UNIVERSITÀ DEGLI STUDI DI VERONA

Automatic 2D Mosaicing

COMPUTER ENGINEERING FOR ROBOTICS AND SMART INDUSTRY

Nicola Marchiotto - VR462317

2022

Contents

1 Introduction

The goal of the project is construct a software application for the generation of automatic 2D mosaic from a set of 2D images related by an homography. The procedure is also known in literature as 2D image stitching. The software must be able to automatic estimate the correspondences between the images, implement a robust estimation of the homography transformation and deal with colour blending.

1.1 Algorithm overview

The implemented algorithm is described by the following steps

- An image is arbitrary chosen as the starting one
- The starting image is augmented with a black contour, this to create image space where the chosen warped image to stitch will be placed. The starting image is then compared with all the remaining images in the dataset for extracting salient points
- The image with the largest number of correspondences is chosen as the image to stitch to the starting one
- The homography transformation matrix is robustly estimated with Ransac and the image is stitched to the starting one
- Colour blending operations are performed on the image resulted from the stitching operation
- The output image becomes the starting one, the procedure is repeated until no images are left in the dataset
- If during an iteration, the largest correspondences found are less than 80, number chosen heuristically, the mosaic algorithm is interrupted and a new execution is started with the dataset composed with the remaining images of the previous execution

2 Conjugate point extraction

To automatically detect the conjugate points from 2 given images, the Scale Invariant Feature Transform, SIFT, was exploited. This technique detects salient points using a Laplacian of Gaussian filter to the Scale Space. The method is invariant to scale, rotation, illumination and point of view. The SIFT algorithm is based on two main ideas: *Scale Space* and *Laplacian Pyramid*.

The *Scale Space* of an image is a function $L(x, y)$ that is obtained by a convolution of a Gaussian kernel (at different scales) with the input image:

$$L(x, y, \sigma) = G(x, y) * I(x, y)$$

The idea is that different blur, with a following binarization, shows features of different dimension, according to the Gaussian windows size σ.

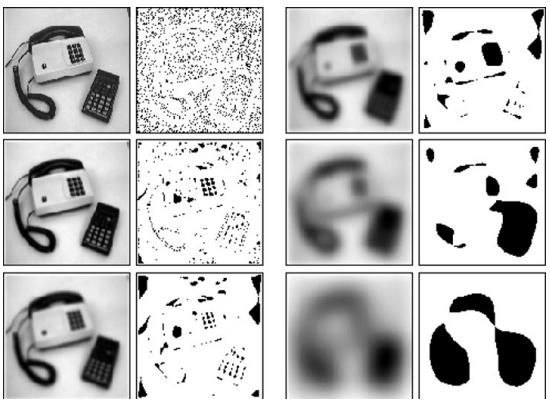

Figure 1: Scale space

Figure 2: Laplacian Pyramid

A *Laplacian pyramid* is a set of images equal to the original one, but at every step the dimension is reduced by half.

The idea is that using images of different size, feature of different dimensions are underlined.

These two approaches are then put together to form the Laplacian of gaussians *LoG*

LoG = Gaussian blur + edge detection with Laplacian

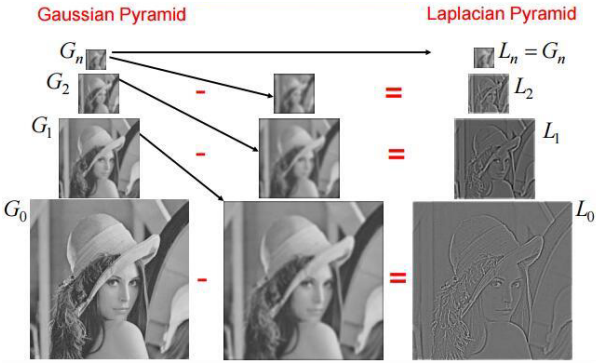

Figure 3: Laplacian Pyramid for edges extraction

The *LoG* kernel is convoluted with the image to find discontinuities i.e. edges, so points which have second derivative equal to zero.

