Journal of Electronic Research and Application, 2025, Volume 9, Issue 4
http://ojs.bbwpublisher.com/index.php/JERA
ISSN Online: 2208-3510
ISSN Print: 2208-3502

Panoramic Image Stitching of the South Campus (East Gate) of Shaanxi University of Technology

Ningxi Wu, Jieyi Tan

School of Mathematics and Computer Science, Shaanxi University of Technology, Hanzhong 723000, China

Abstract: With the development of computer vision technology, panoramic image stitching has been widely used in fields such as scene reconstruction. A single traditional image cannot fully capture the panoramic view of the iconic East Gate of the South Campus of Shaanxi University of Technology. Therefore, this project aims to technically fuse multiple partial images into a complete panoramic image, enabling comprehensive recording and visual presentation of the architectural landscapes and spatial environments in this area. This report first introduces the technical background and application scenarios, clarifying the necessity of panoramic image stitching in campus landscape recording. It then elaborates on the core objectives and practical values, highlighting the role of technical solutions in improving image quality. Technically, a modular system design based on OpenCV is adopted, including modules such as image preprocessing, feature extraction and matching, image registration, fusion, and post-processing. Specifically, the SIFT algorithm is applied for feature extraction, KNN combined with ratio testing is used for feature matching, image registration is achieved by calculating the homography matrix, the fusion process utilizes multiband blending and Laplacian pyramid, and post-processing includes operations such as black area filling and CLAHE contrast enhancement. The experiment was conducted in a specific hardware and software environment using five overlapping images. After preprocessing, stitching, detail enhancement, and black edge repair, a panoramic image was successfully generated. The results show that the panoramic image fully presents the relevant scenery, with concealed seams, balanced exposure differences, and strong hierarchical details. This report provides a systematic description of the project's technical implementation and achievement application.

Keywords: Panoramic image stitching; SIFT algorithm; Multiband blending; Cylindrical projection; CLAHE

Online publication: August 7, 2025

1. Introduction

With the advancement of computer vision technology, panoramic image stitching has found widespread applications in areas such as scene reconstruction, virtual reality, and geographic information systems [1]. In the context of campus landscape recording and display, traditional single images are limited by their perspective range, making it difficult to fully present the entirety of buildings or expansive scenes. The eastern gate of the South Campus of Shaanxi University of Technology, as an iconic area of the campus, has a demand for panoramic image

stitching derived from the comprehensive recording and visual display of architectural landscapes and spatial environments. Technical means are required to fuse multiple partial images into a complete panoramic view.

2. Related theories

2.1. Principles of image stitching

Image stitching involves synthesizing multiple images with overlapping regions into a panoramic image. The basic process includes image acquisition, feature extraction, image registration, projection transformation, and image fusion. In 2020, Fu Ziqiu mentioned that for scenes such as campus buildings and natural landscapes, cylindrical projection is used to map the scene onto a cylindrical surface, reducing horizontal perspective distortion and adapting to wide-angle outdoor scenes. Simultaneously, multi-band fusion technology is employed to process overlapping regions in the frequency domain, reducing the visibility of stitching seams and enhancing the visual continuity of the panoramic image [2].

2.2. Feature extraction and matching

This paper utilizes the SIFT algorithm, as mentioned by Xiang Ziwei, to extract image features. Scale-invariant keypoints are detected through the Difference of Gaussians (DOG) pyramid, and 128-dimensional descriptors are generated based on the gradient directions in the keypoint neighborhoods. This makes the features robust to rotation, scaling, and illumination changes [3]. In the feature matching stage, K-Nearest Neighbors (KNN) combined with a ratio test (threshold of 0.75) is adopted. By comparing the ratio of the distances between the nearest neighbor and the second nearest neighbor matches, only matching pairs with a distance ratio less than 0.75 are retained. This filters out reliable matching points, reduces mismatches, and improves the accuracy of subsequent image registration.

2.3. Image registration

Image registration aligns adjacent images by computing a homography matrix, which describes the projection transformation between two planes. This matrix can be solved using the RANSAC algorithm with at least 4 pairs of feature points. OpenCV's Stitcher class automatically handles multi-image global registration, projecting the images onto a unified coordinate system (such as a cylindrical coordinate system). If registration fails, it is often due to insufficient image overlap or low-quality feature matching, requiring adjustments to the shooting angle or increasing the overlap rate.

