Computer Vision: Technical Report on Project 2

Papadopoulos Aristeidis A.M.:57576

- ## Angle Detection:

The positions of the angles are computed by SIFT(Scale Invariant Feature Transform) & SURF (Speeded Up Robust Features) algorithms. Harris' Corner Detection is not used because the objects of the given images are in different scale. Therefore, a detector invariable to scale needs to be used. So, the SIFT algorithm, and its faster variance SURF are going to be used in this project. An angle is defined as points in image that display variation in luminosity in both directions(vertical and horizontal). In this way, SIFT detector utilizes the maxima or minima that are computed by the difference of two Gaussian distributions with different variance (σ), in a way that maximizes this difference. Then, for each point of interest, a descriptor is created, i.e., a vector that contains certain features for each point. The descriptor is computed with this method:

1. In each point of interest, a window with dimensions 16x16 is placed.
2. Each window is divided into 4 sub-windows, where each window contains pixels that are placed into 4x4 windows.
3. For each pixel, its gradient direction is computed.
4. Using thresholding, the gradients lower than the threshold are discarded.
5. This way, a histogram, sized 32 is created for each sub-window, 8^{th} dimensional, due to the eight different directions of the gradient. Finally, the descriptor of 128 places is created(8 dimensions multiplied by 16 sub-windows.

SURF descriptor is a quicker and more robust version of SIFT, where the same procedure is followed, except that the new descriptor has a size of 64 (4 dimensions multiplied by 16 sub-windows). The four dimensions are summation elements, independent of each image/sub-window. In the following images, the points of interest, or keypoints for each given image can be seen, first for SIFT algorithm, followed by those created by SURF algorithm. Due to the different number of elements in each descriptor(128 vs 64) different results are created. I tried using the 128 feature number in SURF algorithm, but the end result was something completely different and slower.

Image 1 Keypoints for image hotel-00 using the SIFT Algorithm.

Image 2 Keypoints for image hotel-00 using the SURF Algorithm.

Image 3 1 Keypoints for image hotel-01 using the SIFT Algorithm.

Image 4 Keypoints for image hotel-01 using the SURF Algorithm.

Image 5 Keypoints for image hotel-02 using the SIFT Algorithm.

Image 6 Keypoints for image hotel-02 using the SURF Algorithm.

Image 7 Keypoints for image hotel-03 using the SIFT Algorithm.

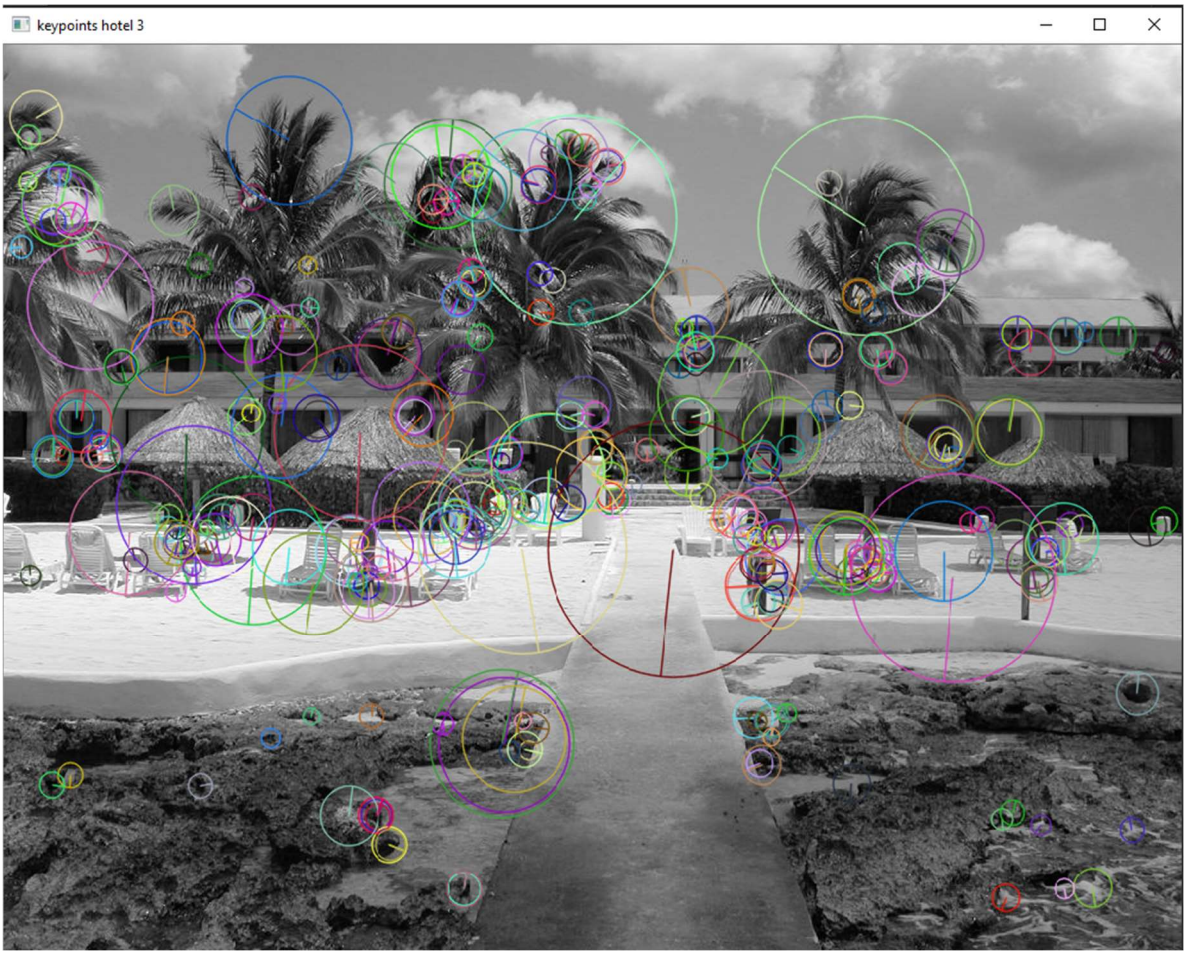


Image 8 Keypoints for image hotel-03 using the SURF Algorithm.

Image 9 Keypoints for panorama between images hotel01-hotel00 using SIFT Algorithm.

Image 10 Keypoints for panorama between images hotel01-hotel00 using SURF Algorithm.

Image 11 Keypoints for panorama between images hotel02,hotel01,hotel00 using SIFT Algorithm.

Image 12 Keypoints for panorama between images hotel02,hotel01,hotel00 using SURF Algorithm.

- **Matching & cross-checking**

```
def matchcross(d, d1):
    matches = []
    for i in range(d.shape[0]):
        vector = d[i, :]
        euclid_dis = np.linalg.norm(d1 - vector, axis=1)
        idx = np.argmin(euclid_dis)
        min_dist = euclid_dis[idx]
        for j in range(d1.shape[0]):
            vector1 = d1[j, :]
            euclid_dis1 = np.linalg.norm(d - vector1, axis=1)
            idx1 = np.argmin(euclid_dis1)
            if idx == idx1:
                matches.append(cv2.DMatch(i, idx1, min_dist))
    return matches
```

Image 13 Function implementation of matching and cross-checking.

Matching and cross-checking common keypoints is based on a function displayed during the laboratories hours of this course. Firstly, each keypoint descriptor of the first image is isolated into one variable, and then Euclidean Distance is calculated (using numpy.lingalg.norm) between the isolated descriptor and each descriptor of the second image, with which we are looking for common keypoints. Then, the minimum of those distances is found. Furthermore, the same process is used for the pair of those images, and a check is performed if the minimum distance from the first pair (first-second image) corresponds with the minimum distance of the second pair (second-first image). If this check returns True, then a common keypoint has been found, and is saved. Following this process, all the common keypoints can be found. Then, the two images are connected, as it can be seen in the bellow images. Similarly, almost the same results can be produced using brute force matching.