The Difference of Gaussian filter is used to approximate the Laplacian of Gaussian filter:

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y)$$
$$= L(x, y, k\sigma) - L(x, y, \sigma)$$

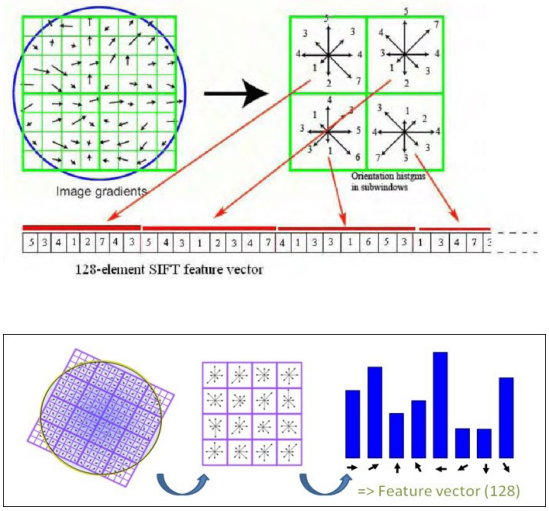

Figure 4: Point Descriptors

Once the salient points have been found, it's necessary to uniquely identify them using a signature.

At each salient point is associated an orientation. This is done centring a $16x16$ window on each salient point, the histogram of gradients is then computed in this region.

The point descriptor, the signature for the point, is then computed. This is a 128-vector obtained from the concatenation of the 16 histograms of gradients!

Finally the point matching is obtained from the comparison of the descriptors

Salient points matches were computed using FLANN (Fast Library for Approximate Nearest Neighbors), which allows to perform approximate nearest neighbor searches in high dimensional spaces very easily.

From the various algorithm offered by the library, KNN was chosen to search correspondences between the images.

Figure 5: Point Matching

3 Homography Computation

To compute the homography matrix, the opencv library was exploited. This matrix is the perspective transformation H between the source and the destination image, computed by assuming that the two images are related by a common plane.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

The homography computation requires only four pairs of points theoretically. Since these points can be affected to error, given that they are automatically extracted, the matrix was computed using RANSAC.

The method iteratively computes the homography using a subset of the extracted pairs to minimize the projection error, until a threshold error or a maximum number of iteration is reached. The following equation represent the projection error to be minimized during the estimation procedure

$$\sum_i \left[\left(x' - \frac{h_{1,1}x_i + h_{1,2}y_i + h_{1,3}}{h_{3,1}x_i + h_{3,2}y_i + h_{3,3}} \right)^2 + \left(y' - \frac{h_{2,1}x_i + h_{2,2}y_i + h_{2,3}}{h_{3,1}x_i + h_{3,2}y_i + h_{3,3}} \right)^2 \right]$$

4 Colour Adjustment Operations

After warping the chosen image to stitch to the starting one, colour correction operations are necessary on the overlapping region, since we can't simply add the intensity level of the two images pixel by pixel.

Figure 6: Without colour blending

The implemented method consists in computing two weights for our images and the apply a weighted sum of these. The formula is the following

$$I_{blend} = \frac{I_1 * w_1 + I_2 * w_2}{w_1 + w_2}$$

w_1 and w_2 are matrix of weights and are computed using the `distance_transform_edt()` function of the python scipy library.

The multiplication $I_i * w_i$ is like applying a Gaussian weight to the image, since the `distance_transform_edt()` return a matrix mask which gives more importance to the pixel in the center of the image and less to the ones at the edges.

