STEFANO BINOTTO: 2052421
GIONATA GROTTO: 2052418

REPORT STRUCTURE FROM MOTION

PART 1

We computed the features and descriptors for each pair of images using the ORB detector already initialized in the code. Furthermore, we extracted the color of each feature.
We also tried to use the SIFT detector and we found that its performance was slightly better than the performance of ORB, however we decided to stick with the provided detector since the SIFT approach was giving us compiling problems when tested in the virtual machine environment and not in our personal machine.

PART 2

To compute the matches we used a Brute Force descriptor matcher that is computationally expensive but it's the best matcher available.
We then computed the homography and essential matrix to find out the inliers using the RANSAC algorithm. Testing with several different combinations of thresholds we found out that the best results were given with the threshold equal to 1 for the homography and 3 for the essential matrix.
We saved the inliers that were selected either by the homography or the essential matrix.

PART 3

We computed the essential and the homography matrix and checked if the numbers of inliers of the essential matrix were higher than the number of inliers of the homography matrix. If that was the case then we recovered the rotation matrix and the translation vector using the essential matrix. In the end we checked that the transformation was mainly given by a sideward motion. To do so, we checked if the motion along the y and z axis was below a certain threshold. Initially we understood also to exclude the rotational movement, so from the axis-angle representation we derived the angle magnitude and set a threshold for it. But confronting with other groups we understood that the check was to be done only on the translational part of the transformation, so we removed this part (but it's still present in the code, commented).

PART 4

We recovered the rotation matrix and the translation vector for the camera and the new camera considered using the Rodrigues algorithm. We used these matrices to triangulate the points in the 3D space. We checked the cheirality constraint in order to execute the provided code.

PART 5

We implemented the Functor that computes the Reprojection error for each single point provided.

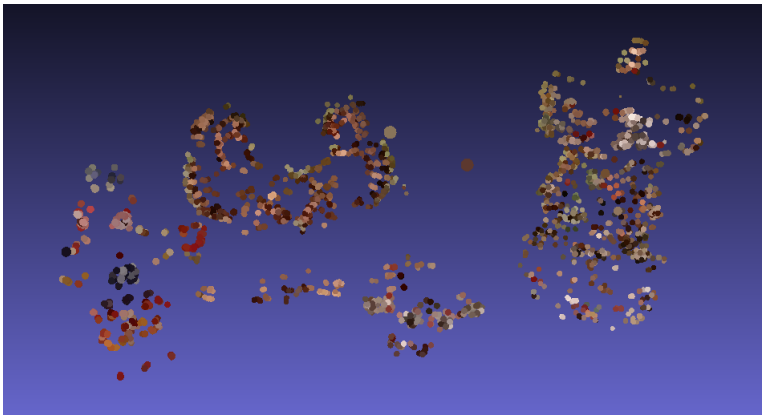
PART 6

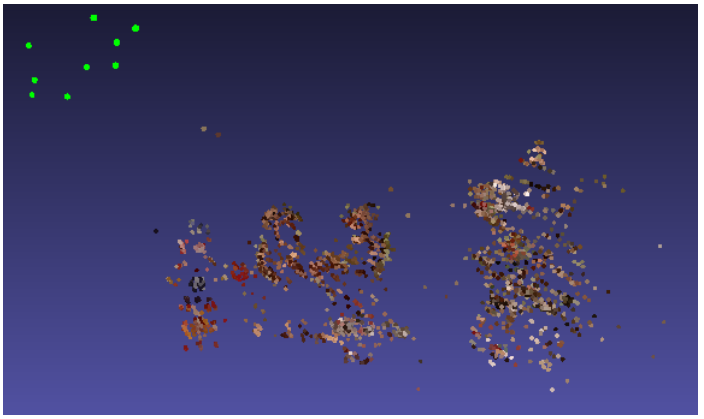
We built the residual block for each observation using the ReprojectionError struct implemented in part 5 and the Cauchy loss function with the suggested parameter.

CALIBRATION & TESTING

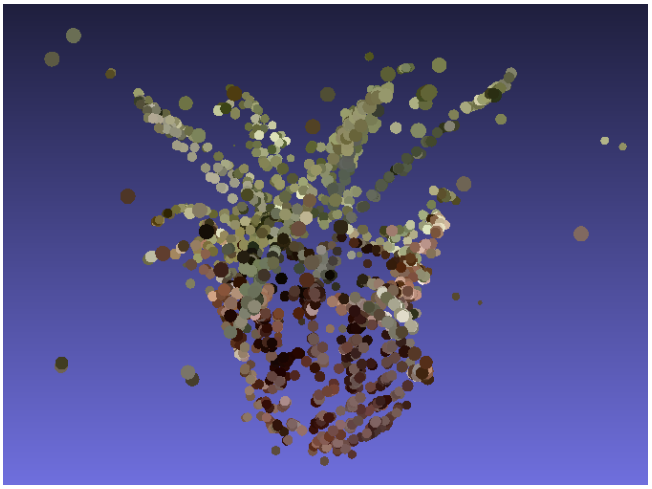
Here are the point clouds we obtained running our code on the “images_1” and “images_2” datasets, with the camera parameters (and focal length = 1,1) already provided in the folder.
The first screenshot of each point cloud is taken from the cameras' point of view, the second is taken from a lateral point of view.