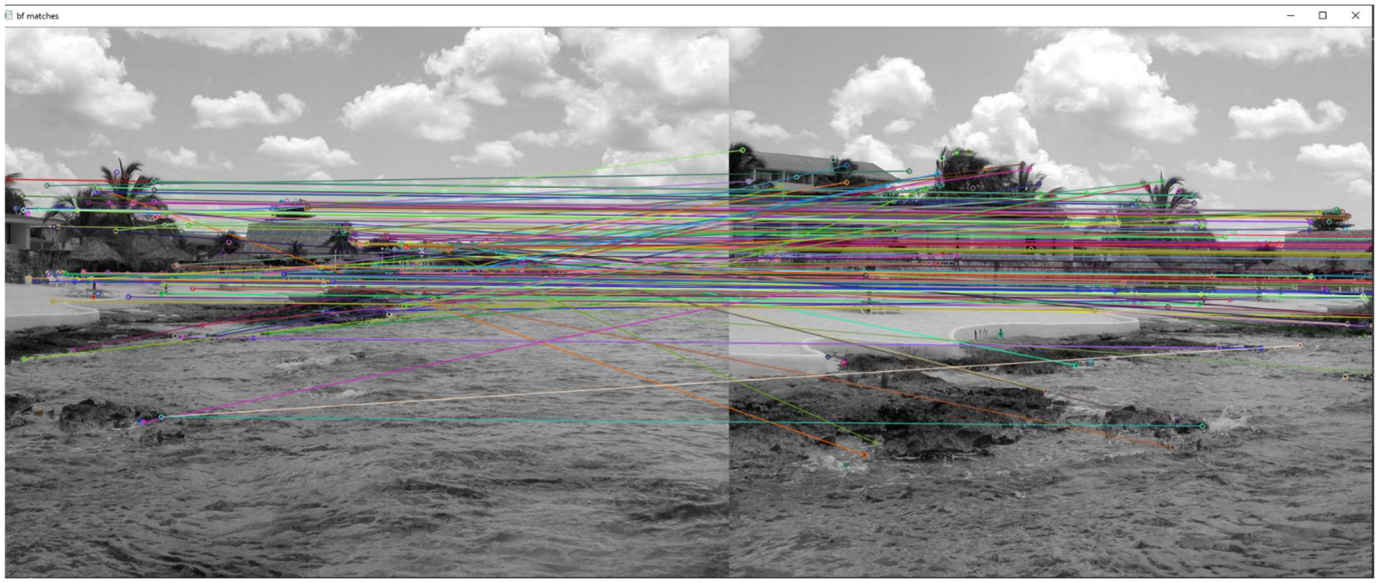


Image 14 Common keypoints for images hotel00(left) and hotel01(right), using SIFT descriptor and brute force matcher.

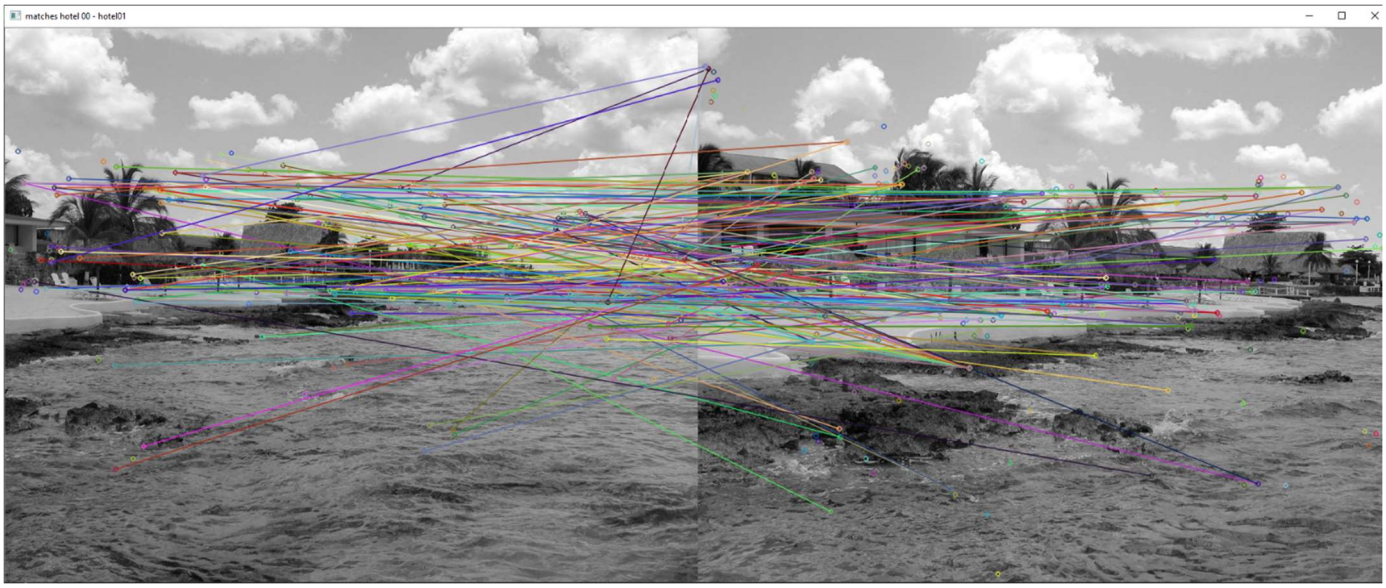


Image 15 Common keypoints for images hotel00(left) and hotel01(right), using SIFT descriptor and custom matching function.

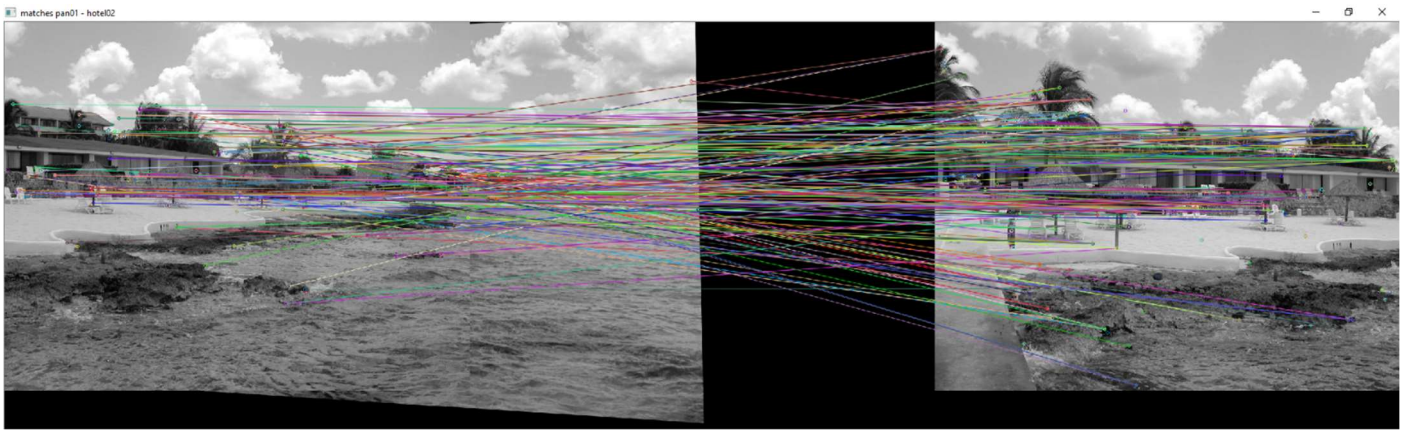


Image 16. Common keypoints for images pan01(left) and hotel02(right), using SIFT descriptor.

