ICE2607 Lab 3: SIFT

SJTU-SEIEE 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:

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

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

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

```
                weights=near_magnitudes)
            dst.extend(hist)
    # Descriptor normalization
    dst = np.array(dst)
    dst = dst / np.linalg.norm(dst)
    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.

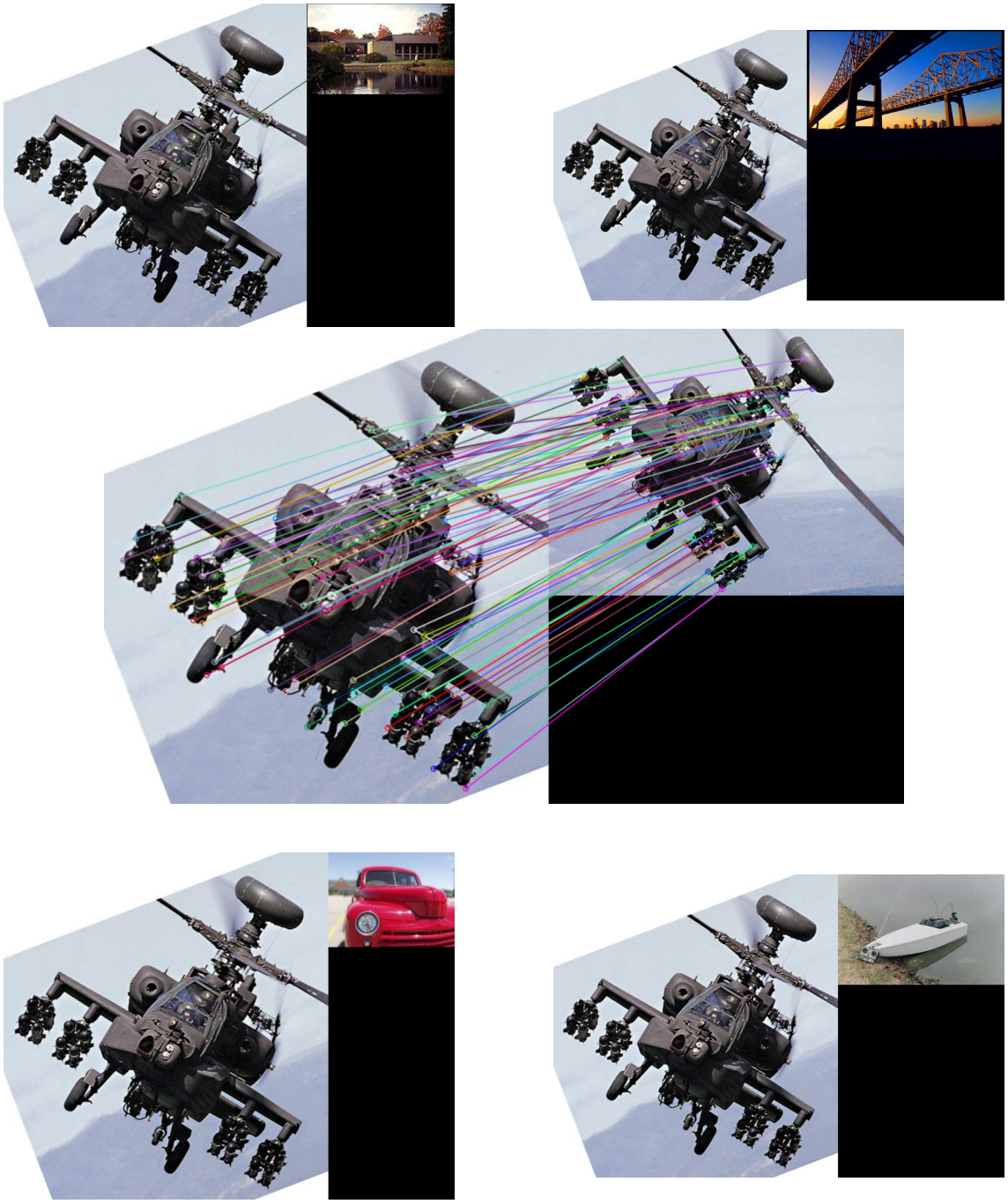


Figure 1 Resulting Images of our SIFT Implementation

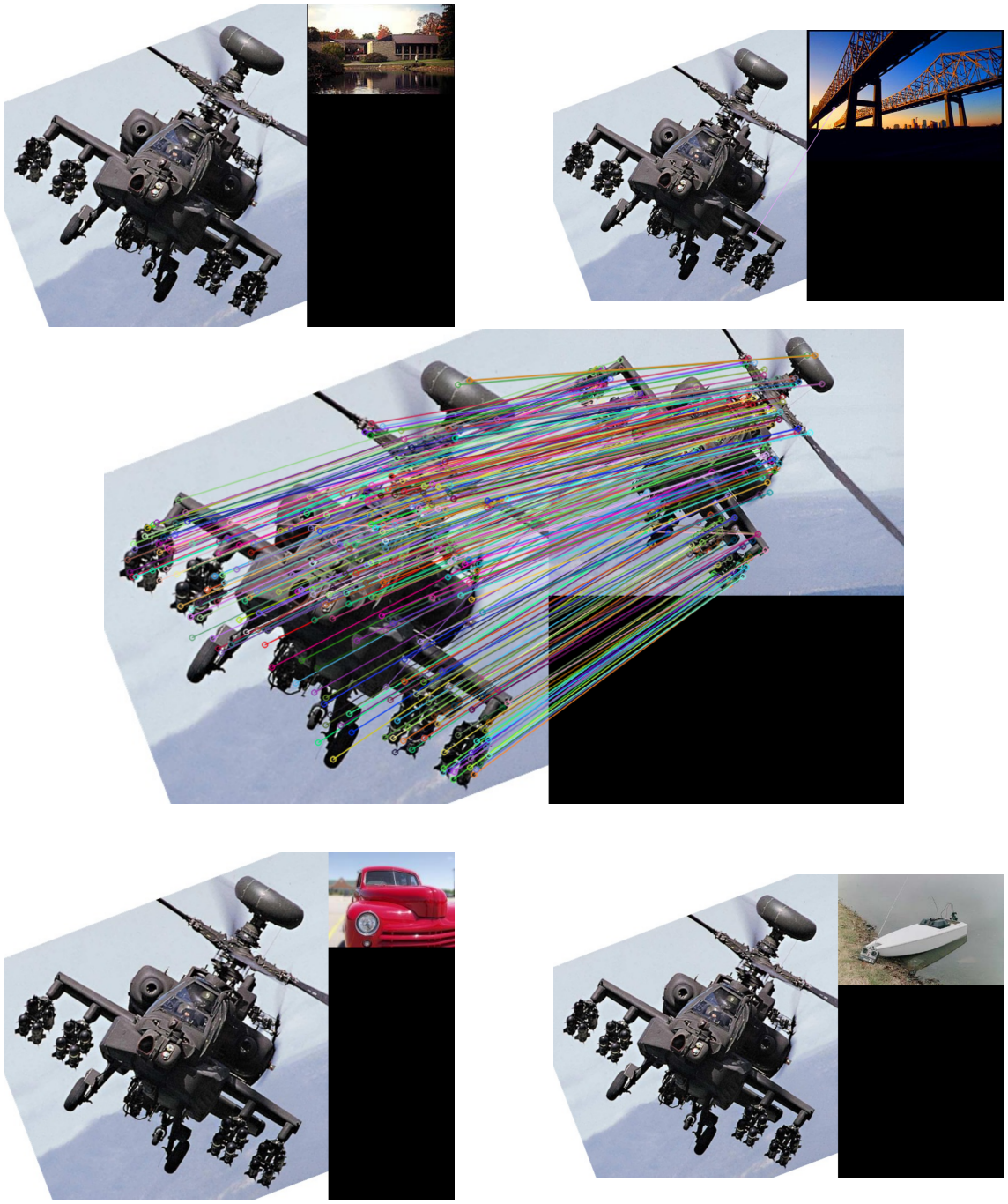


Figure 2 Resulting Images of OpenCV's SIFT Algorithm

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.

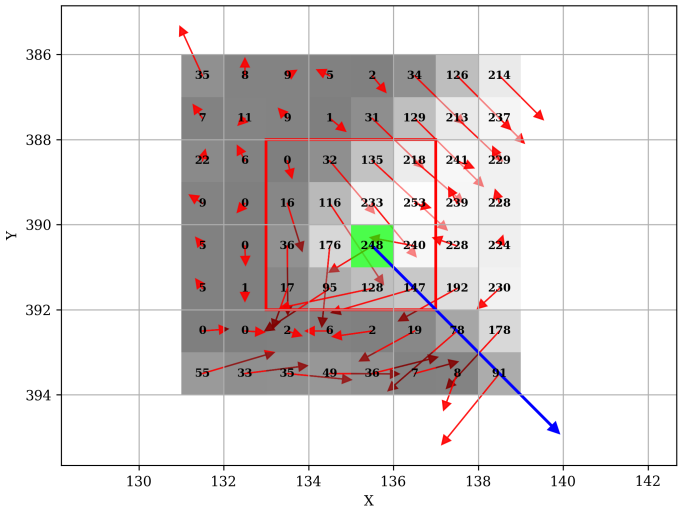


Figure 3 Illustration of the Influence of Local Gradients on Main Orientation

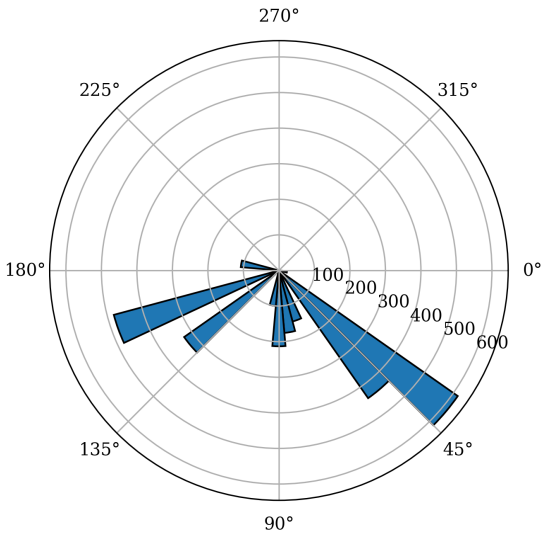


Figure 4 Polar Histogram of Gradient Directions near the Keypoint

