Image Processing - Exercise 4

Shachar Cohen, shahar.cohen1_, 206532418

Introduction

The goal of this exercise is to implement and evaluate various image alignment methods, including global alignment (e.g., applying uniform transformations across the entire image) and feature-based alignment (e.g., utilizing detected keypoints). The objective is to assess the effectiveness of these methods in different settings, such as dynamic scenes or varying viewpoints, and to analyze their strengths and limitations in practical scenarios.

For feature-based alignment, the primary methods include Harris Corner Detection paired with MOPS or SIFT, while for global alignment, the focus is on Lucas-Kanade, Normalized Cross-Correlation, or regular Cross-Correlation techniques. The exercise involves creating dynamic stereo mosaics and changing viewpoint mosaics by stitching aligned frames strips to display different object angles or temporal variations of the same object.

Although precise blending of strips from each frame is not required, various optimal stitching methods discussed in class can enhance the output. These include min-cut (which identifies transitions with minimal pixel differences along the cut), the dynamic cut approach (which optimizes transitions under specific restrictions of the cut shape), and pyramid blending (which produces more natural-looking blends).

Implementation Details

1. **Finding The Transformation Between Consecutive Frames**

To align the images, I opted for feature detection, even though the global transformation approach might seem more intuitive for this exercise. The decision was based on performance, as feature detection produced better results. I began by blurring both images to minimize noise and then used SIFT to detect features in both images. These features were matched using the 1-nn-to-2-nn ratio test. By setting a relatively low threshold for the ratio, I obtained fewer matches, but they were of higher quality (as shown in the image below). Once the matches were identified, I applied RANSAC to determine the best transformation between the images. In each iteration, I randomly selected two matches and calculated the transformation matrix as follows: I computed theta as the angle between the lines formed by the sampled points in each image, rotated the points in the first image by theta, and then calculated the average tx and ty by subtracting the coordinates of the rotated points in first image from the rotated coordinates of the second image.

Feature matching results

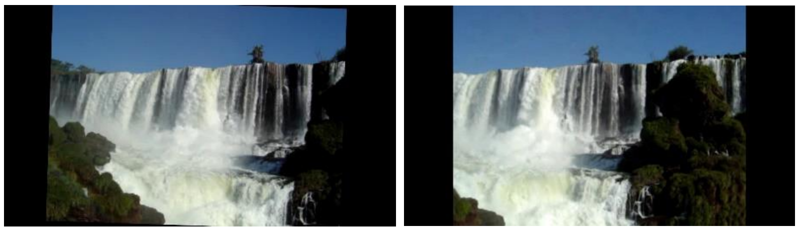
```
Algorithm 1
Input: Two input images, nn threshold
Steps:
1. Blur both images to reduce noise and improve feature detection.
2. Detect edge coordinates in both images and compute their descriptors using SIFT.
3. For each edge coordinate in the first image, find the two nearest neighbors in the second image by comparing descriptor distances.
4. Apply the nearest neighbor ratio test: if the ratio of the distance to the first nearest neighbor and the second nearest neighbor is below the threshold, add the first nearest neighbor and the corresponding coordinate to the matches array.
5. Use RANSAC to compute the rigid transformation between the images:
        Randomly select two matches.
        Calculate the transformation parameters (e.g., rotation and translation) based on these points.
6. Repeat RANSAC to identify the transformation that maximizes inliers and best aligns the images.
```

2. Finding the Transformation Between Each Frame and The Reference Frame

This algorithm computes cumulative transformation matrices and horizontal translation offsets tx to align frames in a sequence relative to a chosen reference frame j*. It processes frames in two parts: frames before j* (moving right-to-left) and frames after j* (moving left-to-right). For each frame, it calculates the transition matrix to the adjacent frame, extracts and stores the tx value, zeros out the tx in the matrix, and updates the cumulative transformation by multiplying with the current matrix. The results for frames before j* are reversed to maintain proper order, and the identity matrix is added for the reference frame. The final output consists of cumulative transformation matrices and the corresponding tx offsets for all frames relative to j*. After calculating the matrices the frames are aligned to the reference frame j* using backward warping with bilinear interpolation with the inverse of the corresponding calculated matrix.