Figure 7: With colour blending

5 Results

5.1 Andalo

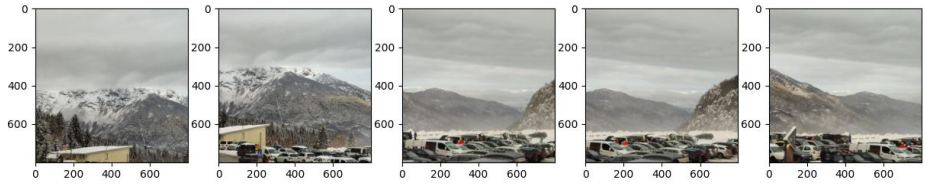

Figure 8: Some of the used images

Figure 9: Result

5.2 Arena

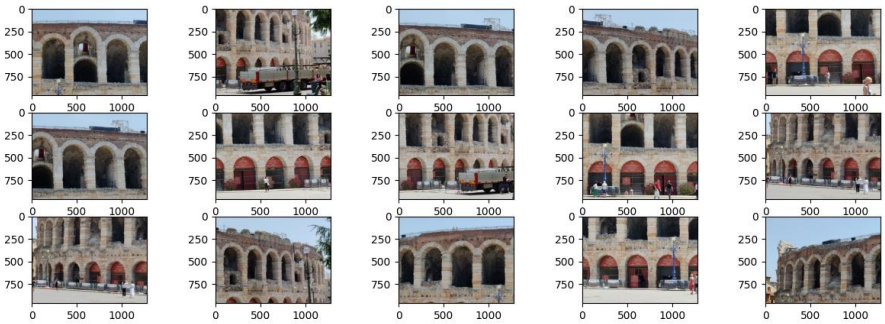

Figure 10: Some of the used images

Figure 11: Result

5.3 Mansion

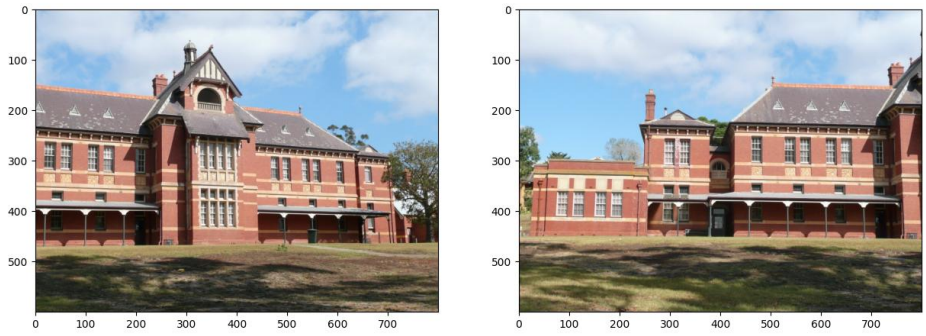

Figure 12: Some of the used images

Figure 13: Result

5.4 Bridge

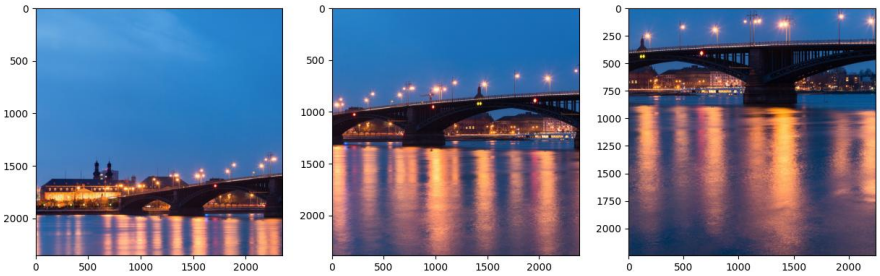