2.4. Image fusion

Image fusion is a critical step in eliminating stitching seams. The multi-band fusion algorithm used in the code decomposes images into different frequency channels (low frequencies represent overall structure, while high frequencies represent details). Weighted fusion is performed separately on each channel, and then the image is reconstructed. Compared to simple linear weighted fusion, this method better preserves image details and balances exposure differences. Additionally, the code utilizes the cv2.inpaint function with the Telea algorithm to repair blank areas resulting from projection transformations based on neighboring pixel information, further enhancing the integrity of the panoramic image.

2.5. Image scaling principles

During the image loading phase, the cv2.resize function is used to scale the image proportionally by specifying a scaling factor. This principle is based on image sampling theory. For image reduction, pixels are selected at intervals to generate a new image, while for enlargement, interpolation algorithms (bilinear interpolation by default) are used to estimate the values of newly added pixels. In this project, reducing the image size can decrease the computational load of feature extraction, matching, and stitching, thereby improving processing efficiency and reducing the impact of noise on feature extraction.

2.6. Correlation between Laplacian pyramid and multi-band fusion

In their 2019 paper "Comparison of Multi-band Image Fusion Rules Based on Laplacian Pyramid Transform Method," Huang Fusheng and Lin Suzhen mentioned that the multi-band fusion algorithm is implemented based on the Laplacian pyramid [4]. Firstly, the image is downsampled and filtered through a Gaussian pyramid to obtain image layers of different resolutions. Then, the difference between the original image and the upsampled downsampled image is taken to obtain the various layers of the Laplacian pyramid. The low-frequency part reflects the overall structure of the image, while the high-frequency part embodies detailed information. During fusion, a wide weighted signal is used for the low-frequency layer to ensure the coherence of the overall structure, while a narrow-weighted signal is applied to the high-frequency layer to highlight edge and texture details, thereby eliminating stitching seams and achieving high-quality image fusion.

2.7. Principles of Contrast Limited Adaptive Histogram Equalization (CLAHE)

When enhancing the quality of panoramic images, CLAHE is adopted for contrast enhancement. CLAHE divides the image into specified small blocks (e.g., 8x8) and performs histogram equalization on each block separately. Finally, bilinear interpolation is used to recombine the blocks, which not only enhances the overall contrast of the panoramic image but also preserves details and suppresses noise, clearly showing the details and layers of campus buildings.

2.8. Principles of inpainting

In panoramic image stitching, projection transformations (such as cylindrical projection) may result in black blank areas at the edges of the image. This occurs because the pixels of the source image cannot cover all positions of the target projection plane. The cv2.inpaint function adopted in the code is based on the Telea algorithm to fill in these black areas.

3. Design of panoramic image stitching system

3.1. Overall architecture (including framework diagram)

Overview of system architecture: The panoramic image stitching system adopts a modular design, with OpenCV as the core framework. It implements a complete link from image input to panoramic image output through a multi-stage processing flow. The system mainly consists of an input module, preprocessing module, feature processing module, registration module, fusion module, post-processing module, and output module. These modules work together through data flow. The framework diagram is shown in **Figure 1**.

Module description and data flow:

(1) Input layer: Reads multiple images from a specified folder. The preprocessing module reduces pixel count and computational complexity by scaling (e.g., 0.5x).

(2) Feature processing layer: Uses the SIFT algorithm to extract key points and descriptors. Reliable feature pairs are filtered through KNN matching combined with a ratio test. Feedback is provided to the preprocessing layer.

(3) Registration layer: Calculates the homography matrix based on feature matching results. All images are aligned to a cylindrical coordinate system through global graph optimization, addressing perspective differences.

(4) Fusion layer: Employs multi-band blending to process overlapping areas, generating an initial panorama in combination with cylindrical projection transformation to eliminate seams.

(5) Post-processing layer: Fills in black areas after projection using the Telea algorithm, enhances contrast with CLAHE, and crops invalid edges.

(6) Output layer: Saves and visually displays the final panorama, supporting customizable output paths and display modes via command line parameters.