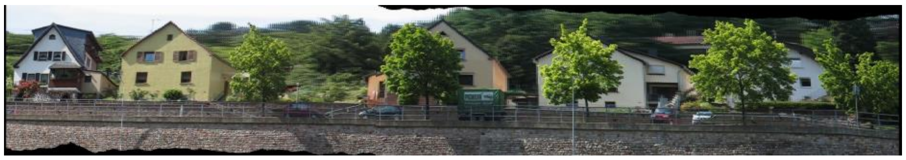
```
Algorithm 2
Input: array of frames A, reference frame index j*
Steps:
Right ← empty array
Right_tx ← empty array
Current_matrix ← identity matrix of size 3*3
For each index i in [0,j* - 1] inclusive in reversed order:
        Matrix ← computed transition matrix from A[i] to A[i + 1] (using the first algorithm)
        Append the absolute value of element in the 3'rd column and 1'st row (|tx|) to right tx
        Set the element the 3'rd column and 1'st row (tx) in Matrix to be zero
        Current_matrix ← Matrix @ Current_matrix
        Append current matrix to Right
```

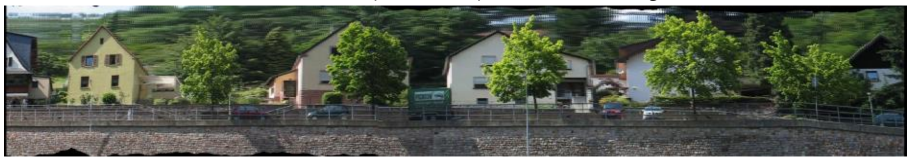
```
Reverse Right and Right_tx
Left ← empty array
Left_tx ← empty array
Current_matrix ← identity matrix of size 3*3
For each index i in [j*+1, number of frames - 1] inclusive:
        Matrix ← computed transition matrix from A[i] to A[i - 1] (using the first algorithm)
        Append the absolute value of the element in the 3'rd column and 1'st row (|tx|) to left tx
        Set the element the 3'rd column and 1'st row (tx) in Matrix to be zero
        Current_matrix ← Matrix @ Current_matrix
        Append current matrix to Right
Return Right + [identity matrix] + Left, right_tx + left_tx + [0] (aligned frames and movement between consecutive frames)
```

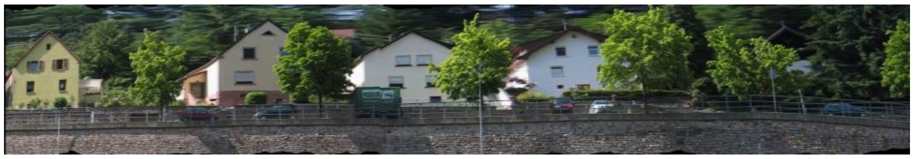

Alignment result (right – frame 0 aligned by Y and rotation parameters to frame 50, left - frame 50)

3. Creating An Array of Dynamic Panoramas

This algorithm constructs a panorama by sequentially placing vertical strips from aligned frames onto a canvas. It determines the left and right gaps for each frame based on the translation offsets tx - representing shifts from the previous frame to the current frame and from the current frame to the next frame, respectively - and extracts corresponding vertical strips. These strips are positioned on a black canvas, whose width accommodates the total frame widths plus the cumulative tx offsets. The result is a stitched panorama that showcases portions of each aligned frame, with the center of the strips determined by the desired viewpoint. For instance, if j* is placed on the right side of the image, the panorama presents the scene from a left viewpoint, and vice versa. I used this algorithm both for the dynamic panorama stereo and the viewpoint panorama stereo.


