles < -np.pi] += 2 * np.pi
20                    near_angles[near_angles > np.pi] -= 2 * np.pi
21                    # Update the descriptor
22                    hist, _ = np.histogram(near_angles, bins=8, range=(-np.pi, np.pi)),

```

```

        weights=near_magnitudes)
22     dst.extend(hist)
23     # Descriptor normalization
24     dst = np.array(dst)
25     dst = dst / np.linalg.norm(dst)
26     return dst

```

In practical scenarios, observing a fixed number of pixels can significantly impact the quality of results due to various image sizes and may lead to suboptimal matching outcomes. Hence, we introduced a flexible parameter, n , to regulate the scope we intend to observe. The selection of n will be deliberated on subsequently.

3 Experimental Results (实验结果)

In the experiment, our descriptor generation algorithm is employed to search for objects similar to the one depicted in the target image within all images in the dataset folder, with the results of good matches being graphically represented. Additionally, a comparative analysis is conducted with the SIFT function from OpenCV to assess the performance and efficacy of our SIFT implementation in object detection and matching tasks.

3.1 Experimental Environment

The experiments were conducted on macOS Sonoma 14.6.1 with Python 3.13.0. Key libraries included OpenCV-Python 4.10.0.84, Numpy 2.1.3 and Matplotlib 3.9.2.

3.2 Matching Results

Figure 1 illustrates the matching outcomes derived from our custom SIFT implementation, while Figure 2 showcases the results obtained using OpenCV’s SIFT algorithm.

The resulting images clearly demonstrate that both our SIFT implementation and OpenCV’s SIFT algorithm effectively match the target image with the corresponding image in the dataset. Numerous connecting lines are plotted to illustrate the similarity between keypoints. In contrast, incorrect matches display either no lines or very few connections.

However, OpenCV’s SIFT algorithm surpasses our implementation in terms of both accuracy and computational efficiency. This superiority can likely be attributed to the more sophisticated keypoint selection and descriptor generation methods employed by OpenCV, compared to our more simplistic approaches.

4 Analysis and Discussion (分析与思考)

4.1 SIFT Visualization

To gain an intuitive understanding of the processes happened behind SIFT algorithm, several figures are plotted during the experiment.