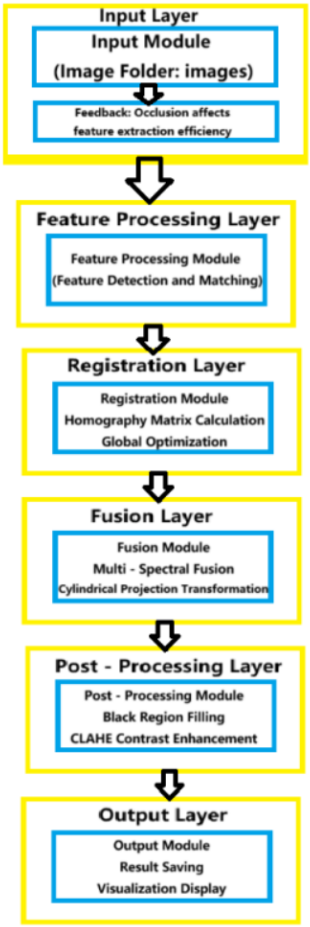

Figure 1. Framework diagram

3.2. Detailed design of feature extraction and matching module

Functional positioning: This module is responsible for extracting scale-invariant features from input images and establishing cross-image feature correspondences, laying the foundation for subsequent registration.

Technical implementation:

(1) Feature extraction: The SIFT algorithm detects key points and generates 128-dimensional descriptors. It achieves scale invariance through a Difference of Gaussians pyramid, exhibiting robustness to rotation, scaling, and illumination changes [5]. In the code, the detector is initialized via cv2.SIFT_create() and images are processed in batches using the detectAndCompute method.

(2) Feature matching: KNN matching combined with a ratio test (threshold of 0.75) is employed to filter matching points. Violent matching is implemented via cv2.BFMatcher. Only matching pairs with a ratio less than 0.75 between the distances of the nearest and second nearest neighbors are kept, reducing mismatches.

(3) Visualization and saving: Matching results are drawn using cv2.drawMatches, and the visualization images of the first 50 matching points are saved for debugging and verifying matching quality.

3.3. Detailed design of image registration module

Functional positioning: This module calculates the geometric transformation relationships between multiple images, aligning them to a unified coordinate system and addressing perspective differences between images.

Technical implementation:

(1) Local registration: For adjacent images, the RANSAC algorithm is used to solve the homography matrix, which requires at least 4 pairs of feature points. In the code, OpenCV's Stitcher class automatically handles homography matrix estimation.

(2) Global consistency optimization: A graph optimization strategy is employed to unify all homography matrices into a cylindrical coordinate system. This minimizes global registration errors and eliminates cumulative errors in multi-image stitching.

3.4. Detailed design of image fusion module

Functional positioning: Fuse registered images into a complete panorama, eliminating seams and optimizing visual continuity while repairing blank areas resulting from projection transformations.

Technical implementation:

(1) Laplacian pyramid-based image frequency domain decomposition: Wide weighting is applied to fuse the overall structure in the low-frequency layer, while narrow weighting preserves details in the high-frequency layer. This is implemented using cv2.detail_MultiBandBlender, which better balances exposure differences compared to simple linear weighting.

(2) Cylindrical projection: Maps the scene onto a cylindrical surface, reducing perspective distortion in the horizontal direction. This is suitable for wide-angle outdoor scenes. Projection is set using cv2.detail_CylindricalWarper, ensuring that architectural lines remain straight after stitching.

(3) Black area filling: Utilizes the Telea algorithm to repair blank areas after projection. Black regions are identified through threshold segmentation and filled in an order based on isophote lines, prioritizing pixels that contribute significantly to the structure and ensuring texture consistency.

3.5. Post-processing module design

Functional positioning: Optimize the visual quality of the panorama, enhancing contrast and cropping invalid edges.

Technical implementation:

(1) Contrast enhancement: CLAHE is employed, dividing the image into 8x8 blocks and limiting the contrast enhancement threshold (clipLimit = 3.0) for each block. This prevents noise amplification and enhances the layering of architectural details.

(2) Edge cropping: Contour detection is used to find the bounding rectangle of the valid area in the panorama, cropping away black edges to ensure the output image is complete and without redundancy.

3.6. Parameter configuration module design

Functional positioning: Provide a flexible user interaction interface supporting custom input/output paths, scaling factors, and other parameters.

Technical implementation:

(1) Command line parameters: argparse is used to parse parameters, supporting options such as --input to specify the image folder, --output to specify the save path, and --resize-factor to adjust the scaling ratio, meeting the needs of different scenarios.

(2) Logging system: The logging module records key operations like image loading and stitching status, facilitating debugging and error troubleshooting, and enhancing system maintainability.

