Computer Vision - Final Project (LAB4)

Povegliano Giorgio 2089066

The objective of this lab is to implement an efficient approach for replacing corrupted regions in an image by utilizing data obtained from patches. Each dataset comprises the corrupted image, normal patches, and "_t" patches, which have been subject to transformations to intensify the complexity of the problem, including rotation, scaling, flipping, and noise addition. I have implemented several methods to address this task, and their effectiveness, particularly with transformed patches, will be examined and discussed in the subsequent sections.

Starting off we have the **basic implementation**: it exploits the SIFT (Scale Invariant Feature Transform) to extract feature both for the image and the associated patches. This operation returns a vector of keypoints and descriptors, that will be used to compute the matches between each patch and the corrupted image (using Brute-Force Matcher). We then proceed to keep only the matches that have distance less than *threshold = ratio * min_distance*, in which *ratio* is a user-defined value, and *min_distance* is the minimum distance amongst the matches. At this point we can compute the affine transformation (represented by a matrix) that links each patch with the image, to do so we exploit the RANSAC algorithm implemented in the function *findHomography*. The final step involves overlaying the patches within the corrupted image. To accomplish this, we extract the corners of each patch and position them in the corrupted image using the homography matrix computed earlier. Next, we create a rectangle using the four destination corners and paste the patch into the corresponding region of the image.

In practice I found out that these parameters work well for the feature extraction of the images:

- Number of octave layers (number of octaves in the scale space): 8
- Contrast threshold (filter features with low contrast): 0.04
- Edge threshold (filter features that are too close to edges): 5
- Sigma: 1.6

With these parameters, we obtained the results depicted in the following pages. The first figure represents the valid matches between the patch and the image (as mentioned earlier, all of them have a distance less than the threshold). The second figure shows the patch overlaid on the corrupted image. Finally, the last figure displays the final result of the restored image with all the patches in their corresponding locations.