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.

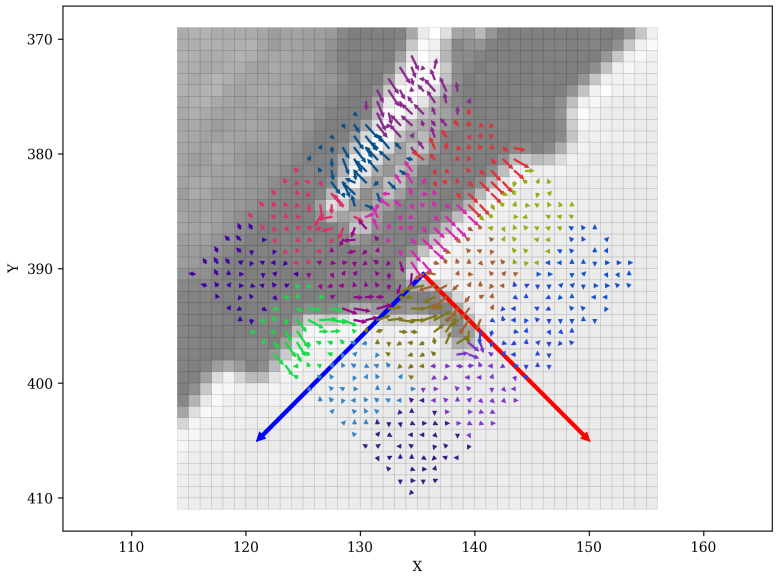
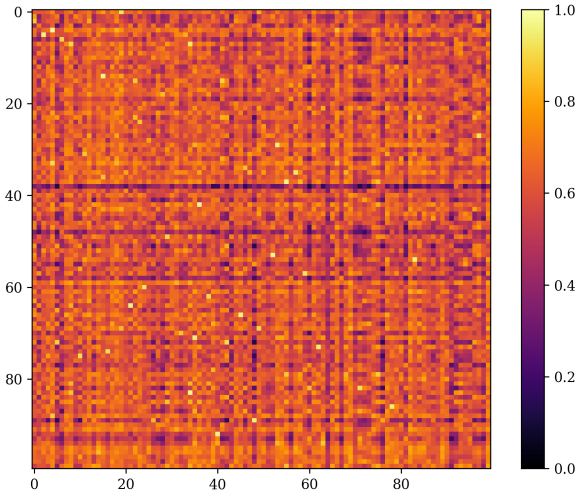


Figure 6 Visualization of the 4×4 Neighborhood of the Keypoint

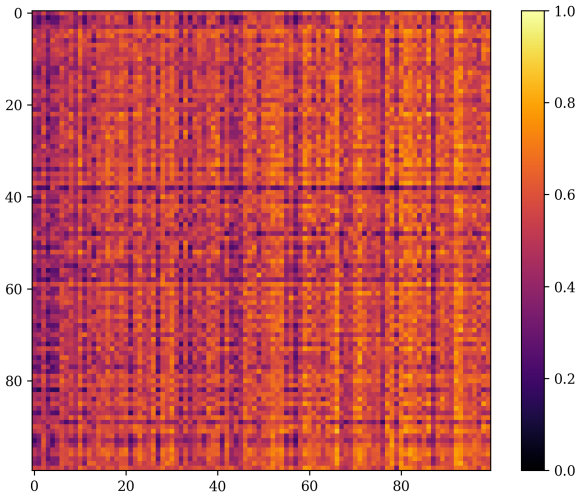
4.1.3 Visualization of Similarities between Descriptors

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:

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

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