Panorama – Left Perspective (strips taken from the right side)


Panorama – Central Perspective (strips taken from the center)



Panorama – Right Perspective (strips taken from the left side)

Algorithm 3 – creating a specific panorama given some center index

```
Input: center index j*, aligned frames, tx array from algorithm 2, width, height
Panorama ← empty canvas (black pixels) of size [width + sum(tx), height]
i = j*
for each idx, frame in enumerate(aligned frames):
        left_gap =  0 if idx == 0 else tx[i-1] // 2
        right_gap = tx[i] // 2
        band ← the vertical strip from (j* - left_gap) to (j* + right gap)
        set the vertical strip from i to (i + left_gap + right_gap) in the panorama to store the values of the band
        i += left_gap + right_gap
return the panorama
```

Algorithm 4 – creating a sequence of panoramas

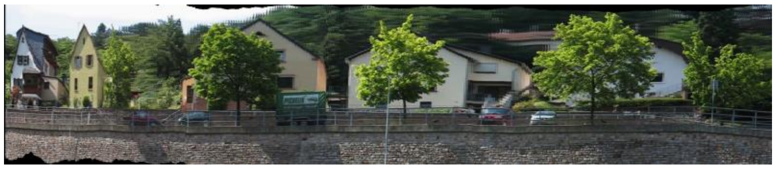
```
Input: frames, width, height
Transition_matrices, tx ← algoritm 3 output with the following inputs: frames and len(frames) / 2
Warp all frames according to their corresponding matrix
Possible centers ← all indices from max(tx) to width – max(tx)
Create a panorama for each center index using algorithm 3
Export the panoramas to a video keeping their order.
```

4. **Entire final algorithm - creating The Stereo (viewpoint / dynamic) –** export the sequence of panoramas into a video. To create a dynamic stereo, reverse the panoramas array.

I implemented all the mentioned algorithms from scratch, with the exception of the SIFT edge detector, backward warping, and feature matching, for which I utilized the OpenCV library's implementation. My implementation steps are mentioned above, in the algorithms section.
My algorithms contain several hyper parameters:

1. RANSAC number of iterations: The maximum number of iterations for the RANSAC algorithm to converge to a solution. I set this hyper parameter to 1000, which is relatively high, to ensure I find the best transformation.
2. RANSAC inlier threshold - The maximum allowable distance between points for them to be considered inliers during the RANSAC process. I set this hyper parameter to 1 which is low enough to exclude outliers yet doesn't filter out too many points.
3. Nearest neighbor distance ratio - The ratio used to filter feature matches, ensuring the best match is significantly closer than the second-best match. I used 0.75.
4. Blurring factor – before finding the features in each image, I blurred the images with a gaussian kernel (sigma = 3) to clean noises (false positive edges).

One of the main challenges I faced was dealing with the parallax effect: closer objects in the picture move faster while distant object move slower. Consequently, if a significant number of edges were detected on either closer or more distant objects, the calculated x translation factor could become biased. This effect was especially prominent in the boat video – most of the features were taken from close objects. I therefore tried to eliminate this effect by calculating the features based only in the more distant objects – the upper part of the frames. The result turned out pretty bad:

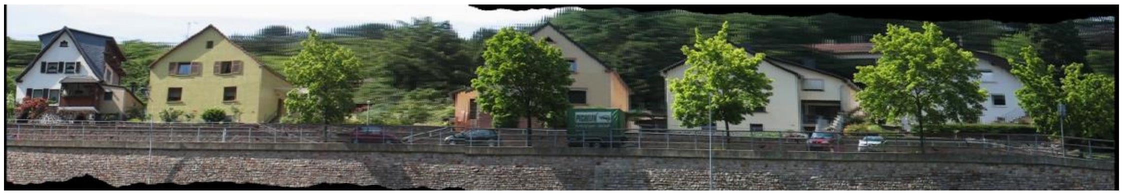


So I stayed with the original method.