Figure 14: Some of the used images

Figure 15: Result

5.5 Building Site

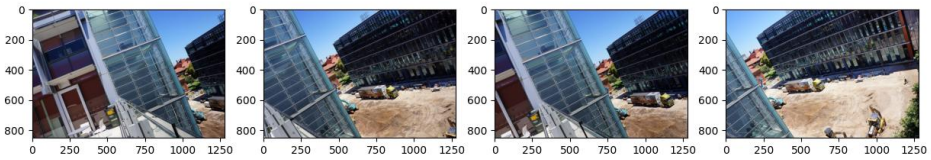

Figure 16: Some of the used images

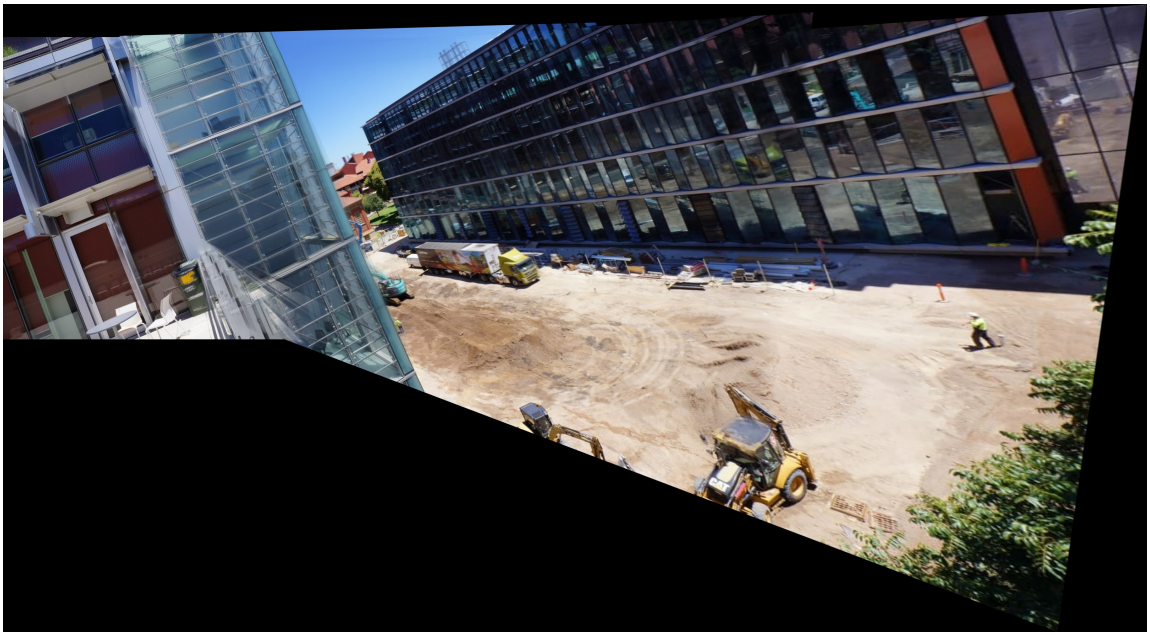

Figure 17: Result

5.6 Hills Panorama

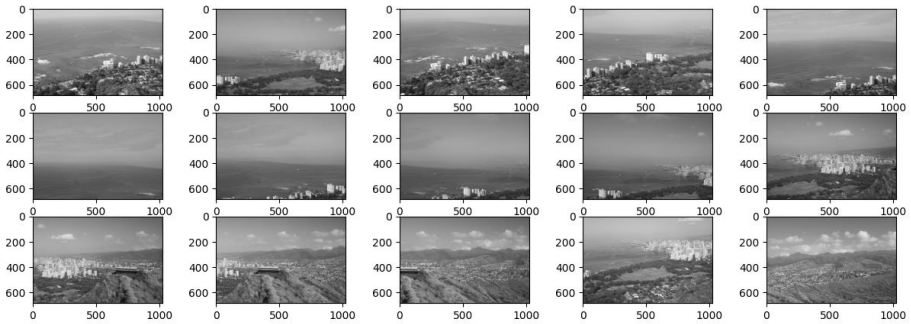

Figure 18: Some of the used images

Figure 19: Result

5.7 Fish Bowl

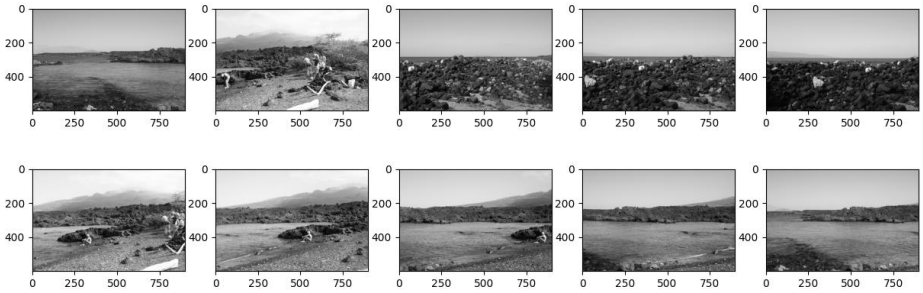