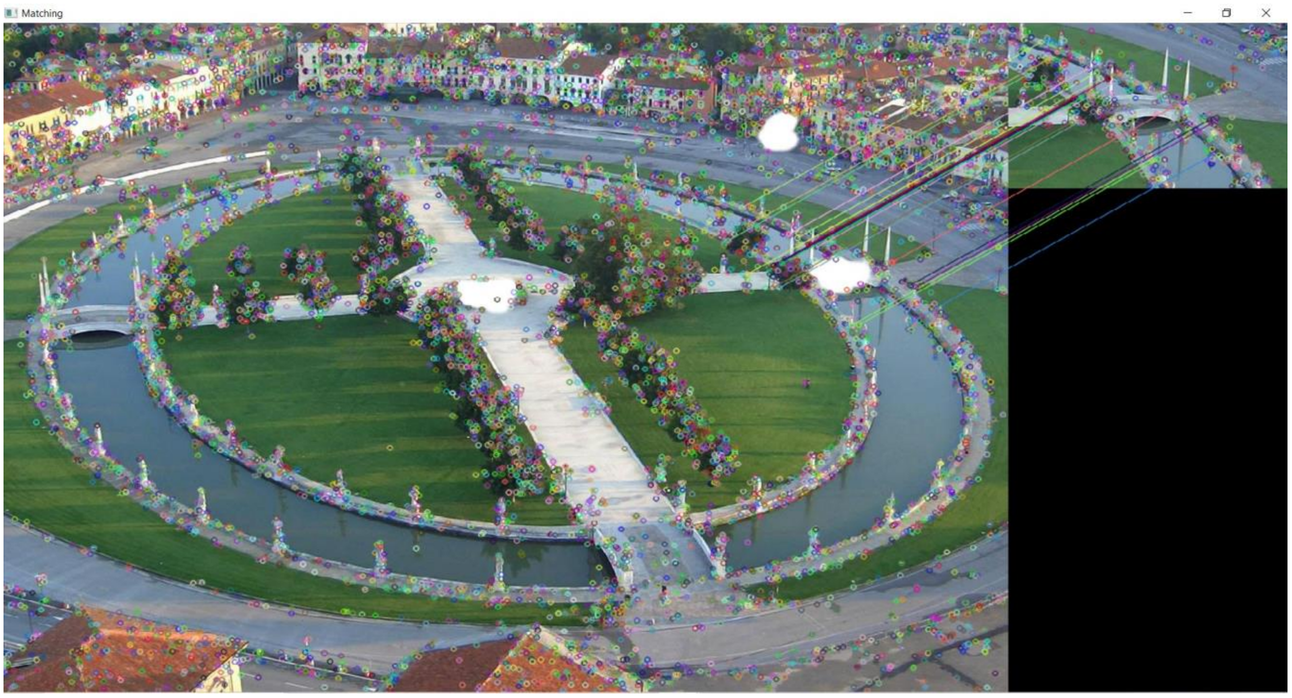

Figure 1: matchings for the first patch.

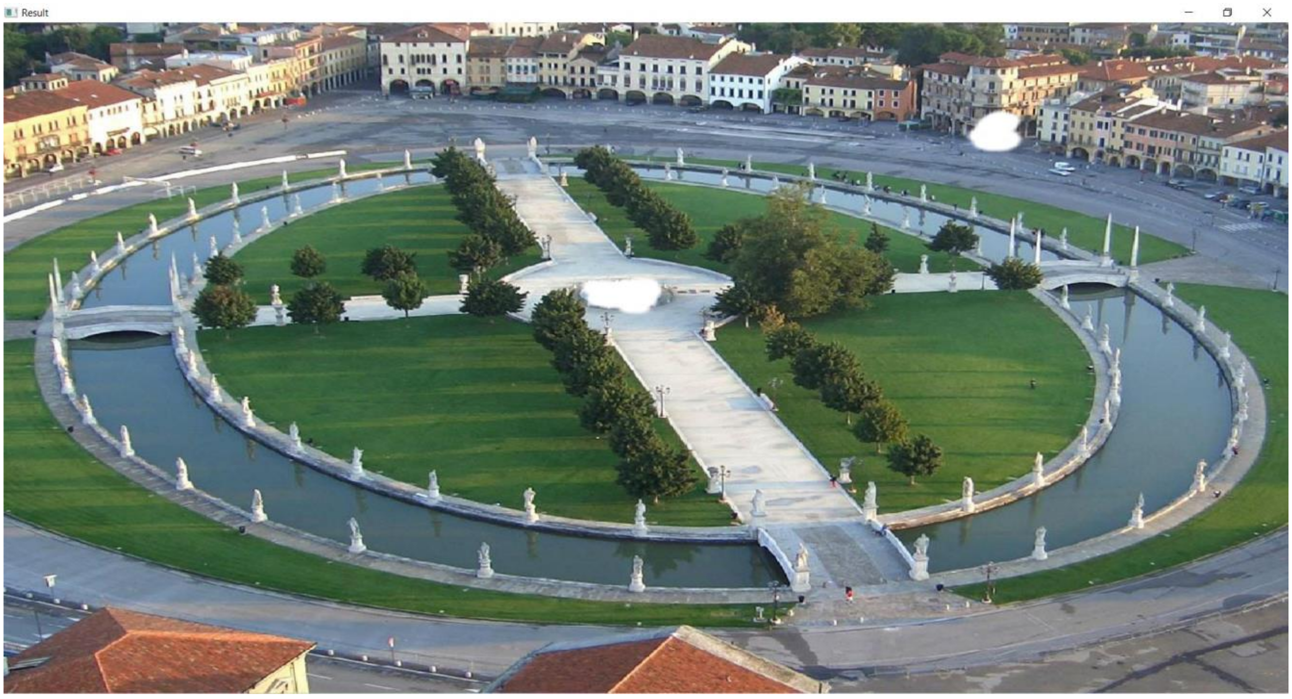

Figure 2: first patch overlayed in the corrupted image.

