DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

COMPUTER VISION 2023 - LAB 4
REPORT

FERIDUN CEMRE GÜLTEN

ID: 2080660

1) Introduction

The task is a patch-based image completion algorithm using Scale-Invariant Feature Transform (SIFT) features and the Random Sample Consensus (RANSAC) algorithm. Also different methods and approaches were tried and results tried to be obtained conveniently. The goal is to replace corrupted regions in an image with patches extracted. There are **optional choices** after running the code to select different tasks .As mentioned , different approaches and methods were tried but here is the code review of the baseline assignment and the steps were performed:

Code Overview:

1. Loading and Displaying the Corrupted Image:
    - The code starts by reading and displaying the corrupted image specified by the file path.
    - The OpenCV **imread()** and **imshow()** functions are used for image loading and display, respectively.
2. Loading Patch Images:
    - The code loads patch images that will be used to replace the corrupted regions.
    - The file path to the patch images is specified, and the OpenCV **glob()** function is used to retrieve the file names.
3. Extracting SIFT Features from the Corrupted Image:
    - SIFT feature extraction is performed on the corrupted image using the **SIFT::create()** function.
    - Detected keypoints and computed descriptors are stored in **keypoints_image** and **descriptors_image**, respectively.
    - The keypoints are visualized by drawing them on the corrupted image using the **drawKeypoints()** function.
4. Matching Patches with the Corrupted Image:
    - SIFT features are extracted from each patch image.
    - The Brute-Force Matcher with L2 norm is created using **BFMatcher**.
    - Matches between the patch descriptors and the image descriptors are obtained using **knnMatch()**.
    - Matches are refined by applying a threshold ratio to filter out less reliable matches.
5. Estimating Transformation using RANSAC:
    - RANSAC is used to estimate the transformation between each patch and the image.
    - Inlier matches are identified using the **findHomography()** function with RANSAC.
    - Inlier matches are drawn on the image using the **drawMatches()**

6. Overlaying Patch on the Image:
    - The patch is warped onto the image using the estimated transformation matrix.
    - The corrupted region in the image is replaced with the overlayed patch using a binary mask.
    - The overlayed patch is added to the image to complete the region.
7. Visualization of Results:
    - The result, with the completed image, is displayed.

Conclusion: The implemented code showcases a patch-based image completion approach using SIFT features and RANSAC. By matching patches with the corrupted image, estimating the transformations, and overlaying the patches, the code provides a way to complete the corrupted regions. The visualization steps help understand the matching process, the identified inlier matches, and the final completed image.

2) Results

2.1) Baseline Assignment

2.1.1) Detecting Keypoints

Figure 1. Image with Keypoints

This image serves as a visual representation of the keypoints detected in the corrupted image, allowing to inspect and analyze the distribution of keypoints before proceeding with the patch-based image completion algorithm.

2.1.2) Images with Matches

1. Image with Inlier Matches:
    - The "Image with Inliers" displays the corrupted image along with the patch, emphasizing the inlier matches between them.
    - The inlier matches are the subset of matches that are estimated to correspond to the same transformation between the patch and the image.
    - The inlier matches are determined using the RANSAC (Random Sample Consensus) algorithm.
    - These matches are considered to be the most reliable matches for estimating the transformation between the patch and the corrupted image.
2. Visualization:
    - The cv::drawMatches() function is used to draw the inlier matches on the image.
    - The inlier matches are typically represented by lines or other visual indicators connecting the matching keypoints.
    - The visualization helps to identify the keypoints that are considered to be inliers and contributes to understanding the alignment between the patch and the corrupted image.

By displaying the "Image with Inliers," you can observe the inlier matches that have passed the RANSAC algorithm's threshold and are determined to be consistent with a particular transformation. This visualization allows you to assess the quality and accuracy of the inlier matches, which are crucial for estimating the transformation accurately.

The "Image with Inliers" provides insights into the success of the RANSAC algorithm in identifying reliable matches and estimating the transformation between the patch and the corrupted image. It helps in evaluating the effectiveness of the matching process and determining whether the selected matches align well with the expected transformation.

