Stitching images in a panoramic view

Mautone Alberto

The source code (**source.cpp**) uses a class called "**PanoramicImage**" that actually implements all the funcition I'm using in the main() in "source.cpp". This class is at first defined in its header file "**PanoramicImage.h**" in which I defined all the functions needed, all implemented in the "**PanoramicImage.cpp**" file. The first two function are kind of simple, since the first is just used to load all the set of images to be stiched in a vector of Mat (it uses the glob(...) function. **NB (hard coding)**: path need to be changed with the one with the starting pictures on your PC), and the second just takes the *cylindrical projection*(...)function. The third function, *cal-homography-matrix(...)* is useful to compute all the **homography matrix** between each consecutive couple of images from the dataset given. It actually uses **ORB** features to extract keypoints and descriptor of each single images, draws the keypoints thanks to the *drawKeypoints(...)* function and finds a match between all of them with the *knnMatch(...)* function, using a **BFMatcher** matcher. After filtering the matches choosing only the best using the **Lowe's ratio test**, I'm almost ready to get my homography matrix: actually the function *findHomography(...)* requires two vectors of Point2f taken from keypoints, and to get them I just iterate the good-matches found before extracting the Point2f of the first keypoint (that I called **object**) with *.queryIdx.pt* and the Point2f of the second image (that I called **scene**) with *.trainIdx].pt*. I'm finally ready to get the homography matrix with the function *findHomography(...)* and return the homography matrix obtained. In the "source.cpp" file I use the function just implemented to get all the homography matrixes and store them in a **vector<Mat> homo**: it's gonna be helpful later on.
The fourth function finds the average distance between 2 matched keypoints (after using *findHomography(...)* mask, as done in the previous function): the distance is computed selecting only the **inliers** (mask entry set to 1), obtained by the mask returned by *findHomography(...)*, and making a simple subtraction between 2 matched keypoint among both x and y axis using again *queryIdx].pt.axis* (axis = x or y) for the first image and *trainIdx].pt.axis* for the second image. Summing all the subtractions among x and y and dividing them by the number of matched keypoints I get average distances, that I store in a vector of **pair<double, double>** called *p* in the "source.cpp" file.
The fifth function is *crop-image(...)*: since when I stitch 2 images I get a blank space (all colored in **black**) I need it to be removed. To do that I apply a **threshold** to the stitched image, and then I get all the contours that do not include the black area: finally I cut all this part to get only the stitched image. This function is supposed to be used **after** the sixth and last function I implemented: *stitch-image(...)* that, as suggested by the name, taked 2 images and stich them using their homography mask passing through a *warpPerspective(...)* and some manipulation about the resulting Mat including the stitched image. The source.cpp file stitches all the images using a **for loop** with **2 indexes** (a tecnique I've also used before): one for the first image until the last -1, and another for the second image until the last (since images need to be stitched in **consecutive couple**), and uses the vector of homographies matrix computed before. That's the **final result**:

All the set of images given stitched in a panoramic one.