Computer Vision Course **(Due: 18 June 2020)**

Panoramic Images Lab #4

Instructor: Prof. Pietro Zanuttigh *Name:* Elena Camuffo, *Id:* 1234370

Introduction

The purpose of this project is to obtain a panoramic image, starting from a set of images, that needs to be stitched. The task is achieved going through five main passages, described in the following, and in all these passages some variations are tried in order to obtain better results. Also some different datsets are used.

Setup

The initial setup provides to load the set of images which will form the panoramic one. The images are numbered from left to right and they will finally be stitched this way. An example of two consecutive images taken from the *kitchen* dataset are shown in figure 1. Other datasets are then employed to test the program, but we take this one as an example to see the most salient steps of the computing.

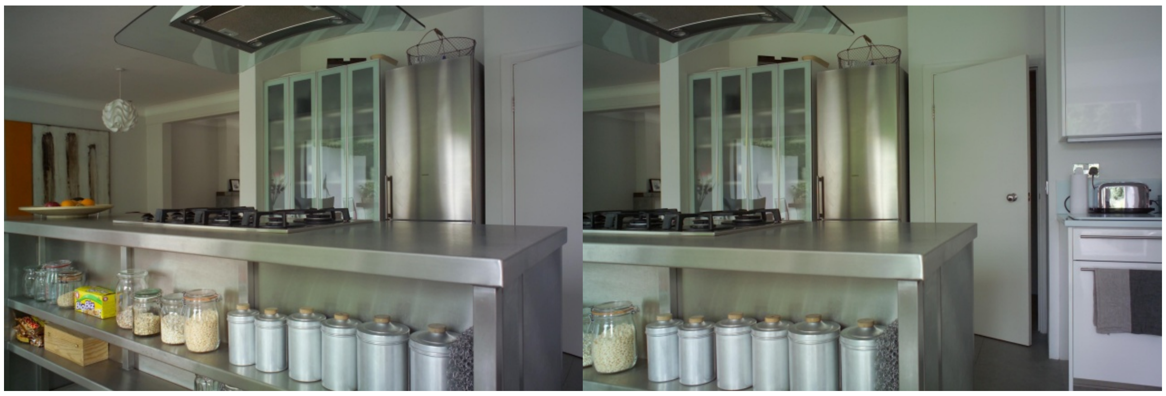

(a) Left (b) Right

Figure 1: Couple of consecutive original images

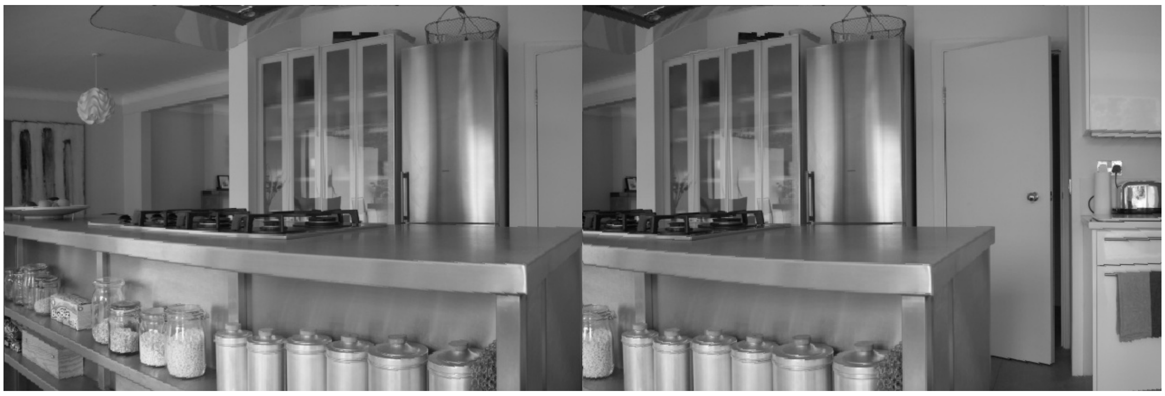
1. Cylindric Projection

Single Channel

The set of images is here trasformed by means of a cylindric projection, exploiting the function *cylindricalProj* provided. The parameter to keep in consideration in this step is the angle, i.e. the half field of view, and it depends on the dataset chosen. For the *kitchen* is set to $angle = 33°$.
The result is shown in figure 2. It is possible to notice that the two images, besides being greyscale, are even more similar and the effect of the angulation is fixed.

