Computer Vision 2021 - Final Project: *Panoramic Image Construction*

Michele Guadagnini - ID 1230663

June 24, 2021

Abstract

The goal of this project is to construct a panoramic image starting from a sequence of images. The procedure adopted starts from projecting the pictures into a cylindrical surface; in this way the transformations between adjacent images consists ideally in a simple translation. To compute the proper translation matrix the *SIFT* features have been used together with the *RANSAC* algorithm to select inlier keypoints. Some extra features are represented by the possibility to produce a panoramic image also in RGB colors and to apply a histogram equalization to the final image.
The code has been developed in *C++* language, making use of the *OpenCV* library.

1 Algorithm description

The algorithm can be summarized as follows:

- Firstly, the images are projected onto a cylindrical surface. This ideally allows to use a simple horizontal translation between the images to build up the panoramic images.
  The projection requires to know the Field of View (**FoV**) used to capture the photos.

- The next step is to apply the Scale-Invariant Feature Transform (**SIFT**) algorithm to detect the local features in pictures. Presented by Lowe in 2004, it is useful in a lot of computer vision applications. Its greatest strength is that it allows to select the best scale for feature extraction.
  In this work we use the local features extracted from the algorithm to match the consecutive images and find the translation that best merge them into one picture.

- The matches resulting from the SIFT features require to be filtered in order to keep only the useful ones. A first refinement is done by selecting only the matches with distance less than their minimum distance multiplied by a certain coefficient to be defined by the user. Then the translation of an image with respect to the other is estimated by exploiting the **RANSAC** algorithm to select the inliers between the SIFT keypoints. Finally the translation is computed by calculating the average distance between inliers on both dimensions, obtaining the two coefficient needed to build the translation matrix.

- The last step is to merge all the projected images into one using the transformations computed above. To do this, the translations are combined together at each step.

2 Code Development

The code development started by implementing the functions needed to complete the task inside a class, *PanoramicUtils*. In Listing 1 it is reported the class definition, contained in the file *panoramic_ utils.h*.

```
19 class PanoramicUtils
20 {
21 public:
22
23     // project an image on the cylindrical surface in gray_scale
24     static cv::Mat cylindricalProj( const cv::Mat& image,
25                                     const double angle
26                                   );
27
28 // Added functions with respect to the provided library ------------------------
29
30     // project an image on the cylinder in RGB
31     static cv::Mat cylindricalProjRGB( const cv::Mat& image,
32                                        const double angle
33                                      );
34
35     // histogram equalization method
36     static cv::Mat intensity_hist_EQ( const cv::Mat& input );
37
38     // extract SIFT descriptors from an image
39     static void SIFTkeypoints( cv::Mat& img,
40                                std::vector<cv::KeyPoint>& keypoints,
41                                cv::Mat& descriptor );
42
43     // find matching features between different images
44     static std::vector<cv::DMatch> SIFTmatches( cv::Mat &desc1,
45                                                 cv::Mat &desc2,
46                                                 double ratio );
47
48     // use RANSAC to estimate translations between adiacent images
49     static cv::Mat find_translation( std::vector<cv::KeyPoint>& kps1,
50                                      std::vector<cv::KeyPoint>& kps2,
51                                      std::vector<cv::DMatch>& matches
52                                    );
53
54     // build and visualize / store the panoramic image
55     static cv::Mat build_panoramic( std::vector<cv::Mat> &images,
56                                     std::vector<cv::Mat> &transforms
57                                   );
58 };
```

Listing 1: PanoramicUtils class definition inside its header file.

The first function, the one performing the cylindrical projection, was already provided, but only to work with gray-scale images. To have the possibility to produce also color panoramics a new function has been created. This new function (*cylindricalProjRGB*) do the same computations of the first one, but, instead of converting the image to gray-scale, it splits the 3 channels, applies the projection to each of them and finally restore the RGB image by merging the projected channels.