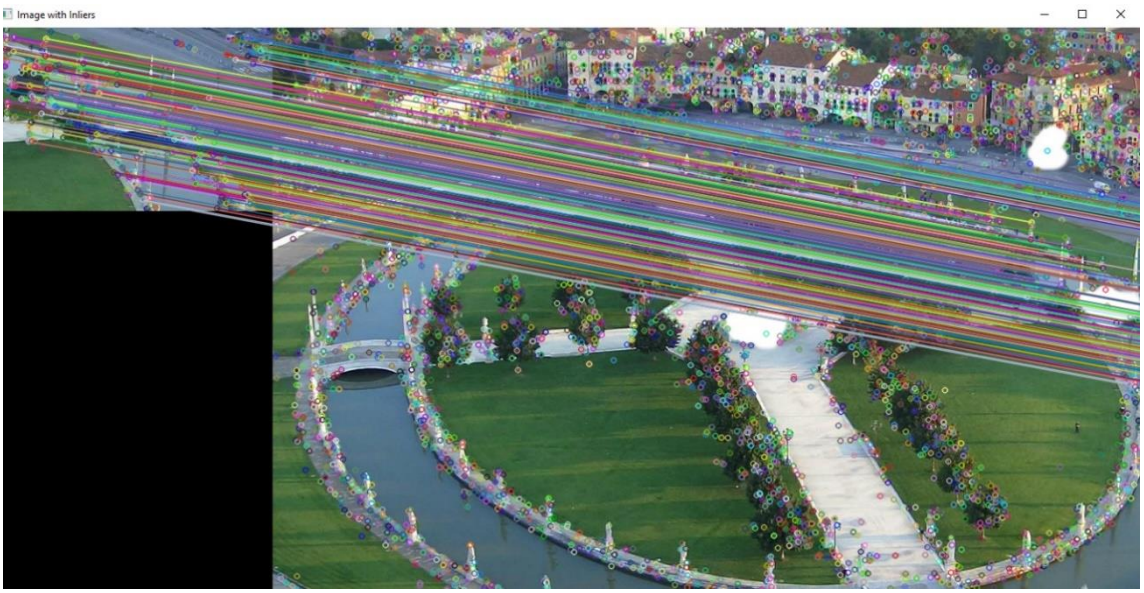


Figure 2. Image with matches for 1st patch

Figure 3. Image with matches for 2nd patch

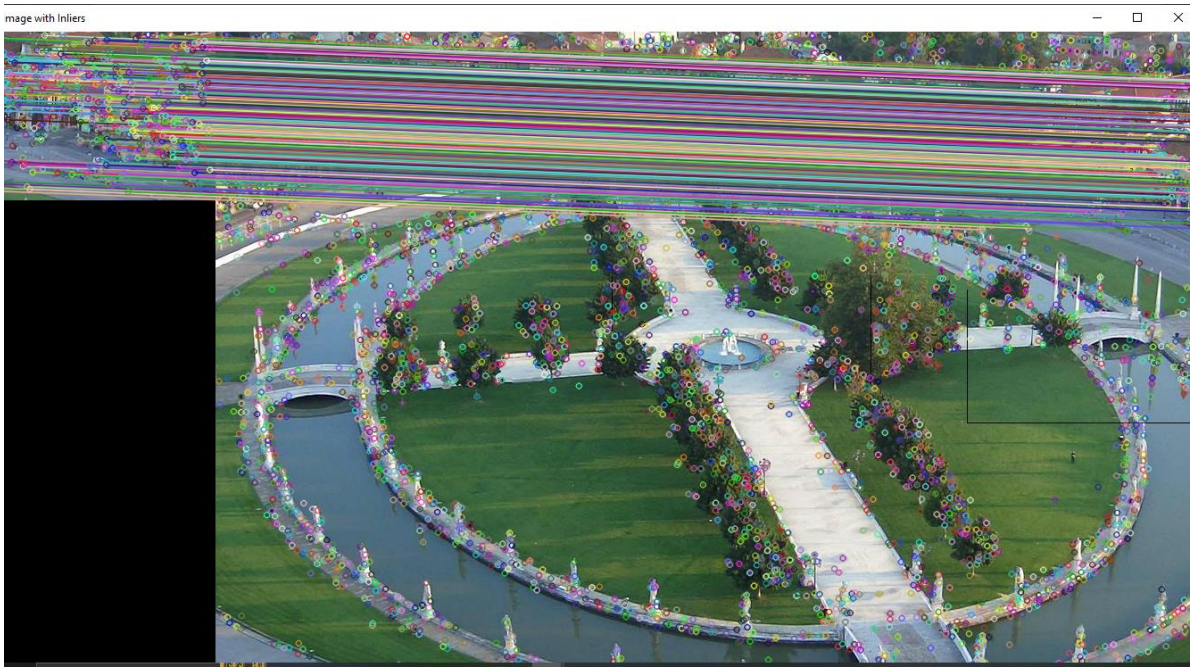


Figure 4. Image with matches for 3rd patch

Figure 5. Final Image fully overlayed

2.2) Optional Step (Manual RANSAC & Affine Estimation)

