COMPUTER VISION Course – A.Y. 2019/2020

LAB 4 – PANORAMIC IMAGES

FINAL PROJECT REPORT

1. Introduction and goals

The task of this lab experience is building a 360° horizontal panorama by stitching a sequence of vertically aligned pictures, taken from a single viewpoint. The procedure will make use of known computer vision techniques for feature detection and descriptors computation, using the well-known OpenCV library. The quality of the obtained panoramas is evaluated by noticing the correct alignment of the pictures, both vertically and horizontally, and the absence of stitching artifacts (vertical lines/structures).

2. Developed algorithm

2.1 Parameters setting

The first section of the C++ program is dedicated to the setting of the program parameters: MASK_PERC, SAMPLE_SELECT, PATCH_SIZE and TRIM_EDGES. The use of those parameters will be discussed in detail in the following sections.

2.2 Images import

Each sequence is stored in a folder of ordered stills of the same size. The first part of the program imports those images in memory as a cv::Mat objects and applies to them the cylindrical projection, both in grayscale (with the provided algorithm in the PanoramicUtils class) and color (with a new C++ method, stored in the same class). This, following the image projection theory, is made to obtain a set of stills whose similar features differ only from vertical and/or horizontal translation. At this stage the parameters are set according to the imported sequence.

After this, the first image is considered the root of the panorama and added to the *output* container. Then, within a “for” loop, each of the remaining stills is processed. Only the grayscale pictures are considered during features computation, at the end of each iteration the obtained information is applied to the color pics. In case of unknown sequence, the program uses the default ones.

2.3 Features computation

The first step that each image has to withstand, is the features point detection, that highlights notable points in the image, and the computation of descriptors in the same locations. The algorithm of choice for this procedure is the “Oriented FAST and Rotated BRIEF” (ORB) which is

built upon the “FAST” corner detection tool for feature extraction and the BRIEF keypoint descriptor to which rotational invariance is added. OpenCV contains already a set of classes that implements the ORB algorithm.

Important note is that features are computed only alongside the right border of the developing panorama and the left border of the image that must be added. Figure 1 shows an example of the used masks. The parameter MASK_PERC defines the width fraction over the whole (added) image horizontal size, notice that both have the same active area width. This should be, ideally, the amount of superposition between adjacent images during capture. The mask building functions are stored in a dedicated class, MaskTool, as static methods.

Figure 1: Feature keypoint masks, MASK_PERC set at 0.3

The ORB tool is initialized with 700 as the max number of features points, and 1.2 as scale step, for a fine scale decomposition. Since we are working with low resolution images and we would like to find features alongside borders, the computation edges size has been reduced to 10. The size of the feature computation patch is set with the PATCH_SIZE parameter.

Using the BFMatcher class the keypoints are compared and the best ones kept. In particular, SAMPLE_SELECT determines the fraction of matches (over the whole list) that are used in the following steps, selecting the ones with the minimum Hamming distance.

2.4 Translation estimation

Having obtained a set of matched points, now we proceed by computing the translation in the X and Y axes. This is done with a customized version of the RANdom SAmple Consensus (RANSAC) procedure.

The RANSAC algorithm optimizes the selection of the best deltas by considering the average only between a set of coherent translations, meaning that the difference between the considered points is under a pre-defined threshold, and outputs only the average over the biggest set.

In practice, the used implementation randomly chooses a pair of points, computes the X and Y deltas, finds between all the deltas the ones similar (or different up to a threshold) and, if the number of elements in the compatible set is the biggest found, stores the sum of the found set of translations. Eventually, the summed deltas are divided by the size of the relative reference set, so that the averages are returned.

The function customRANSAC() in the main() class implements this algorithm.

2.4 New container building and images blending

The translation information achieved with the previous steps is used to determine the final size of the resulting image at this iteration, these new dimensions are immediately used to build overall 4 image containers, each one being:

a. Stores the grayscale panorama, eventually shifted in the bottom direction if the added image positioned higher in the frame.
b. Stores the grayscale version of the newly added image, in the correct relative position with respect to the panorama in the previous container
c. Stores the color panorama, in the same position as the grayscale version
d. Stores the color new image, in the same position as the grayscale version.