```python
"""
File name: sift.py
"""
import copy
import cv2
import numpy as np
from matplotlib import pyplot as plt
from tqdm import tqdm
from typing import Tuple

class SIFT:

    def __init__(self, alpha=700, beta=15):
        self.alpha = alpha # 超参数alpha（见report）
        self.beta = beta # 超参数beta（见report）

    def detectAndCompute(
            self,
            img: np.ndarray, # 灰度图片
            mask: None = None, # 掩码，与OpenCV SIFT接口保持一致
            corner_num: int = 200 # 关键点数量
    ):
        """
        关键点提取及描述子计算
        """
        # 超参数计算
        sigma = (img.shape[0] + img.shape[1]) / self.alpha
        m = int(1.5 * sigma)
        n = int(sigma * self.beta)

        # 关键点提取
        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 <=
                    corner[0][0] <= img.shape[1] - m - 1:
47                  continue
48              kps.append(cv2.KeyPoint(float(y), float(x), 1))
49
50              # 确定主方向
51              main_dir, _ = calc_main_dir(magnitudes, angles, (x, y), m)
52
53              # 描述子计算
54              dst = []
55
56              for offset_x in range(-2, 2):
57                  for offset_y in range(-2, 2):
58                      near_magnitudes = []
59                      near_angles = []
60
61                      for i in range(offset_x * n, offset_x * n + n):
62                          for j in range(offset_y * n, offset_y * n + n):
63
64                              # 坐标变换
65                              ori_x = x + int(round(i * np.cos(main_dir) - j *
                                    np.sin(main_dir)))
66                              ori_y = y + int(round(i * np.sin(main_dir) + j *
                                    np.cos(main_dir)))
67
68                              try:
69                                  near_magnitudes.append(magnitudes[ori_x, ori_y])
70                                  near_angles.append(angles[ori_x, ori_y])
71                              except IndexError:
72                                  pass # 忽略越界像素
73
```