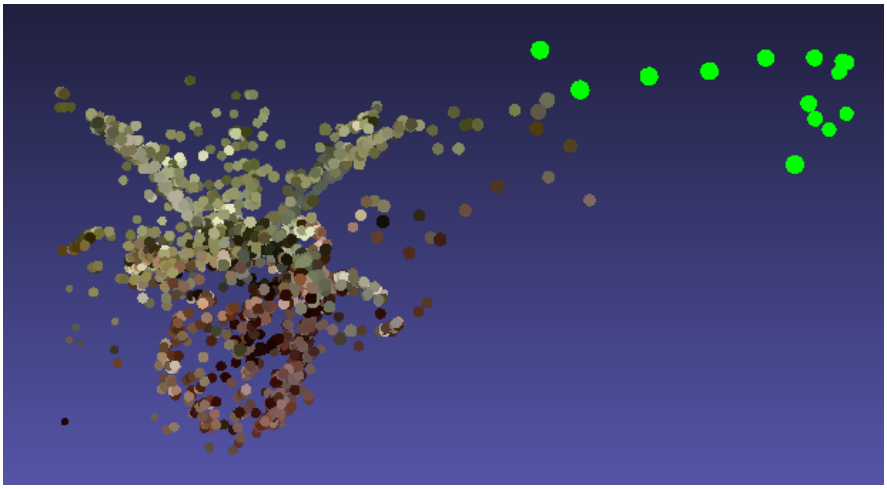
- Images_1:



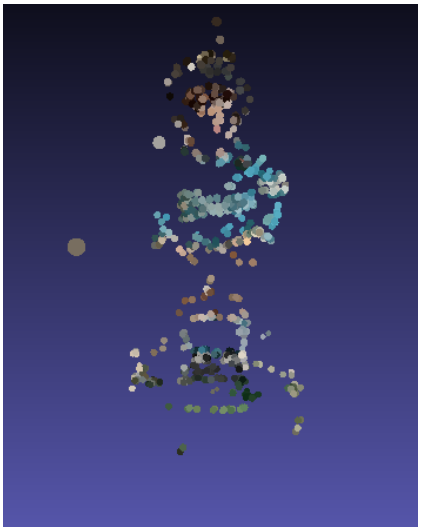


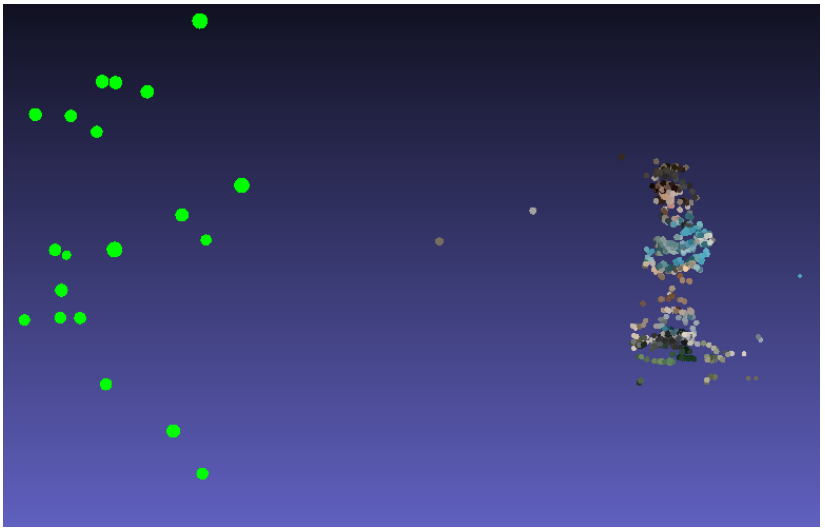
- Images_2:





- Maradona:
We took a bunch of photos of the checkerboard and a little Maradona statue, which we had to rescale to 1280x720, with focal length equal to 1,75.
After having calibrated the camera using our custom dataset of the checkerboard, with total reprojection error equal to 0.29, we tested our code on the testing dataset obtaining very fine results.
Here is one image example of our dataset and below the frontal and lateral point of views of the point cloud.





Note:
The calibration parameters, our custom dataset and the three point clouds we obtained are attached in the folder we submitted.
We developed the whole project together via Zoom.