Çağrı Çakıroğlu

December, 2023

AIN433 ASSIGNMENT3

Panorama Image Stitching with Keypoint Descriptors

1-Feature Extraction

1. Initialization of Feature Detector

A feature detector is initialized using the SIFT algorithm. In computer vision, SIFT is widely used for extracting distinctive features from images which can be used to perform reliable matching between different views of an object or scene. Here, cv2.SIFT_create() is called to create a SIFT feature detector object. This object will be used to detect keypoints and compute descriptors in the given image.

-

2. Feature Detection and Computation

The actual detection of keypoints and computation of descriptors in the grayscale image is carried out. The method detectAndCompute is invoked on the feature_detector object, with the grayscale version of the image (gray_image) as the input. This method finds keypoints in the image and computes descriptors for each keypoint. Keypoints are points of interest in the image, and descriptors are vectors that uniquely describe the keypoints. The time taken for this process is measured using time.time() before and after the detection process, helping in assessing the performance of the feature extraction.

3. Drawing Keypoints on the Original Image

The keypoints detected by the SIFT algorithm are then drawn on the original image. This is achieved using the cv2.drawKeypoints function. The keypoints are marked with green circles color=(0,255,0)), and cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS flag is used to draw the circles with their size representing the scale and the orientation of the keypoint.

4. Updating the Feature Points Plot

Finally, the feature points plot is updated. If no existing plot is provided (existing_feature_plot is None), the updated plot is just the original image with keypoints. However, if there is an existing plot, the new keypoints image is concatenated with this existing plot along the horizontal axis (axis=1). This updated or merged image is then displayed using a function (presumably display_feature_points), showing the feature points from the current subset of the image, identified by subset_name.

2- Feature Matching

1. Preparing Descriptors

The descriptors of the current (cur_descs) and next (next_descs) images are ensured to be in the float32 format. This step is crucial as the feature matching algorithm requires the descriptors to be in a specific data type for accurate computation. The conversion is performed only if the existing data type of the descriptors is not float32.

2. Checking for Sufficient Descriptors

A conditional check is performed to ensure that both the current and next images have more than a specified number of descriptors (NEAREST_NEIGHBOR_NUM). This number represents the minimum required for effective feature matching. If either image lacks sufficient descriptors, the function proceeds to concatenate the images and save them without performing the matching process.

3. Creating the FLANN Matcher and Finding Matches

If sufficient descriptors are available, a FLANN (Fast Library for Approximate Nearest Neighbors) based matcher is created using cv2.FlannBasedMatcher(). FLANN is utilized for its efficiency in handling large datasets. The matcher then finds the k-nearest neighbors for each descriptor, where k is specified by NEAREST_NEIGHBOR_NUM.

4. Applying the Ratio Test

A ratio test is applied to the matches to filter out the good ones. This test compares the distance of the nearest neighbor to that of the second-nearest neighbor. If the distance of the nearest neighbor is less than 70% (0.7) of the second-nearest, the match is considered good. This step is critical for eliminating false matches and retaining only those that are most likely to be correct.

5. Drawing and Saving the Matches

If good matches are found, they are drawn on the images, and the resultant image is saved. This is done using a function (presumably draw_matches_and_save) which visually represents how points in one image correspond to points in the other image. If no good matches are found, or if the images lack sufficient descriptors, the two images are simply concatenated and saved using another function (presumably concatenate_images_and_save).

3- Finding Homography

1. execute_RANSAC Function

This function employs the RANSAC (Random Sample Consensus) algorithm to robustly estimate a homography matrix between matched feature points (match_pairs) from two images.

- Random Match Selection and Homography Calculation: In each iteration, four random matches are selected from match_pairs. Then, a homography matrix (H) is computed using these matches via the find_homography function.
- Inlier Calculation: The number of inliers is determined by computing the geometric distance for each match pair with respect to the homography matrix. A match is considered an inlier if this distance is less than a predefined threshold (INLIER_THRESH).
- Best Homography Matrix Selection: The homography matrix that results in the highest number of inliers is updated as the best homography matrix (best_H). The process iterates for a predefined number of times (RANSAC_ITERATION) or stops early if the number of inliers exceeds a certain threshold (RANSAC_THRESH).

2. find_homography Function

This function calculates the homography matrix for a given set of match pairs.

- Matrix Construction: For each pair of matched points, a set of linear equations is constructed based on the homography transformation properties. These equations are aggregated into a matrix (matrixA).
- Singular Value Decomposition (SVD): The SVD of matrixA is computed. The homography matrix (H) is then derived from the last row of the matrix vh (which is the result of the SVD), and normalized so that its last element is 1.

3. compute_homographies Function

This function computes homographies between consecutive images in a subset.

