# Homework 2 Computer Vision Author: Cristiano Colpo, ID: 1245205

The laboratory that was selected for this homework is lab 4, where it was requested to create a panoramic image from a set of photos by implementing some computer vision algorithms.

## Features

- panorama image with resizing
- panorama with colors or black and white images
- histogram equalization
- support for ORB and SIFT

## How it works

Setup the environment: if you are using Ubuntu 18.04 you can use this guide for install openCV and cuda here.

The class PanoramicImage takes as argument the location of the photos for example: `/home/seleucio/Documenti/dante1/*.jpg` and the FoV (e.g. `54`). You can decide to use SIFT instead of ORB by passing the flag true in the method `createPanoramicImage`.

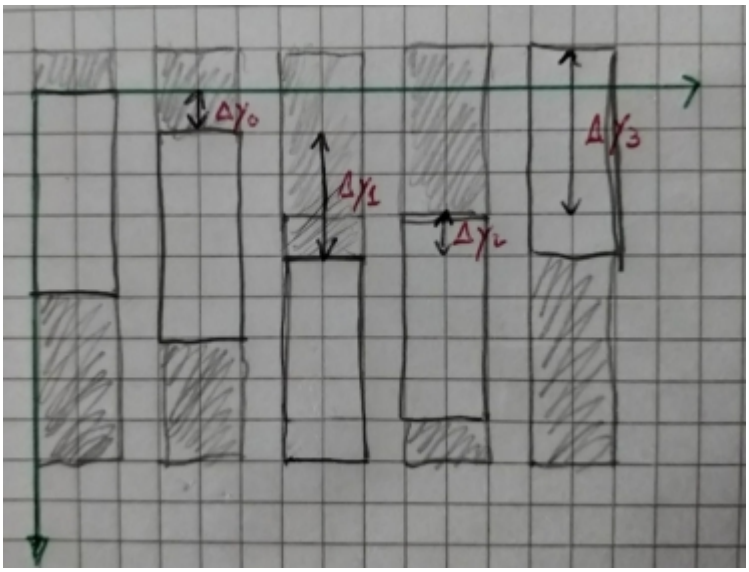
The algorithm that is executed by the program follows the specification of the request:

- load images in the path defined by the user in local memory;
- execute the projection on the cylinder with ToCylinder and return a `vector<Mat>` grayscale images with no equalization;
- calculate keypoints and descriptors with the overloaded method `detectAndComputeKD()` by using ORB or SIFT depending on the advanced flags for each consecutive image pair in `vector<Mat>`;
- detect the matches with `hardmatch()`;
- clean the matches with `cleaning()` by taking only the possible matches that could possibly be only on the right side of the first image and on the left side of the second. Another condition for defining a true match is by checking the `distance` property of a match and if this value is less than `ratio * min_dist`. Other filters can be implemented in the `filters()` method;
- The good_matches returned by `cleaning()` are then displayed in the dir good_matches as pairs of images by the method `drawMatch()`;
- For each image, we get the mask from findHomography that is computed by `getMaskFromHomography()` and the RANSAC algorithm included in that method and we extract the inliers with `extractInlier()`;
- We load the colored images with the histogram equalization in LAB;
- The final method that creates the panoramic image is `stacking()`

The idea behind the method `stacking()` is: the consecutive pairs of images have in common a specific area so we create a *stack* of images by the first one to the last one by superimposing each image by their *common area* (we will see later how we can manage this specific area).

Firstly the algorithm calculates the coordinates `deltaX` and `deltaY`of the *shift vectors* of each keypoints for each consecutive pairs of images, the true shift of the image is calculated by taking the mean of deltaX and deltaY.

Secondly, the algorithm creates the `vector<int> h_pad_from_baseline` a vector for taking into account the vertical shift of the image from the baseline that is the origin of the first image(an inverted 2D cartesian coordinate system as the one that opencv use).




The two vector `l_pad_final` and `h_pad_final` contain the lower and higher padding for each image, we add the padding on each image with the method `copyMakeBorder()`.

The last operation is the stack of each image, we can decide to eliminate the common area on one image or we can mix these areas for example by using a gradient ramp as a mask(there is a bit of ghosting artifact). The output image has vertical padding so we can crop the dark area.

the FoV of the photo from Col di Dante was taken with a smartphone with Sony Exmor IMX398, a sensor with diagonal 6.40 mm and focal length 24mm so:

`FOV [°] = 2 * arctan ( d [mm] / (2 * f [mm]) )= 2*arctan(6.40/(2*24)`

## Results

ORB:

Dolomiti SIFT SIFT:

Col di Dante (VI)

Dolomiti SIFT

## Room for improvments

- Performance improvements by using better the memory (it will take some seconds even in a good system for computing the panoramic image for 16mpx images);
- Performance improvements by using cuda for some methods CUDA SIFT;
- Use a slope filter for eliminating the spurious matching vectors that are not in the mean slope (interval defined by a user-defined threshold) in case of using ORB;