REPORT LAB 2:
Structure from Motion

Boscolo Simone	simone.boscolo.2@studenti.unipd.it	N.Matricola:	2026826
Forin Paolo	paolo.forin@studenti.unipd.it	N.Matricola:	2019146

1.INTRODUCTION

Structure from motion (SfM from now on) is a computer vision technique that tries to reconstruct a 3D scene from a collection of images taken from different points of view that frame the interested scene. An SfM application generally takes as input a set of images (and the camera parameters, if they are available) and as output it provides the camera positions, the camera parameters and a 3D sparse reconstruction (in our case, a points cloud).

2. IMPLEMENTATION

Our implementation of SfM is divided into two classes: `features_matcher.cpp` and `basic_sfm.cpp`.

`features_matcher.cpp` performs the 'Correspondence Search', in particular we have developed:

1. features extraction and descriptor computation (point 1/5 of the assignment);
2. features matching (point 2/5 of the assignment);
3. geometric verification (point 2/5 of the assignment).

POINT 1 is done by using AKAZE detector and feature descriptor, that is an accelerated version of KAZE detector that exploits the benefits of nonlinear scale spaces
POINT 2 is implemented with an exhaustive matching among all possible image pairs by using `BFMatcher`, which is provided by OpenCV, and the parameter `crossCheck` is set to `true` in order to remove some outliers. Because we are using AKAZE we have set `normType = NORM_HAMMING`, as suggested by the OpenCV documentation.
POINT 3 computes the homography and the essential matrices in order to have two set of inliers, then the set with more inliers is kept to extract the observations later on the code.

Point 1 is developed inside the `extractFeatures` function.
Points 2 and 3 are developed inside the `exhaustiveMatching` function.

`basic_sfm.cpp` performs the 'Incremental Reconstruction' and we have developed only few parts of this process, in particular:

1. part of the seed pair selection (point 3/5 of the assignment);
2. minimize the reprojection error inside the bundle adjustment (point 4/5 of the assignment);
3. define the error function for the reprojection error (point 5/5 of the assignment).

POINT 1 computes the homography and essential matrices and if the provided inliers by the essential matrix are strictly more than the provided inliers by the homography matrix, then the seed pair is found and the camera pose is recovered from the essential matrix; otherwise it passes to the successive pair. After that the pair seed is found, only the inlier points are stored. All the other checks, as the 'already tested pair' for example, are already developed in the provided code.
POINT 2 for each observation (2D point), it minimizes the reprojection error by adjusting the correlated 3D point and camera pose. This is done by adding a residual block inside the Ceres solver with a defined cost function (the reprojection error, see point 3) and by setting as loss parameter the Cauchy loss function.
POINT 3 defines the cost function used as the reprojection error in the previous point. The function takes as input the camera parameters, a 3D point and an observation (the 2D point in the actual image). The function computes the residual among the observation and the 2D projection of the input 3D point for the given camera parameters.

Point 3 is developed inside the `solve` function.
Point 4 is developed inside the `bundleAdjustmentIter` function.
Point 5 is developed inside the `ReprojectionError` struct, that also provides a static function to create a pointer to the `ceres::CostFunction` object that is later passed to the Ceres solver.

3.RESULTS AND COMMENT

The most challenging part has been the feature matching, in particular it has been hard finding a good detector that was able to detect enough features.
We have tested almost all the feature detectors present in the OpenCV library and, after a few tests, ORB and AKAZE were the ones that were providing us the best results in terms of number of features (as output of the `matcher` application), therefore we have decided to study deeper these features detector by changing their default parameters.

Also, in order to obtain good results, in terms of quality of the final point cloud, the `basic_sfm` application needs to be run more times; this is due to the fact that we need to minimize a nonlinear function (the reprojection error) through Ceres solver and this gives different result at each iteration.