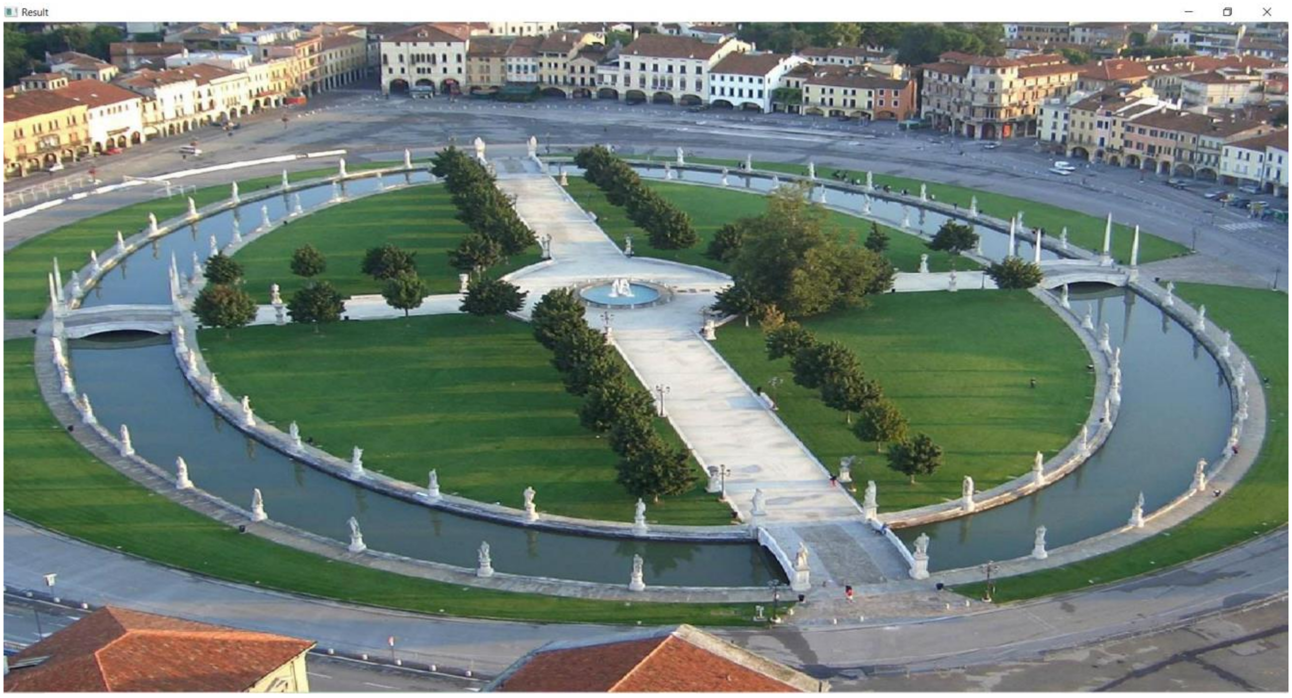

Figure 3: result with normal patches.

We now look at the transformed patches, setting the "*transformed"* variable to 1 we obtain:

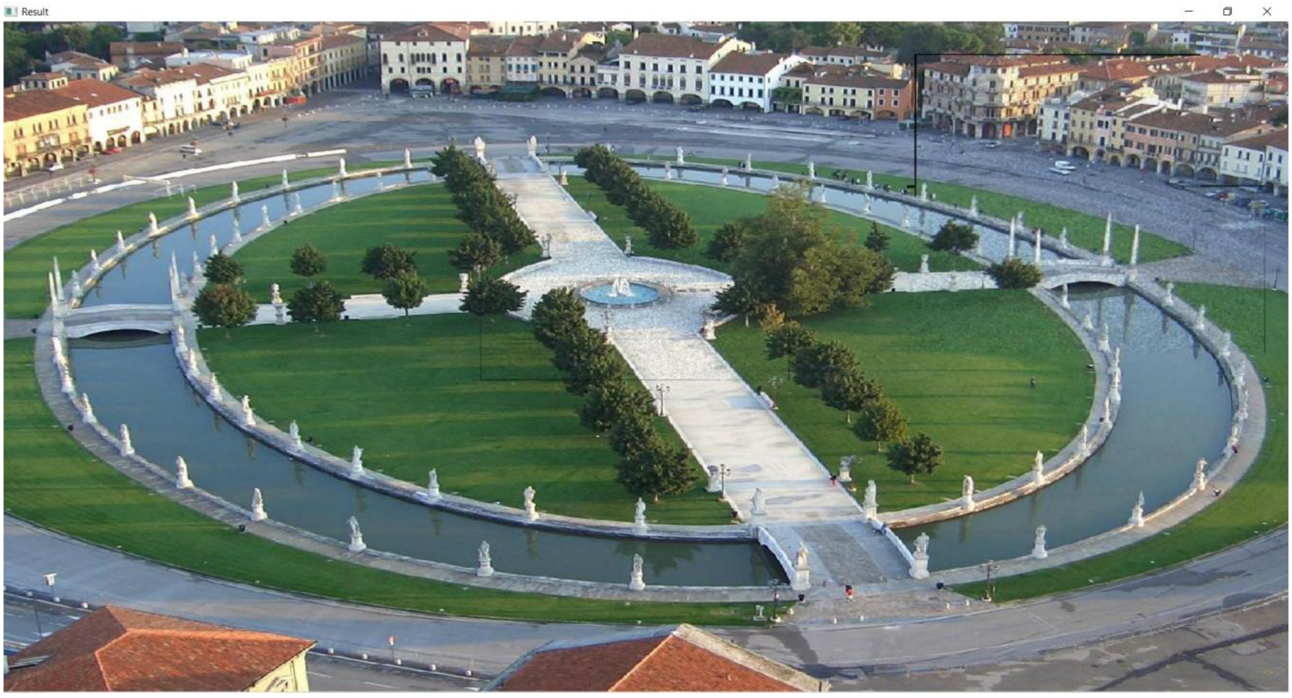

Figure 4: result with transformed patches.

Upon closer inspection of the above figure, it is evident that the patch's position within the image is accurate, but it is not perfectly aligned due to the presence of added noise. To resolve this issue, we can adjust the top-left and bottom-right corners of the rectangle after wrapping, ensuring that the black border no longer appears.

There is another observation to be made regarding the transformed patches. In the case of the patch containing the fountain, it appears to have been flipped, requiring additional calculations to properly position it. For this reason, it is necessary to compute the matches twice: once with the original patch and once with the flipped patch. The distances of all the matches are then summed, and the matches with the lower distances are retained (also updating the flipped patch if needed).

Additional implementations:

I also attempted to resolve the problem using the **ORB feature descriptor**. From a code perspective, there are not many changes required to implement this solution, as ORB is also implemented within OpenCV. The main difference to keep in mind is that the distance of the matches is calculated using the Hamming distance instead of the Euclidean distance. This implies that we cannot compute the threshold in the same way as we did for SIFT. In the case of ORB, if two exact same points are matched, their Hamming distance will be 0, resulting in a threshold of 0 as well. To solve this issue, we can consider the threshold as a fixed integer value.

We can observe that ORB is generally faster than SIFT, but it is also less robust. When analyzing the results, we notice that ORB performs well with normal patches, but it tends to make more mistakes when working with transformed patches. These mistakes can include incorrect matches or inaccurate alignment of the patches within the image. For example, here are its performance on the "Scrovegni" dataset, which includes patches that have been subject to a rotation.