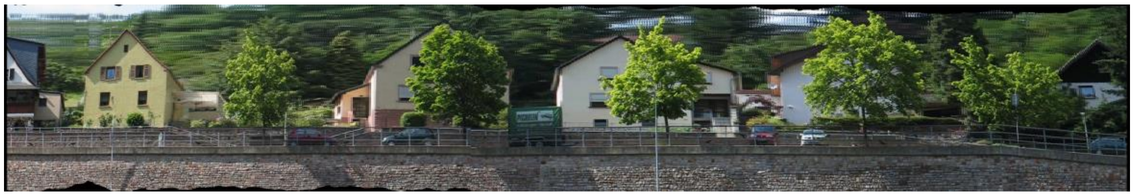
Another challenge was having too many feature points in the image – which made the calculations slower and the transformations less precise. To deal with this problem, I blurred the image – only the most prominent features were left.

Visual Results

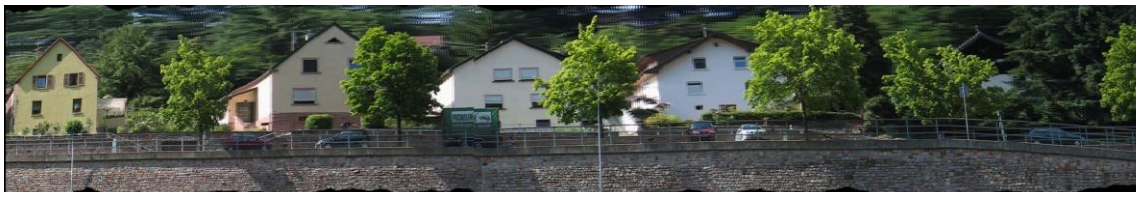
<u>Changing Viewpoint Results:</u>



Panorama – Left Perspective (strips taken from the right side)



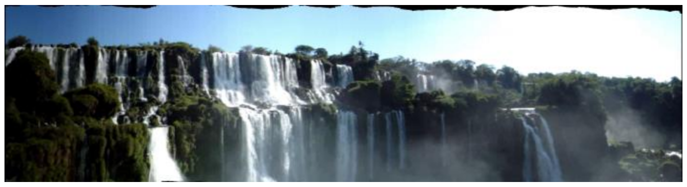
Panorama – Central Perspective (strips taken from the center)

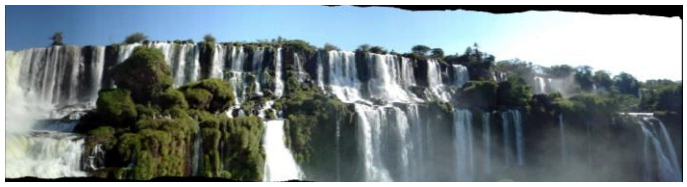


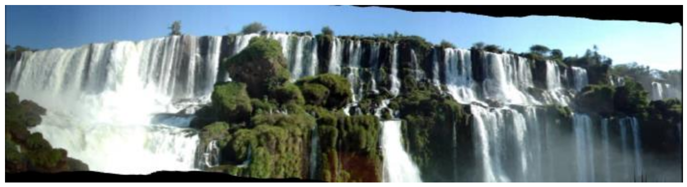
Panorama – Right Perspective (strips taken from the left side)

As the visual results exemplify, the alignment between consecutive frames isn't perfect. That happens because the video has parallax effect – closer objects move faster than distant ones. Since SIFT finds many feature points in the bricks and leaves, the wall and trees are aligned almost perfectly while the houses appear a bit wider than they actually are. Yet, the Y and rotation parameters are calculated quiet precisely as demonstrated in these panoramas.