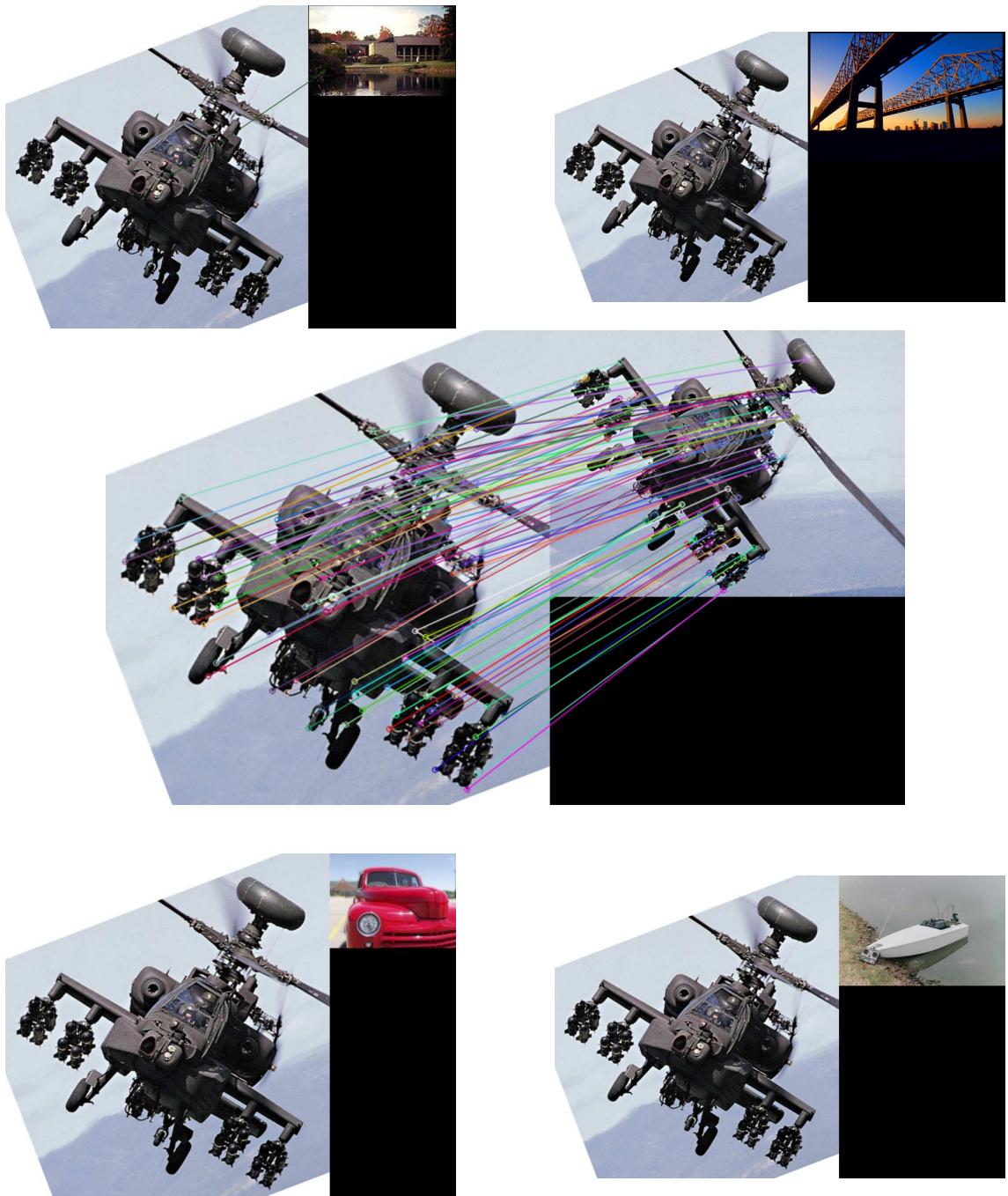


Figure 1 Resulting Images of our SIFT Implementation

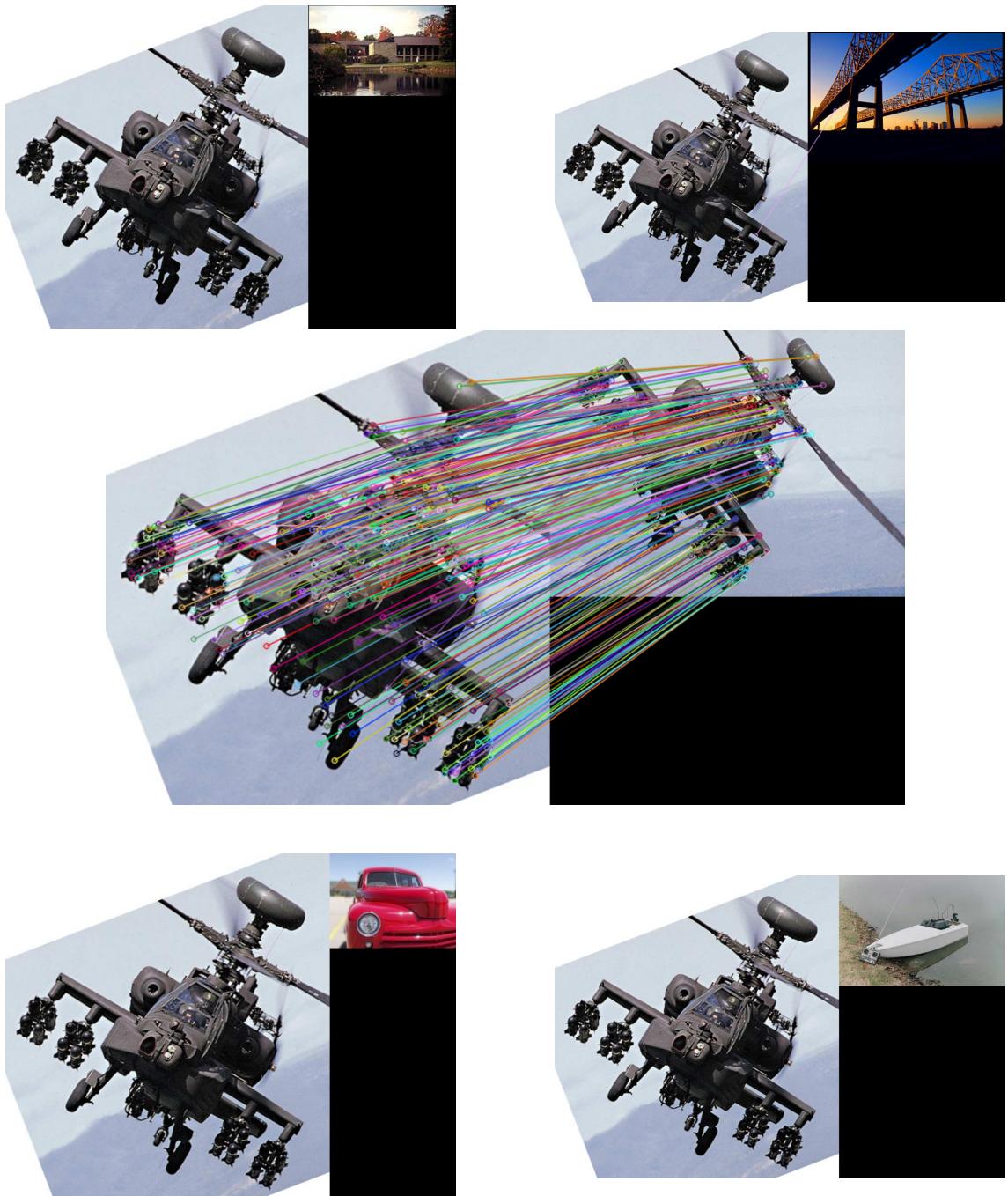


Figure 2 Resulting Images of OpenCV's SIFT Algorithm

4.1.1 Visualization of Main Orientations

Figure 3 visually illustrates how gradients of nearby pixels influence the determination of the main orientation of keypoints. In this depiction, the green pixel represents a keypoint in the target image. The numerical values denote the grayscale intensities at each pixel, while the red arrows indicate the magnitude and direction of the gradients at these points. Notably, the pixels enclosed within the red rectangle are the ones considered when determining the primary orientation, which is indicated by the blue arrow.

The alignment of the blue arrow with the predominant direction of the "long" arrows within the rectangle corroborates our intuitive understanding of the main orientation of keypoints. This voting process is further elucidated by the polar histogram showcased in Figure 4, where the primary orientation prominently emerges.

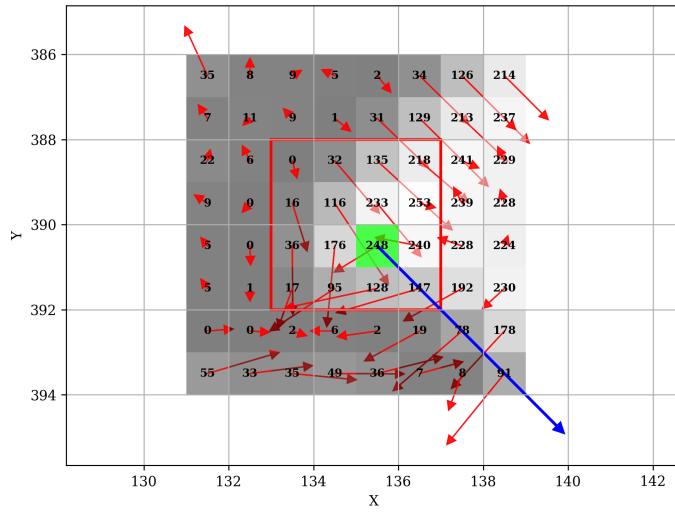


Figure 3 Illustration of the Influence of Local Gradients on Main Orientation

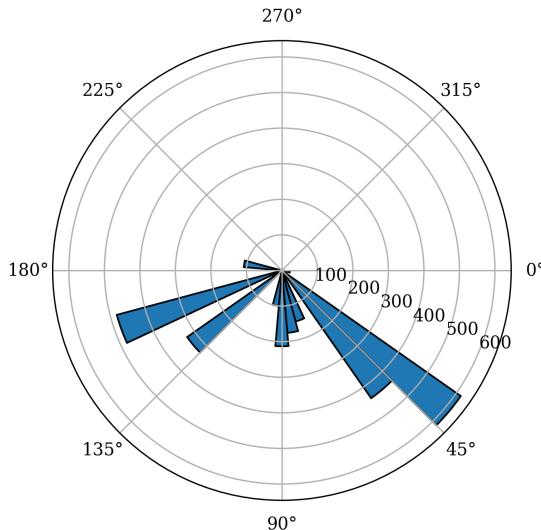


Figure 4 Polar Histogram of Gradient Directions near the Keypoint

Figure 5 shows the main orientations of all keypoints within the target image. Notably, the selected keypoints are all corner points and their main orientations generally point towards regions that appear "brighter". This observation corresponds well with our expectation regarding the primary orientation of gradients.



Figure 5 Main Orientations of All Keypoints in the Target Image

4.1.2 Visualization of Keypoint Neighborhood and Descriptors

Figure 6 illustrates the neighborhood of keypoints that we consider when generating the descriptors. In the figure, the red and blue arrows represent the axes of the object coordinate system. The 16 blocks, each marked with arrows of various colors, depict the 16 groups of pixels that we analyze along with their corresponding gradients. This visualization offers us some insights into what our descriptors are describing.

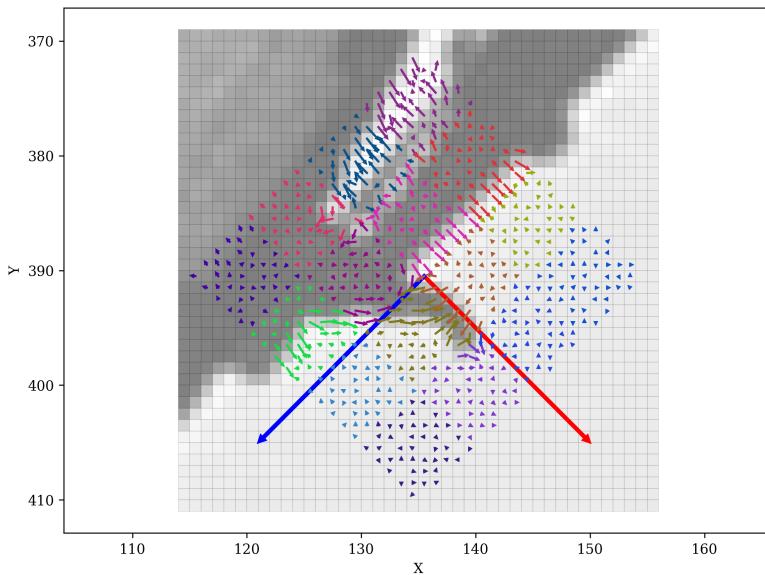
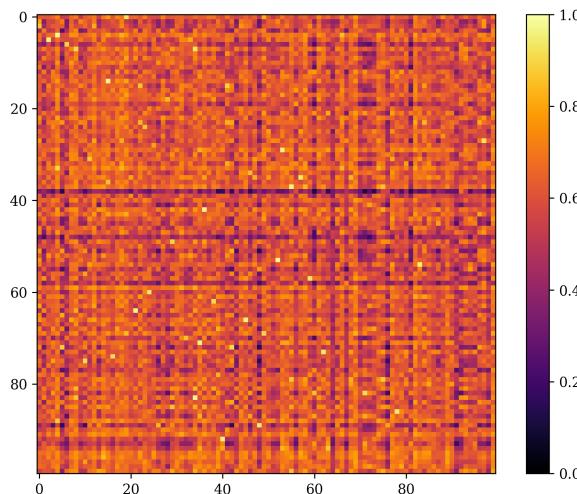


Figure 6 Visualization of the 4×4 Neighborhood of the Keypoint

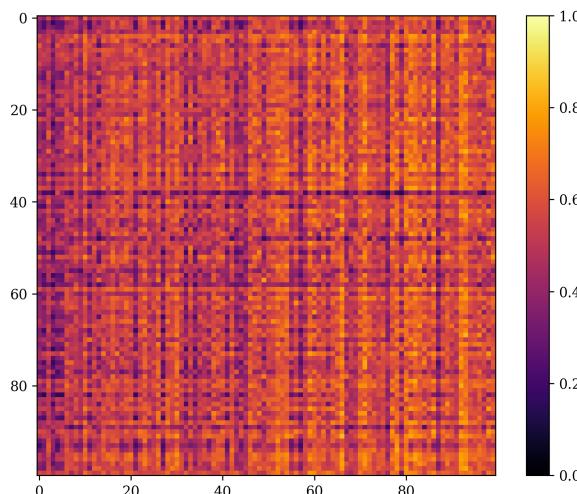
