COMPUTER VISION FINAL PROJECT
FIXING CORRUPTED IMAGE WITH FEATURE MATCHING
ÖMER CEM TABAR - 2081169

1. INTRODUCTION

Aim of this project is to fix the corrupted images which contain missing parts with given patches. Programming language of this project is C++ and the main library used for this project is OpenCV (version 4.7.0). There is one corrupted image and 8 corresponding patches for the missing parts. In the report you can find the detailed explanation about the steps and result that is obtained.

2. STEPS IN DETAILS

- **STEP 1:** For the first step, I created a function that handles the whole image reading process. Images() function does not take any argument. Inside the function I am mainly defining the paths of the corrupted image and patches, and reading them to separate matrices. For simplicity, I collected these image matrices into another matrix vector and returned it for later usages. Also there exists another function that is implemented called imageAndPatchShow() which takes the images vector as an argument and displays the images on the screen.

- **STEP 2:** After I managed to read all required images, I extracted the SIFT features (keypoints and descriptors) for the corrupted image i.e image to complete. For this purpose, since we are going to use the SIFT object multiple times, I implemented a function called siftReturner() which takes an image and returns its keypoints and descriptors as tuples. Since we are collecting the images in a vector of matrices, the function is also taking an integer index value in order to find the wanted image in the vector. Inside the function, I defined an empty keypoints vector and descriptor matrix and with the SIFT object I passed these to the detectAndCompute method in order to be filled. Before obtaining the keyPoints and Descriptors of the images with SIFT, a SIFT object must be created (which was also sent to the function as an argument). SIFT object created at the beginning of the main and used each time for keyPoint and Descriptor obtainment.
    - As a side note, I created another function called visualizeKeypoints() for visualization purposes of the keypoints. visualizeKeypoints() function takes the image vector, image index for finding the image in the vector and its keyPoints as an argument and displays the image with keyPoints to the screen.

- **STEP 3:** As in the previous step, in this step keyPoints and Descriptors are obtained with the SIFT object and its detectAndCompute() method for each patch. Previous function was used and keyPoints and Descriptors for each patch were obtained.

- **STEP 4:** Since we have obtained all keypoints and descriptors of the images, in this step matching was done between the descriptors by using Brute-Force Matcher (BFMatcher). After creating the BFMatcher object with parameters NORM_L2 and crossCheck = true, with the “match” method the descriptors of the patches are passed and matched one-by-one with the corrupted image descriptors to obtain possible matches. Since we are in the loop, in each iteration matches are collected in a <DMatch> type vector in order to keep every information about the match. Working principle of the match method is that the primarily passed (first parameter) descriptor point is matched with all the descriptor points in the secondly passed descriptor (second parameter), and the best match is returned in each iteration. Meaning of the best match is the smallest distance match is returned as a best match by doing comparison element-wise. In the second part of the Step 4, best matches are refined in accordance with the minimum distance approach. getMinimumDistance() function below is dedicated for obtaining the minimum distance within the matches array for each patch iteration. In accordance with the obtained minimum distance approach, ratioThreshold is calculated manually, by doing comparisons, as 0.07f and comparisons are made element-wise within the matches vector. During the iteration within the vector, the current position distance value is compared with the first position distance value which is multiplied with the ratio. And if the distance value of the current position is less, then the match is collected in the bestMatch vector.

- **STEP 5:** After I obtain the refined matches, key points (both for corrupted image and patch) are filtered according to best match positions meaning that only the key points of the best matches are filtered and collected as points. Points are collected with definition of <Point2f> and collected in a vector in order to pass the values to the findHomography() function. After the filtering operation, with the help of findHomography() function, I passed the points that are filtered and I obtained the perspective transformation of the given patch in accordance with the corrupted image.

- **STEP 6:** With the obtained perspective transformation, warpPerspective() function is used. Image matrices (corrupted image and patch), their keypoints and found homography are passed to the warpPerspective() function. By this approach, found best matches for the patches are located correctly on the corrupted image and copied on that position in order to fix the corrupted image. As a result, a fixed image is displayed on the screen.

You can find the results that I obtained during the process of implementation in the results section. Each obtained result is shown as screenshots that are taken during the execution and annotated with the required information.