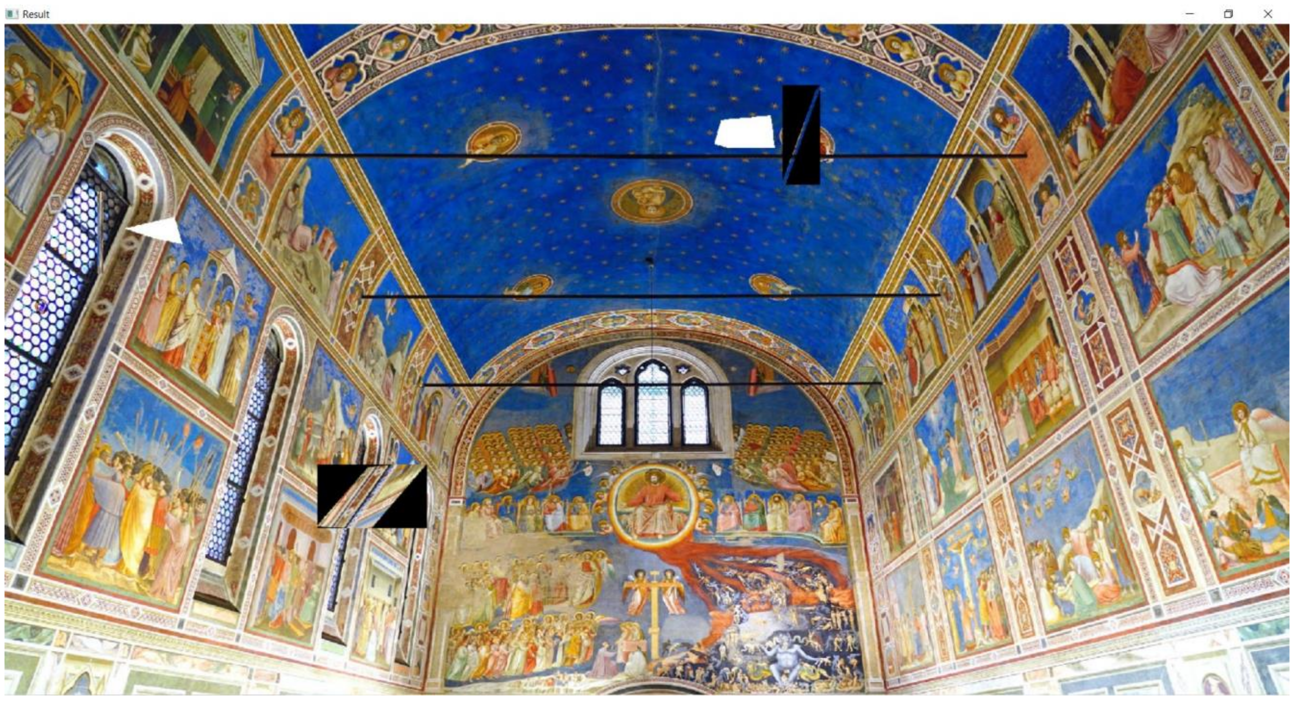

Figure 5: ORB performance with transformed patches.

While ORB may be advantageous in scenarios where speed is a critical factor, such as real-time applications, it may not be the ideal choice when dealing with more challenging image transformations or complex scenes (like this one). In such cases, SIFT's higher level of robustness and better handling of transformed patches make it a preferred choice despite its relatively slower computation speed.

As an additional step, I also created my own dataset called "**totem**". In this dataset, I erased portions of the image and then cut a patch that is larger than the hole. This approach allows the algorithm to match keypoints between the patch and the corrupted image. The following results demonstrate the effectiveness of the ORB feature descriptor in efficiently solving the task.