After some experiments this is what we have obtained:

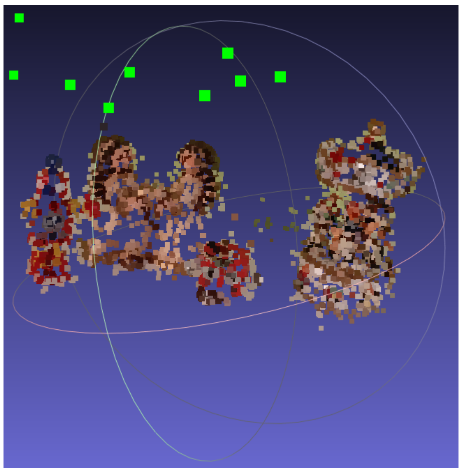
I. AKAZE

PARAMETERS USED

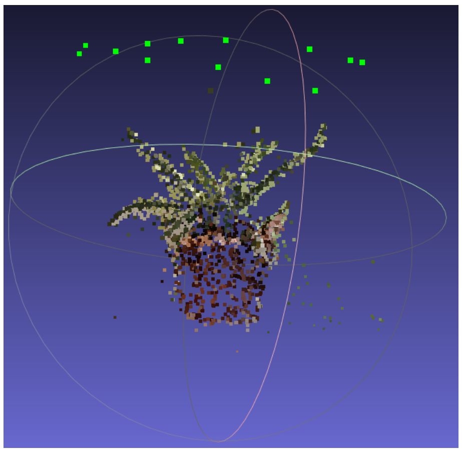
- `threshold = 1e-4`
- `nOctaves = 10`
- `nOctavesLayers = 20`

RESULTS

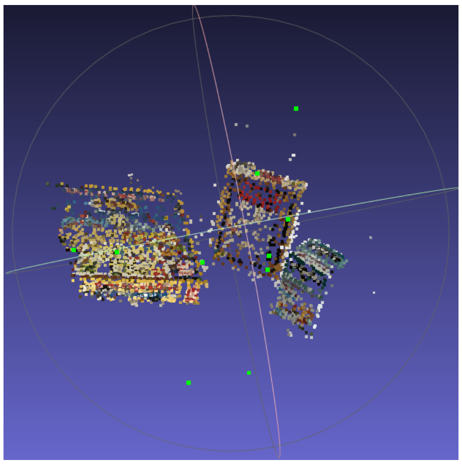
A. Statuettes dataset presents 9 images, in fact the final output presents 9 cameras (green points, if we pass the .ply file to MeshLab application) and a sparse representation of the scene. The obtained result presents 5178 points and, as mentioned before, we needed to re-run the `basic_sfm` app several times.



B. In order to obtain the following good result from the Aloe dataset, we have removed the strict majority condition in `basic_sfm.cpp`, specifically when the program checks if the number of provided inliers by the essential matrix are strictly more than the number of provided inliers by the homography matrix. Also we have removed image 6 from the dataset because it was creating some problems, in particular Ceres was always returning NaN values for the translation and rotation matrices.