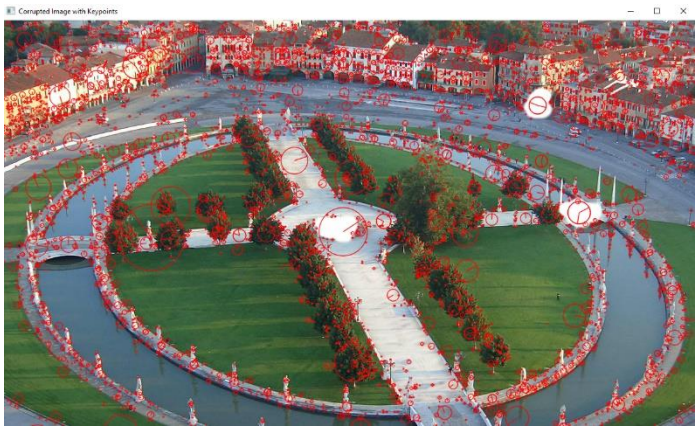
Code Overview:

1. The function starts by displaying a message indicating the execution of **Main Function 2**.
2. It then loads a corrupted image using the OpenCV library and displays it using the **imshow** function. The program waits for a key press before proceeding.
3. Next, the code specifies the path to the patches covering the corrupted regions. It uses the **cv::glob** function to retrieve the file paths of the patches and stores them in a vector.
4. SIFT (Scale-Invariant Feature Transform) features are extracted from the corrupted image using the **cv::SIFT** class provided by OpenCV. The keypoints and descriptors of the image are computed and stored.
5. The image is then displayed with the extracted keypoints marked using the **drawKeypoints** function.
6. A brute-force matcher is created using **cv::BFMatcher**. This matcher will be used to match the SIFT features between the patches and the image.
7. A ratio threshold is set to filter out unreliable matches during the matching process.
8. For each patch, SIFT features are extracted using the same process as for the image.
9. The matcher is used to find matches between the patch descriptors and the image descriptors. The matches are then refined based on the ratio threshold.

10. RANSAC (Random Sample Consensus) is employed to estimate an affine transform that aligns the patch with the corresponding region in the image. It randomly selects three matches and uses them to compute the transform.
11. The inlier count is calculated by applying the estimated transform to the patch keypoints and comparing them with the corresponding keypoints in the image. A distance threshold is used to determine whether a match is an inlier.
12. The best model (transform) and its associated inlier count are updated if the current iteration produces more inliers than the previous best.
13. The selected patch is then overlaid onto the image using the best model. The corrupted region is replaced by the overlayed patch.
14. The overlayed image is added to the corrupted image to accumulate the patches.
15. The image with the overlayed patch is displayed, and the program waits for a key press.
16. Steps 8-15 are repeated for each patch in the collection.
17. Finally, the completed image is displayed, representing the final result of the image restoration process.

This optional step is a patch-based image completion technique using SIFT features and RANSAC to align patches with a corrupted image. It iteratively selects patches, matches them to the image, estimates transforms, and overlays them onto the corrupted regions. The process continues until all patches have been processed, resulting in the restoration of the corrupted image.

2.2.1) Corrupted image and Keypoints



2.2.2) Images with Overlayed Patches

2.3) Optional Step (Template Matching)

Code Overview:

1. A copy of the corrupted image is created and stored in the resultImage variable. This image will be modified to replace the corrupted regions with the matched patches.
2. For each patch path in the collection, the code performs template matching between the corrupted image and the patch using the cv::matchTemplate function. The resulting correlation map, result, is obtained.
3. The minimum and maximum values in the correlation map are calculated using the cv::minMaxLoc function. The maximum value and its corresponding location (top-left corner of the matched region) are stored.
4. A rectangle is drawn on the resultImage to indicate the matched region using the cv::rectangle function. The rectangle is defined by the top-left corner (maxLoc) and the size of the patch.
5. The matched patch is copied onto the resultImage by replacing the corresponding region defined by the match rectangle. This is achieved by using the copyTo function.
6. The image with the matched patches is displayed using imshow, and the program waits for a key press.

7. The function then returns 0 to indicate successful execution.

In summary, it implements a template matching approach to restore a corrupted image. It iterates over a collection of patches, performs template matching, identifies the best match for each patch, and replaces the corrupted region in the resultImage with the matched patch. The resulting image displays the original image with the patches successfully matched and placed in their corresponding regions.

2.3.1) Matching Results

Each red rectangle corresponds to a patch that has been successfully matched to a region in the corrupted image.During the template matching process, the code compares each patch with the corrupted image using correlation analysis. The highest correlation score is found, indicating the best match.