```
                # 角度变换及归一化
                near_angles = np.array(near_angles) - main_dir
                near_angles[near_angles < -np.pi] += 2 * np.pi
                near_angles[near_angles > np.pi] -= 2 * np.pi

                # 更新描述子
                hist, _ = np.histogram(near_angles, bins=8, range=(-np.pi, np.pi),
                    weights=near_magnitudes)
                dst.extend(hist)

            # 描述子归一化
            dst = np.array(dst)
            dst = dst / np.linalg.norm(dst)
            dsts.append(dst)

        kps = np.array(kps)
        dsts = np.array(dsts, dtype=np.float32)
        return kps, dsts


def calc_grad(
        img: np.ndarray
):
    """
    计算梯度幅值和方向
    """
    img = img.astype(np.int32)
    img_border = cv2.copyMakeBorder(img, 1, 1, 1, 1, cv2.BORDER_DEFAULT) # 边界填充
    grad_x = img_border[2:, 1:-1] - img_border[:-2, 1:-1] # x方向梯度
    grad_y = img_border[1:-1, 2:] - img_border[1:-1, :-2] # y方向梯度
    magnitudes = np.sqrt(grad_x ** 2 + grad_y ** 2) # 梯度幅值
    angles = np.arctan2(grad_y, grad_x) # 梯度方向
    return magnitudes, angles


def calc_main_dir(
        magnitudes: np.ndarray,
        angles: np.ndarray,
        coord: Tuple[int, int],
        m: int
):
    """
    计算主方向
    """
```

```
    x, y = coord
    neighbor_magnitudes = magnitudes[x - m:x + m, y - m:y + m].flatten()
    neighbor_angles = angles[x - m:x + m, y - m:y + m].flatten()
    hist, bin_edges = np.histogram(neighbor_angles, bins=36, range=(-np.pi, np.pi),
        weights=neighbor_magnitudes)
    main_dir = (bin_edges[np.argmax(hist)] + bin_edges[np.argmax(hist) + 1]) / 2
    return main_dir, hist


if __name__ == '__main__':
    sift = SIFT()

    image1 = cv2.imread("target.jpg")
    gray1 = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)
    kps1, dst1 = sift.detectAndCompute(gray1)
    image2 = cv2.imread("dataset/3.jpg")
    gray2 = cv2.cvtColor(image2, cv2.COLOR_BGR2GRAY)
    kps2, dst2 = sift.detectAndCompute(gray2)

    bf = cv2.BFMatcher()
    matches = bf.knnMatch(dst1, dst2, k=2)
    good_matches = [m for m, n in matches if m.distance < 0.7 * n.distance]
    match_img = cv2.drawMatches(image1, kps1, image2, kps2, good_matches, None,
        flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
    match_img = cv2.cvtColor(match_img, cv2.COLOR_BGR2RGB)
    plt.imshow(match_img)
    plt.axis("off")
    plt.show()
```

```
"""
File name: main.py
"""
import os
import sys
import glob
import warnings
import argparse
import cv2
import matplotlib.pyplot as plt
from tqdm import tqdm
from sift import SIFT


def parse_args(args):
```

```
    """
    读取命令行参数
    """
    parser = argparse.ArgumentParser()
    parser.add_argument("--image-dir", type=str, default="dataset", help="path to the
        input image or folder of input images")
    parser.add_argument("--opencv", type=bool, default=False, help="use OpenCV or not")
    parser.add_argument("--target-dir", type=str, default="target.jpg", help="path to the
        target image")
    parser.add_argument("--output-dir", type=str, default="results", help="path to folder
        of output images")
    parser.add_argument("--output-type", type=str, default="png", help="layout of output
        images")

    args = parser.parse_args(args)
    return args


def get_image_paths(input_dir, extensions = ("jpg", "jpeg", "png", "bmp")):
    """
    找到所有图片文件路径
    """
    # 输入是图片文件路径
    if os.path.isfile(input_dir):
        assert any(input_dir.lower().endswith(extension) for extension in extensions)
        return [input_dir]