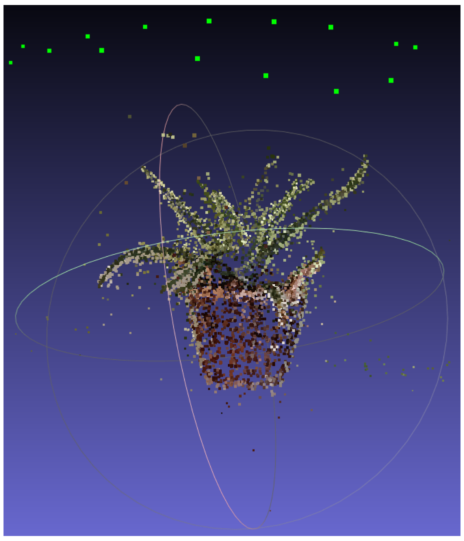


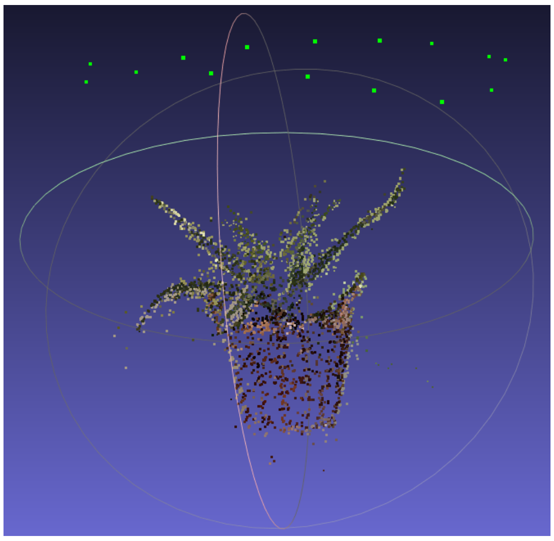
C. The following result is obtained with our datest: AKAZE works very well and the SfM returns 4833 points with all cameras in a correct position.



FURTHERMORE CONSIDERATIONS

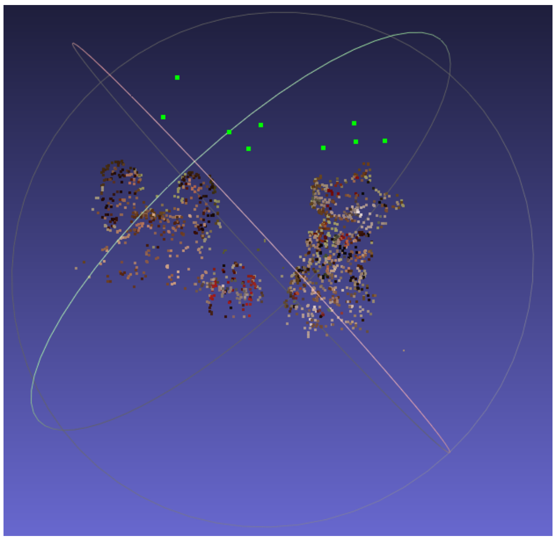
In `featurers_matcher.cpp` we have also tested the `crossCheck` for the initialization of the `BFMatcher`. In the already presented results, the parameter is set to `true`, as in the early presented implementation; but we try to set it to `false` and these are the results:





As we can see, the aloe dataset works very well and this is due to the fact that there are more correspondences between the pairs (more inliers, but at the same time more outliers). For instance, the correspondences obtained with this test are 20/30 times more than the ones obtained with the previous implementation (`crossCheck = true`). The two images present about 7000 points (image on the right) and about 4000 points (image on the left). But, we can notice that there are more outlier points in these images with respect to the ones presented in the previous section.

Unfortunately, this parameter setting doesn't work with the other two datasets (statuettes and our datesets), since the number of outliers for both are too much.
But the best result obtained, with one of these two datasets, is the following one with about 1600 points and all cameras in the correct position. (With our personal dataset we obtained only bad results).