4.1.3 Visualization of Similarities between Descriptors

The matching of figures involves evaluating the similarity between descriptors. Figure 7 consists of two heat maps that illustrate the similarity levels of descriptors from different images, with the level indicated by varying colors.

In the first heat map, which compares descriptors from the target image with those of the matching image, several prominent light yellow pixels stand out, indicating a very high level of similarity. Conversely, in the second heat map, where descriptors from the target image are compared with those from an unmatched image, the absence of such pixels signifies a globally low level of similarity. This observation is in accordance with the matching results.



Comparison with 3.jpg in the dataset (matching image)



Comparison with 2.jpg in the dataset (non-matching image)

Figure 7 Heat Maps of Similarity between Descriptors of the Target Image and Images from the Dataset

4.2 Influence of Receptive Field

During the experiment, it is found that the receptive field plays an important role in matching accuracy. Specifically, the number of pixels around keypoints that we consider is crucial for determining the main orientation and generating descriptors.

Two parameters, denoted as m and n , were formulated to regulate the receptive field. Here, m signifies the radius used in main orientation calculation, while n denotes the radius of the small block during descriptor generation.

For m , a balance must be struck to ensure it is sufficiently small to accurately capture local gradients. However, an excessively small m may result in fewer points and reduced accuracy.

For n , it must be set to an optimal size, neither too small nor too large, to effectively represent the gradient surroundings near the keypoint. Actually, since the matching image is derived from single rotations and scaling of the target image, a larger n tends to yield improved matching outcomes.

It's crucial to highlight that both m and n should adjust according to the input image size. This sensitivity ensures that σ appropriately scales with the image dimensions. Failure to account for this sensitivity could lead to significant discrepancies in the regions described by descriptors, subsequently impacting matching performance.

In practical applications, the following parameters can be utilized:

```
1 alpha, beta = 700, 15
2 sigma = (img.shape[0] + img.shape[1]) / alpha
3 m = int(1.5 * sigma)
4 n = int(sigma * beta)
```

4.3 Highlights of Codes

- **Object-Oriented Approach:** Utilizing an object-oriented design to enhance code readability and maintainability.
- **Integration of tqdm Progress Bar:** Incorporating a tqdm progress bar to display real-time progress updates and indicate successful saves, improving user experience and task monitoring.
- **Command-Line Interface:** Implementation of a command-line interface allowing for the input of various parameters, enhancing versatility, usability, and cross-system compatibility.

5 Reflection and Conclusion (实验感想)

5.1 Learning from Experiments

Through the course of these experiments, we have deepened our comprehension of the renowned SIFT algorithm, refined our skills in parameter selection, and further honed our proficiency in LaTeX. These experiences have not only expanded our technical knowledge but also fostered creativity and innovation within our research endeavors.

5.2 Challenges Encountered

A key challenge encountered during the experiments was the unexpected orientation difference in image dimensions between the default behavior of OpenCV and the actual image data loaded, resulting in a swap of the x and y axes. This discrepancy required careful adjustments to ensure proper data handling and processing for accurate analysis and interpretation of the experimental results.

5.3 Conclusion

Throughout the experiment, we utilized the SIFT algorithm to successfully match the target image with images in the dataset, yielding notable results. Additionally, we employed visualizations to enhance our comprehension of the algorithm's inner workings and discuss the selection of hyperparameters.

A Source Code File List

Table 1 File List

File Name	Description
sift.py	SIFT class implementation
main.py	Matching process and main function
plot.py	Functions for visualizing SIFT algorithm

B Source Code

