Lab 5 - Keypoints, Descriptors and Matching

Luca Scattolaro

15 May 2021

1 Report

To do this Laboratory I follow primarly the opencv online guide and the theoretical slides of the lab.
The goal of this lab is to Create a panoramic image given a sequence of unstitched images.
For this porblem I create a Class called PanoramicImage and then lab5.cpp to test the functions of the class.

1.1 PanoramicImage Class

In this class I create different methods, in the image below we can see the most important ones:

```
class PanoramicImage{
public:
    // constructor
    PanoramicImage(std::string pathDataset_);
    //-- Methods
    void computeCylindricalProj(double angle);
    void stichingImages(double ratio);
    cv::Mat getResultImage();
    cv::Mat getResultImageEqualized();

// Private
private:
    // Data
    std::string pathDataset;
    std::vector<cv::Mat> imagesDataset;
    cv::Mat resultImage;
```

Figure 1: PanoramicImage Class

In the class there are also other secondary methods with the aim to solve different sub-tasks of the principal method: $stichingImages(..)$.

1.2 Computational Steps

1. **Load a set of images**
   I create an instance of the class PanoramicImage passing as parameter the path of the dataset that contains all the consecutive images.
   Name of the instance: $panoramicImage$.
   In the constructor I extract all the Images contained in the dataset saving them in $std :: vector < cv :: Mat > imagesDataset$

2. **Project the images on a cylinder surface**
   I asked to the user to insert the value of the angle to compute the Cylindrical Projections. If the user insert a wrong value I will use as default the value 33 (That is ok for all the datasets but the dolomites one).
   And I called the method $panoramicImage.computeCylindricalProj(angle)$;
   This method called a method given by the professors contained in the $PanoramicUtils$ Class.

3. **Compute the Panorama** I asked to the user to insert the value of the ratio that is used to extract from all the matches found only the good ones (the ones with $distance < ratio * minDistance$).
   And I called the method $panoramicImage.stichingImages(ratio)$;
   This method of the class computes different steps:

(a) **Extract the SIFT features from the images**
I create a SIFT Detector: $dectector = cv :: SIFT :: create()$;
I create 2 variable in order to contains for each image in th dataset the keypoints and also the descriptor:
$std :: vector < std :: vector < cv :: KeyPoint >> keypoints$;
$std :: vector < cv :: Mat > descriptors$;
For each image I applied the method $dectector- > detectAndCompute(...)$ In order to get the keypoints and the descriptor.

(b) **Compute and refine the match between images**
I create a matcher: $cv :: Ptr < cv :: BFMatcher > matcher = newcv :: BFMatcher(cv :: NORML2)$; and a vector for the matches found for each pair of consecutive images: $std :: vector < std :: vector < cv :: DMatch >> matches$; and I fill it with the result obtained using the method $matcher- > match(...)$; for each pair of descriptors of consecutive images.
After that among all these matches found for each pair of consecutive images I calculate the minimum distance and I take only the good matches (the ones with $distance < ratio * minDistance$).

Figure 1: Example of good matches between the first 2 images. (Circles: keypoints, Line: good matches)

(c) **Compute the final panorama**
To compute the final image for each pair of consecutive images I compute the homography matrix and I calculate the average translation between the 2 images.
The aveavgTranslation allows me to "cut" the second image and save the cuttedVersion in a vector.
After this cycle into $std :: vector < cv :: Mat > cuttedImage$ I have all the images rightly cutted and ready to get concatenated in order to get the final Panorama Image.

1.3 Results

Figure 2: Result Obtained

Figure 3: Equalized Result Obtained