Computer Vision - Project 1 Report

Carlo Facchin (1234374)

23, June 2021

1 Introduction

The goal of this project is to create a panoramic image given a sequence of images.
In order to do that the provided code :

- Load the set of images
- Project the images on a cylinder surface using the provided static method *cylindricalProj()* of the *PanoramicFunctions* class (*PanoramicUtil* class modified containing all the necessary functions).
- Extract the ORB features from the images.
- Compute the match between the different features extracted.
- Refine the matches found above selecting them with a thresholding process.
- Compute the translation between matching features using the *RANSAC* algorithm.
- Compute the final panorama merge of the input images.

The coded software has also the possibility of working with colour images and eventually equalize them during the loading process, provide an additional merging function that fade the edges of consecutive images blending/mixing the transition, close the loop connecting the last and the first image.
Is also possible to try the code on two different set of images personally acquired with two different lenses (resized and compressed to reduce size and computational demand).

2 PanoramicFunctions class

The software code is divided in the executable file with the *main()* function, the *Extended_panoramic_utils.cpp* and the *.h* header file that implement the class *PanoramicFunctions*. The functions are the following:

```
Mat RGB_cylindricalProj(const Mat& image, const double angle);
```

Similar function of the already provided *cylindricalProj()* with the difference of no grayscale conversion. The parameters are the input image Matrix and the angle of the half of the used camera Field of View. It returns the projection of the colour image on a cylinder surface.

```
vector<Mat> img_load(string path_dir, string img_format,
int n, bool equalization);
```

Images loading function that given the directory path of the images, their format and number return a *vector<Mat>* of all the loaded images. The flag *equalization* enable the equalization with *equalizeHist* of the three color channels. Has been used this method instead of the luminance equalization because of the possible use .png format.

```
static void img_proj(vector<Mat> input, vector<Mat>& output,
double half_FoV_angle, bool RGB_select=false);
```

Function that given in input the *vector<Mat>* that contains all the loaded images and the angle value of the half FoV, compute the cylindricalProjection for each image using the provided function as a *vector<Mat>*. The choice between the *cylindricalProj()* function or the RGB one is decided over the choice of the boolean flag *RGB_ select*.

```
static void ORB_extract(vector<Mat> input, vector<Mat>& descriptors_vec,
vector<vector<KeyPoint>>& keypoints_vec);
```

Function that given in input the *vector<Mat>* that contains all the loaded images extract the ORB features in a *vector<vector<KeyPoint»* and compute descriptors in *vector<Mat>* for each image.

```
static void feature_match(vector<Mat> descriptors_vec,
vector<vector<DMatch>>& matches_vec, bool verbose=true);
```

Function that given the features previously found, compute the matching of the features of two consecutive images storing them in *vector<vector<DMatch»* (a vector containing all the matches for each image, vectors as well).

```
static void match_select(vector<vector<DMatch>> matches_vec,
vector<vector<DMatch>>& selected_matches_vec, float ratio);
```

Function that refine the matches previously found by thresholding the matches with the parameter value *ratio*. Has been used a ratio = 1.5. In order to avoid a selection/threshold too severe if the number selected of matches is less than a fixed value (80 in this case), the ratio is increased allowing a looser filtering.

```
static void RANSAC_homography(vector<vector<DMatch>> selected_matches_vec,
vector<Mat>& output, vector<vector<KeyPoint>> keypoints_vec);
```

Function that given as parameters the selected matches and keypoints, implement the RANSAC algorithm to compute the homography matrices of two consecutive images. In the second part of the function the same computation has been made separately between the last and the first image, in order to close the loop of the panoramic matching.

```
static Mat ToCanvas(vector<Mat> input, vector<Mat> H_mats,
const string& path = "/", bool save_img = false);
```

Function that given the vector of the loaded images and the vector of homography matrices merge the images copying them in a canvas. Return the output canvas (right border for the closing loop merging). The *save_img* flag enable the saving of the output panoramic .jpg image in the defined path.

```
static Mat ToCanvas_smooth(vector<Mat> input, vector<Mat> H_mats, int d1,
const string& path = "/", bool save_img = false);
```

Merging function that addictionally fade the edges of consecutive images. Since the input images have all the same size is possible to use the OpenCV function *addWeighted()*. It is a collaging and compositing function that apply a transparency gradient mask to the image that affect the transparency of the image. The parameter *d1* select the section of image affected by the transparency gradient. Since the merge of the edges require a cross-fading effect when the portion of an image decrease in transparency, the other one symmetrically becomes more visible. The transparancy gradient section used is of 30px and maintain a good transition of the edges without being too smooth and large, allowing to notice the overlap of two semi-transparent section of different images.

3 Results

The software has been tested at first with the "dolomites", "kitchen" sets and the two self acquired set of images "cortile" and "terrazzo". Both for the "lab-manual-ok" and the kitchen set the number of image has been reduced avoiding pictures with the spacial section already totally covered. Then lastly a further test has been made on the lab-auto-challenging set that differently also from the others lab sets do not perform because of the camera

rotation change during the images acquisition.

Figure 1: "Cortile", greyscale, No Eq, Standard merging

Figure 2: "Cortile", greyscale, No Eq, Smooth merging

Figure 3: "Cortile", greyscale, Eq, Smooth merging

Figure 4: "Cortile", BGR, No eq, Smooth merging

Figure 5: "Cortile", BGR, Eq, Smooth merging

The difference between the use of the merging function is noticeable since the edge transition become smoother and the horizontal alignment imperfections less noticeable however a simple BGR channel equalization doesn't affect much the difference in brightness between the images (the photos has been taken with automatic setting instead of using fixed parameters). Infact the effect obtained by the equalization is only a color saturation and an increased contrast, and a different approach affecting the image luminance of the

image would have better result.

Other test result examples:

Figure 6: "terrazzo", BGR, No Eq, Standard merging

Figure 7: "terrazzo", BGR, No Eq, Smooth merging

Figure 8: "kitchen", BGR, No Eq, Smooth merging

Figure 9: "lab", BGR, No Eq, Smooth merging

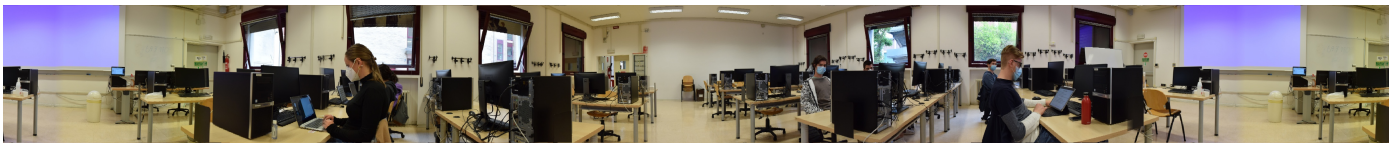

Figure 10: "lab-manual-ok", BGR, No eq, Smooth merging

Figure 11: "dolomites", BGR, No eq, Smooth merging