```
1 """
2 File name: sift.py
3 """
4 import copy
5 import cv2
6 import numpy as np
7 from matplotlib import pyplot as plt
8 from tqdm import tqdm
9 from typing import Tuple
10
11 class SIFT:
12
13     def __init__(self, alpha=700, beta=15):
14         self.alpha = alpha # 超参数alpha (见report)
15         self.beta = beta # 超参数beta (见report)
16
17     def detectAndCompute(
18         self,
19         img: np.ndarray, # 灰度图片
20         mask: None = None, # 掩码, 与OpenCV SIFT接口保持一致
21         corner_num: int = 200 # 关键点数量
22     ):
23
24         """
25             关键点提取及描述子计算
26         """
27
28         # 超参数计算
29         sigma = (img.shape[0] + img.shape[1]) / self.alpha
30         m = int(1.5 * sigma)
31         n = int(sigma * self.beta)
32
33         # 关键点提取
34         corners = cv2.goodFeaturesToTrack(img, corner_num, 0.01, 10).astype(np.int32)
```

```

33
34     # 梯度计算
35     magnitudes, angles = calc_grad(img)
36
37     kps = []
38     dsts = []
39
40     # 逐个处理关键点
41     for corner in tqdm(corners, desc='Processing keypoints'):
42
43         y, x = corner.ravel()
44
45     # 边界检查
46     if not m + 1 <= corner[0][1] <= img.shape[0] - m - 1 or not m + 1 <=
47         corner[0][0] <= img.shape[1] - m - 1:
48         continue
49
50     kps.append(cv2.KeyPoint(float(y), float(x), 1))
51
52     # 确定主方向
53     main_dir, _ = calc_main_dir(magnitudes, angles, (x, y), m)
54
55     # 描述子计算
56     dst = []
57
58     for offset_x in range(-2, 2):
59         for offset_y in range(-2, 2):
60             near_magnitudes = []
61             near_angles = []
62
63             for i in range(offset_x * n, offset_x * n + n):
64                 for j in range(offset_y * n, offset_y * n + n):
65
66                     # 坐标变换
67                     ori_x = x + int(round(i * np.cos(main_dir) - j *
68                         np.sin(main_dir)))
69                     ori_y = y + int(round(i * np.sin(main_dir) + j *
70                         np.cos(main_dir)))
71
72                     try:
73                         near_magnitudes.append(magnitudes[ori_x, ori_y])
74                         near_angles.append(angles[ori_x, ori_y])
75                     except IndexError:
76                         pass # 忽略越界像素
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93

```

```

74     # 角度变换及归一化
75     near_angles = np.array(near_angles) - main_dir
76     near_angles[near_angles < -np.pi] += 2 * np.pi
77     near_angles[near_angles > np.pi] -= 2 * np.pi
78
79     # 更新描述子
80     hist, _ = np.histogram(near_angles, bins=8, range=(-np.pi, np.pi),
81                           weights=near_magnitudes)
82     dst.extend(hist)
83
84     # 描述子归一化
85     dst = np.array(dst)
86     dst = dst / np.linalg.norm(dst)
87     dsts.append(dst)
88
89     kps = np.array(kps)
90     dsts = np.array(dsts, dtype=np.float32)
91
92     return kps, dsts
93
94
95 def calc_grad(
96     img: np.ndarray
97 ):
98     """
99     计算梯度幅值和方向
100
101     Parameters:
102         img (ndarray): Input image.
103
104     Returns:
105         magnitudes (ndarray): Gradient magnitude.
106         angles (ndarray): Gradient angle.
107
108     """
109     img = img.astype(np.int32)
110     img_border = cv2.copyMakeBorder(img, 1, 1, 1, 1, cv2.BORDER_DEFAULT) # 边界填充
111     grad_x = img_border[2:, 1:-1] - img_border[:-2, 1:-1] # x方向梯度
112     grad_y = img_border[1:-1, 2:] - img_border[1:-1, :-2] # y方向梯度
113     magnitudes = np.sqrt(grad_x ** 2 + grad_y ** 2) # 梯度幅值
114     angles = np.arctan2(grad_y, grad_x) # 梯度方向
115
116     return magnitudes, angles

```