4. Experimental steps

4.1. Experimental environment

(1) Hardware environment:

CPU: Intel Core i5-1135G7 2.4GHz Quad-Core Eight-Thread

Memory: 16GB DDR4 - Storage: 512GB SSD

Graphics Card: Integrated Intel Iris Xe Graphics

(2) Software environment:

Operating System: Windows 10 64-bit

Programming Language: Python 3.8.10

(3) Main libraries:

OpenCV 4.5.5 (Core stitching library supporting Stitcher class and cylindrical projection) - NumPy 1.21.2 (Numerical computation)

Matplotlib 3.4.3 (Result visualization)

tqdm 4.62.3 (Progress bar display)

argparse 1.4.0 (Command line parameter parsing)

(4) Development Environment: PyCharm 2021.3.2

4.2. Experimental data (Figure 2)

Photography scene: East Gate of Shaanxi University of Technology South Campus (including main building, gate columns, plaques, and surrounding environment)

Photography equipment: OnePlus ACE 2 Pro (rear 50MP main camera)

Photography method: Handheld camera rotated horizontally, capturing 5 overlapping images with an overlap rate of approximately 50–70%

Image format: JPEG, with a single original resolution of 1706x1279 pixels

File location: Relevant images (img1.jpg, etc.) placed in the “images” folder under the current directory.

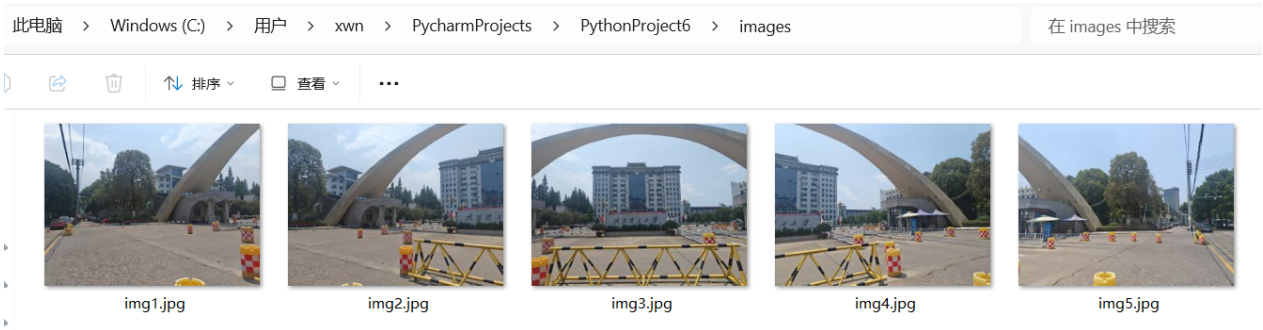

Figure 2. Experimental data

4.3. Experimental process

(1) Image preprocessing: Prepare 5 overlapping images of the east gate of the South Campus of Shaanxi University of Technology. Preprocess the images by using Gaussian filtering to denoise and utilizing AI functions to remove dynamic objects such as people. Resize the images to 0.5 times their original size using cv2.resize to reduce computational load and noise interference. For example, see **Figure 3** before and after processing.

Figure 3. Sample image of image preprocessing

(2) Image stitching processing: Extract feature points using the SIFT algorithm. Filter reliable matching pairs through KNN matching and ratio testing. Calculate the homography matrix using OpenCV's Stitcher class. Generate an initial panorama through cylindrical projection and multi-band blending, as shown in **Figure 4**.

Figure 4. Image stitching processing diagram

(3) Detail enhancement processing: Enhance the contrast of the panorama using the CLAHE algorithm. Divide the image into 8x8 blocks and limit the contrast enhancement threshold. Compensate for detail loss caused by stitching by overlaying high-frequency details through a Laplacian pyramid, as shown in **Figure 5**.

Figure 5. Detail enhancement processing diagram

(4) Black edge repair processing: Mark the black areas after projection through threshold segmentation. Fill these areas using the Telea algorithm in the order of isophote lines. Locate the effective area through contour detection and crop redundant black edges to output a complete panoramic image, as shown in **Figure 6**.