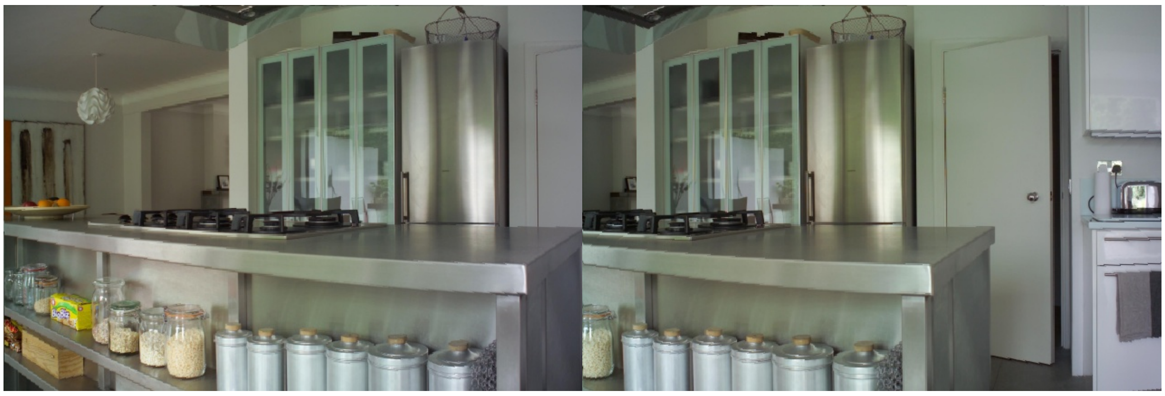
Multiple Channels

To improve the result, the colors were added modifying in some way the function *cylindricalProj* provided: instead of flattening to one channel the image, each channel is computed on its own and then they are merged again. The result is shown in figure 3 and the considerations are the same of the black and white image, but it is possible to do a best visive comparison w.r.t. the original images.
The following passages are all developed working on the color images.



(a) Left (b) Right

Figure 2: Couple of consecutive greyscale images obtained with cylindric projection



(a) Left (b) Right

Figure 3: Couple of consecutive color images obtained with cylindric projection

2. Features Extraction

The algorithm used to extract the keypoints and the associated descriptors is **ORB** (Oriented FAST and Rotated BRIEF), which is an efficient alternative to SIFT or SURF. It uses the FAST corner detector for feature extraction and adds rotation invariance to BRIEF.

3. Matching Features

Then the descriptors are computed for each couple of consecutive images, finding out the possible matches. A couple of images matched is shown in figure 4. It is possible to see that the matches found at this point are many, and a relevant amount of them are not good matches.

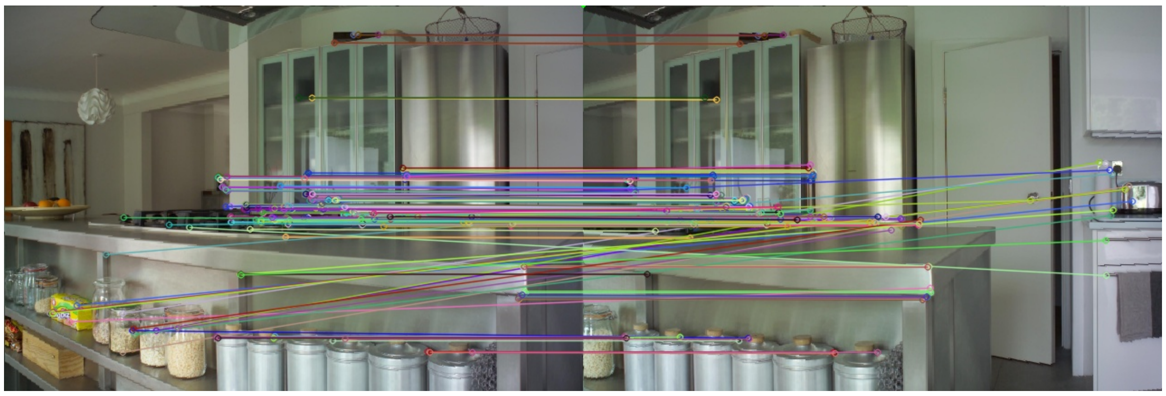


Figure 4: Matches found computing ORB descriptors

This way we refine the matches found by selecting only the matches with:

$$d < ratio \cdot d_{min}$$

where d is the distance beteen two matches, d_{min} is the minimum distance achieved by two matches and $ratio$ is a parameter which depends on the set of images chosen. For the example *kitchen* dataset is set to $ratio = 7$. The result is shown in figure 5 and we can notice that almost only relevant matches are still present.

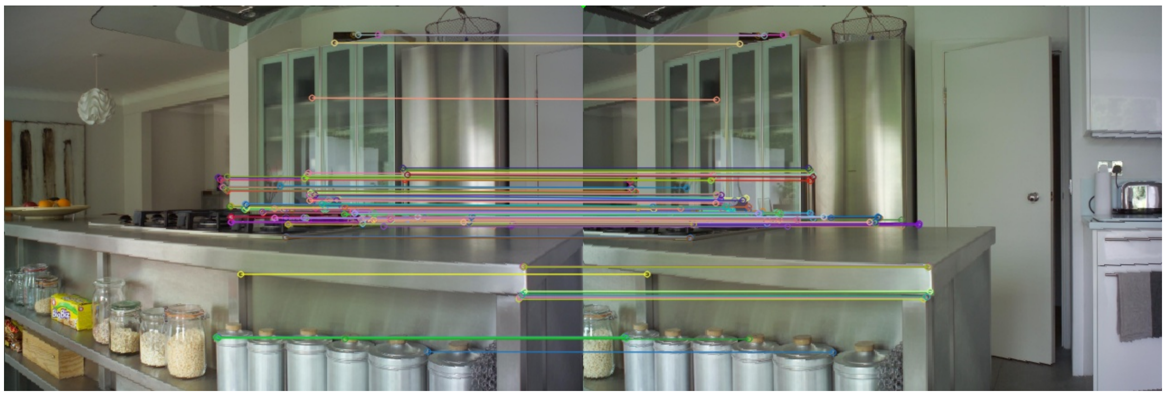


Figure 5: Good matches found applying threshoding

4. Translation Estimation

Once computed the matches, to stitch the images together we need to estimate the translations on the x and y axis of each image w.r.t. the previous one. The algorithm used here is **RANSAC** (RANdom SAmple Consensus).

RANSAC algorithm

The algorithm proceeds following these passages:

- Select a random match and get the corresponding shift on the x and y axis, i.e. Δx and Δy.
- Count how many matches are consistent with the selected one, applying a threshold on the difference from the estimated shift, i.e. the ones which satisfy $(|\Delta x_n - \Delta x| + |\Delta y_n - \Delta y|) < t_R$ for $n = 0, 1...N$, where N is the total number of matches and t_R is the threshold value, here set to $t_R = 25$.
- Iterate k times and keep the match with the largest compatible set.
- Compute the average shift $(\overline{\Delta x}, \overline{\Delta y})$ using only the compatible points.