    # 输入是文件夹路径
    pattern = f"{input_dir}/**/*"
    img_paths = []

    for extension in extensions:
        img_paths.extend(glob.glob(f"{pattern}.{extension}", recursive=True))

    if not img_paths:
        raise FileNotFoundError(f"No images found in {input_dir}. Supported formats are:
            {', '.join(extensions)}")

    return img_paths


def main(args):
    args = parse_args(args)

```

```
    # 初始化SIFT
    sift = SIFT() if not args.opencv else cv2.SIFT_create()

    # 提取目标图像的特征
    image1 = cv2.imread(args.target_dir)
    gray1 = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)
    kps1, dst1 = sift.detectAndCompute(gray1, None)

    img_paths = get_image_paths(args.image_dir)
    for img_path in tqdm(img_paths, desc='Processing images'):

        # 提取待匹配图像的特征
        image2 = cv2.imread(img_path)
        gray2 = cv2.cvtColor(image2, cv2.COLOR_BGR2GRAY)
        kps2, dst2 = sift.detectAndCompute(gray2, None)

        # 匹配特征（KNN）
        bf = cv2.BFMatcher()
        matches = bf.knnMatch(dst1, dst2, k=2)
        threshold = 0.8 if not args.opencv else 0.4
        good_matches = [m for m, n in matches if m.distance < threshold * n.distance]
        match_img = cv2.drawMatches(image1, kps1, image2, kps2, good_matches, None,
            flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)

        # 保存匹配结果
        if not os.path.exists(args.output_dir):
            os.makedirs(args.output_dir)
        original_filename = os.path.splitext(os.path.basename(img_path))[0]
        type = "my" if not args.opencv else "opencv"
        save_pth = os.path.join(args.output_dir,
            f"{original_filename}-{type}SIFT.{args.output_type}")
        if os.path.exists(save_pth):
            warnings.warn(f"File '{save_pth}' already exists. Existing file will be
                overwritten.", UserWarning)

        match_img = cv2.cvtColor(match_img, cv2.COLOR_BGR2RGB)
        plt.imshow(match_img)
        plt.axis("off")
        plt.savefig(save_pth, bbox_inches='tight', pad_inches=0, dpi=300)

    print(f"{len(img_paths)} figures successfully saved to {args.output_dir}")


if __name__ == '__main__':
```

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

```
"""
File name: plot.py
"""
import random
import cv2
import numpy as np
from copy import deepcopy
import colorsys
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from typing import Tuple, Literal, Optional
from sift import calc_grad, calc_main_dir, SIFT

# 设置全局字体
plt.rcParams['font.family'] = 'serif'
plt.rcParams['font.size'] = 10
plt.rcParams['axes.labelsize'] = 10
plt.rcParams['xtick.labelsize'] = 10


def generate_random_colors(num_colors, alpha=1.0):
    """
    生成随机颜色
    """
    colors = []
    for _ in range(num_colors):
        h = random.random() # 色相随机
        s = random.uniform(0.7, 1.0) # 饱和度高
        v = random.uniform(0.5, 0.9) # 亮度适中
        r, g, b = colorsys.hsv_to_rgb(h, s, v)
        colors.append((r, g, b, alpha))
    return colors


def gray_to_rgba(gray_value, alpha=0.5):
    """
    灰度值转RGBA
    """
    grayscale = gray_value / 255.0
    return (grayscale, grayscale, grayscale, alpha)


```