```

117     x, y = coord
118     neighbor_magnitudes = magnitudes[x - m:x + m, y - m:y + m].flatten()
119     neighbor_angles = angles[x - m:x + m, y - m:y + m].flatten()
120     hist, bin_edges = np.histogram(neighbor_angles, bins=36, range=(-np.pi, np.pi),
121                                     weights=neighbor_magnitudes)
122     main_dir = (bin_edges[np.argmax(hist)] + bin_edges[np.argmax(hist) + 1]) / 2
123
124
125 if __name__ == '__main__':
126     sift = SIFT()
127
128     image1 = cv2.imread("target.jpg")
129     gray1 = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)
130     kps1, dst1 = sift.detectAndCompute(gray1)
131     image2 = cv2.imread("dataset/3.jpg")
132     gray2 = cv2.cvtColor(image2, cv2.COLOR_BGR2GRAY)
133     kps2, dst2 = sift.detectAndCompute(gray2)
134
135     bf = cv2.BFMatcher()
136     matches = bf.knnMatch(dst1, dst2, k=2)
137     good_matches = [m for m, n in matches if m.distance < 0.7 * n.distance]
138     match_img = cv2.drawMatches(image1, kps1, image2, kps2, good_matches, None,
139                                 flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
140     match_img = cv2.cvtColor(match_img, cv2.COLOR_BGR2RGB)
141     plt.imshow(match_img)
142     plt.axis("off")
143     plt.show()

```

```

1 """
2 File name: main.py
3 """
4 import os
5 import sys
6 import glob
7 import warnings
8 import argparse
9 import cv2
10 import matplotlib.pyplot as plt
11 from tqdm import tqdm
12 from sift import SIFT
13
14 def parse_args(args):

```

```

16 """
17 读取命令行参数
18 """
19
20 parser = argparse.ArgumentParser()
21 parser.add_argument("--image-dir", type=str, default="dataset", help="path to the
22     input image or folder of input images")
23 parser.add_argument("--opencv", type=bool, default=False, help="use OpenCV or not")
24 parser.add_argument("--target-dir", type=str, default="target.jpg", help="path to the
25     target image")
26 parser.add_argument("--output-dir", type=str, default="results", help="path to folder
27     of output images")
28 parser.add_argument("--output-type", type=str, default="png", help="layout of output
29     images")
30
31 args = parser.parse_args(args)
32 return args
33
34
35 def get_image_paths(input_dir, extensions = ("jpg", "jpeg", "png", "bmp")):
36     """
37     找到所有图片文件路径
38     """
39     # 输入是图片文件路径
40     if os.path.isfile(input_dir):
41         assert any(input_dir.lower().endswith(extension) for extension in extensions)
42         return [input_dir]
43
44     # 输入是文件夹路径
45     pattern = f"{input_dir}/**/*"
46     img_paths = []
47
48     for extension in extensions:
49         img_paths.extend(glob.glob(f"{pattern}.{extension}", recursive=True))
50
51     if not img_paths:
52         raise FileNotFoundError(f"No images found in {input_dir}. Supported formats are:
53             {'.'.join(extensions)}")
54
55     return img_paths
56
57
58 def main(args):
59     args = parse_args(args)

```

```

55 # 初始化SIFT
56 sift = SIFT() if not args.opencv else cv2.SIFT_create()
57
58 # 提取目标图像的特征
59 image1 = cv2.imread(args.target_dir)
60 gray1 = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)
61 kps1, dst1 = sift.detectAndCompute(gray1, None)
62
63 img_paths = get_image_paths(args.image_dir)
64 for img_path in tqdm(img_paths, desc='Processing images'):
65
66     # 提取待匹配图像的特征
67     image2 = cv2.imread(img_path)
68     gray2 = cv2.cvtColor(image2, cv2.COLOR_BGR2GRAY)
69     kps2, dst2 = sift.detectAndCompute(gray2, None)
70
71     # 匹配特征 (KNN)
72     bf = cv2.BFMatcher()
73     matches = bf.knnMatch(dst1, dst2, k=2)
74     threshold = 0.8 if not args.opencv else 0.4
75     good_matches = [m for m, n in matches if m.distance < threshold * n.distance]
76     match_img = cv2.drawMatches(image1, kps1, image2, kps2, good_matches, None,
77                                 flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
78
79     # 保存匹配结果
80     if not os.path.exists(args.output_dir):
81         os.makedirs(args.output_dir)
82     original_filename = os.path.splitext(os.path.basename(img_path))[0]
83     type = "my" if not args.opencv else "opencv"
84     save_pth = os.path.join(args.output_dir,
85                            f"{original_filename}-{type}SIFT.{args.output_type}")
86     if os.path.exists(save_pth):
87         warnings.warn(f"File '{save_pth}' already exists. Existing file will be
88                         overwritten.", UserWarning)
89
90     match_img = cv2.cvtColor(match_img, cv2.COLOR_BGR2RGB)
91     plt.imshow(match_img)
92     plt.axis("off")
93     plt.savefig(save_pth, bbox_inches='tight', pad_inches=0, dpi=300)
94
95     print(f"{len(img_paths)} figures successfully saved to {args.output_dir}")
96
97 if __name__ == '__main__':

```