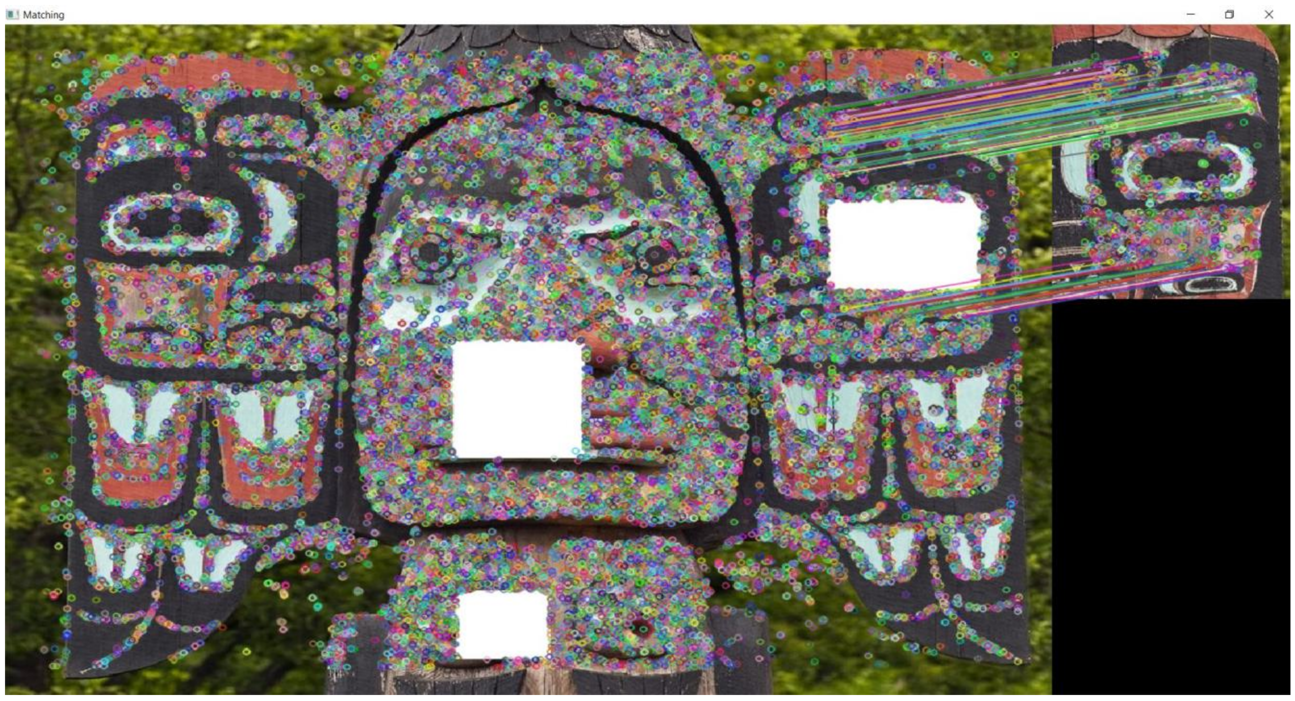

Figure 6: matchings of the first patch using ORB feature descriptor.

