Report Laboratory 2:

Structure from Motion

Christian Francesco Russo 2087922
Matteo Spinato 2087634

Task 1/7: Feature extraction

For the feature extraction task, we used ORB feature detector and descriptor. Then, for each keypoint found, we extracted and stored the corresponding color value of its pixel.

We tested many other different alternatives: AKAZE, BRISK, SIFT, SURF, KAZE, FAST, however, ORB in general was the one which allowed to obtain the better results and the largest number features.

Task 2/7: Descriptors matching and Geometric validation

For the descriptors matching task, we used a brute force matcher `cv::BFMatcher` (using Hamming norm). In particular, to enhance the results:

- We did a k-NN matching using `cv::knnMatch` with $k = 2$, which allows to detect 2 correspondences in image j for each descriptor in image i.
- Then, we used Lowe's ratio test to filter the false positive matches and to keep the best match in image j for each feature descriptor in image i.

In this way, what we obtained in the end was a 1:1 correspondence between the descriptors of image i and of image j (i.e., the good matches). Then, we stored these good matches and their corresponding keypoints.

For the geometric validation task, we estimated the inlier masks for the essential matrix E, using `cv::findEssentialMat()`, and for the homography matrix H, `cv::findHomography()`, where for both we used RANSAC with threshold 1.0.
Then we selected the best model among the two by taking the one with the highest number of inliers. For the selected model, we extracted and stored the corresponding inlier matches.
Finally, if the number of inlier matches was greater than 5, we set the matches between the two images.

Task 3/7: Rigid body transformation between seed pairs

To compute the rigid body transformation between `seed_pair_idx0` and `seed_pair_idx1`, we first estimated the essential matrix E and the homography matrix H (using `cv::findEssentialMat()` and `cv::findHomography()`) between the inlier matches found in the previous point by using RANSAC with threshold 0.0015.

Then, we checked that the number of inliers for E was higher than the number of inliers for H, in that case, we recovered the rigid body transformation R, t between the two seed pair cameras using the function `cv:: recoverPose()`.

Then, we checked whether the recovered rigid body transformation R, t was mainly sideward motion, by checking that the absolute translation along the x coordinate is greater than the translation along the z coordinate (lateral camera motion), or that the absolute translation along the y coordinate is greater than the translation along the z coordinate (up-down camera motion), in order to discard bad scenarios in which the camera was moved toward or away from the scene. If this condition was true, then we stored the rigid body transformation between the two seeds pair cameras.

Task 4/7: 3D point triangulation

To triangulate each observation seen by the two seed pair cameras, we first extracted and stored the 2D observation points stored in the vector `observations_`. Afterwards, from `cam0_data` and from `cam1_data containing` the 6D pose blocks, we extracted and built the rotation matrix R and the translation vector t for both cameras (w.r.t. the reference camera). Then, we exploited these data to triangulate the 3D point seen from the two cameras using the function `cv::triangulatePoints()`.

At this point, we checked the cheirality constraint by verifying that the depth (i.e., the dehomogenized z coordinate of the 3D point) was positive, that means that the point is positioned in front of the two cameras. If this condition was true, we executed the lines of code provided to store the 3D point.

Task 5/7: Ceres cost function

To implement the struct `ReprojectionError`, which defines the Ceres auto-differentiable cost function of our Bundle Adjustment problem, we just implemented the same code that can be found in the official documentation of Ceres for Bundle Adjustment (at: http://ceres-solver.org/nnls_tutorial.html#bundle-adjustment), with the only difference that in our case we are dealing with a calibrated undistorted camera, so we do not have to use distortion parameters to obtain the final projected point position (i.e., we only have 6 camera parameters, instead of 9).

Task 6/7: Ceres residual block

For creating the residual block we used the Ceres function `ceres::AddResidualBlock()`, where for the cost function we used the `ReprojectionError` struct implemented before, and for the loss function we created a Cauchy loss function with parameter $2 \cdot$ `max_reproj_error`.

Task 7/7: Divergence detection

For the divergence detection task, we implemented two different approaches.

The first approach consists in computing the volume of the current bounding box that encloses all the points of the scene recorded so far, and to compare it with the volume of the bounding box of the previous iteration. If the percentage of growing of this volume is higher than a certain threshold it could be that we are facing an unrecoverable divergence in the registration, so we return false and we reset the computation to start from scratch with a new seed pairs.