The other functions visible in Listing 1 follows the steps explained above in the algorithm description. It is worth to mention that the *SIFT* descriptions has been computed by using the *OpenCV* class **cv::SIFT** and the matches between them with the class **cv::BFMatcher**; they allowed to reduce the code needed for these tasks to few lines (see the implementation in *panoramic_ utils.cpp* for details).

As suggested in the assignment, to apply the *RANSAC* algorithm to select the best matches between keypoints the function **cv::findHomography** has been used and its argument *mask* allowed to retrieve the inliers keypoints.

In the function *build_panoramic*, after merging all the images into one, it has been added a cropping of the panoramic image in order to remove the borders left by some small vertical translations. The code of this step is reported below in Listing 2.

```
292         // cut on y-axis to hide vertical translations effects
293         int crop_top    = static_cast<int>(max_y_trl+1.5);
294         int crop_bottom = std::abs( std::min(static_cast<int>(min_y_trl-1.5), 0) );
295 
296         panoramic = panoramic( cv::Rect( 0,
297                                          crop_top,
298                                          panoramic.cols,
299                                          panoramic.rows - (crop_bottom+crop_top) )
300                                );
```

Listing 2: Code used to eliminate black borders from panoramic image.

The last function to be mentioned is the one that allows to perform the histogram equalization of the final image, *intensity_hist_EQ*, which is reported in Listing 3. This function can equalize the histogram of a gray-scale image, but also a color image. To do this, the input image is firstly converted from **RGB** to **YCrCb** space, then equalization is applied to the **Y** (intensity) channel. By merging the channels again and converting back to RGB space, the equalized color image is obtained.

```
119     // Histogram equalization method
120     cv::Mat PanoramicUtils::intensity_hist_EQ( const cv::Mat& input )
121     {
122         if(input.channels() == 3)       //RGB
123         {
124             cv::Mat ycrcb;
125             cv::cvtColor(input, ycrcb, cv::COLOR_BGR2YCrCb);
126 
127             std::vector<cv::Mat> channels;
128             cv::split(ycrcb, channels);
129 
130             // equalization is done on the intensity channel
131             cv::equalizeHist(channels[0], channels[0]);
132 
133             cv::Mat result;
134             cv::merge(channels, ycrcb);
135             cv::cvtColor(ycrcb, result, cv::COLOR_YCrCb2BGR);
136 
137             return result;
138         }
139         else {                          //gray-scale
140             cv::Mat result;
141             cv::equalizeHist(input, result);
142 
143             return result;
144         }
145     }
```

Listing 3: Function to perform histogram equalization to an image.

2.1 Compilation and usage

The program has been compiled with the following command:

```
g++ -o panoramic Panoramic_main.cpp panoramic_utils.cpp `pkg-config --cflags --
  libs opencv`
```

Below it is reported a usage explanation of the executable:

```
./panoramic "images_path"  FoV  ratio  output_name  color_mode  equalization
```

The program also print a help message with usage instructions in case a wrong number of arguments have been passed.

3 Results

The first result presented is the panoramic image of the lab, reported also in the project description as an example. The image has been obtained also with colors, Figure 1 and 2. Figure 3 instead reports the equalized panoramic view of the lab.

Also the panoramic images of the *dolomites* and *kitchen* datasets are reported in Figure 4, both in gray-scale and colors.

Figure 1: Laboratory panoramic view in gray-scale.

Figure 2: Laboratory panoramic view with colors.

Figure 3: Laboratory panoramic view with colors after histogram equalization.

4 Conclusions

Looking at the panoramic views, the stitching position are visible in all the images due to color differences between them. Also the translations are not perfect as sometimes the overlap is noticeably wrong.

Some possible improvements of the project could be:

- more advanced methods instead of simple translation;
- a simple idea to improve the results could be a small local blurring in the region between two images;
- it is alos visible that the final equalization on the panoramics is not effective; it could be better to do a histogram specification using one of the images as reference to remove color jumps.

Figure 4: Other panoramic images constructed.