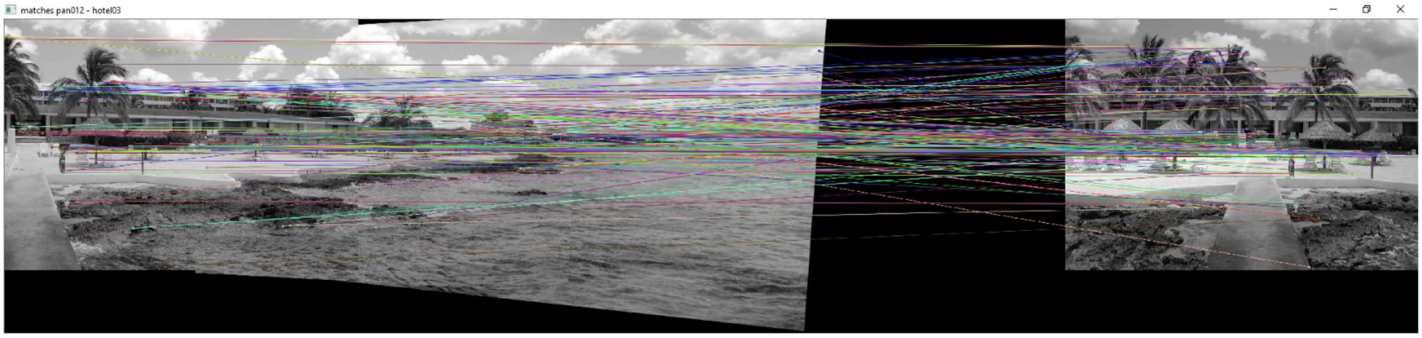


Image 17 Common keypoints for images pan012(left) and hotel03(right), using SIFT descriptor.

In order to create the final panorama, first the images hotel00 and hotel01 were connected. Then, descriptors where used in the connected results, and using the descriptors for the image hotel02, the same procedure was followed, to create the panorama between images hotel00,hotel01 & hotel02. Lastly, using the same algorithm, using descriptors for the image hotel03, the final panorama was created. The following imaged depict the matches found using the SURF algorithm.

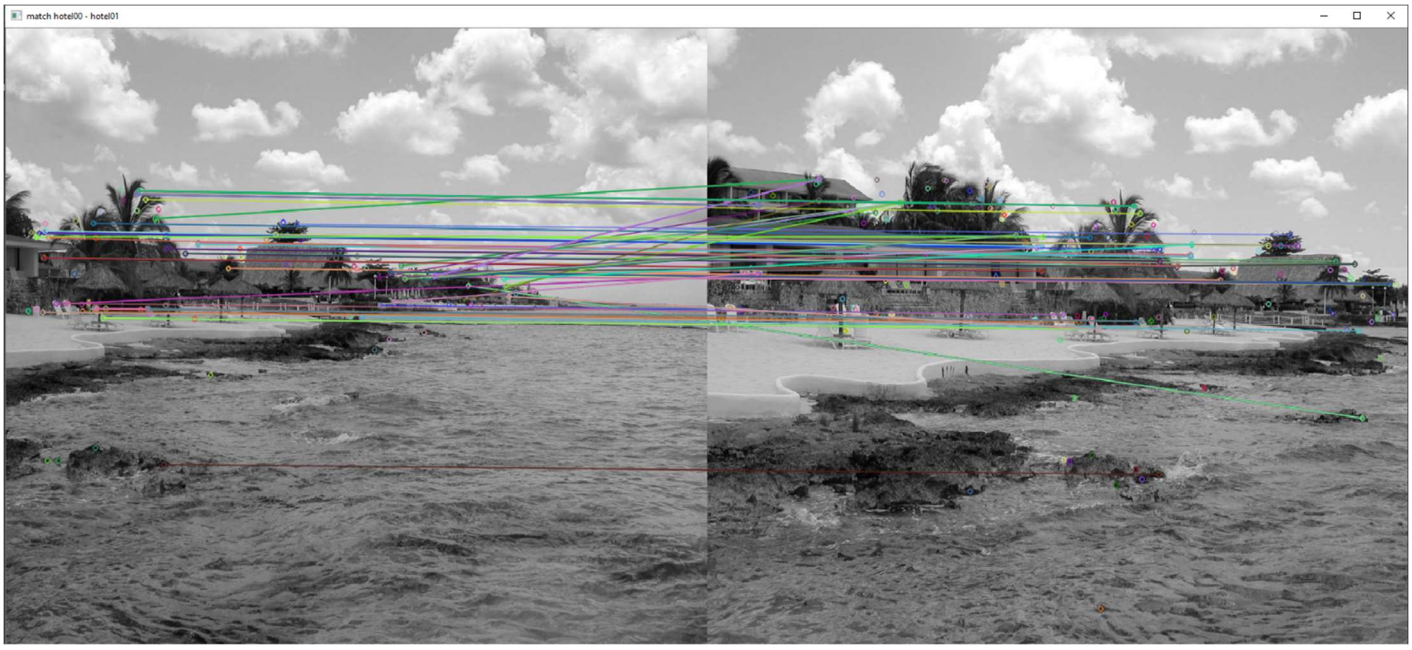


Image 18 Common keypoints for images hotel00(left) and hotel01(right) using SURF Algorithm

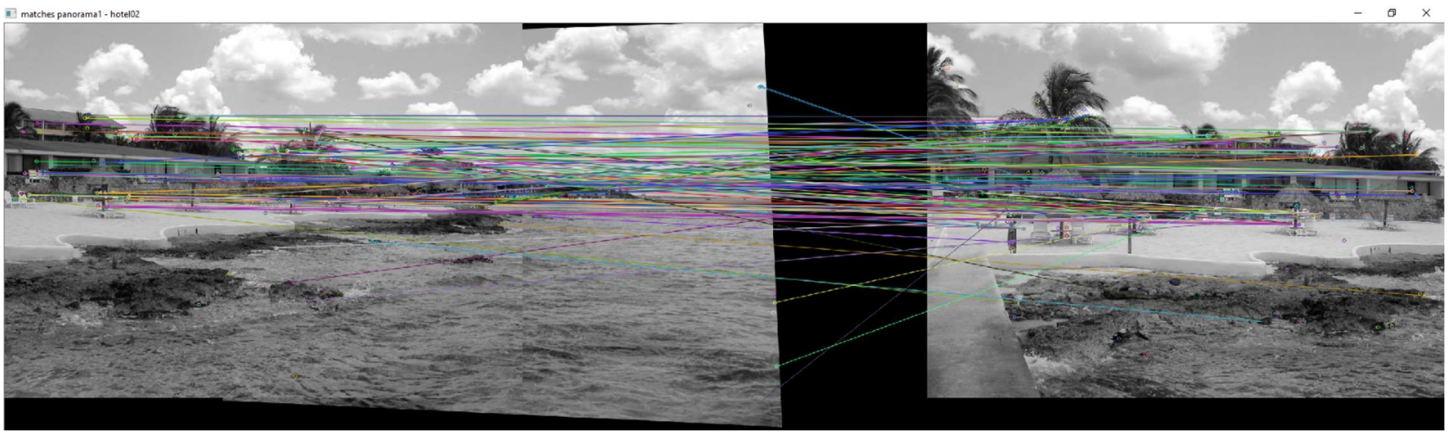


Image 19 Common keypoints for the panorama image of hotel00,hotel01(left) and hotel02(right) using SURF Algorithm

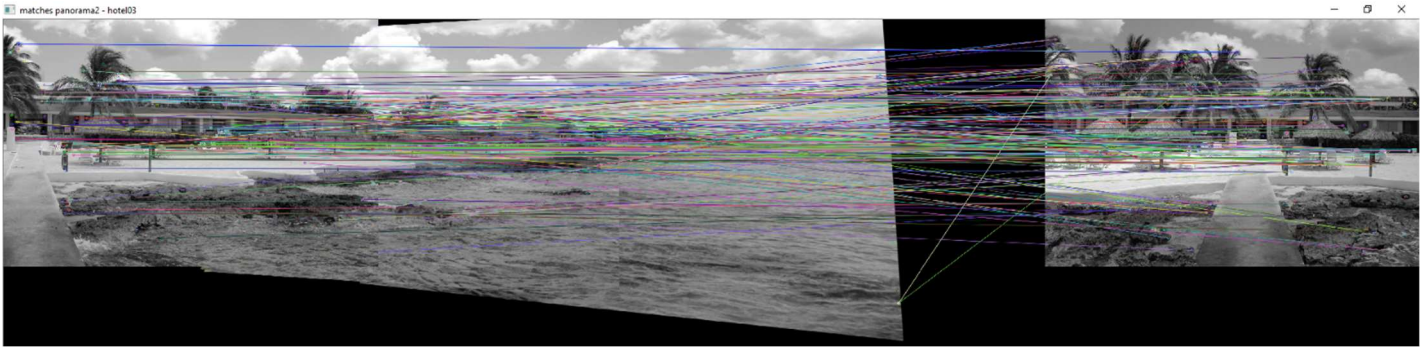


Image 20 Common keypoints for the panorama image of hotel00,hotel01,hotel02(left) and hotel03(right) using SURF Algorithm

Some matches are missed, i.e., they don't exist, or exist but are not found by this implementation. Those matches will be discarded using RANSAC algorithm, as it will be explained later on.