RESULTS

Image 1: Corrupted Image with KeyPoints

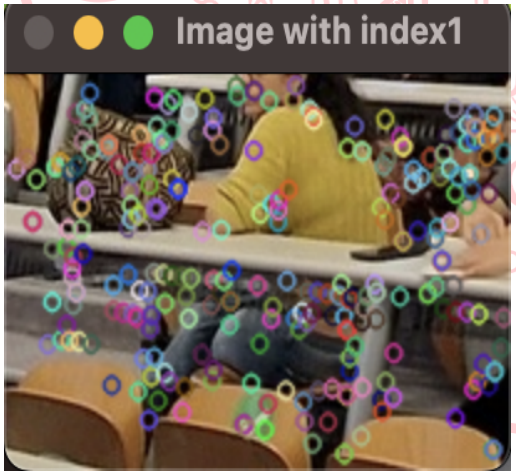

Image 2: Patch_0 with KeyPoints

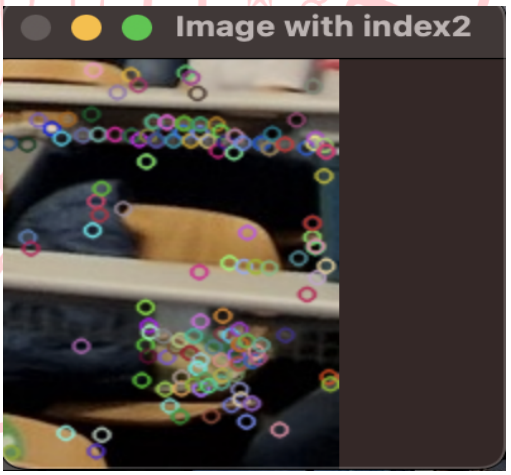

Image 3: Patch_1 with Keypoints

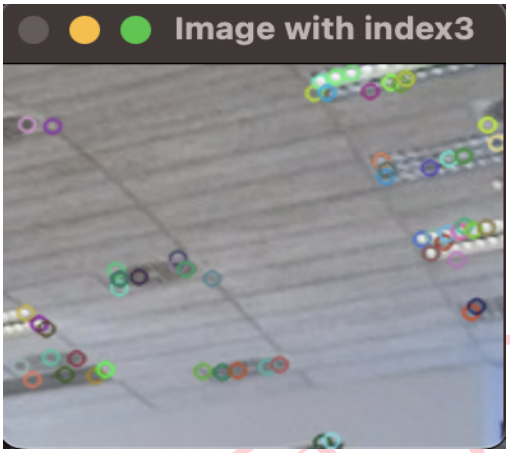

Image 4: Patch_2 with KeyPoints

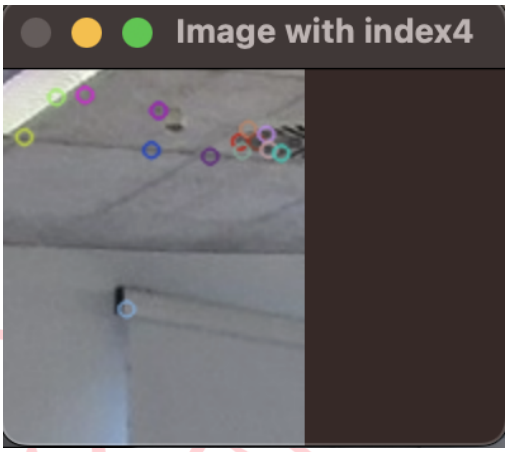

Image 5: Patch_3 with Keypoints

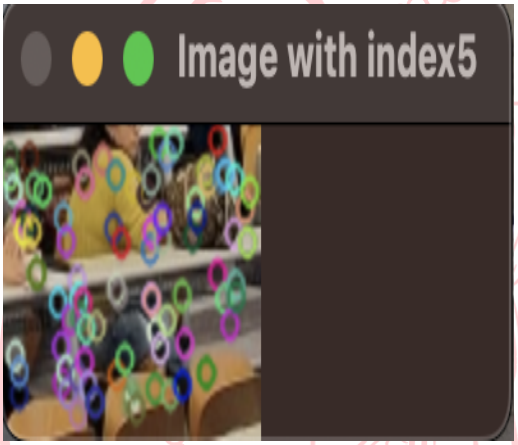

Image 6: Patch_t_0 with KeyPoints

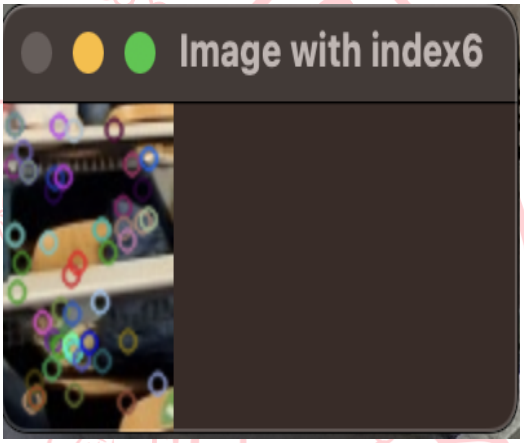

Image 7: Patch_t_1 with KeyPoints

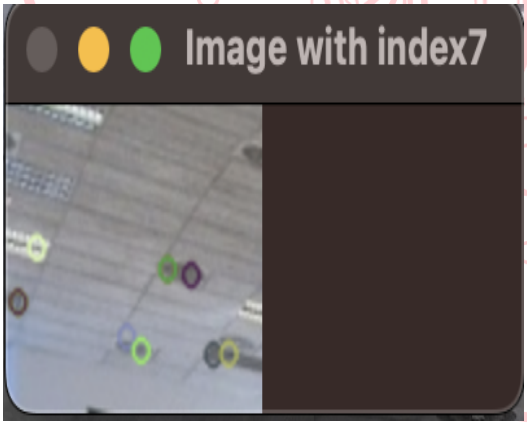

Image 8: Patch_t_2 with KeyPoints

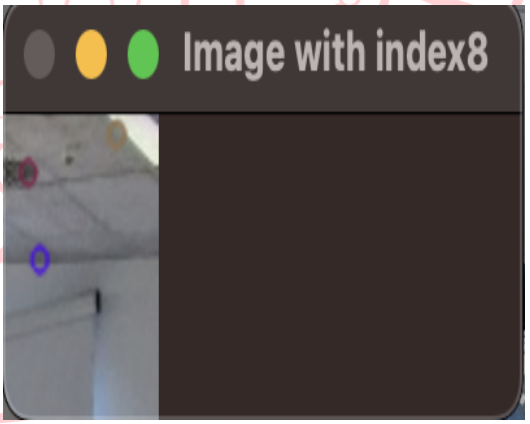

Image 9: Patch_t_4 with KeyPoints

Figure 10: Fixed version of the corrupted image