```python
def plot_arrow(ax, x, y, angle, magnitude, **kwargs):
    """
    绘制箭头
    """
    ax.arrow(x, y, magnitude * np.cos(angle), magnitude * np.sin(angle), **kwargs)


def find_corners(
        img: np.ndarray # 彩色图片
):
    """
    输出角点坐标
    """
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    corners = cv2.goodFeaturesToTrack(img, 100, 0.01, 10)
    return corners.astype(np.int32)


def plot_local_grad(
        img: np.ndarray, # 彩色图片
        coord: Tuple[int, int], # 关键点坐标
        type: Literal['figure', 'hist'] = 'figure' # 可视化类型
):
    """
    局部梯度可视化
    """
    y, x = coord
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    sigma = (img.shape[0] + img.shape[1]) / 700
    m = int(1.5 * sigma)
    magnitudes, angles = calc_grad(img)

    # 绘制直方图（极坐标）
    if type == 'hist':
        _, hist = calc_main_dir(magnitudes, angles, (x, y), m)
        bin_edges = np.linspace(0.0, 2 * np.pi, 36, endpoint=False)
        hist = np.concatenate((hist[18:], hist[:18]))

        fig, ax = plt.subplots(subplot_kw={'projection': 'polar'})
        ax.set_theta_direction(-1)
        bars = ax.bar(bin_edges, hist, width=(bin_edges[1] - bin_edges[0]),
            edgecolor='black')
        plt.savefig("figures/local_hist.png", dpi=300)
        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',
                            fontdict={"weight": 'bold', "size": 8})
96                      ax.add_patch(patches.Rectangle((j, i), 1, 1, linewidth=0,
                            edgecolor='none', facecolor=gray_to_rgba(img[i, j])))
97                      plot_arrow(ax, j + 0.5, i + 0.5, angles[i, j], magnitudes[i, j] * 0.008,
                            color=(1, 0, 0, 0.9), linewidth=1, head_width=0.2, head_length=0.2)
98
99          ax.add_patch(patches.Rectangle((y - m, x - m), 2 * m, 2 * m, linewidth=2,
                edgecolor='red', facecolor='none'))
100         ax.add_patch(patches.Rectangle((y, x), 1, 1, linewidth=0, edgecolor='none',
                facecolor=(0, 1, 0, 0.7)))
101         plot_arrow(ax, y + 0.5, x + 0.5, main_dir, 6, color=(0, 0, 1, 1), linewidth=2,
                head_width=0.2, head_length=0.2)
102
103         plt.xlabel("X")
104         plt.ylabel("Y")
105         plt.gca().invert_yaxis()
106         plt.axis("equal")
107         plt.grid(True)
108         plt.savefig("figures/local_grad.png", dpi=300)
109         plt.show()
110
111
112 def plot_global_grad(
113         img: np.ndarray # 彩色图片
114 ):
115     """
116     全局梯度可视化
117     """
118     img_copy = deepcopy(img)
119     img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
120     sigma = (img.shape[0] + img.shape[1]) / 700
121     m = int(1.5 * sigma)
122
123     corners = cv2.goodFeaturesToTrack(img, 100, 0.01, 10)
```

```
    corners = corners.astype(np.int32)
    magnitudes, angles = calc_grad(img)

    for corner in corners:
        y, x = corner.ravel()
        if not m + 1 <= x <= img.shape[0] - m - 1 or not m + 1 <= y <= img.shape[1] - m -
            1:
            continue
        main_dir, _ = calc_main_dir(magnitudes, angles, (x, y), m)

        cv2.circle(img_copy, (y, x), 1, (0, 0, 255), -1)
        end_x = int(x + 20 * np.cos(main_dir))
        end_y = int(y + 20 * np.sin(main_dir))
        cv2.arrowedLine(img_copy, (y, x), (end_y, end_x), (0, 0, 255), 1)

    grad_img = cv2.cvtColor(img_copy, cv2.COLOR_BGR2RGB)
    plt.imshow(grad_img)
    plt.axis('off')
    plt.tight_layout()
    plt.savefig("figures/global_grad.png", dpi=300)
    plt.show()