- ## Finding Homography matrix and creating panorama picture

```
def points(k, k1, match):
    pt1 = []
    pt2 = []
    for x in match:
        pt1.append(k[x.queryIdx].pt)
        pt2.append(k1[x.trainIdx].pt)
    pt1 = np.array(pt1)
    pt2 = np.array(pt2)
    return pt1, pt2


def creating_pan(pt1, pt2, img1, img2):
    M, _ = cv2.findHomography(pt1, pt2, cv2.RANSAC)
    pan_out = cv2.warpPerspective(img1, M, (img1.shape[1]+img2.shape[1], img1.shape[0] + img2.shape[0]))
    pan_out[0:img2.shape[0], 0:img2.shape[1]] = img2
    return pan_out
```

Image 21 Implementation of functions calculating matching coordinates, homography array and panorama creation.

In order to calculate the homography matrix, which will show how one picture must be transformed to match with another one, matching points in both pictures must be found. This process is followed in the function "points", where keypoints of the first image, keypoints of the second image and a list containing their matches are used as inputs. For each match, their corresponding coordinates are placed in a list. Afterwards, homography matrix is calculated, while wrong matches are discarded using RANSAC method (RANdom Sample Consensus). This method contains the following steps:

1. A match is selected, followed by its orientation, and then matches with the same orientation are found and elements with the higher probability to be matches or inliers.
2. Elements with the highest variety of matches are saved per orientation, while the rest are considered outliers and are discarded.
3. This process is repeated until the number of elements or the elements themselves stop changing.

Afterwards, one of the images is rotated according to the homography matrix and the second image is pasted on top of the first one, covering their common pixels. The process I followed to create the final panorama image is described as follows:

1. Beginning with the images hotel00,hotel01, their panorama is created, tentatively named pan01.
2. The keypoints and the descriptors of pan01 are calculated and the panorama between this image and image hotel02 is created, tentatively named pan012.
3. Step two is repeated between images pan012 and hotel03.

The inverse method was not followed, i.e. panorama between images hotel03, hotel02 first e.t.c., because elements of image hotel03 are shifted in a way that are placed outside of our viewpoint and as such cannot be viewed. The panorama images are followed, with each descriptor.

Image 22 Panorama between images hotel00 (right) and hotel01 (left).

Image 23 Panorama between images hotel02 (left) and the previous panorama image (right).

Image 24 Final panorama image using SIFT Descriptor.

Image 25 Panorama image between images hotel00(right) and hotel01(left) using SURF descriptor.

Image 26 Panorama between images hotel02(left) and previous panorama image(right).

Image 27 Final panorama image using SURF descriptor.

The final panorama images using either one of the descriptors are identical. One can observe a discontinuity between the panorama image of photos hotel00 and hotel01, along the vertical axis where the two images are concatenated. The same observation can be made for every other panorama created. This is due to the method used to create the panorama images, as one image is pasted on top of the other and due to different visual content between consecutive shots, such as the waves on the water.

Image 28 Example of discontinuity between images.

- **Panorama comparison with Image Composite Editor (ICE)**

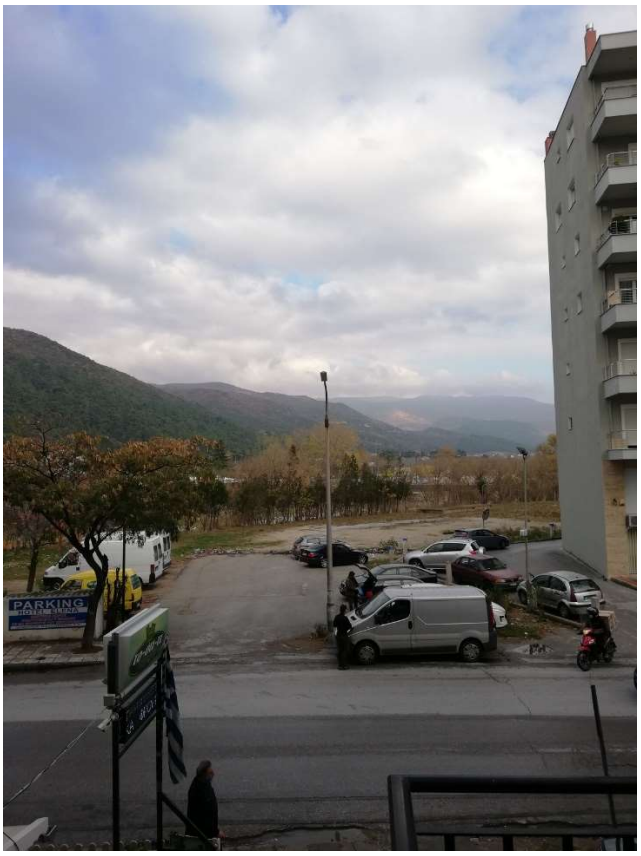
Image 29 Initial panorama using Image Composite Editor

Firstly, the panorama image using ICE can be seen in the above image. The differences spotted between this implementation and ICE are due to the fact that ICE solves the problem mentioned before. More specifically, the ICE-panorama is created using as center pieces the images hotel02-hotel03 and then transforms the rest. However, toying with the setting of ICE can produce a similar result to the panorama created by this implementation.

Image 30 Panorama created by toying with the setting of ICE.

- **Process repetition using custom images**

The images used have dimensions of 1200x1600, as larger images would need much more time to create the desired panorama. The images used can be seen below:




Images 31,32 Initial images used, named view0(left) and view1(right).


Images 33,34 Initial images used, named view2(left) and view3(right).

Following the same process as before, first common keypoints are found, as it can be seen in the images below, while wrong matches are discarded using RANSAC. Furthermore, in the images below the matchings between each image can be seen and the panorama images that were created.

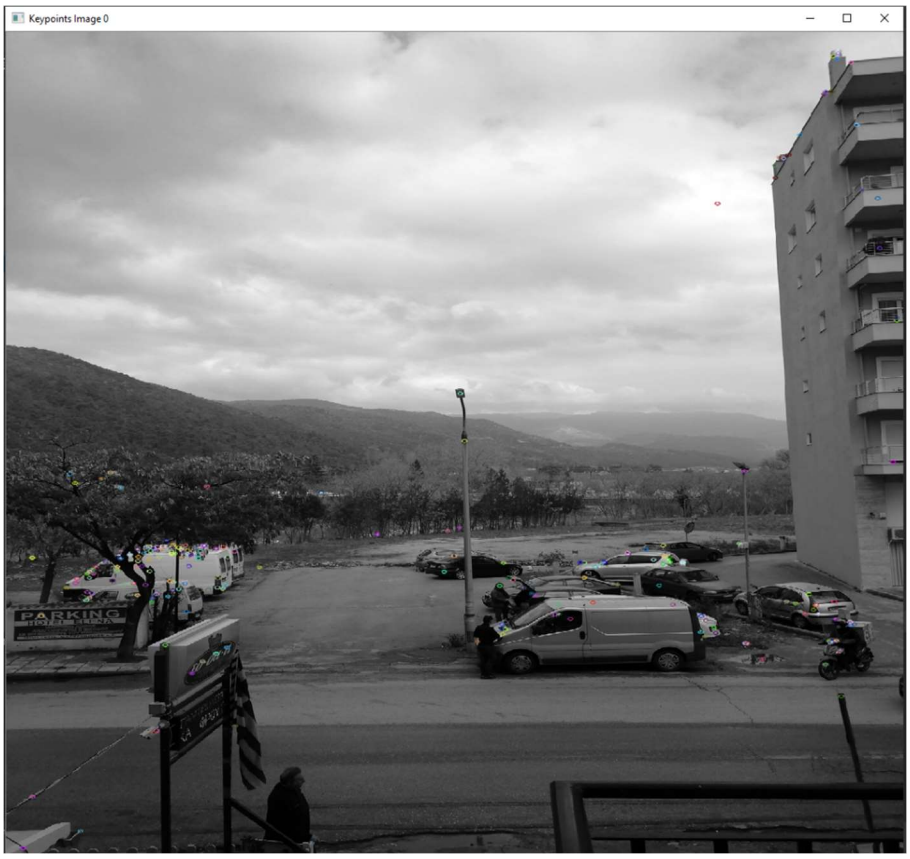


Image 35 Keypoints using SIFT descriptor for image view0.