```
96     main(sys.argv[1:])
```

```
1 """
2 File name: plot.py
3 """
4 import random
5 import cv2
6 import numpy as np
7 from copy import deepcopy
8 import colorsys
9 import matplotlib.pyplot as plt
10 import matplotlib.patches as patches
11 from typing import Tuple, Literal, Optional
12 from sift import calc_grad, calc_main_dir, SIFT
13
14 # 设置全局字体
15 plt.rcParams['font.family'] = 'serif'
16 plt.rcParams['font.size'] = 10
17 plt.rcParams['axes.labelsize'] = 10
18 plt.rcParams['xtick.labelsize'] = 10
19
20
21 def generate_random_colors(num_colors, alpha=1.0):
22     """
23     生成随机颜色
24     """
25     colors = []
26     for _ in range(num_colors):
27         h = random.random() # 色相随机
28         s = random.uniform(0.7, 1.0) # 饱和度高
29         v = random.uniform(0.5, 0.9) # 亮度适中
30         r, g, b = colorsys.hsv_to_rgb(h, s, v)
31         colors.append((r, g, b, alpha))
32     return colors
33
34
35 def gray_to_rgba(gray_value, alpha=0.5):
36     """
37     灰度值转RGBA
38     """
39     grayscale = gray_value / 255.0
40     return (grayscale, grayscale, grayscale, alpha)
41
42
```

```

43 def plot_arrow(ax, x, y, angle, magnitude, **kwargs):
44     """
45     绘制箭头
46     """
47     ax.arrow(x, y, magnitude * np.cos(angle), magnitude * np.sin(angle), **kwargs)
48
49
50 def find_corners(
51     img: np.ndarray # 彩色图片
52 ):
53     """
54     输出角点坐标
55     """
56     img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
57     corners = cv2.goodFeaturesToTrack(img, 100, 0.01, 10)
58     return corners.astype(np.int32)
59
60
61 def plot_local_grad(
62     img: np.ndarray, # 彩色图片
63     coord: Tuple[int, int], # 关键点坐标
64     type: Literal['figure', 'hist'] = 'figure' # 可视化类型
65 ):
66     """
67     局部梯度可视化
68     """
69     y, x = coord
70     img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
71     sigma = (img.shape[0] + img.shape[1]) / 700
72     m = int(1.5 * sigma)
73     magnitudes, angles = calc_grad(img)
74
75     # 绘制直方图（极坐标）
76     if type == 'hist':
77         _, hist = calc_main_dir(magnitudes, angles, (x, y), m)
78         bin_edges = np.linspace(0.0, 2 * np.pi, 36, endpoint=False)
79         hist = np.concatenate((hist[18:], hist[:18]))
80
81         fig, ax = plt.subplots(subplot_kw={'projection': 'polar'})
82         ax.set_theta_direction(-1)
83         bars = ax.bar(bin_edges, hist, width=(bin_edges[1] - bin_edges[0]),
84                       edgecolor='black')
85         plt.savefig("figures/local_hist.png", dpi=300)
86         plt.show()

```

```

86
87     # 绘制局部梯度示意图
88     elif type == 'figure':
89         main_dir, _ = calc_main_dir(magnitudes, angles, (x, y), m)
90
91         fig, ax = plt.subplots(figsize=(10, 8))
92         for i in range(0, img.shape[0]):
93             for j in range(0, img.shape[1]):
94                 if x - 2 * m <= i < x + 2 * m and y - 2 * m <= j < y + 2 * m:
95                     ax.text(j + 0.5, i + 0.5, str(img[i, j]), ha='center', va='center',
96                             fontdict={"weight": "bold", "size": 8})
97                     ax.add_patch(patches.Rectangle((j, i), 1, 1, linewidth=0,
98                                         edgecolor='none', facecolor=gray_to_rgba(img[i, j])))
99                     plot_arrow(ax, j + 0.5, i + 0.5, angles[i, j], magnitudes[i, j] * 0.008,
100                                color=(1, 0, 0, 0.9), linewidth=1, head_width=0.2, head_length=0.2)
101
102                     ax.add_patch(patches.Rectangle((y - m, x - m), 2 * m, 2 * m, linewidth=2,
103                                         edgecolor='red', facecolor='none'))
104                     ax.add_patch(patches.Rectangle((y, x), 1, 1, linewidth=0, edgecolor='none',
105                                         facecolor=(0, 1, 0, 0.7)))
106                     plot_arrow(ax, y + 0.5, x + 0.5, main_dir, 6, color=(0, 0, 1, 1), linewidth=2,
107                                head_width=0.2, head_length=0.2)
108
109                     plt.xlabel("X")
110                     plt.ylabel("Y")
111                     plt.gca().invert_yaxis()
112                     plt.axis("equal")
113                     plt.grid(True)
114                     plt.savefig("figures/local_grad.png", dpi=300)
115                     plt.show()
116
117     """
118     全局梯度可视化
119     """
120     img_copy = deepcopy(img)
121     img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
122     sigma = (img.shape[0] + img.shape[1]) / 700
123     m = int(1.5 * sigma)
124
125     corners = cv2.goodFeaturesToTrack(img, 100, 0.01, 10)

```

