High-Low-Resolution-Blending

Avinoam Nukrai

Introduction

This project focuses on developing an algorithm to seamlessly blend a low-resolution image with a corresponding high-resolution part. By leveraging techniques such as feature extraction, feature matching, RANSAC, and image warping, the algorithm aims to align and blend the images effectively. This task is essential for applications like image enhancement, restoration, and super-resolution. By applying concepts learned in class, such as feature extraction and matching, the project demonstrates the practical application of these techniques in real-world scenarios.

Algorithm

The main algorithm for blending low-resolution and high-resolution images orchestrates a series of steps, each facilitated by sub-algorithms, to seamlessly merge the two images. Here's an overview of how the main algorithm utilizes the sub-algorithms:

Step 1 - Feature Extraction:

- Explanation: Feature extraction is performed using the Scale-Invariant Feature Transform (SIFT) algorithm. SIFT detects distinctive keypoints in images and computes descriptors to characterize each keypoint's appearance.
- Inputs: Low-res img, high-res part img, contrastThreshold and edgeThreshold.
- Outputs: Keypoints and descriptors for both images.

Step 2 - Feature Matching:

- Explanation: In feature matching, keypoints from both low-resolution and high-resolution images are matched using FLANN for efficient nearest neighbor search. Lowe's ratio test is then applied to filter unreliable matches based on distance ratios between the closest and second-closest matches.
- Inputs: Keypoints and descriptors for both images, algorithm, trees, checks and k.
- Outputs: Good matches between keypoints.

Step 3 - Perspective Correction:

- Explanation: Perspective correction aligns the low-resolution image with the high-resolution part by estimating a transformation matrix. This matrix is computed using the RANSAC algorithm, which handles outliers and inaccuracies in feature matching to ensure precise alignment.

- Inputs: cv2.RANSAC, src_pts, dst_pts, ransacReprojThreshold, high-resolution image, homography matrix, dimensions and cv2.INTER_LINEAR flag.
- Outputs: Corrected high-resolution image.

Step 4 - Image Blending:

- Explanation: After computing the perspective transformation matrix, we align the low-resolution image with the high-resolution part and blend them together. A binary mask is created from the alpha channel of the high-resolution part image to selectively blend pixels from both images, ensuring smooth transitions.
- Inputs: Corrected high-resolution part image, low-resolution image.
- Outputs: Blended image.

In summary, the main algorithm coordinates sub-algorithms to blend low-resolution and high-resolution images effectively, resulting in a visually appealing composite image. Each sub-algorithm contributes to different stages of the process, ensuring successful blending.

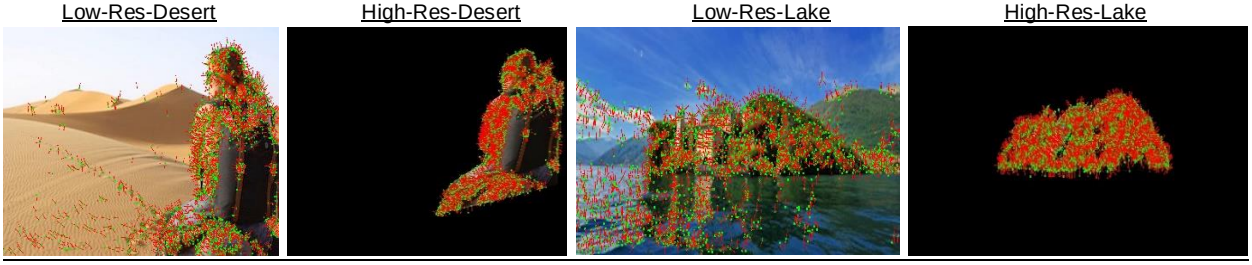
Implementation Details

We first enter to the blend_images() function which takes the two given images, low resolution and high resolution, this function first calculates the correct perspective of the high resolution image, for being able to blend it correctly with the low resolution after wards, for doing this correction she's enter to the correct_perspective() function which gets the 2 images and dose the following 1-3 steps:

Step1 - Feature Extraction:

- Keypoints and descriptors are extracted from both the low-resolution and high-resolution part images using the Scale-Invariant Feature Transform (SIFT) algorithm provided by OpenCV (cv2.SIFT_create(), sift.detectAndCompute()) I set also contrastThreshold=0.04 specifies the minimum contrast required for a keypoint to be detected, while edgeThreshold=17 determines the sensitivity to edges, with higher values resulting in fewer keypoints near edges.
- Visualization (Keypoints + Direction (part of the descriptor of each keypoint)):

Low-Res-Desert | High-Res-Desert | Low-Res-Lake | High-Res-Lake



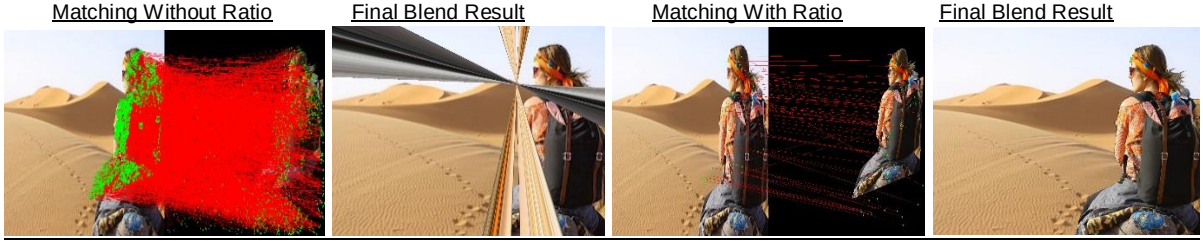
Step 2 - Feature Matching:

- Initializing FLANN: I first set up FLANN parameters. flann_index_kdtree indicates the algorithm to be used (in this case, a KD tree). index_params and

search_params are dictionaries specifying parameters for building the index and searching for nearest neighbors, respectively. trees=5 in index_params sets the number of trees in the index, while checks=20 in search_params determines the number of times to search in each tree.

- Feature Matching: FLANN's knnMatch function is called to find the k nearest neighbors (in this case, k=2) for each descriptor in the two images. This generates a list of potential matches.
- Lowe's Ratio Test: Lowe's ratio test is applied to filter out unreliable matches. For each pair of potential matches, if the distance ratio of the closest neighbor to the second-closest neighbor is less than 0.6, the match is considered good and added to the good_matches list.
- Extracting Matched Keypoints: We extract the matched keypoints from both images. This involves retrieving the coordinates of the matched keypoints, which are necessary for subsequent processing steps.
- Ensuring Sufficient Matches: Finally, we check if there are enough good matches for RANSAC. If the number of good matches is less than 4 (a minimum requirement for RANSAC), an error is raised as there are insufficient matches for robust perspective transformation estimation.
- Visualizations (With Lowe's ratio ves without Lowe's ratio): We will notice the huge difference between filtering the keypints by Lowes ratio and not filtering them, without the ratio we get a huge number of points that a lot of them considered as unreliable which causes our result to be really distorted, on the other hand after running the ratio we get a good enough filtering for a smaller number of points and then our result is perfect.

Matching Without Ratio | Final Blend Result | Matching With Ratio | Final Blend Result



Step 3 - Perspective Correction:

- Find The Homography: We using cv2.findHomography() to estimate a perspective transformation matrix (M) aligning keypoints from the low-resolution image to the high-resolution part. We sending the src_pts, des_pts and choosing cv2.RANSAC as the method for computing this matrix, ensuring robustness to outliers. The ransacReprojThreshold parameter is set to 5.0, defining the maximum allowed reprojection error for inliers, balancing accuracy and robustness.
- Warping High Resolution Inage: We sends the high resolution image to correct its perspective using a provided perspective transformation matrix (M), wanted dimensions and the cv2.INTER_LINEAR flag that specifies bilinear interpolation

for smoother transformation. The resulting image (corrected_image) matches the perspective of the low_resolution image, as wanted.