Image 36 Keypoints using SIFT descriptor for image view1.

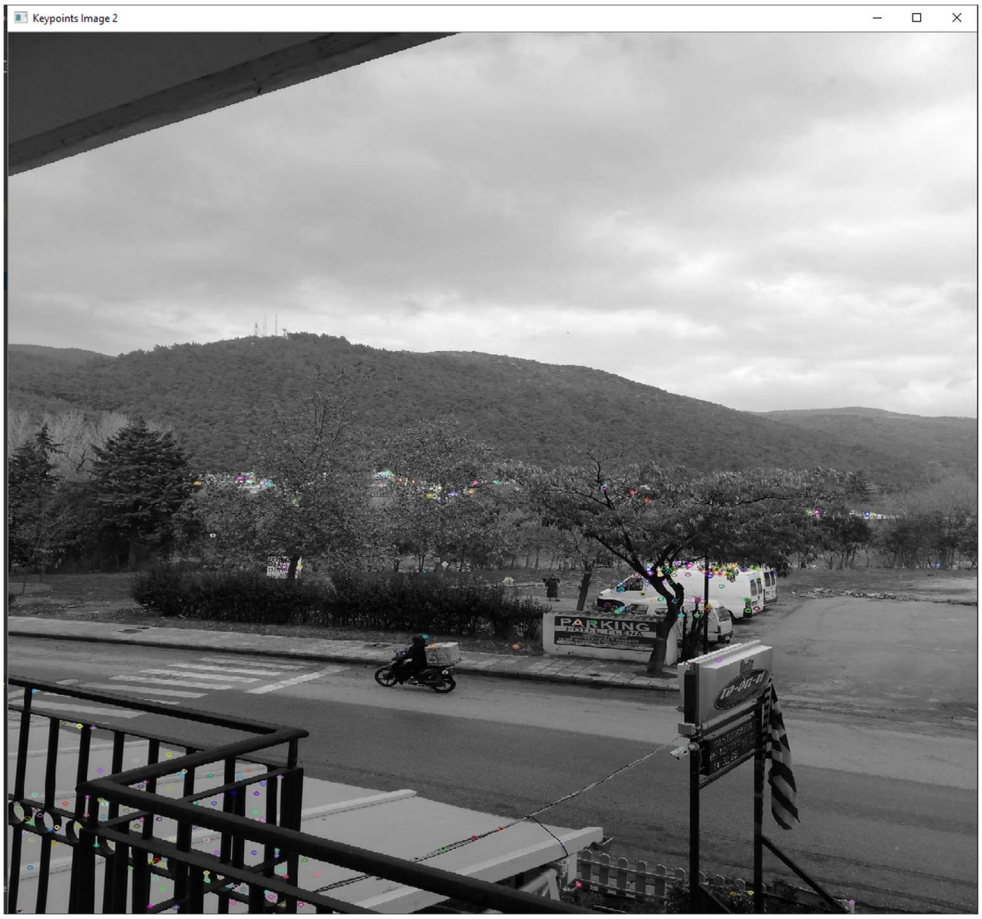


Image 37 Keypoints using SIFT descriptor for image view2.

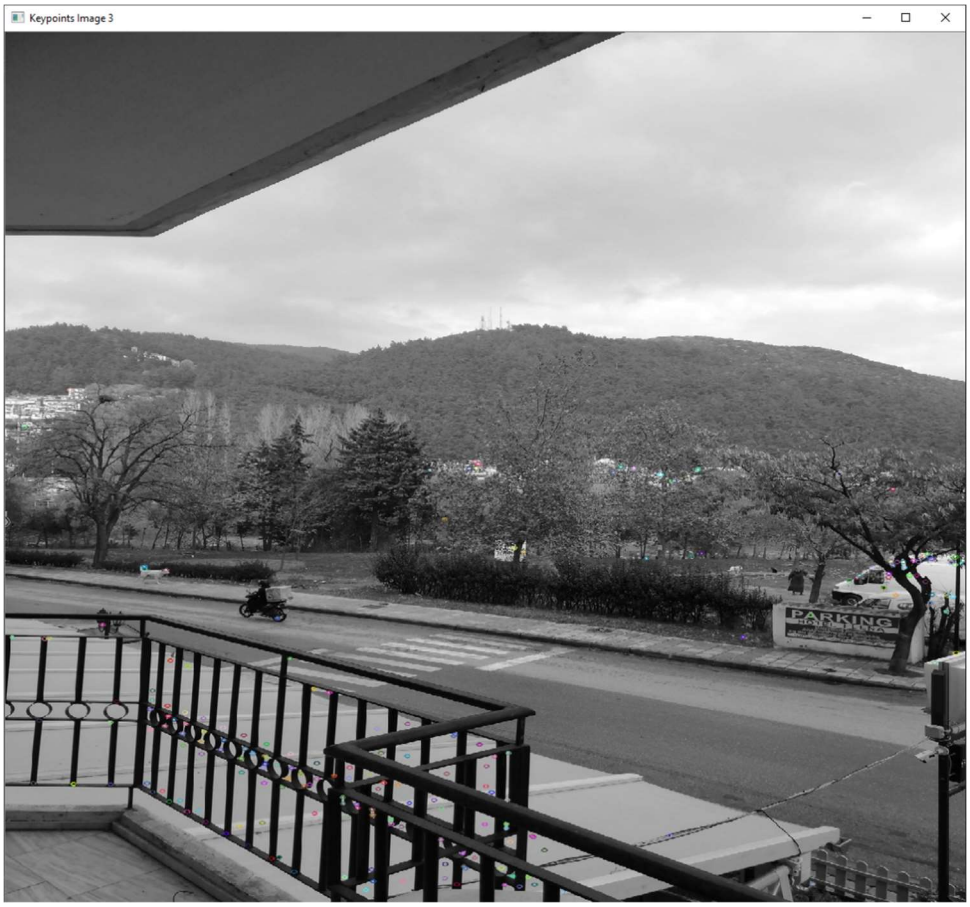


Image 38 Keypoints using SIFT descriptor for image view3.