The version implemented here below is a little modified w.r.t. the original version. It gives the possibility to choose whether the algorithm has to be randomized or not. In the version used here it is not, and the match is selected sequentially iterating along the vector containing all the good matches. However the procedure leads to very similar results.

```
/* FUNCTION TO COMPUTE RANSAC ALGORITHM */

Point2f PanoramicImage::computeRansac(vector<DMatch> good_matches, vector<KeyPoint> keypoints_src, vector<KeyPoint> keypoints_dst,
        int thresh, bool random, int max_iter) {

        int max_size = 0;
        int max_idx = 0;
        if (!random) max_iter = good_matches.size();

        for (int t = 0; t < max_iter; t++) {

                // Select a random correspondence and get the corresponding shift
                int j = t;
                if (random) int j = rand() % good_matches.size();
                DMatch selected_match = good_matches[j];
                Point2f shift = keypoints_src[selected_match.queryIdx].pt - keypoints_dst[selected_match.trainIdx].pt;

                // Count how many correspondences are consistent with the selected one
                vector<int> consistent_matches;
                for (int i = 0; i < good_matches.size(); i++) {
                        DMatch candidate_match = good_matches[i];
                        Point2f this_shift = keypoints_src[candidate_match.queryIdx].pt - keypoints_dst[candidate_match.trainIdx].pt;

                        //update best match index
                        if ((abs(shift.x - this_shift.x) + abs(shift.y - this_shift.y)) < thresh) {
                                consistent_matches.push_back(i);
                        }
                }

                // Update large compatible set index and size
                if (consistent_matches.size() > max_size) {
                        max_size = consistent_matches.size();
                        max_idx = j;
                }
        }

        //Keep the correspondence with the largest compatible set
        Point2f shift = keypoints_src[good_matches[max_idx].queryIdx].pt - keypoints_dst[good_matches[max_idx].trainIdx].pt;

        vector<Point2f> compatible_pts;
        for (int i = 0; i < good_matches.size(); i++) {
                DMatch candidate_match = good_matches[i];
                Point2f this_shift = keypoints_src[candidate_match.queryIdx].pt - keypoints_dst[candidate_match.trainIdx].pt;
                if ((abs(shift.x - this_shift.x) + abs(shift.y - this_shift.y)) < thresh) {
                        compatible_pts.push_back(this_shift);
                }
        }

        // Compute the average using only the compatible points
        Point2f average(0, 0);
        for (Point2f pt : compatible_pts) {
                average.x += pt.x;
                average.y += pt.y;
        }
        average.x = average.x / compatible_pts.size();
        average.y = average.y / compatible_pts.size();

        return average;
}
```

5. Image Stitching

The images are finally stitched together in the following way:

- The first image is simply pasted.
- The second one is shifted by Δx horizontally and by Δy vertically, w.r.t. the position of the first image.
- The procedure is repeated for the other images in the set, considering as a first image, the result of the stitching of all the previous ones.

The result of the stitching is shown in figure 6, and the program is finally tested on the other datasets: the *lab* dataset in shown in figure 13 and the *dolomites* is shown in figure 11.
It is possible to notice that the result is good, but there are some vertical lines (and horizontal also, especially if we consider the *dolomites* dataset which has many Δy shifts, fig. 11). These lines are due to the fact that the image stitching is made simply pasting the next image over the previous one.

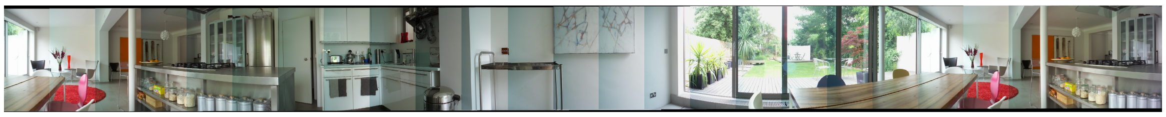
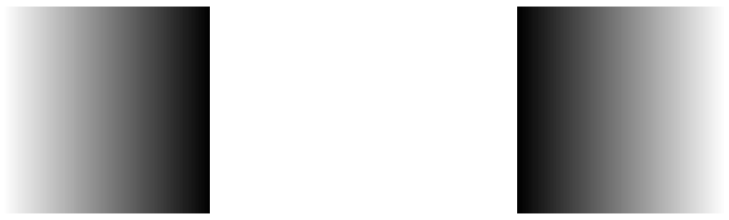