- Image Processing: Consecutive images are read and converted to grayscale.
- Homography Computation: The stitch_images function (not detailed here) is presumably called to compute the homography matrix for each pair of consecutive images. This function likely uses feature extraction, feature matching, and the execute_RANSAC function to find the homography.
- Homography Matrix Storage: The computed homography matrices are stored in a list (homographies). If the homography matrix for a pair is not found, the function returns the computed homographies up to that point.

4-Merging by Transformation.

1. finalize_and_store_panorama Function

This function is responsible for the final enhancement and storage of the panorama image.

- Brightness Enhancement: The brightness of the final panorama is enhanced using the enhance_image_brightness function. This function adjusts the brightness of the image to make it visually more appealing.
- Saving the Panorama: After enhancement, the panorama image is saved to a file with a specific name related to the subset using the save_panorama function.

2. enhance_image_brightness Function

This function enhances the brightness of an image.

- Color Space Conversion: The image's color space is converted from BGR to HSV. HSV (Hue, Saturation, Value) is often used for color and brightness adjustments.
- Brightness Adjustment: The value channel (V) of HSV is adjusted by multiplying it with a BRIGHTNESS_FACTOR and then clipped to ensure the values remain within the 0-255 range. This step increases the brightness of the image.
- Conversion Back to BGR: The modified HSV image is converted back to BGR color space for standard image format.

3. save_panorama Function

This function saves the enhanced panorama image in a specific folder.

- Folder Path Creation: A path for the subset folder is created using os.path.join. If the folder does not exist, it is created using os.makedirs.
- Saving the Image: The image is saved in the subset folder with a filename "result.png" using cv2.imwrite.

4. merge_images Function

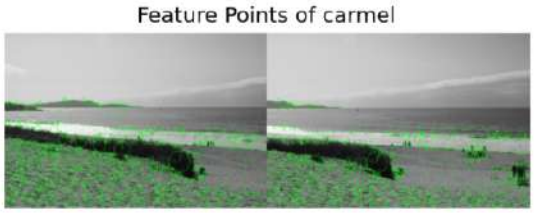
This function merges two images into a panorama using a homography matrix (H).

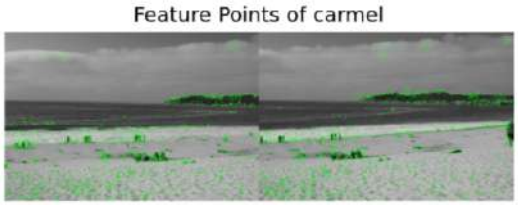
- Image Transformation: The next_image is transformed using the homography matrix. This involves calculating new indices for each pixel in the next_image using np.dot with the homography matrix.
- Brightness Enhancement: The transformed image's brightness is enhanced by multiplying with a brightness factor and clipping the values.
- Merging Images: The function creates a mask to identify non-black pixels in cur_image. These non-black pixels are then copied into the corresponding locations in the transformed next_image.
- Cropping the Image: The merged image is cropped using crop_image to remove black borders resulting from the transformation.
- Saving the Merged Image: The merged image is saved using a function (presumably save_image).

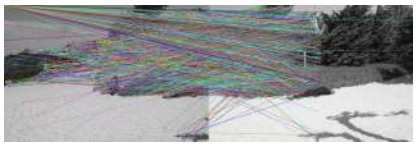
5. crop_image and transpose_and_copy Functions

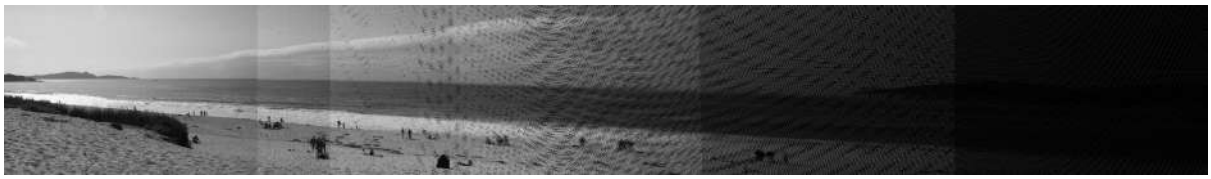
- Cropping: crop_image removes black borders from the image by recursively cropping rows and columns that consist entirely of black pixels.
- Transposing and Copying: transpose_and_copy transposes image data (swaps the first two dimensions). If the data type is 'matrix', the transposed data is converted to an unsigned 8-bit integer format.