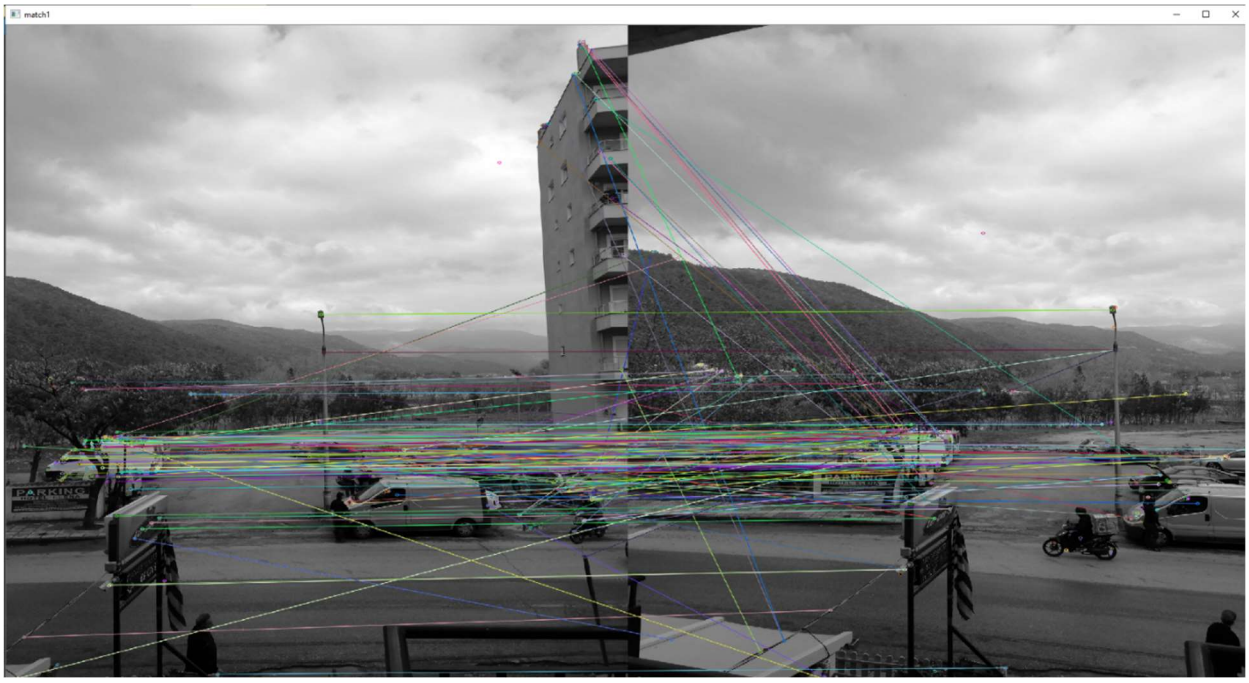


Image 39. Matching keypoints for images view0(left) and view1(right) using SIFT descriptor.

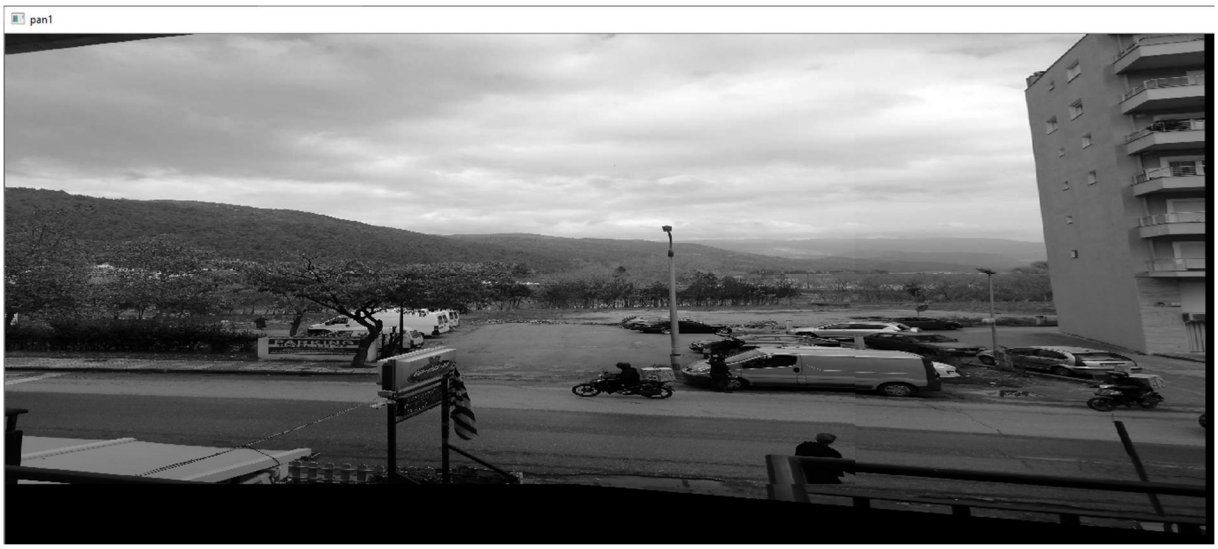


Image 40 Panorama created using images view0 and view1, using SIFT descriptor.

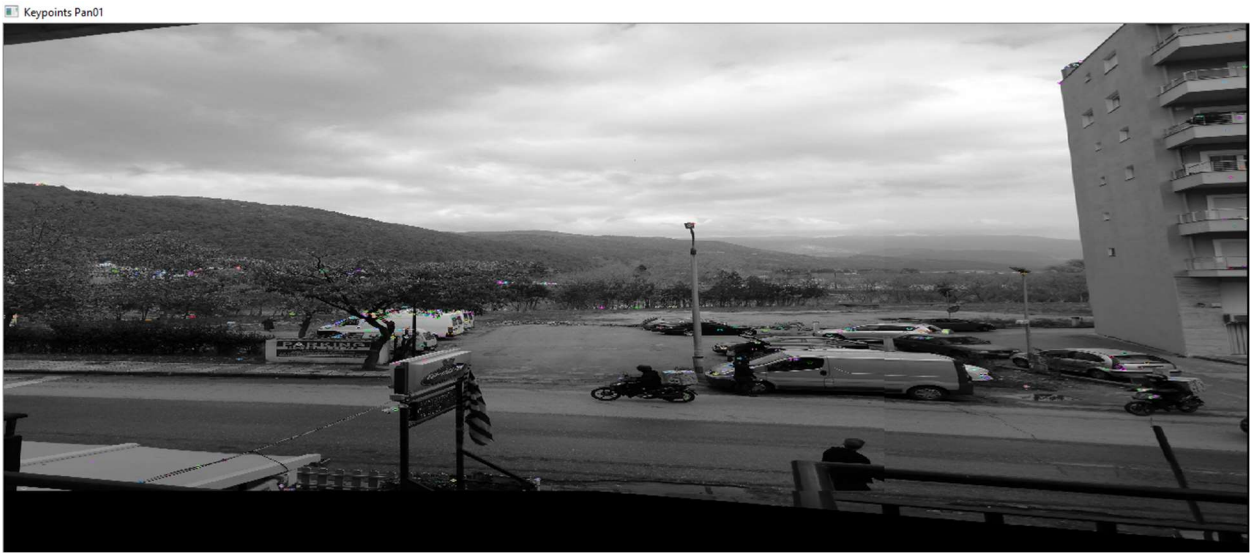


Image 41 Keypoints in the previous panorama.

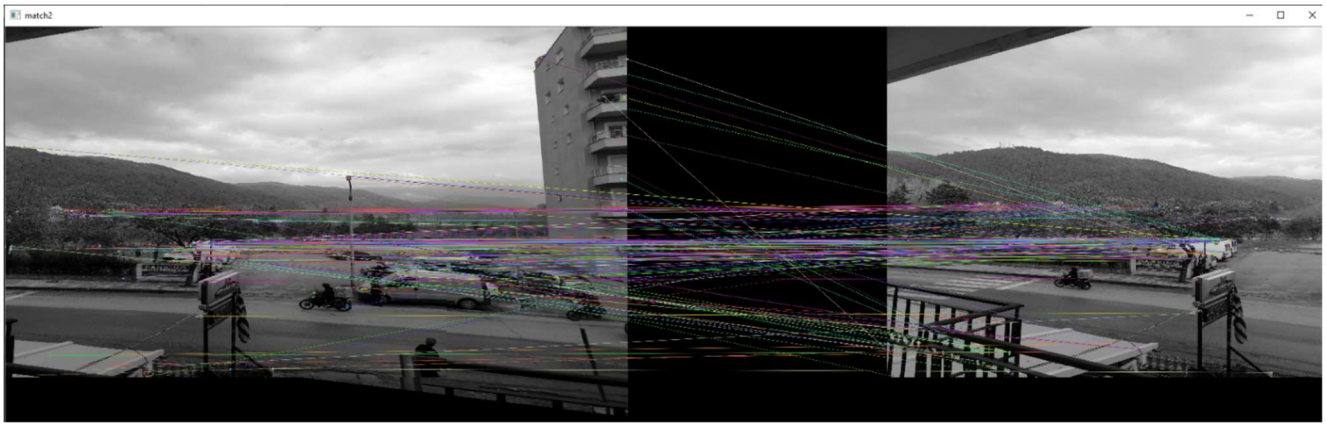


Image 42 Matching keypoints of previous panorama(left) with image view2(right), using SIFT descriptor.