Figure 6: Panoramic image of the kitchen

Blending

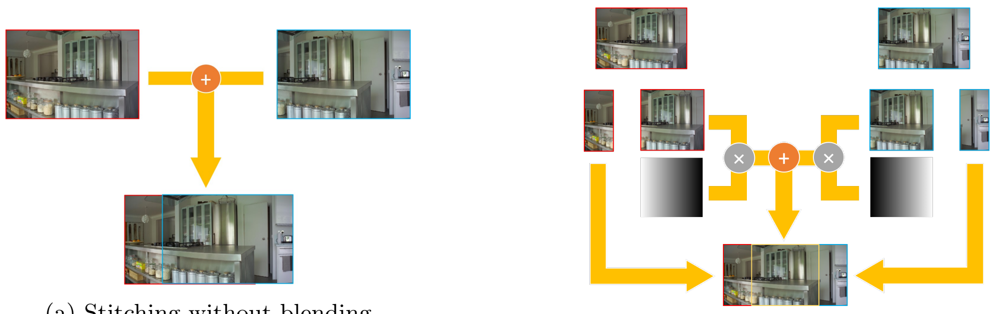
To improve the result compensating the vertical lines due to the stitching, a linear blending is used. To do that the following procedure (scheme of figure 8b) is employed:

- Apart from the absolute first image, the left image involved is split into two parts: the part which will be visible at all and the part which will overlap with the right image involved.
- The same procedure is repeated for the right image.
- The parts not involved in the overlap are pasted as they are in the result, while the right part of the first image and the left part of the second image (which will overlap), are weighted according to the linear blend masks of figures 7a and 7b respectively and then overlapped.



(a) Mask for right part of the first image (b) Mask for left part of the second image

Figure 7: Masks applied to the images to be stitched.



(a) Stitching without blending (b) Stitching with linear blending

Figure 8: Diagrams showing the procedures followed.

The final result is shown in figure 9. It is possible to notice that this technique leads to very noticable improvements w.r.t. the simple stitching of images.
However the overlapping regions in some situations creates some "duplications" effects. This because the RANSAC algorithm gives good results, but not perfect ones in some situations, and if the shifts are little wrong the two images result being not perfectly aligned.

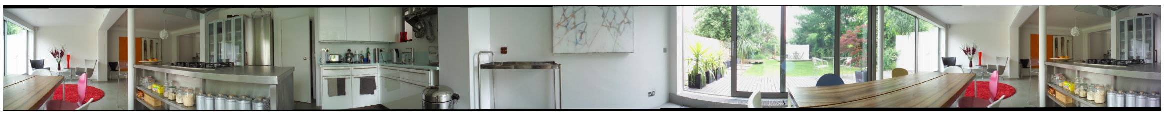


Figure 9: Panoramic image of the kitchen with linear blending on the ovelapping regions

Filtering

To attenuate also this problem is possible to work a priori, changing the values of the RANSAC threshold t_R or the *ratio* for the matches acceptance. But some errors are even introduced.
It is also possible to try to solve them a posteriori, performing a filtering operation on the final image. In particular, a **bilateral filterer** is here a good choice because, tuning wisely the parameters we can achieve a good result preserving the clearness of the image, as shown in figure 10.

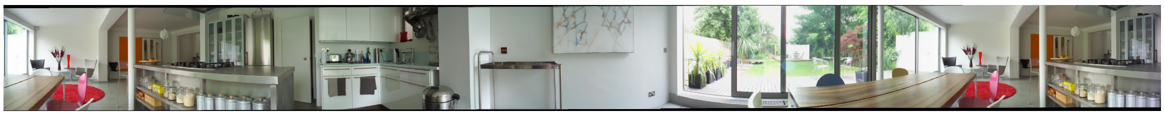


Figure 10: Panoramic image of the kitchen with linear blending, filtered with a bilateral filter with parameters $\sigma_s = 120, \sigma_r = 15, w_s = 15$. The image is a little more defined.

Many other tests can be performed, in particular using a "homemade" dataset, acquired with a mobile phone. Figure 15 shows the result. It is possible to notice that the quality is a little lower w.r.t. the datasets provided, but the result is very satisfactory, especially if compared with the image stitched with the software of my mobile phone (*Samsung Galaxy S7*), showed in figure 16.