Figure 20: Some of the used images

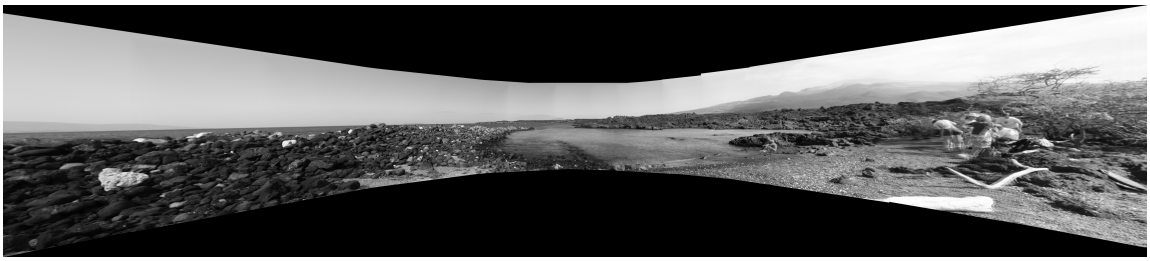

Figure 21: Result

5.8 Golden Gate

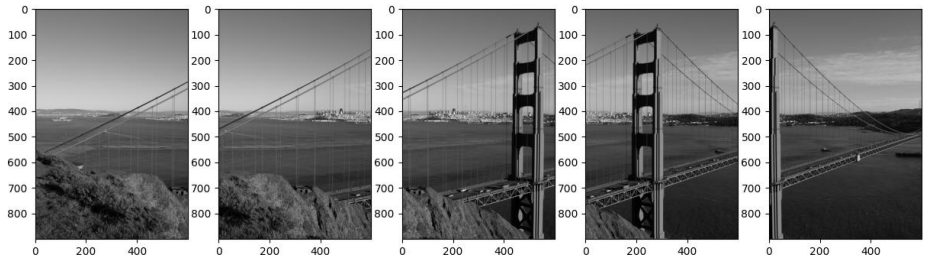

Figure 22: Some of the used images

Figure 23: Result

5.9 Half dome

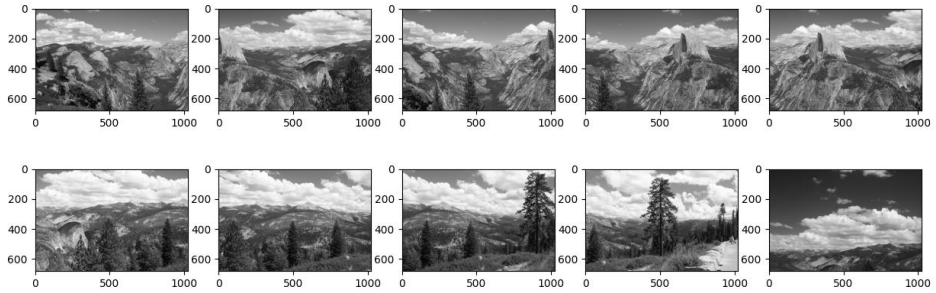

Figure 24: Some of the used images

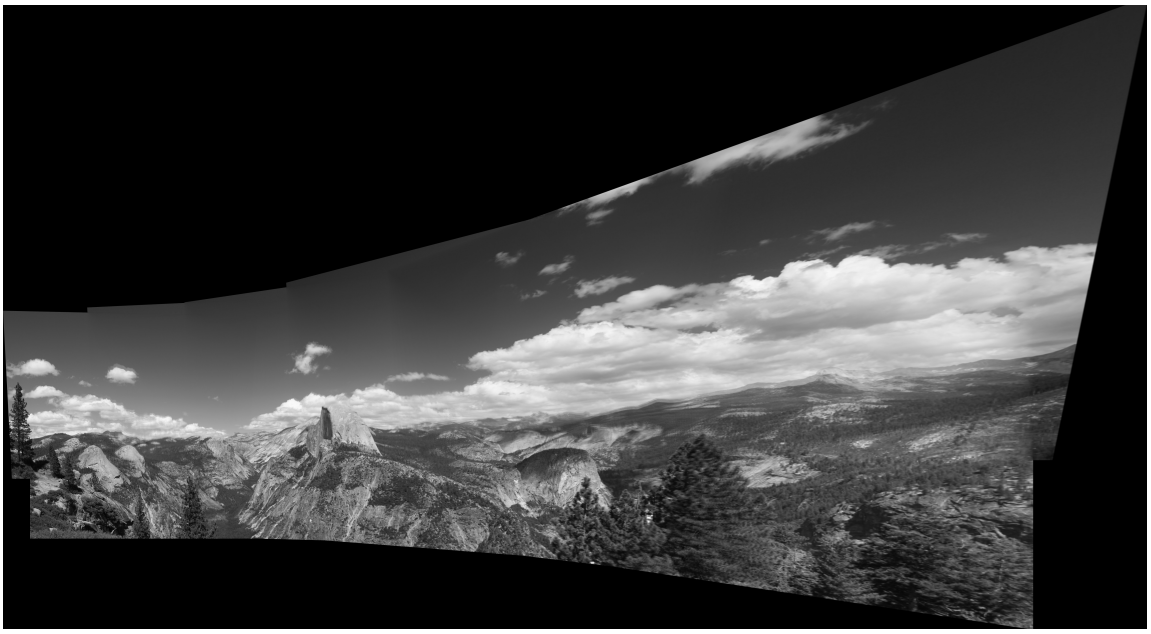

Figure 25: Result

5.10 Beach Hotel

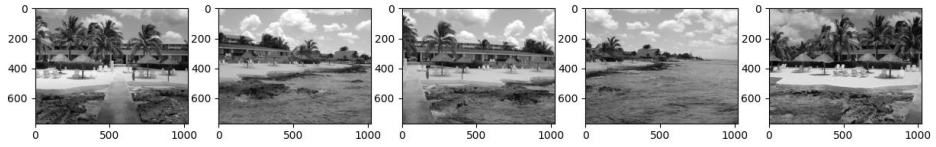

