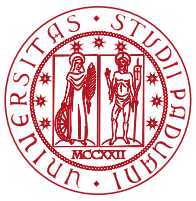


UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Computer Vision, A.Y. 2019/2020
Master Degree in ICT for Internet and Multimedia

Report Lab 4

Deadline: June 18, 2020

Student: Aniello Xie (1234362)

Introduction

The objective of the project is to create a panoramic image given a sequence of unstitched images. For this purpose, the main steps are:

1. Project the images on a cylinder surface (already provided function).
2. Extract the features from the projected images.
3. Compute the matching features between adjacent images.
4. Compute the translation between matching features.
5. Stitch images together.

In addition, we added edge blurring affect between adjacent images during the stitching passage to smooth the image transition area and merged the last image with the first to close the loop.

In the end we implemented a simple panoramic image viewer that allow us to see the merged image and rotate the view to left or right without restrictions.

The implemented functions work also with colour images.

Implementation

We used OPENCV 4.3 lib for C++ and implemented Panoramic class that includes all the functions that we need:

- **Panoramic::load_images(string path, int n, bool show = false)**:
  load n images from given path in the program inside a vector<Mat>. The parameter show is default false, if set true we can see each loaded image inside a different window.

- **Panoramic::cylindrical_projection(vector<Mat> input, vector<Mat>& output, double half_fov, bool color = false, bool show = false)**:
  vector<Mat> input contains all the loaded images and the output is the vector<Mat> where we will store the projected images. As input we also require the half_fov parameter of the camera. The color parameter is default false, so the function will use the given projection

function that return grey-scale projected images. If we set color true will be used the modified projection function that maintains all the BGR colors. In the end the show parameter is used if we want to see each image in different window.

- **Panoramic:: compute_ORB(vector<Mat> input, vector<vector<KeyPoint>>& KeyPoints, vector<Mat>& Descriptors)**:
  we extract ORB features and compute the descriptors of each image and store them inside the input parameter KeyPoints and Descriptors. We keep the default parameter for the ORB features because the results are not bad but is possible to tune the parameter inside the implementation in the *panoramic.cpp* file.

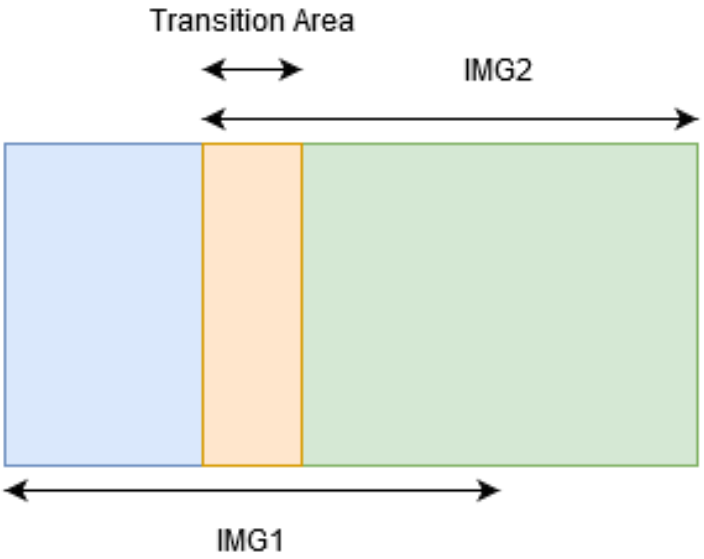
- **Panoramic::compute_match(vector<Mat> Descriptors, vector<vector<DMatch>>& Matches)**:
  we compute the matching features between adjacent images and save the matches inside the parameter Matches.