- Visualizations:

Before Warping | After Warping | Before Warping | After Warping

After those 3 steps, the blend_image() function is starting to produce the actual blended image, doing Step 4 as follows:

Step 4 - Image Blending:

- Creating Binary Mask: The alpha channel of the high-resolution (png) part image is thresholded to create a binary mask. Pixels with alpha values greater/smaller then 128 will get W/B color. using cv2.threshold() with cv2.THRESH_BINARY flag.
- Inverting Binary Mask: The binary mask is inverted using cv2.bitwise_not() to obtain. This inversion swaps the black and white regions of the mask.
- Resizing Low-Resolution Image: The low-resolution image is resized to match the dimensions of the high-resolution part image using cv2.resize().
- Blending Images: the blending is performed using bitwise operations as follows:
    - The resized low-resolution image is combined with itself using cv2.bitwise_and() and the inverted mask (inverted_mask). This effectively removes the areas where the high-resolution part image will be overlaid.
    - The high-resolution part of the image (excluding the alpha channel) is combined with itself using cv2.bitwise_and() and the original binary mask (binary_mask). This retains the high-resolution part image's content where it will be overlaid on the low-resolution image.
    - These two masked images are then added together using element-wise addition (+=) to obtain the final blended image (blended_img).

More Details (Hyper-Parameters, Thresholds, Choices and Challenges)

Hyper-Parameters & Thresholds: The hyper-parameters and thresholds used in the algorithm were carefully chosen to optimize key aspects of the image processing pipeline. For instance, in the cv2.SIFT_create() function, contrastThreshold and edgeThreshold were set to 0.04 and 17, respectively. These values control keypoint detection sensitivity based on contrast and edge intensity difference, crucial for accurate feature matching. Similarly, in cv2.FlannBasedMatcher(), trees and checks were set to 5 and 20, respectively, balancing accuracy and efficiency in nearest neighbor search. The ransacReprojThreshold parameter in cv2.findHomography() was set to 5 to balance outlier rejection and transformation accuracy. Additionally, the thresh parameter in cv2.threshold() was set to 128 to control the binary mask's sensitivity to image details.

After extensive experimentation, these parameters were chosen for their ability to produce the most accurate results.

Choices & Challenges: During implementation, various challenges were encountered and addressed. Initially, understanding the algorithm's structure and identifying the necessary building blocks posed a challenge. However, leveraging the functionalities provided by the cv2 library significantly expedited development and minimized bugs. Precise warping of high-resolution images presented another challenge, often resulting in distorted outputs. Adjusting parameters, particularly in cv2.warpPrespective(), helped achieve smoother results. Insufficient matching keypoints between images hindered progress to RANSAC calculations in some cases. By adjusting the threshold for good matches and the edgeThreshold in SIFT, more keypoints were detected, resolving the issue. Additionally, refining the algorithm's performance for specific images, such as blurring the high-resolution image before processing, further improved accuracy and smoothness, leading to satisfactory outcomes. Overall, iterative refinement and parameter tuning addressed implementation challenges and enhanced algorithm performance.

Visual Results

Throughout the report, various visualizations of the sub-algorithms were presented, along with additional visualizations and final results. While running the algorithm, it was observed that the image of the desert yielded satisfactory results, whereas the image of the lake showed some imprecision in blending (e.g., noticeable double roofs). To address this, a pre-processing step of blurring the high-resolution image before blending was introduced. This slightly blurred the final result, leading to a smoother blending effect. Despite efforts to mitigate artifacts like the thin black band around the attachment of the woman on the camel by blurring edges, complete elimination was not achieved. Here are the final outcomes of the project.

Final Desert Blended Image

Lake Non-Blurring Blended Image

Final Lake Blended Image

Conclusion

In this project I learned a lot about advanced pasting of photos, about combining photos and matching them to each other (not a trivial and simple task at all), I learned to use significant tools of CV2 and I also had a lot of fun and was even a little proud of my results.