```

124     corners = corners.astype(np.int32)
125     magnitudes, angles = calc_grad(img)
126
127     for corner in corners:
128         y, x = corner.ravel()
129         if not m + 1 <= x <= img.shape[0] - m - 1 or not m + 1 <= y <= img.shape[1] - m -
130             1:
131             continue
132
133         main_dir, _ = calc_main_dir(magnitudes, angles, (x, y), m)
134
135         cv2.circle(img_copy, (y, x), 1, (0, 0, 255), -1)
136         end_x = int(x + 20 * np.cos(main_dir))
137         end_y = int(y + 20 * np.sin(main_dir))
138         cv2.arrowedLine(img_copy, (y, x), (end_y, end_x), (0, 0, 255), 1)
139
140         grad_img = cv2.cvtColor(img_copy, cv2.COLOR_BGR2RGB)
141         plt.imshow(grad_img)
142         plt.axis('off')
143         plt.tight_layout()
144         plt.savefig("figures/global_grad.png", dpi=300)
145         plt.show()
146
147     def plot_neighbour_grad(
148         img: np.ndarray, # 彩色图片
149         coord: Tuple[int, int], # 关键点坐标
150         magnify: Optional[int] = None # 局部放大
151     ):
152         """
153         邻域梯度可视化
154         """
155         y, x = coord
156         img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
157         sigma = (img.shape[0] + img.shape[1]) / 700
158         m = int(1.5 * sigma)
159         n = int(sigma * 5)
160
161         magnitudes, angles = calc_grad(img)
162
163         main_dir, _ = calc_main_dir(magnitudes, angles, (x, y), m)
164
165         fig, ax = plt.subplots(figsize=(8, 6))
166         for i in range(0, img.shape[0]):
167             for j in range(0, img.shape[1]):
168                 if x - 3 * n <= i < x + 3 * n and y - 3 * n <= j < y + 3 * n:

```

```

167         rect = patches.Rectangle((j, i), 1, 1, linewidth=0.05, edgecolor='black',
168                               facecolor=gray_to_rgba(img[i, j]))
169         ax.add_patch(rect)
170
171     # 绘制关键点主方向
172     plot_arrow(ax, y + 0.5, x + 0.5, main_dir, 20, color=(1, 0, 0, 1), linewidth=3,
173                 head_width=0.3, head_length=0.3)
174     plot_arrow(ax, y + 0.5, x + 0.5, main_dir + np.pi / 2, 20, color=(0, 0, 1, 1),
175                 linewidth=3, head_width=0.3, head_length=0.3)
176
177     # 绘制邻域梯度
178     colors = generate_random_colors(16)
179     cnt = 0
180
181     for off_i in range(-2, 2):
182         for off_j in range(-2, 2):
183             cnt += 1
184
185             for i in range(off_i * n, off_i * n + n):
186                 for j in range(off_j * n, off_j * n + n):
187                     ori_i = x + int(round(i * np.cos(main_dir) - j * np.sin(main_dir)))
188                     ori_j = y + int(round(i * np.sin(main_dir) + j * np.cos(main_dir)))
189
190                     plot_arrow(ax, ori_j + 0.5, ori_i + 0.5, angles[ori_i, ori_j],
191                                magnitudes[ori_i, ori_j] * 0.005, color=colors[cnt - 1],
192                                head_width=0.15, head_length=0.15, linewidth=1.5)
193
194     plt.xlabel('X')
195     plt.ylabel('Y')
196     plt.axis('equal')
197
198     # 是否局部放大
199     if magnify:
200
201         plt.xlim(y - magnify * n, y + magnify * n)
202         plt.ylim(x - magnify * n, x + magnify * n)
203
204         plt.gca().invert_yaxis()
205         plt.grid(False)
206
207         plt.tight_layout()
208
209         plt.savefig("figures/neighbour_grad_magn.png", dpi=300)
210
211         plt.show()
212
213
214     def plot_similarity(
215         dst1: np.ndarray,
216         dst2: np.ndarray,
217     ):

```

```
206     """
207     描述子相似度可视化
208     """
209     similarity_matrix = np.dot(dst1, dst2.T)
210     plt.figure(figsize=(8, 6))
211     plt.imshow(similarity_matrix, cmap='inferno', interpolation='nearest', vmin=0, vmax=1)
212     plt.colorbar()
213     plt.savefig("figures/similarity5.png", dpi=300)
214     plt.show()
215
216
217 if __name__ == '__main__':
218     image1 = cv2.imread("target.jpg")
219     # print(find_corners(image1))
220     # plot_local_grad(image1, (135, 390), type='figure')
221     # plot_local_grad(image1, (135, 390), type='hist')
222     # plot_global_grad(image1)
223     plot_neighbour_grad(image1, (135, 390), magnify=None)
224
225     # image1 = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)
226     # image2 = cv2.cvtColor(cv2.imread("dataset/5.jpg"), cv2.COLOR_BGR2GRAY)
227     # sift = SIFT()
228     # kp1, dst1 = sift.detectAndCompute(image1, corner_num=100)
229     # kp2, dst2 = sift.detectAndCompute(image2, corner_num=100)
230     # plot_similarity(dst1, dst2)
```