Conclusions

In conclusion we can assert that the use of the blending leads to a particular improvement in the quality of the final image. The filtering, on the other hand, helps in attenuating the errors intruduced by the superposition of adjacent images.
Even if it is immediate to think that an histogram equalization could have brought more uniformity among the images, it doesn't work well making them unreal and not more similar each other. Probably an histogram specification only on the region where the superposition occurs would have been better than the equalization. Anyway it was not performed because the result with only the blending is sufficiently satisfactory.
On the other hand, using more complex algorithms than ORB, like SIFT or SURF could have probably lead to a little improvement, but the result achieved this way is anyway very good in terms of visual quality.
Finally, it is to remember that the dataset influences the quality of the result in a larger part, and is important to have a set of pictures aligned. Acquiring the pictures with a camera and using a triple leads for sure to the best results.

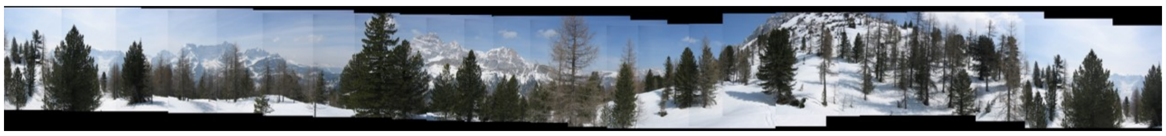


Figure 11: Panoramic image of the dolomites

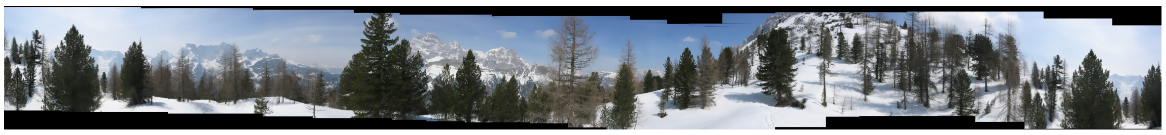


Figure 12: Panoramic image of the dolomites with linear blending, filtered

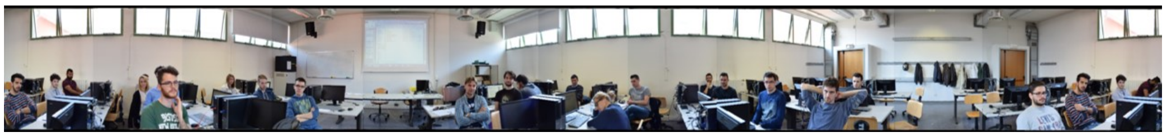


Figure 13: Panoramic image of the lab

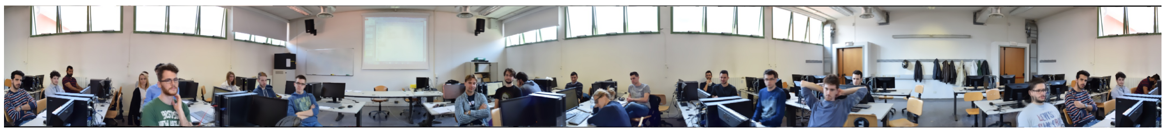


Figure 14: Panoramic image of the lab with linear blending, filtered

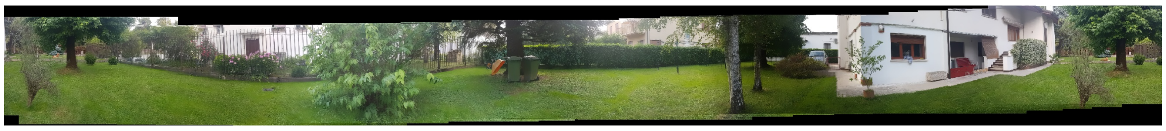


Figure 15: Panoramic image of my garden, with linear blending, filtered. It is possible to notice that the images are not perfectly aligned

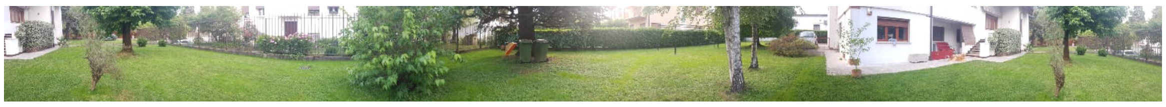


Figure 16: Panoramic image of my garden acquired and stitched directly with the mobile phone *Samsung Galaxy S7*