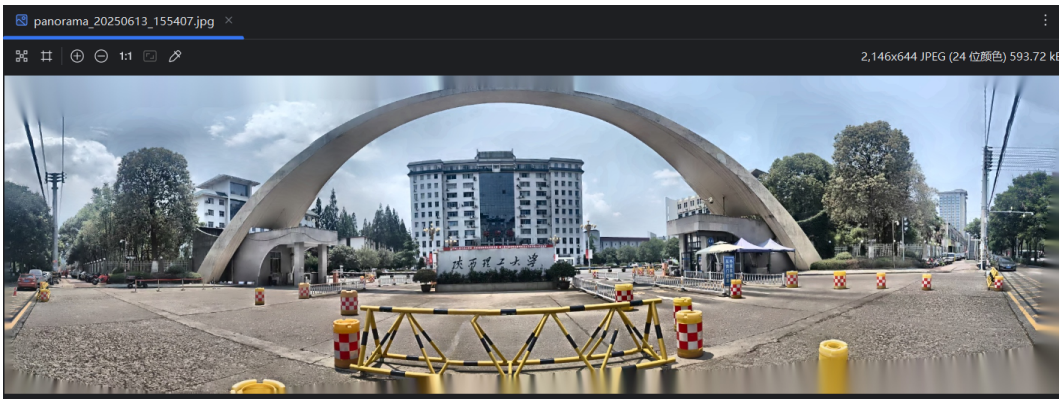

Figure 6. Black edge repair processing diagram

4.4. Experimental results and analysis

Experimental results are shown in **Figure 7**.

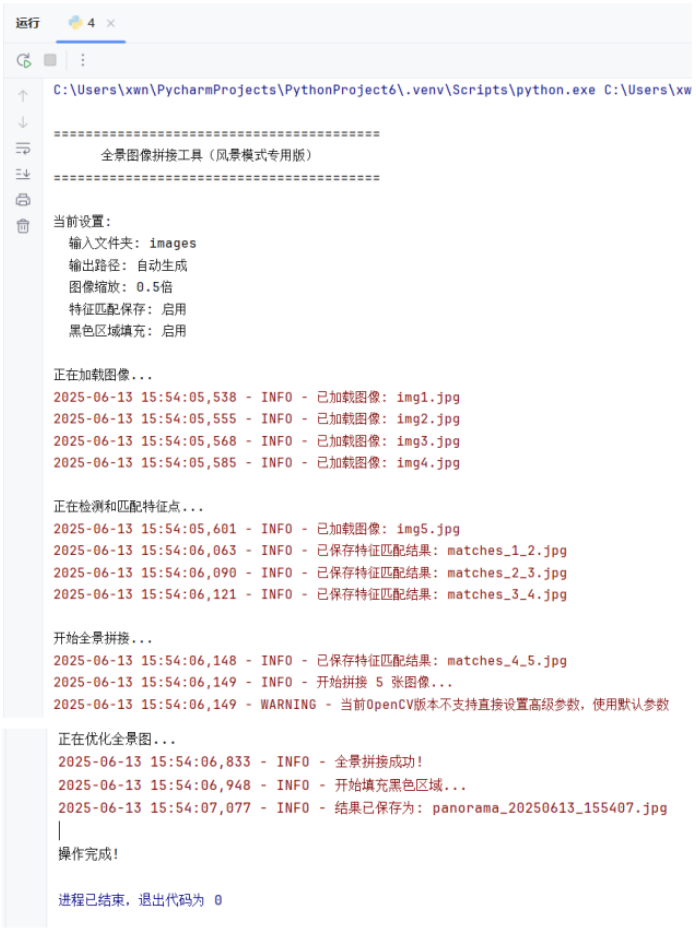

Figure 7. Code output result diagram

Feature matching results are shown in **Figures 8, 9, 10,** and **11**.

Figure 8. match_1_2.jpg

Figure 9. match_2_3.jpg

Figure 10. match_3_4.jpg

Figure 11. match_4_5.jpg

Final stitching result is shown in **Figure 12**.