Figure 26: Some of the used images

Figure 27: Result

5.11 Mountain Landscape

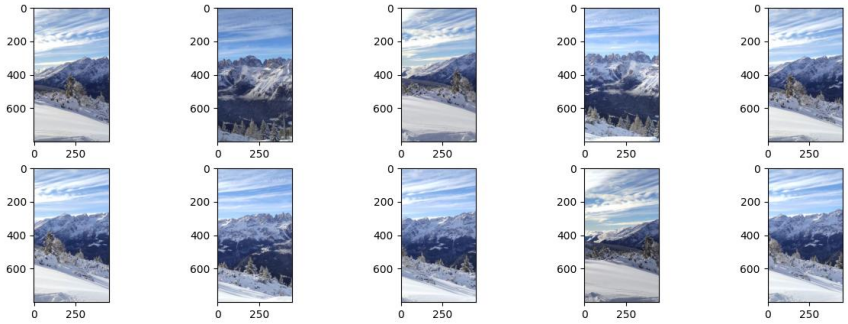

Figure 28: Some of the used images

Figure 29: Result

5.12 Office

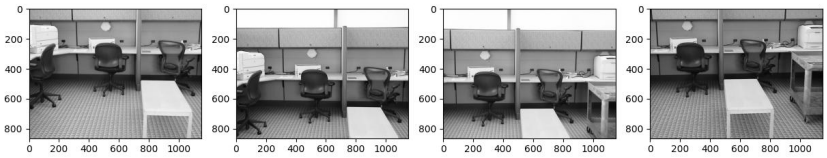

Figure 30: Some of the used images

Figure 31: Result

5.13 Ponte Nuovo

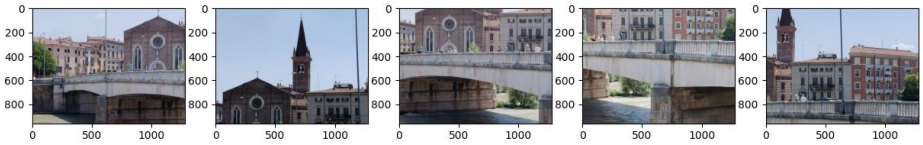

Figure 32: Some of the used images