Image 43 Panorama image using the previously shown common keypoints.

Image 44 Keypoints for previous panorama image.

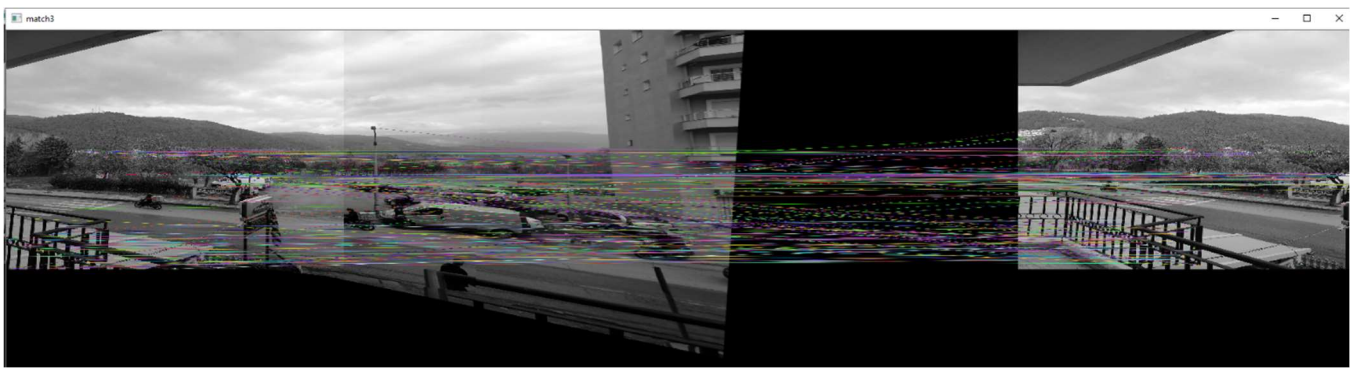


Image 45 Matching keypoints for previous panorama image(left) and image view3(right), using SIFT descriptor.

Image 46. Final panorama image, produced with SIFT descriptor.

The final panorama image, while with its imperfections, is an acceptable result. The imperfections are due to the fact that several objects, such as a moving motorbike, appear multiple times in the images. Furthermore, several discontinuities appear, for example on the right side of the panorama image. In contrast, the panorama created by ICE, seems to fix all the aforementioned problems.

Image 47. Panorama resulting from using ICE

The results using SURF descriptor follow, where the findings and the problems are the same as the ones using SIFT descriptor.

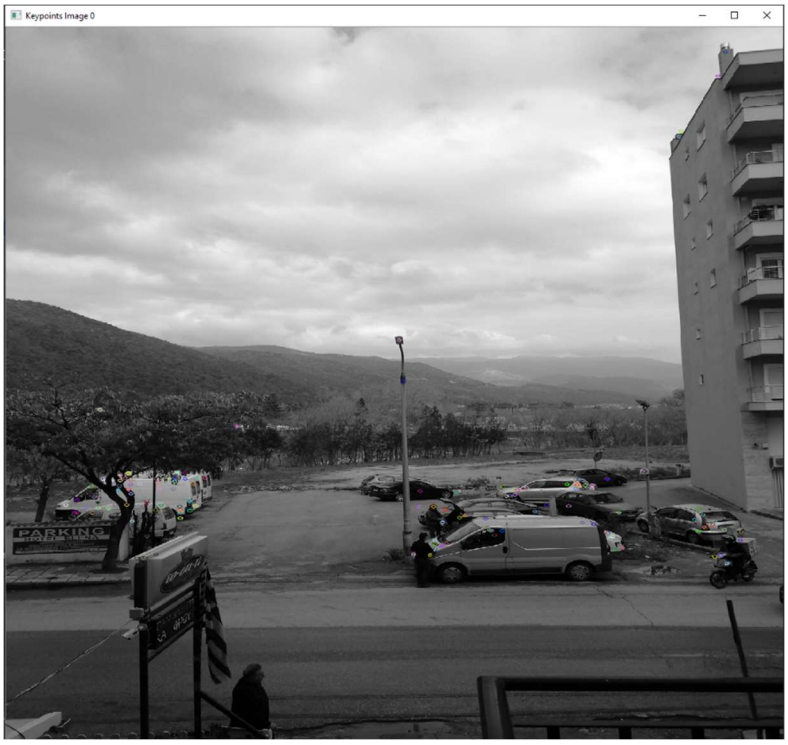


Image 48 Points of interest using SURF Descriptor, for image view0,

Image 49. Points of interest using SURF Descriptor, for image view1.

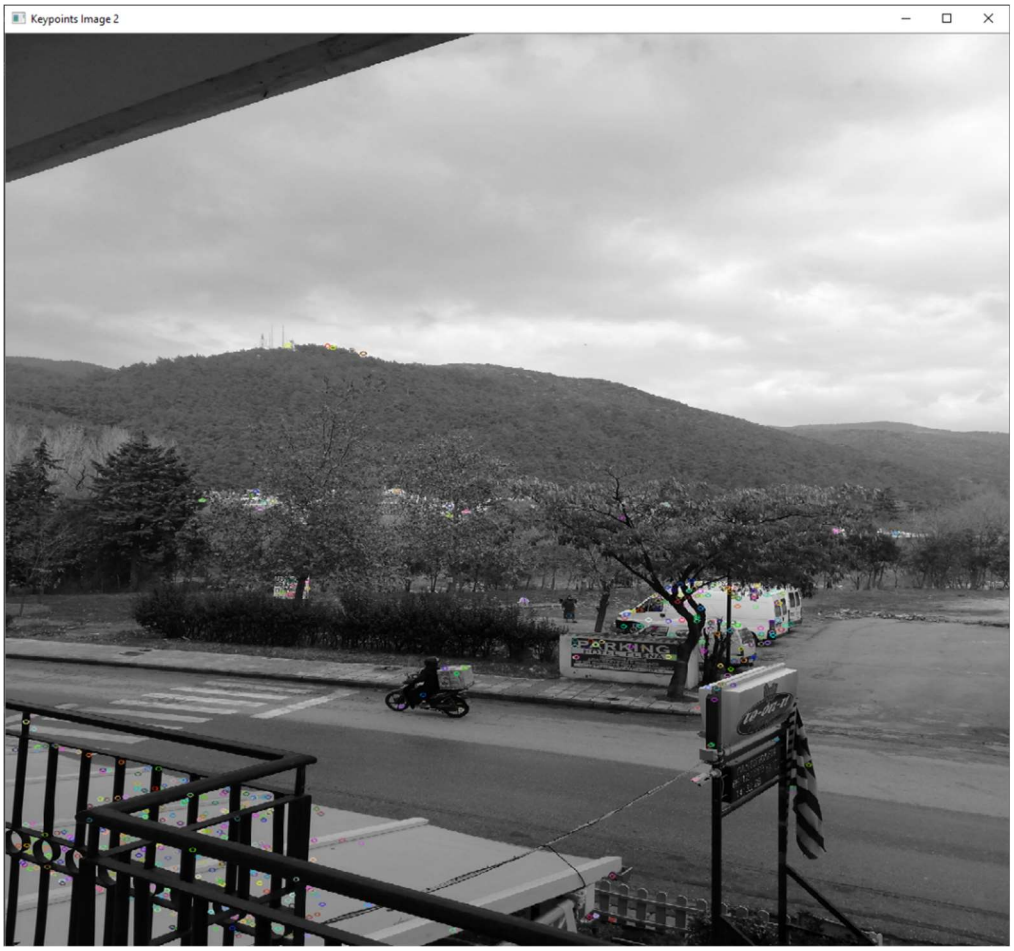


Image 50. Points of interest using SURF Descriptor, for image view2.