CARMEL



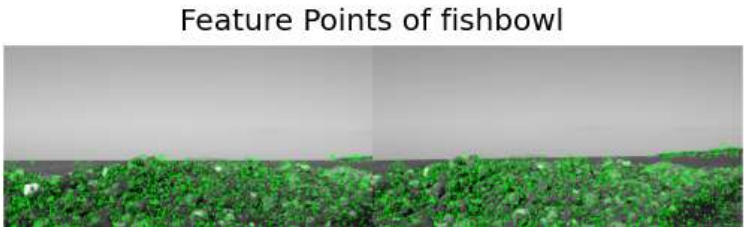


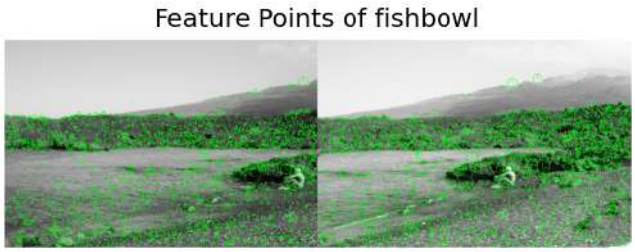


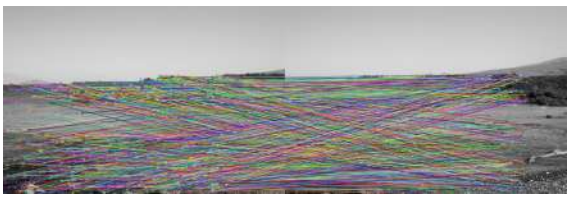


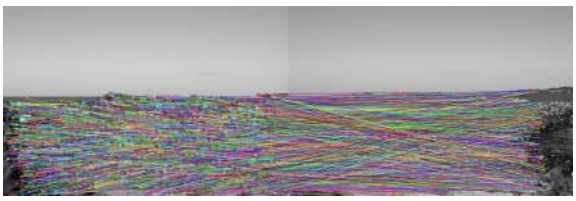
RESULT

FISHBOWL



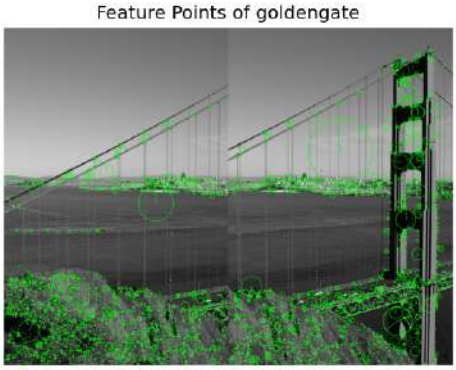


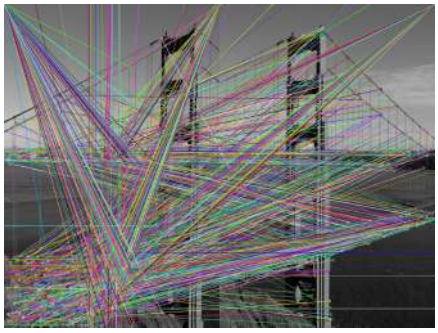


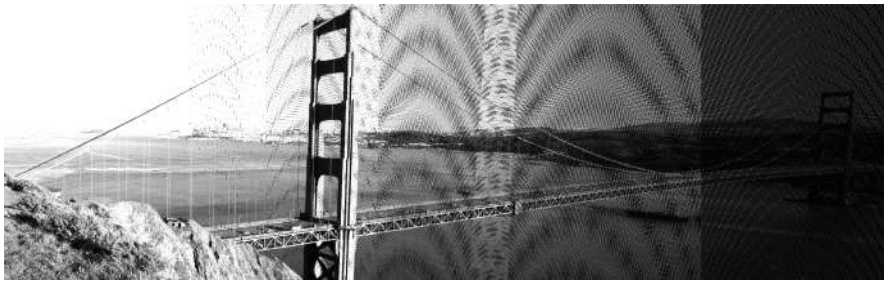


FISHBOWL RESULT

Golden gate








Feature Points of hotel


Feature Points of hotel


Feature Points of yard

AVG RUNTIME FOR SIFT
FOR EACH CLASS
0.07854998812955968
0.06900426745414734
0.056888580322265625
0.1013862235205514
0.10415752232074738

During feature extraction, interest points on each image were identified. These points are significant because they represent distinctive areas that can be easily matched across different images. Following this, feature matching was employed to find corresponding points between images, which is a crucial step in aligning images in preparation for merging.

The process of finding homography involves computing the perspective transformation that aligns each image with a common viewpoint. This transformation is critical as it allows for the seamless blending of images by accounting for the differences in perspective, camera motion, and scene structure.

However, it seems that there were challenges encountered in the merging step. The resultant panorama images exhibit artifacts and distortions, which indicate issues during the image alignment and blending phases