II. ORB

PARAMETERS USED

- `nFeatures = 10 000`
- `scaleFactor = 1.2`
- `nlevels = 35`

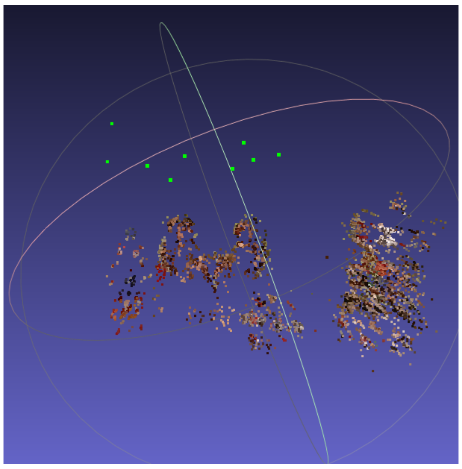
RESULTS

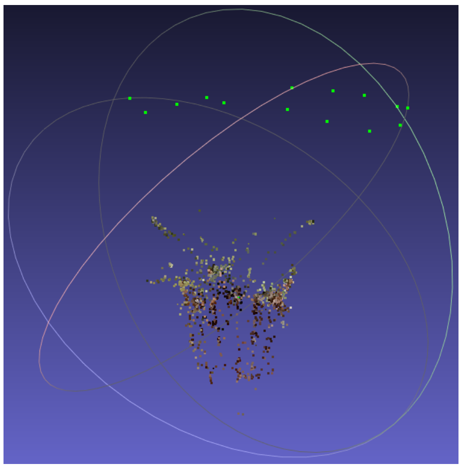
We have tested different parameter settings, but with these we have never obtained the best result, moreover with the Aloe dataset it wasn't able to find the 10 000 features required.
Also, in order to obtain good results, we needed to set the seed pair selection differently, depending on the dataset used as input.
For the statuettes, the seed pair was selected as usual, instead for aloe the seed pair is chosen only by selecting the Essential matrix that was providing more inliers.
Anyway, during the brute force matching, cross checking was always performed in order to remove some outliers.

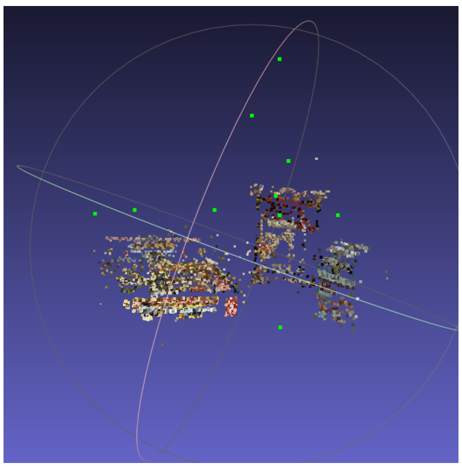
For all dataset we have obtained the correct positions of the cameras and, regarding the points:

- statuettes: 4600 points
- aloe: 1300 points
- personal dataset: 5600 points

We can see the results in the following images:


STATUETTES


ALOE


PERSONAL DATASET

All the images shown in this section can be found as .ply file in the delivered assignment.

SOME NUMERICAL RESULTS

For a better comparison, we have decided to report two tables with some numerical results.

I. `matcher`

In the following table are reported the number of detected points from `matcher` application for each test done with implementations presented above.

Detector	`crossCheck`	Statuettes	Aloe	Personal Dataset
AKAZE	`True`	10566	11630	17608
	`False`	9951	12867	16541
ORB	`True`	11712	6755	14539
	`False`	11282	7425	13376

II. `basic_sfm`

Instead the following table reports the number of 3D points that the `basic_sfm` application has been able to detect in 10 runs.
All the executions are done by providing as input the `matcher` result that uses AKAZE as detector and `crossCheck = true`.

NOTICE THAT *the number of poin1486ts includes the cameras, more precisely it' the value of '*`element vertex`*' present in the .ply file.*

i-th run	Statuettes	Aloe	Personal Dataset
1	3	14	10
2	2559	14	10
3	9	14	10
4	3	14	10
5	9	1486	10
6	4094	3	12
7	3	3	10
8	3	1809	10
9	9	14	10
10	3575	2020	4139

4. CONCLUSION

After several tests and after the considerations done with the presented results, we have decided to keep the previously presented implementation because it provides the best results in terms of visual quality, that are:

- AKAZE as feature detector and descriptor, with the previous presented parameters settings;
- `crossCheck = true`, as parameter for the `BFMatcher`;
- in `basic_sfm.cpp`, the essential matrix is chosen if it strictly provides more inliers than the homography one.

Furthermore, all the results shown in the previous pictures are the best ones and they are obtained by re-running the `basic_sfm` application multiple times (as we can see in the previous tables), since Ceres at each run gives different results and doesn't perform very well (and also differently in different machines).

Also, we have to report that we have found some problems with the provided CMakeLists, therefore we have modified it. We have used:

```
set(CMAKE_CXX_STANDARD 14)
```

instead of

```
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAG} -03 -std=gnu+11 -g").
```

NOTICE THAT: some code lines are commented with a prefix '`//DEGUG:`' these are just standard output code lines that were allowing us to understand the behavior of the algorithm.