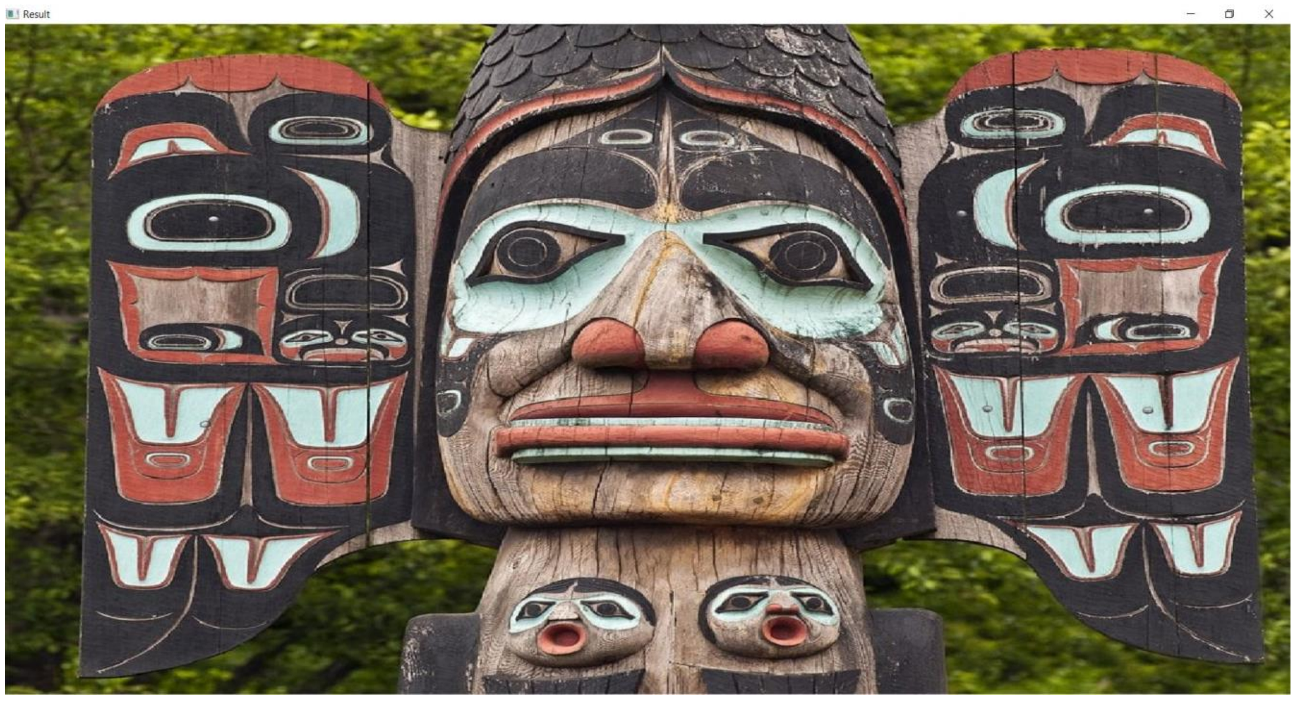

Figure 7: final result using ORB.

Template matching is another technique used for matching and locating patches that I implemented. It involves comparing a template or a small image patch with the larger image in a sliding window fashion. The template is moved across the image, and at each position, a similarity metric is computed to determine the resemblance between the template and the corresponding image region. It is a computationally expensive approach, especially when performed in large images with many different templates. Template matching is primarily effective for non-transformed patches, as it is not robust to image scaling and rotation. Its performance for normal patches may sometimes be even worse than ORB. This technique is particularly unsuitable for images with repeating patterns (like the “totem” dataset), as it may not provide accurate results in such cases.

Here is an example where template matching performs well:

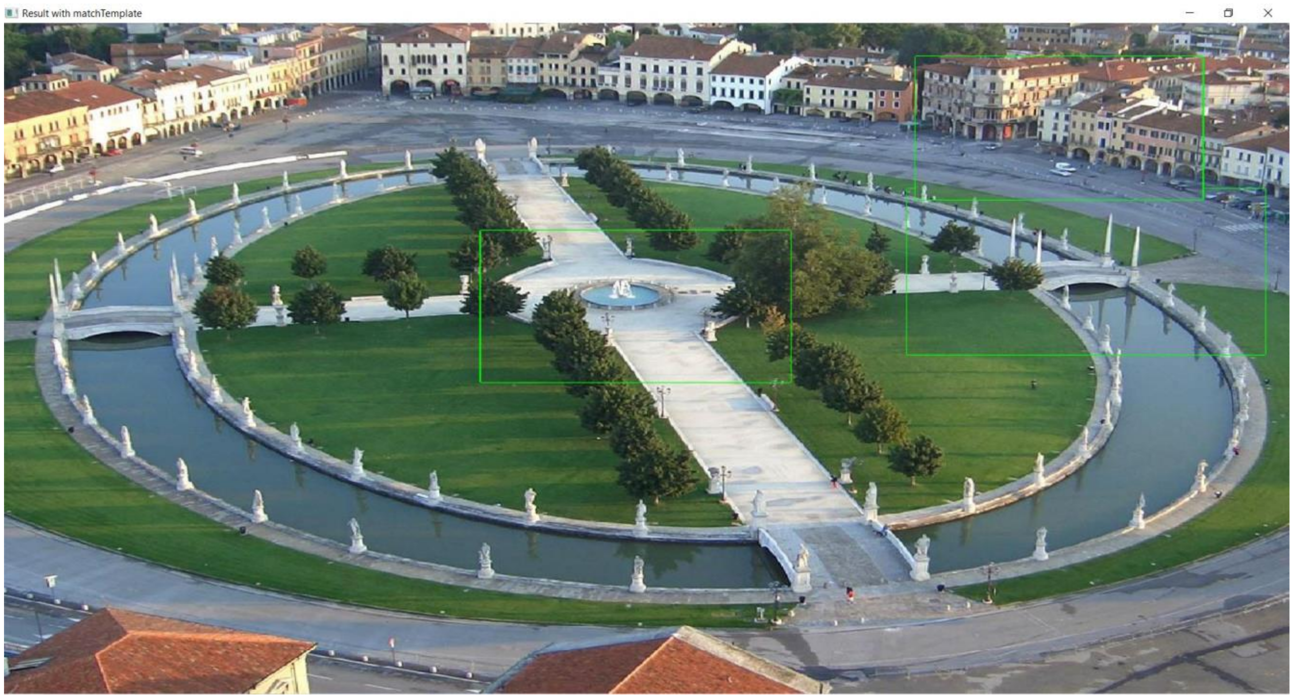

Figure 8: good performances of template matching in “pratodellavalle”

While in *Figure 9* is showed a case in which it does not lead to good outcomes.