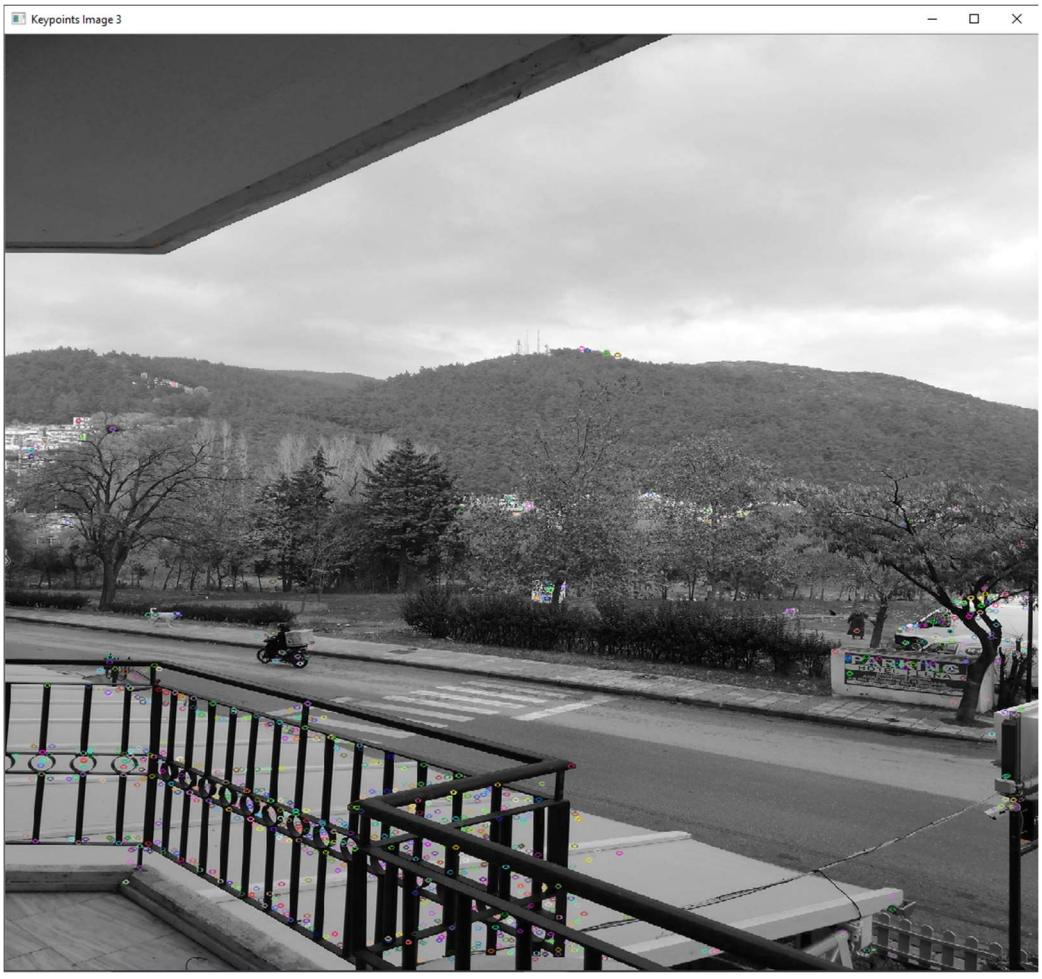


Image 51. Points of interest using SURF Descriptor, for image view3.

Image 52. Common keypoints for images view0(left) and view1(right) using SURF Descriptor.

Image 53. Panorama created using images view0 and view1.

Image 54. Points of interest using SURF Descriptor, for the previous panorama image.

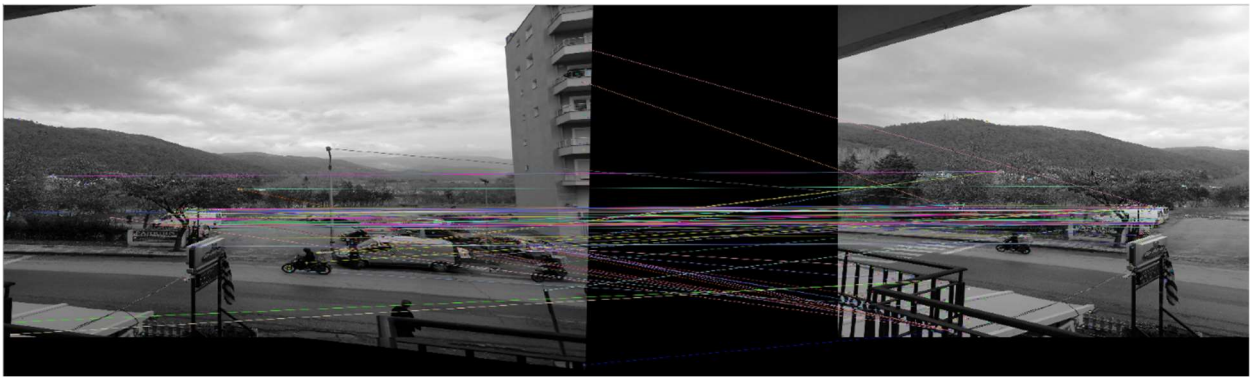


Image 55. Common keypoints for the previous panorama (left) and image view2 (right) using SURF Descriptor.

Image 56. Panorama created using images view0,view1,view2.

Image 57. Keypoints for previous panorama.

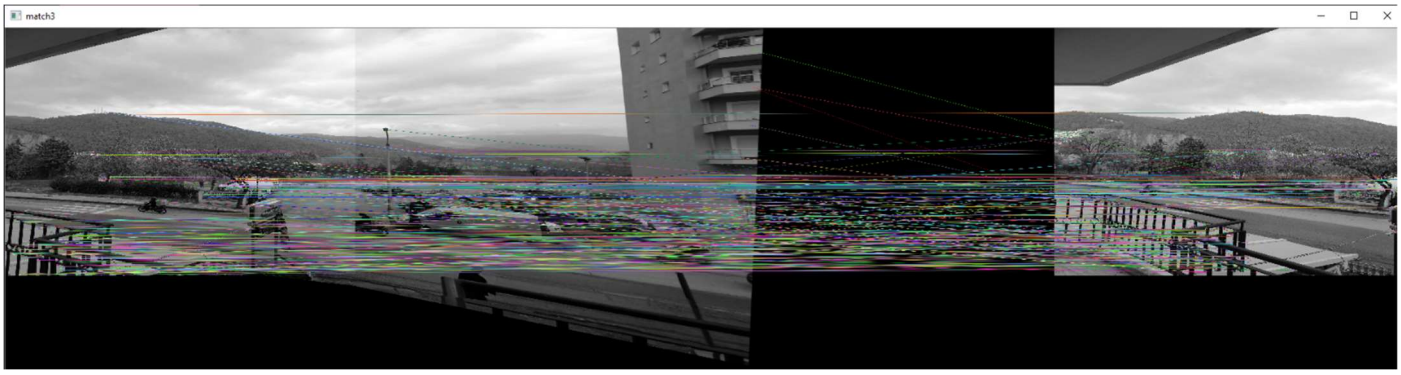


Image 58. Common keypoints for previous panorama image(left) and image view3(right).

Image 59. Final panorama image using SURF Descriptor.

In conclusion, both panorama images result in the same image, with almost identical mistakes and mismatches. Those mistakes could be rectified by using a different implementation or by using different images. However the end result is satisfactory, considering the objects depicted in these images.

- **References**
  - Richard Szeliski: Computer Vision Algorithms and Applications
  - NumPy Documentation : https://numpy.org/doc/stable/contents.html
  - OpenCV Documentation : https://docs.opencv.org/3.4/index.html
  - David G. Lowe : Distinctive Image Features from Scale-Invariant Keypoints - https://www.robots.ox.ac.uk/~vgg/research/affine/det_eval_files/lowe_ijcv2004.pdf