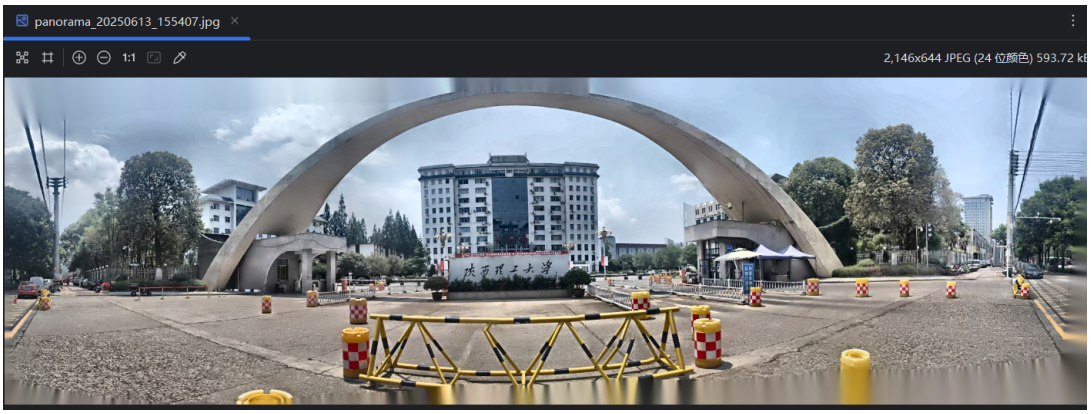

Figure 12. Final stitching result

Result analysis: The experimental results show that the stitched panorama has a size of 2153x645 pixels, fully presenting the main building, gate columns, and surrounding scenery of the east gate of the South Campus of Shaanxi University of Technology. Cylindrical projection effectively reduces horizontal perspective distortion, keeping architectural lines straight and continuous. Multi-band blending uses a Laplacian pyramid to layer blend overlapping regions, resulting in concealed stitching seams and natural texture transitions, balancing exposure differences. The Telea algorithm fills projection edges and, after cropping, leaves no visible black edges. CLAHE divides the image into 8x8 blocks and limits the contrast threshold, enhancing the layered texture of details such as brick patterns and plaque inscriptions. Combining this with the overlaying of high-frequency details through a Laplacian pyramid enhances edge sharpness, such as reliefs on gate columns and window grilles, compensating for blurring caused by downsampling.

Disclosure statement

The authors declare no conflict of interest.

References

[1] Hu J, Wang S, Yang M, 2025, Sparse Depth Feature Infrared Image Stitching Algorithm. Infrared Technology, 47(05): 584–590.

[2] Fu Z, Zhang X, Yu C, et al., 2020, Cylindrical Image Stitching Method Based on Fast Camera Calibration in Multiple Scenes. Opto-Electronic Engineering, 47(04): 74–86.

[3] Xiang Z, Wang Y, Yan X, 2025, Research on Field Crop Root Image Stitching Method Based on Improved SIFT Algorithm. Acta Agronomica Sinica, 1–17.

[4] Huang F, Lin S, 2019, Comparison of Multiband Image Fusion Rules Based on Laplacian Pyramid Transform Method. Infrared Technology, 41(01): 64–71.

[5] Luo Q, Xu W, Li Y, et al., 2025, Robust Stitching Method for Borehole Inner Wall Images under Multi-Interference Imaging Conditions. Chinese Journal of Liquid Crystals and Displays, 40(06): 895–904.