- **Panoramic:: filter_best_match(vector<vector<DMatch>> Matches, vector<vector<DMatch>>& BestMatches, float ratio)**:
  we filter the best matches computed before using the parameter ratio. At first we find the minimum match distance and use the ratio parameter to set the max distance in which we can keep the match (for our test we use ratio=1.5). Inside the function we used a support function, *autotune_matches (Matches[i], min_dist, instance_ratio)*, that check and push the in-range match inside the bestMatches vector. As baseline we set that we want at least 120 best matches, and if we cannot reach the number of matches we double the ratio value and re-iterate the function.

- **Panoramic::compute_homography (vector<vector<DMatch>> BestMatches, vector<vector<KeyPoint>> KeyPoints, vector<Mat>& output)**:
  we use the computed best matches key-points to compute the homography between adjacent images, getting a vector of 3x3 translation matrices.

- **Panoramic::merge(vector<Mat> img, vector<Mat> H, bool save, const string& path)**:
  first version of image merging function that just stitch them with a copy&paste method using only the x axes translation (because we have prior knowledge that the images are took just rotating the camera). In the end we save the result in the given path if the parameter *save* is set true.

- **Panoramic:: blur_edge_merge(vector<Mat> img, Mat& panoramic, vector<Mat> H, int delta, bool save = false, const string& path = "/")**:
  modified version of the merge function that add a transition effect with given width in input as parameter *delta.* The effect is obtained using *addWeighted* OpenCV function that makes the first image more and more transparent, while the second more and more visible inside the delta area[1].
  In the end when all the images are all merged, we use this transition effect also on the last image and the beginning of the panoramic image to close the loop. In this way the first and the last part of our panoramic image will have an identical area.



Merging images with transition area.

- **Panoramic::panoramic_viewer(Mat img, int width, int step_size, int base_img_cols)**: this last function is a basic panoramic viewer that need in input the panoramic image, the width of the window that we want to use to show the image, the step_size is the number of pixel that will be “translated” when we rotate left/right, and base_img_cols is the number of columns of our basic image (in our case is 640px).

Results

We tested the algorithm on all 3 image dataset that were given, the bareback Copy Paste result is not very good in fact even if the images are matched at correct x-offset, is possible to see very clearly the separation line between one image and another.


Dolomites C&P Panoramic


Kitchen C&P Panoramic (with only 20/23 images)


Lab C&P Panoramic

The following are the results obtained with the edge-blurring function: in this case the pictures are merged better and is more difficult to see their transition border. We also closed the loop between the last image and the beginning of the panoramic, in fact is possible to see the same scene in the beginning and the ending. For these experiments we used a transition area of 40px, we also tried different transition area width with different results:

- Augmenting transition area we get better merge, but there will be significative “ghosting effect” of objects that are in different position in the consecutive images.
- Decreasing transition area, we reduce the “ghosting effect” but the transition line becomes more evident.

Dolomites Edge-Blurred Panoramic

Kitchen Edge-Blurred Panoramic (with only 20/23 images)

Lab Edge-Blurred Panoramic

We also tried to work with color images adding a modified cylinder projection function that were given at beginning, obtaining the following results:

Dolomites Edge-Blurred Color Panoramic

Kitchen Edge-Blurred Color Panoramic (with only 20/23 images)

Lab Edge-Blurred Color Panoramic

In the color ones the change of image is a bit more evident respect their grey-scale counterpart due to the change of illumination in the same point between one image and another, some color balance or illumination correction algorithms should be tested in order to improve under this point of view.

To have a better panoramic view experience we built a simple function that create a dynamic canvas that update the image inside when we press “a” to rotate left and “d” to rotate right.

In the beginning part of our panoramic image we have visible ghosting effect due to the double blurring-merge that we have made to connect the end to the beginning and the first image to the second one.

For the kitchen dataset we decided to use 20/23 images because the last ones cover exactly the position of the first, second and third images, so we decided to not use them.

References

1- Ahmed Bdr Eldeen Ahmed Mohammed, Fang Ming & Ren Zhengwei, “Color Balance for Panoramic Images”, Modern Applied Science Vol. 9 No. 13, pp. 142-143, 2015.