Figure 33: Result

5.14 Rio de Janeiro

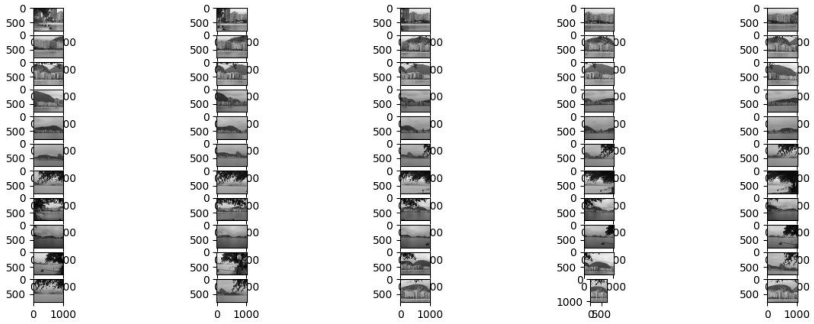

Figure 34: Some of the used images

Figure 35: Result

5.15 River

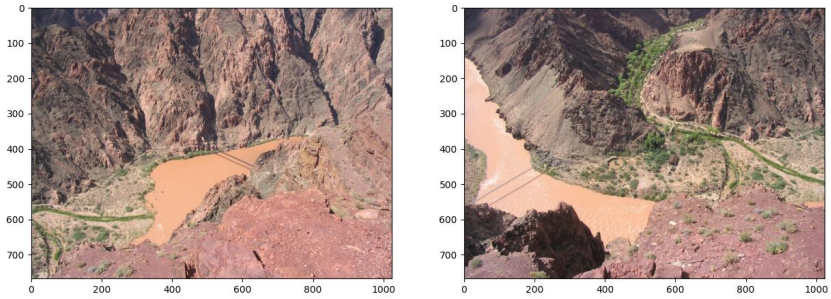

Figure 36: Some of the used images

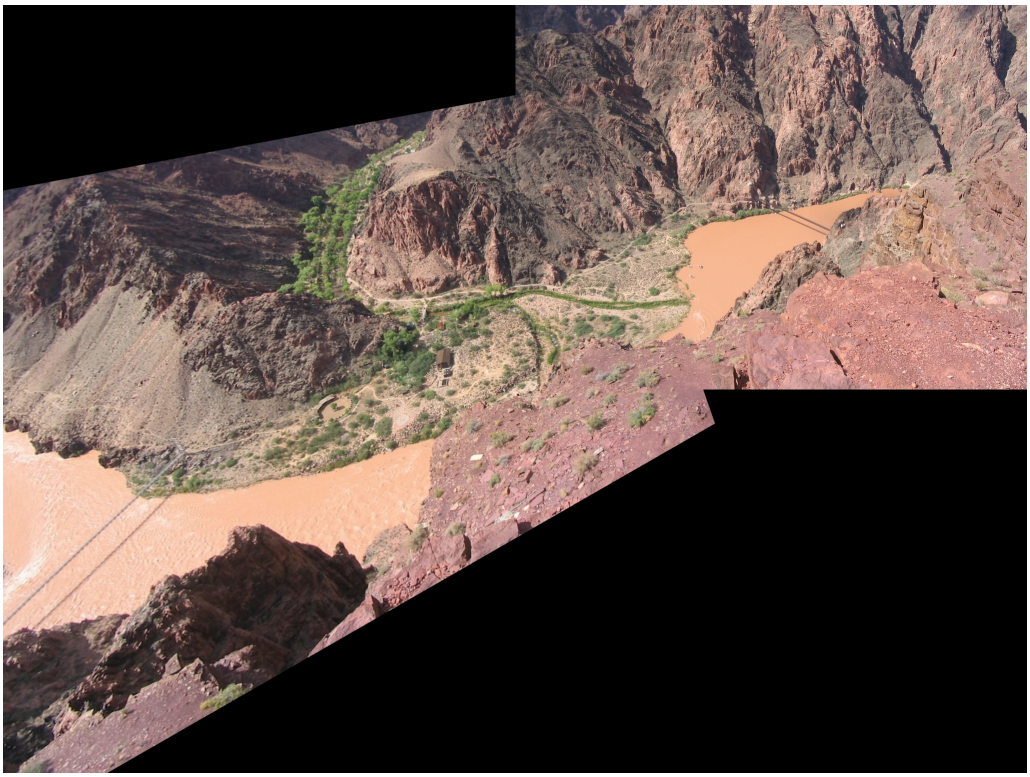

Figure 37: Result

5.16 Roof

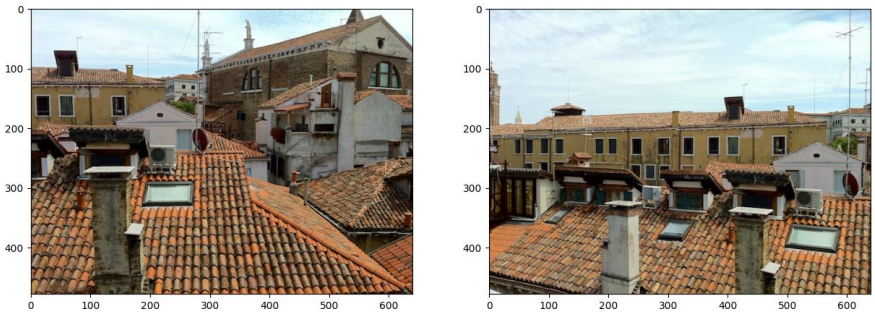

Figure 38: Some of the used images

Figure 39: Result

5.17 San Pietro