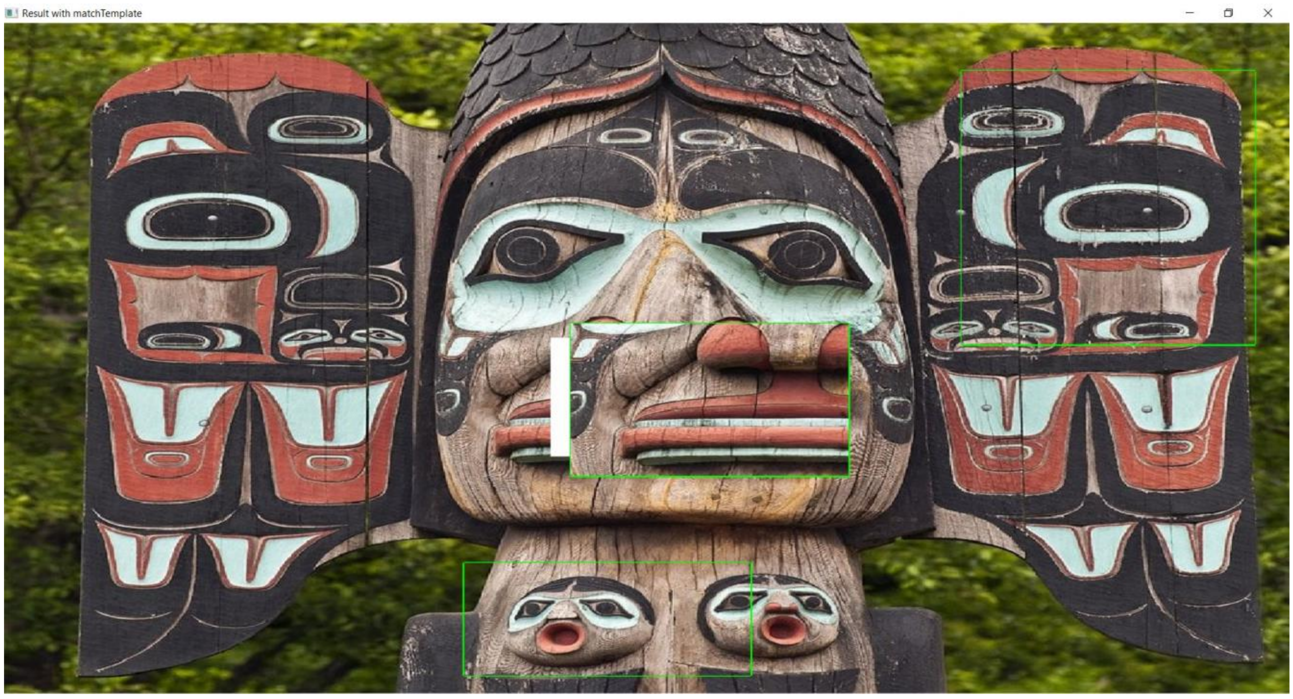

Figure 9: bad performances of template matching in "totem"

Conclusion:

The analysis of different feature descriptors for patch recognition reveals distinct characteristics and performance levels. SIFT, with its robustness to transformations and ability to handle variations in scale and orientation, proves to be a reliable choice in various scenarios. ORB, on the other hand, demonstrates faster processing times but may struggle with transformed patches, making it less suitable in those cases. Template matching, while effective for non-transformed patches, lacks robustness when faced with scaling and rotation variations, particularly in images with repetitive patterns.