Dynamic Panorama Results:

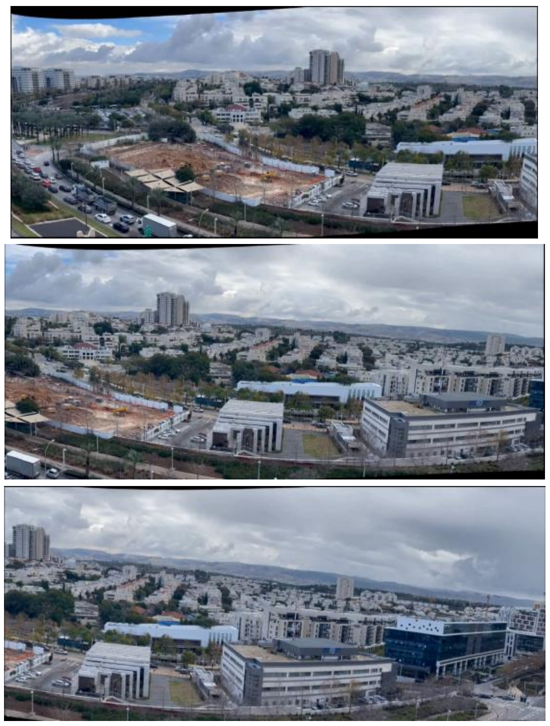

First panorama


Middle Panorama


Last Panorama

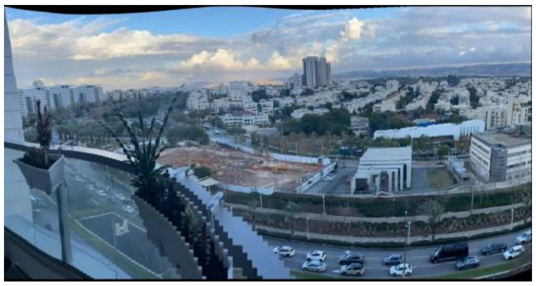
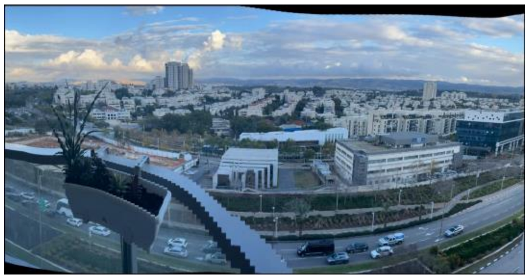
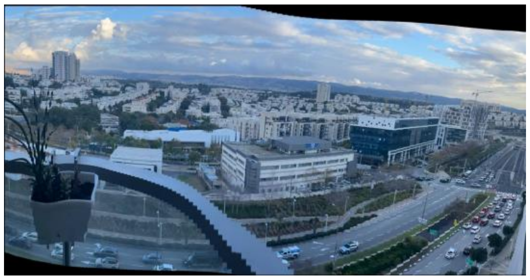
Here the alignment between consecutive frames improves as the camera rotates around the eyelet (=no parallax).

Good Result:



The video fulfills the assumptions of our panorama creator model – there is a big overlap between two consecutive frames, the frames contain (many) detectable features, the transformation between two consecutive frames is approximately rigid.

Bad Result:

In this scenario, the failure occurs because scaling is required to transition between consecutive frames, violating the rigid transformation assumption of the panorama model. The created panoramas look distorted.

Conclusion

This exercise highlighted the strengths and limitations of global and feature-based image alignment methods in creating panoramas for dynamic scenes and changing viewpoints. Feature-based alignment, using SIFT and RANSAC, proved robust for significant transformations but struggled with repetitive patterns, low-texture areas, and parallax effects, where closer objects moved faster than distant ones, causing distortions. Attempts to mitigate parallax by focusing on distant objects were ineffective, underscoring the need for depth-aware models. Failures often arose with scaling transformations and excessive parallax, which violated rigid transformation assumptions. Blurring images to reduce feature overabundance improved computation efficiency and precision. While the methods worked well for ideal videos, challenges with "bad" videos revealed the need for depth integration and advanced blending to enhance performance in complex real-world scenarios.