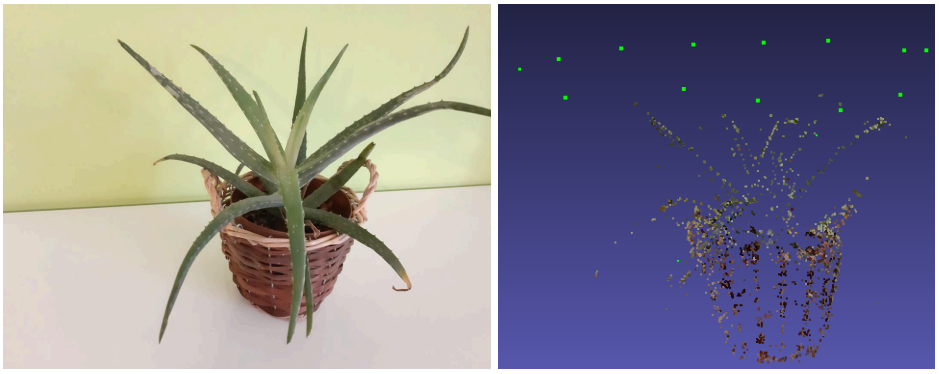
*3D DATA PROCESSING - LAB 2 (**GROUP ASSIGNMENT**)*



Topic: Structure from Motion
Goal: Estimate the 3D structure of a small scene taken by your smartphone from a sequence of images with some field-of-view overlaps.
Groups: Students should organize in groups of two members and do the assignment together. Only one submission per group is sufficient (just clearly indicate the names and the "matricola" (IDs) of the components in the report)

Introduction

For this assignment, it is required to complete some code portions of a simple Structure from Motion pipeline. This pipeline includes two main modules:

- A feature extractor and matcher module, by means of the `matcher` app defined in `matcher_app.cpp` (you shouldn't edit this source file). This application requires as input the calibration parameters of the camera used to acquire the images, a folder containing the images used for the 3D reconstruction, and the name of the file where the processed data will be stored. Basically, this file will be filled with all the information needed to build the scene graph (e.g., the collected observations) that will be used in the subsequent reconstruction module.

- An incremental structure from motion module, by means of the `basic_sfm` app defined in `sfm_app.cpp` (you shouldn't edit this source file) that requires as input the outfile produced by the matcher app and the name of `.ply` file (Polygon File Format) to save the point cloud resulting from the 3D reconstruction.

Groups are required to complete two chunks of code within the source file **`features_matcher.cpp`**, namely:

1) Extract salient points, descriptors, and features colors from images (task 1/7, function `extractFeatures()`);
2) Match descriptors between images performing Essential + Homography geometric verification (task 2/7, function `exhaustiveMatching()`).

and four chunks of code within the source file **`basic_sfm.cpp`**, namely:

1) Extract Essential matrix E and Homograph matrix H and check the number of inliers for both models to extract the seed pair. In a good seed pair the number of inliers for E should be greater than the number of inliers for H. For a suitable pair, recover from E the initial rigid body transformation and check if the recovered transformation is mainly given by a sideward motion. In case, promote the pair as seed pair, setting the estimated transformation as initial rigid body transformation (task 3/7, function `solve()`);
2) Triangulate a set of newly registered 3D points from a pair of observations (2D projections) (task 4/7, function `solve()`);
3) Implement an auto-differentiable cost function for the Ceres Solver problem (task 5/7, struct `ReprojectionError()`);.
4) Add a residual block inside a Ceres Solver problem (task 6/7, function `solve()`).
5) Check at each registration iteration if the reconstruction has diverged. In this case, just reset the reconstruction and start from scratch with a new pair of seeds.

Optional (in `basic_sfm.cpp`)) Implement an alternative next-best view selection strategy, e.g., the one presented in class(see Structure From Motion Revisited paper, sec. 4.2).

Note: In basic_sfm.cpp, all observations (2D projections) are normalized, i.e. they refer to the canonical camera, i.e. a camera with matrix K equal to the identity and no distortion.

Detailed descriptions of what is required is reported directly inside the code. Chunks of code to be completed are marked with the preamble:

```
////////////////////// Code to be completed (x/y) ///////////////////
```

followed by a description of the required functionalities.

Please check the README and code carefully, reading all the comments and trying to understand the flow.

What you need to deliver

- Source code (without objects and executables)
- A short written report with:
    - A brief description of the work done, and the problems encountered, if any, clearly specifying who did what;
    - Some qualitative results on at least one of the two provided datasets and at least one dataset acquired from the group with a smartphone or a camera.
- Your (small) dataset (Avoid too high-resolution images: in case rescale them. Frame a simple scenes!) and the obtained point clouds (.ply files) for all the tested scenes.
- **Supplementary work for groups of 3 students**: these groups, if any, in addition to the above activities, should qualitatively test the effectiveness of a deep learning-based feature detection/description scheme.

The results in the report can be shown through point-cloud screenshots (for example, you can use MeshLab to visualize the produced .ply file, see for instance the figure above): they should be “reasonably good” and makes sense with the framed scene. To install MeshLab on a Linux Debian-based distro, type in a terminal window:

```
sudo apt install meshlab
```

To calibrate your camera and produce the calibration parameter file for the matcher app, follow the instructions shared on the course webpage.

HINTS

1) Open CV provides several utility functions for geometry and 3D reconstruction:

https://docs.opencv.org/4.2.0/d9/d0c/group__calib3d.html

Among others:

`void cv::Rodrigues ()`
convert to axis-angle rotation representation and vice-versa.

`Mat cv::findEssentialMat()`
`Mat cv:: findHomography()`
Estimate the Essential matrix and the Homography matrix from two sets of correspondences

`int cv::recoverPose()`
Recovers the relative camera rotation and the translation from an estimated essential matrix

…

2) Read the Ceres Solver bundle adjustment tutorial carefully:

http://ceres-solver.org/nnls_tutorial.html#bundle-adjustment