def plot_neighbor_grad(
        img: np.ndarray, # 彩色图片
        coord: Tuple[int, int], # 关键点坐标
        magnify: Optional[int] = None # 局部放大
):
    """
    邻域梯度可视化
    """
    y, x = coord
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    sigma = (img.shape[0] + img.shape[1]) / 700
    m = int(1.5 * sigma)
    n = int(sigma * 5)
    magnitudes, angles = calc_grad(img)

    main_dir, _ = calc_main_dir(magnitudes, angles, (x, y), m)

    fig, ax = plt.subplots(figsize=(8, 6))
    for i in range(0, img.shape[0]):
        for j in range(0, img.shape[1]):
            if x - 3 * n <= i < x + 3 * n and y - 3 * n <= j < y + 3 * n:
```

```
            rect = patches.Rectangle((j, i), 1, 1, linewidth=0.05, edgecolor='black',
                facecolor=gray_to_rgba(img[i, j]))
            ax.add_patch(rect)

    # 绘制关键点主方向
    plot_arrow(ax, y + 0.5, x + 0.5, main_dir, 20, color=(1, 0, 0, 1), linewidth=3,
        head_width=0.3, head_length=0.3)
    plot_arrow(ax, y + 0.5, x + 0.5, main_dir + np.pi / 2, 20, color=(0, 0, 1, 1),
        linewidth=3, head_width=0.3, head_length=0.3)

    # 绘制邻域梯度
    colors = generate_random_colors(16)
    cnt = 0
    for off_i in range(-2, 2):
           for off_j in range(-2, 2):
               cnt += 1

               for i in range(off_i * n, off_i * n + n):
                   for j in range(off_j * n, off_j * n + n):
                       ori_i = x + int(round(i * np.cos(main_dir) - j * np.sin(main_dir)))
                       ori_j = y + int(round(i * np.sin(main_dir) + j * np.cos(main_dir)))

                       plot_arrow(ax, ori_j + 0.5, ori_i + 0.5, angles[ori_i, ori_j],
                           magnitudes[ori_i, ori_j] * 0.005, color=colors[cnt - 1],
                           head_width=0.15, head_length=0.15, linewidth=1.5)

    plt.xlabel('X')
    plt.ylabel('Y')
    plt.axis('equal')
    # 是否局部放大
    if magnify:
        plt.xlim(y - magnify * n, y + magnify * n)
        plt.ylim(x - magnify * n, x + magnify * n)
    plt.gca().invert_yaxis()
    plt.grid(False)
    plt.tight_layout()
    plt.savefig("figures/neighbour_grad_magn.png", dpi=300)
    plt.show()


def plot_similarity(
        dst1: np.ndarray,
        dst2: np.ndarray,
):
```

```
    """
    描述子相似度可视化
    """
    similarity_matrix = np.dot(dst1, dst2.T)
    plt.figure(figsize=(8, 6))
    plt.imshow(similarity_matrix, cmap='inferno', interpolation='nearest', vmin=0, vmax=1)
    plt.colorbar()
    plt.savefig("figures/similarity5.png", dpi=300)
    plt.show()


if __name__ == '__main__':
    image1 = cv2.imread("target.jpg")
    # print(find_corners(image1))
    # plot_local_grad(image1, (135, 390), type='figure')
    # plot_local_grad(image1, (135, 390), type='hist')
    # plot_global_grad(image1)
    # plot_neighbor_grad(image1, (135, 390), magnify=None)

    # image1 = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)
    # image2 = cv2.cvtColor(cv2.imread("dataset/5.jpg"), cv2.COLOR_BGR2GRAY)
    # sift = SIFT()
    # kp1, dst1 = sift.detectAndCompute(image1, corner_num=100)
    # kp2, dst2 = sift.detectAndCompute(image2, corner_num=100)
    # plot_similarity(dst1, dst2)
```