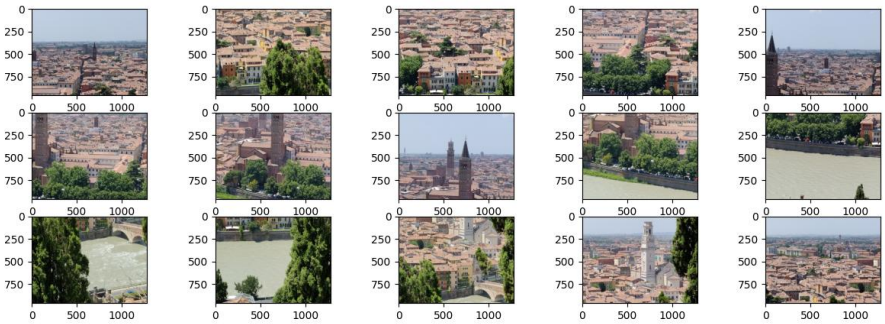

Figure 40: Some of the used images

Figure 41: Result

5.18 Shangai

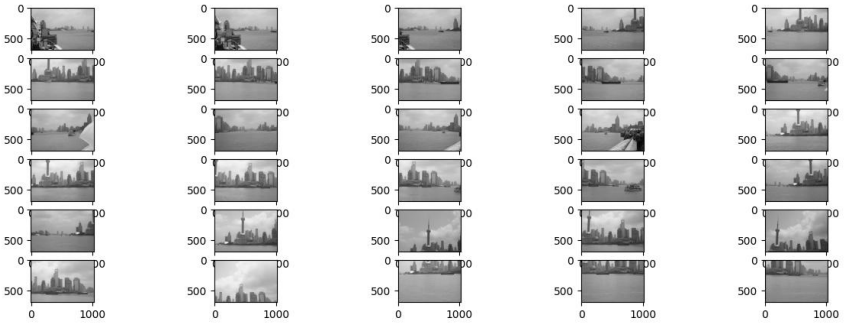

Figure 42: Some of the used images

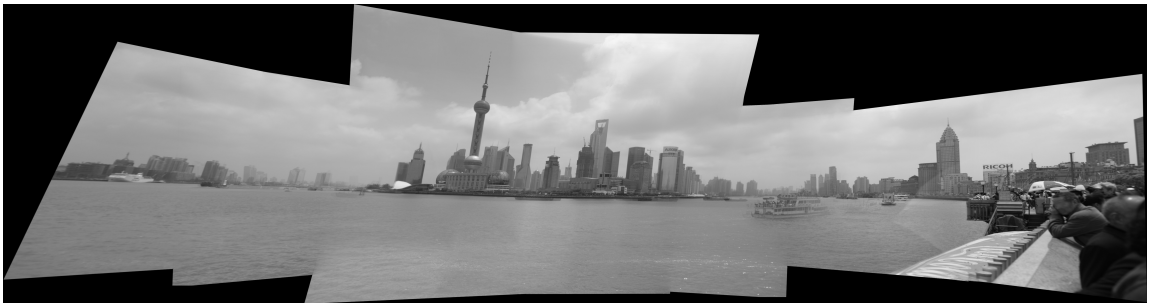

Figure 43: Result

5.19 Yard

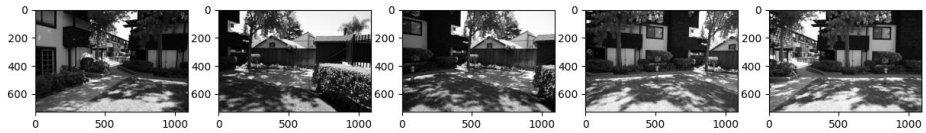

Figure 44: Some of the used images

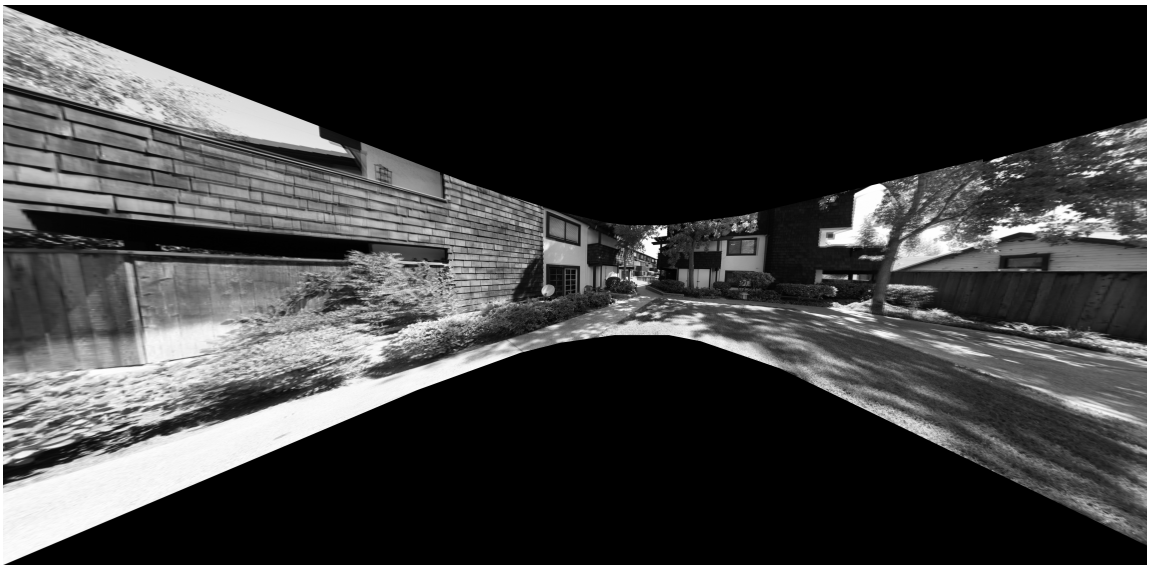

Figure 45: Result