Appendix

Code (There are multiple versions of the code, only the final version is listed here):

```
import cv2
import os
import numpy as np
import argparse
import logging
from datetime import datetime
import matplotlib.pyplot as plt
from tqdm import tqdm

# 配置日志
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

def parse_arguments():
    """解析命令行参数"""
    parser = argparse.ArgumentParser(description='全景图像拼接工具（风景模式专用版）')
    parser.add_argument('--input', '-i', default='images', help='输入图像文件夹路径')
    parser.add_argument('--output', '-o', default=None, help='输出图像路径')
    parser.add_argument('--resize-factor', '-r', type=float, default=0.5, help='图像缩放因子')
    parser.add_argument('--show-progress', '-p', action='store_true', help='显示处理进度')
    parser.add_argument('--show-result', '-s', action='store_true', help='显示拼接结果')
    parser.add_argument('--save-matches', '-M', action='store_true', default=True,
                        help='保存特征匹配结果 ( 默认启用 )')
    parser.add_argument('--fill-black', '-f', action='store_false', default=True, help='禁用黑色区域填充')
    return parser.parse_args()

def load_images(image_folder, resize_factor=0.5, show_progress=False):
    """加载并预处理图像"""
    # 获取文件夹中所有图像文件
    image_files = [f for f in os.listdir(image_folder) if
                   f.lower().endswith(('.jpg', '.jpeg', '.png', '.bmp', '.tiff'))]
    if not image_files:
        logger.error(f"在 {image_folder} 中未找到图像文件")
        return None, None

    # 按文件名排序
    image_files.sort()

    # 创建进度条
    progress_bar = tqdm(image_files, desc="加载图像") if show_progress else image_files

    # 读取并调整所有图像大小
    images = []
    original_sizes = []
    for img_file in progress_bar:
        img_path = os.path.join(image_folder, img_file)
        try:
            img = cv2.imread(img_path)
            if img is not None:
                original_sizes.append(img.shape[:2])  # 保存原始尺寸
                # 调整图像大小以加快处理速度
                img = cv2.resize(img, (0, 0), fx=resize_factor, fy=resize_factor)
```

```
            images.append(img)
            logger.info(f"已加载图像：{img_file}")
        else:
            logger.warning(f"无法加载图像：{img_file}")
    except Exception as e:
        logger.error(f"加载图像 {img_file} 时出错：{str(e)}")

if len(images) < 2:
    logger.error("至少需要两张图像进行拼接")
    return None, None

return images, original_sizes

def detect_and_match_features(images, save_matches=True, show_progress=False):
    """检测特征点并进行匹配"""
    if len(images) < 2:
        return None

    # 创建特征检测器
    detector = cv2.SIFT_create()

    # 检测所有图像的特征点和描述符
    keypoints = []
    descriptors = []

    progress_bar = tqdm(range(len(images)), desc="检测特征点") if show_progress else range(len(images))

    for i in progress_bar:
        img = images[i]
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        kp, des = detector.detectAndCompute(gray, None)
        keypoints.append(kp)
        descriptors.append(des)

    # 创建特征匹配器
    matcher = cv2.BFMatcher()

    # 计算相邻图像之间的匹配
    matches = []

    for i in range(len(images) - 1):
        if descriptors[i] is not None and descriptors[i + 1] is not None:
            # 使用 KNN 匹配
            knn_matches = matcher.knnMatch(descriptors[i], descriptors[i + 1], k=2)

            # 应用比率测试筛选好的匹配
            good_matches = []
            for m, n in knn_matches:
                if m.distance < 0.75 * n.distance:
                    good_matches.append(m)

            matches.append(good_matches)

            if save_matches:
                # 可视化匹配结果
```

```
            img_matches = cv2.drawMatches(
                images[i], keypoints[i],
                images[i + 1], keypoints[i + 1],
                good_matches[:50], None,
                flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS
            )
            output_path = f"matches_{i + 1}_{i + 2}.jpg"
            cv2.imwrite(output_path, img_matches)
            logger.info(f"已保存特征匹配结果 : {output_path}")

    return matches

def stitch_images(images, show_progress=False):
    """拼接图像 ( 风景模式 )"""
    if len(images) < 2:
        logger.error("至少需要两张图像进行拼接")
        return None, -1

    logger.info(f"开始拼接 {len(images)} 张图像 ...")

    # 创建风景模式的全景拼接器
    stitcher = cv2.Stitcher.create(cv2.Stitcher_PANORAMA) if hasattr(cv2, 'Stitcher') else cv2.createStitcher(cv2.Stitcher_PANORAMA)

    # 尝试设置高级参数 , 兼容不同版本的 OpenCV