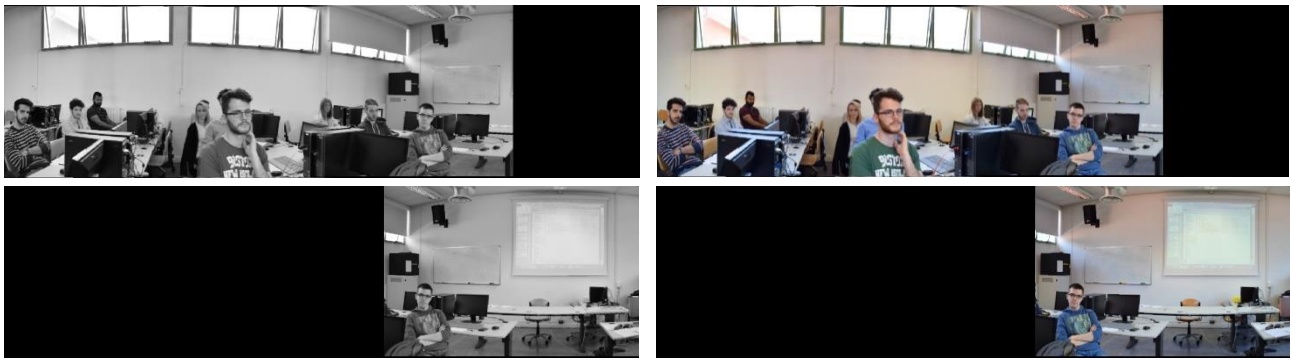

Figure 2a-b-c-d: Working image containers

To obtain a smooth transition between adjacent images an alpha mask is used during the blending process. The mask provides linear gradient transition between the panorama and the added image, assuring that the fading (horizontally) starts after the start of the added image and ends before the end of the built panorama. The mask is created by the hGradientMask() method of the MaskTool class.

Figure 3: Blending alpha mask, referred to stage of Figure 2

The actual blending itself is implemented in the static method available in the BlendTool custom-built class, in the two grayscale and color variants. The simple algorithm works as follows:

1. Multiplies pixel-by-pixel the foreground (the image that should be added on top) with the alpha mask, to obtain a masked foreground;
2. Computes the *beta* mask the complementary mask of alpha, e.g. if *alpha* ranges from [0,1], *beta* = 1 – *alpha* at each pixel location;

3. Multiplies pixel-by-pixel the background (the already processed panorama in its container) with the complementary mask *beta* to obtain the masked background;
4. Adds the two images together to obtain the final blend.

In the grayscale only one layer is used, in the color version, the image is splitted in the 3 channels and the same procedure is applied before merging them back together.

2.5 Image finalization and export

At the end of the loop, the whole grayscale and color panoramas are ready.

The obtained panorama is refined by trimming the top and bottom black bands, which are the result of the vertical shifting of the images. This is done by scanning the image in lines and resizing it whenever a row of black pixel of length bigger than a threshold is detected. Only if the boolean parameter TRIM_EDGES is true this operation is performed.

Finally, the output image is saved as "*inputFolderName* + _out.jpg" in the *out* folder.

3 Use of the program

The user should provide a folder which contains only a sequence of adjacent images, in the correct order, taken by rotating horizontally the head of a tripod. For a good result, achieved in a limited amount of time, it is suggested to compress the images to a height of 400-500 pixels or smaller. A simple *ffmpeg* script (compressor.cmd) is provided to perform this operation, saving the resized images in the *comp* folder.

When executing, the algorithm prompts the user to insert the name folder where the image sequence is stored. If the image sequence is valid, it then asks the relative angle of view and whether the black edges should be trimmed.
Advanced parameters and constants are set at code level.

At each image addition, the program provides information about:

- The processed picture
- The number of keypoints given by the ORB detector that have been matched, and the ones that have been eventually selected.
- The RANSAC algorithm performance: the deltas with the largest coherent set, the set size and the final computed average.
- The panorama size at this step

This, until the end of the execution is notified.

4 Results analysis

Resulting panoramas are presented in the following pages, splitted in half for better display. See the *out* folder for the full image, and the console log during the building process.

Test1 – Lab

ORB mask: 50% of the image
Matches selection: first 30%
Trim black edges: on
Angle of view: 66°

12 images
Images size: 640x427
Panorama size: 4113 x 420

Good results in the whole image.

Test2 – Kitchen

ORB mask: 80% of the image
Matches selection: first 30%
Trim black edges: on
Angle of view: 66°

23 images
Images size: 640x426
Panorama size: 4829 x 411

Good results in the whole image.

Test3 – Asiago

ORB mask: 50% of the image size – restricted on the top half
Matches selection: first 40%
Trim black edges: on
Angle of view: 83°

20 images
Images size: 600x400
Panorama size: 3454 x 390

Good result considering the uniform and repetitive floor, problem solved by using the ORB algorithm only on the top half of the image.