The second approach consists in counting, for both cameras and points, the number of cameras and the number of points whose parameters differ more than a certain threshold from the ones of the previous iteration. If the count is greater than a certain threshold, then it means that the reconstruction may diverge, for example, due to incorrect triangulation or incorrect local minima found during the bundle adjustment. In this case, we return false and we reset the computation to start from scratch with a new seed pairs.

Since the two approaches, even if different, provided both very similar results, we decided to keep the first one, because it is simpler than the second one and it contains only one hyperparameter.

Notice that we also tried using both approaches together, but in this way the test failed too often and very often the algorithm was not able to find any point in the final reconstruction.

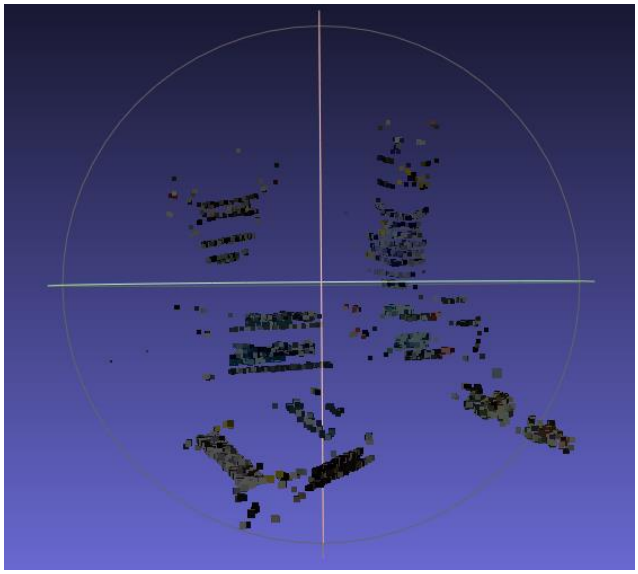
Camera calibration and scene acquisition

The camera calibration was performed using the computer monitor, using a standard A4 paper, and at the end a A3 paper, trying to obtain a good distribution of the corners. The global reprojection error obtained in combination with the best distribution was equal to 0.91. As a result from a cheap camera, it is acceptable.

We acquired more than fifteen scenes, starting from a set of shiny teacups. The problem here was with the artificial light reflection: the single points in the cups reflecting the light were always seen by the algorithm as keypoints to match, instead of considering the actual texturing in the cups, resulting in a totally wrong reconstruction. Then, we used plush toys as a scene, but they were not enough textured to have a good point cloud.

At the beginning, we also tried to keep the camera position fixed and to rotate the scene using a rotating support. Unluckily this idea was not working, maybe because some keypoints are acquired in the rotating part of the image, but some also in the non-rotating part of the plane, creating a confused reconstruction, with all the camera positions condensed in a single position.

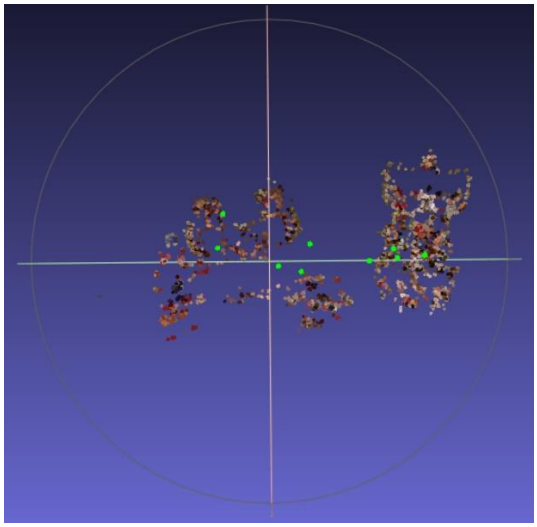
The winning scene representation was obtained using highly textured items, full of text, like cosmetic packages. So, our best performing dataset is composed by 14 images, and it reaches a good number of points in the reconstruction (around 13k points), but still some parts of the items of planar colors are not well reconstructed (like the green part in the proraso package). Here is reported the scene and its reconstruction:



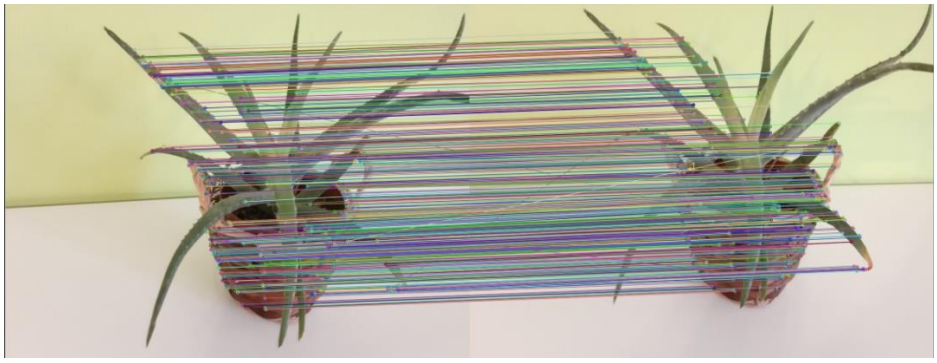
Results

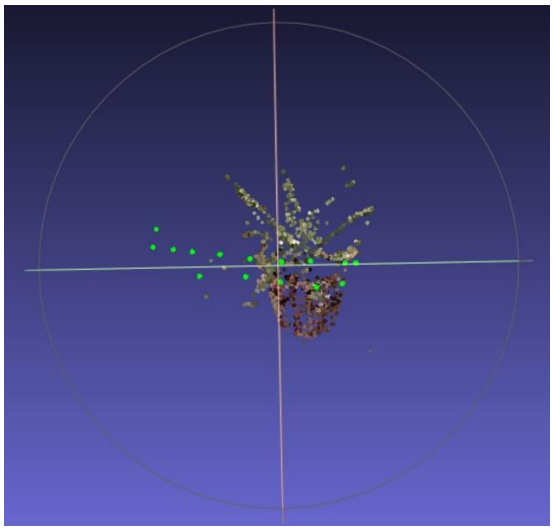
The results were quite satisfying. The ORB method descriptors in combination with our k-NN implementation manage to find a good number of points.

Dataset 1: the final reconstruction contains around 7k points:

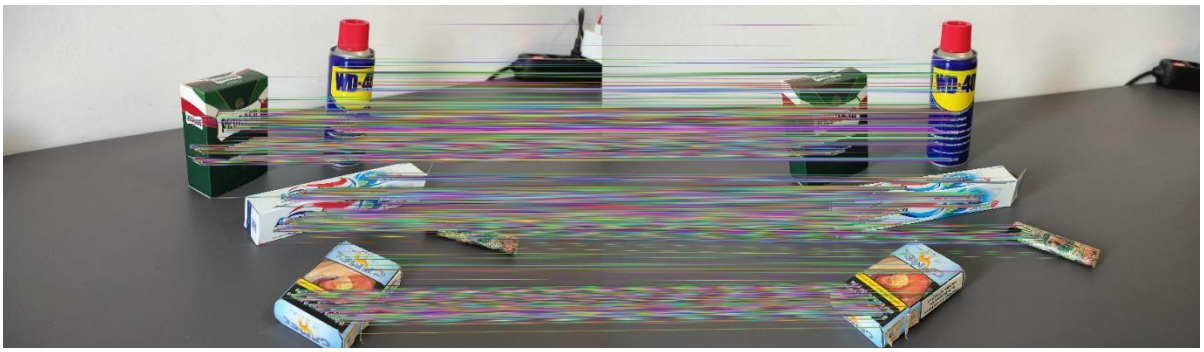


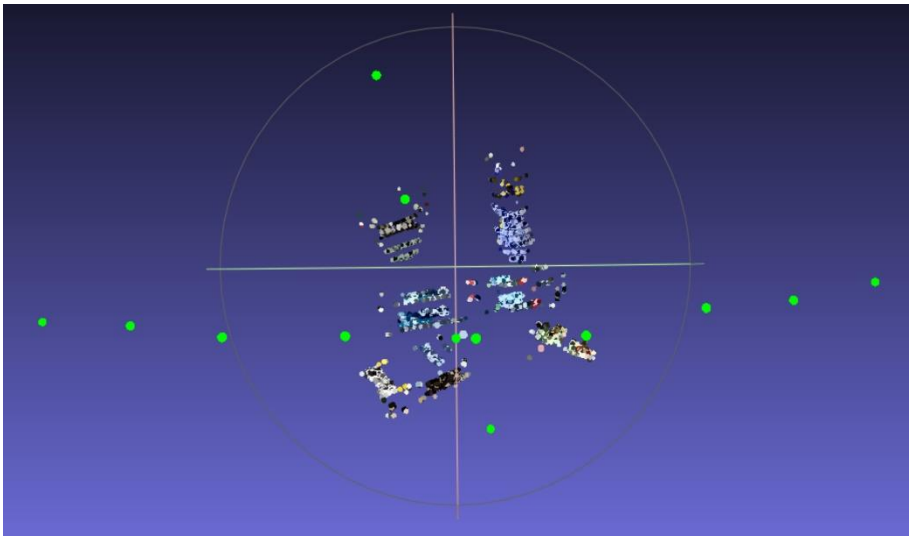
Dataset 2: the final reconstruction contains around $3\mathrm{k}$ points:





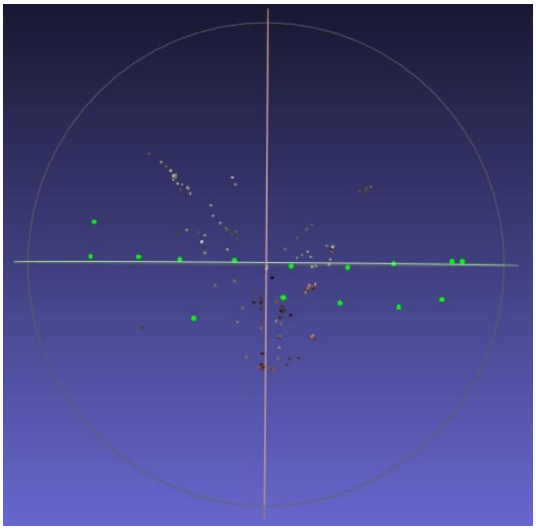
Our dataset: the final reconstruction contains around $13k$ points:



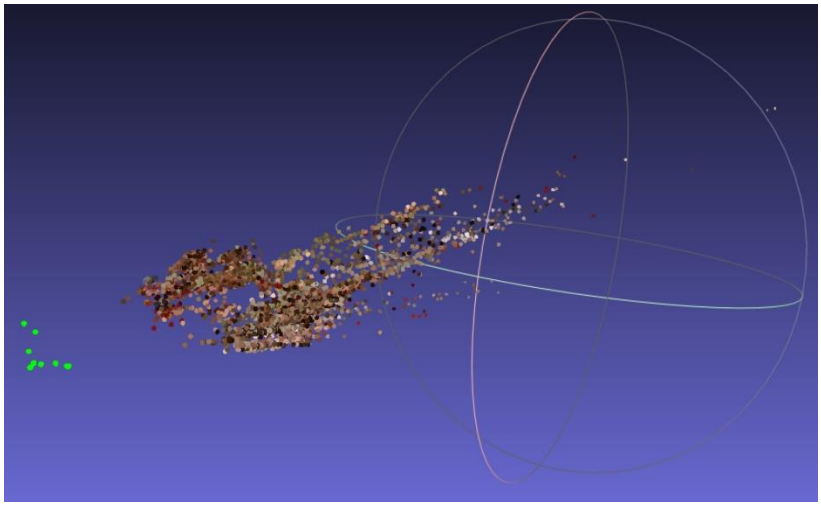


Note: to obtain these reconstructions, the challenge was to find a good value for the RANSAC threshold (at task 3/7) and a good value for the bounding box volume threshold (at task 7/7) that worked well for all the datasets we had.
Bad settings of these variables lead to one of the following possible scenarios:

- Not really good reconstruction with many spurious points or with really few points added, like the one showed below:



- Really bad spread reconstruction, like the one showed below:



- The algorithm is not able to find any good seed pair, so it is not able to add any point to the final reconstruction.

Members contributions

In the first week both of us tried to implement our own code for all the tasks separately, then we meet to discuss and to merge the two implementations. After that, Russo focused on adjusting the code, and Spinato focused on calibration and scene acquisition.

Christian Francesco Russo:

- Implemented code and written report about tasks: 1, 2, 3, 4, 5, 6, 7 and results.

Matteo Spinato:

- Contribution on code about task: 1, 2, 3, 4, 5, 6, 7. Report about camera and scenes and results.