    try:
        # 设置柱面投影 , 适合风景全景
        stitcher.setWarper(cv2.detail_CylindricalWarper())
        # 设置多波段融合 , 减少拼接缝
        stitcher.setBlender(cv2.detail_MultiBandBlender())
        logger.info("已设置风景模式的高级拼接参数")
    except AttributeError:
        logger.warning("当前 OpenCV 版本不支持直接设置高级参数，使用默认参数")
    except Exception as e:
        logger.warning(f"设置高级参数时出错 : {str(e)}，使用默认参数")

    # 执行拼接
    status, result = stitcher.stitch(images)

    # 检查拼接是否成功
    if status == cv2.Stitcher_OK:
        logger.info("全景拼接成功 !")
        return result, status
    else:
        error_messages = {
            cv2.Stitcher_ERR_NEED_MORE_IMGS: "需要更多图像进行拼接",
            cv2.Stitcher_ERR_HOMOGRAPHY_EST_FAIL: "单应性矩阵估计失败",
            cv2.Stitcher_ERR_CAMERA_PARAMS_ADJUST_FAIL: "相机参数调整失败"
        }
        error_msg = error_messages.get(status, f"未知错误 ( 代码 : {status})")
        logger.error(f"拼接失败 : {error_msg}")
        logger.error("可能的原因 : 图像重叠不足、图像质量差或相机参数差异大")
        return None, status

def fill_black_regions(panorama, original_images=None):
```

```
    """填充全景图中的黑色区域"""
    if panorama is None:
        return None

    # 创建掩码，标记黑色区域
    gray = cv2.cvtColor(panorama, cv2.COLOR_BGR2GRAY)
    mask = np.zeros_like(gray)
    mask[gray < 10] = 255  # 黑色区域

    # 检查是否有需要填充的区域
    if np.sum(mask) == 0:
        logger.info("没有检测到黑色区域需要填充")
        return panorama

    logger.info("开始填充黑色区域 ...")

    # 使用 OpenCV 的 inpaint 算法填充黑色区域
    result = cv2.inpaint(panorama, mask, 3, cv2.INPAINT_TELEA)

    return result

def enhance_panorama(panorama, fill_black=True, original_images=None):
    """增强全景图质量"""
    if panorama is None:
        return None

    # 对比度增强
    lab = cv2.cvtColor(panorama, cv2.COLOR_BGR2LAB)
    l, a, b = cv2.split(lab)
    clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8, 8))
    cl = clahe.apply(l)
    limg = cv2.merge((cl, a, b))
    enhanced = cv2.cvtColor(limg, cv2.COLOR_LAB2BGR)

    # 填充黑色区域
    if fill_black:
        enhanced = fill_black_regions(enhanced, original_images)

    # 去除黑色边缘
    gray = cv2.cvtColor(enhanced, cv2.COLOR_BGR2GRAY)
    _, thresh = cv2.threshold(gray, 1, 255, cv2.THRESH_BINARY)
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    if contours:
        cnt = max(contours, key=cv2.contourArea)
        x, y, w, h = cv2.boundingRect(cnt)
        enhanced = enhanced[y:y + h, x:x + w]

    return enhanced

def save_and_display_result(result, output_path=None, show_result=False):
    """保存并显示结果"""
    if result is None:
        return

    # 如果未指定输出路径，使用当前时间戳
```

```
    if output_path is None:
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        output_path = f"panorama_{timestamp}.jpg"

    # 保存结果
    cv2.imwrite(output_path, result)
    logger.info(f"结果已保存为：{output_path}")

    # 显示结果
    if show_result:
        plt.figure(figsize=(12, 8))
        plt.imshow(cv2.cvtColor(result, cv2.COLOR_BGR2RGB))
        plt.title("全景拼接结果")
        plt.axis('off')
        plt.tight_layout()
        plt.show()

def main():
    """主函数"""
    print("\n" + "=" * 60)
    print("    全景图像拼接工具（风景模式专用版）")
    print("=" * 60)

    # 解析命令行参数
    args = parse_arguments()

    # 显示当前设置
    print(f"\n 当前设置：")
    print(f"  输入文件夹：{args.input}")
    print(f"  输出路径：{args.output or '自动生成'}")
    print(f"  图像缩放：{args.resize_factor} 倍")
    print(f"  特征匹配保存：{'启用' if args.save_matches else '禁用'}")
    print(f"  黑色区域填充：{'启用' if args.fill_black else '禁用'}")

    # 检查输入文件夹是否存在
    if not os.path.exists(args.input):
        logger.error(f"输入文件夹不存在：{args.input}")
        return

    # 加载图像
    print("\n 正在加载图像 ...")
    images, original_sizes = load_images(args.input, args.resize_factor, args.show_progress)
    if images is None:
        return

    # 检测并匹配特征点
    print("\n 正在检测和匹配特征点 ...")
    detect_and_match_features(images, args.save_matches, args.show_progress)

    # 执行拼接（风景模式）
    print("\n 开始全景拼接 ...")
    result, status = stitch_images(images, args.show_progress)

    # 如果拼接成功，增强结果
    if status == cv2.Stitcher_OK:
```

```
        print("\n 正在优化全景图 ...")
        enhanced_result = enhance_panorama(result, args.fill_black, images)
        save_and_display_result(enhanced_result, args.output, args.show_result)
        print("\n 操作完成 !")
    else:
        print("\n 拼接失败，请查看日志信息了解详情。")

if __name__ == "__main__":
    main()
```