3D DATA PROCESSING

LAB 2

Giuseppe Labate mat.2095665
Mattia Toffanin mat.2096045

Topic: Structure from Motion
Goal: Estimate the 3D structure of a small scene taken by our smartphone from a sequence of images with some field-of-view overlaps.

- **Before running the executable please read the README file and the notes after task 7 in this file**

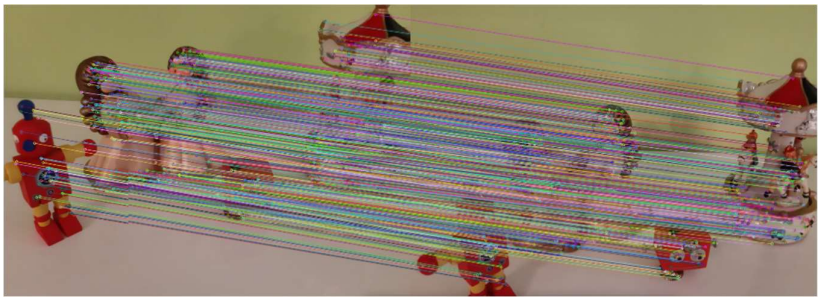
1) Work description

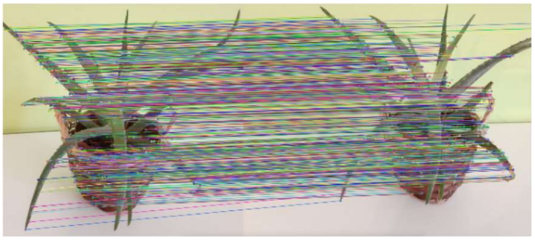
Task 1 Mattia Toffanin

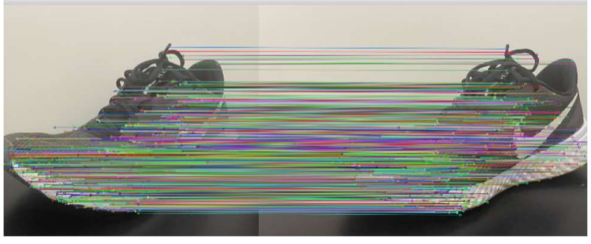
In the first task, we extracted salient points with descriptors using the `cv::detectAndCompute()` function. For each extracted keypoint, the corresponding feature color is stored simply by accessing the corresponding pixel of the image. We tested various detectors such as ORB, BRISK, KAZE, and AKAZE and we decided to use AKAZE (setting accurately its parameters like the threshold, the number of octaves and the number of layers per octave) since it is fast and performs well on all datasets.

Task 2 Mattia Toffanin

In the second task, we exploited a brute-force matcher (`cv::BFMatcher`) using the Hamming norm, since AKAZE descriptors are binary. We implemented `cv::knnMatch()` to find the k best matches (in our case k is equal to 2) and then we applied Lowe's ratio test (with a ratio threshold of 0.8) to discard false positive matches.
After that, we computed the inlier masks for both Essential and Homography matrices, by employing respectively `cv::findEssentialMat()` and `cv::findHomography()` functions with a threshold of 1.0.
While iterating through the matches, we only considered matches present in at least one inlier mask as inlier matches.
In the end, the `setMatches()` function is called if and only if the inlier matches were more than 5.







Task 3 - Giuseppe Labate

In task 3, we had to extract both Essential matrix E and Homograph matrix H to select the seed pair.
To do so, we've used again the `cv::findEssentialMat()` and `cv::findHomography()` functions.
After that, we counted the number of inliers for both models and checked if the number of inliers for E was greater or equal then the one for H.
If this condition wasn't respected, we returned false.
Otherwise, the algorithm proceeded by computing the rigid body transformation between the two camera pairs with `cv::recoverPose()`.
To ensure that we were gaining as much information as possible, we then controlled that the motion of the camera between the two pairs was mostly a sideward one.
We verified this by examining the camera's translation value for the x-axis and ensuring that its absolute value was greater than that of the z-axis.
`if (fabs(t.at<double>(0, 0)) < fabs(t.at<double>(2, 0))).`
Finally, if we pass this control, the rotation matrix and the translation vector are copied respectively to `init_r_mat` and `init_t_vec`.

Task 4 - Giuseppe Labate

In this task, it was requested to triangulate the 3D points by using their observations in the camera poses.
First, we've retrieved the projection matrices `proj_mat0` and `proj_mat1`.
To do so, we've converted the camera rotation parameters (first 3 parameters) from axis angle representation to a rotation matrix, by using `cv::Rodrigues()`.
We then added the translation vector to complete the projection matrix, picking the last three camera parameters.
All these passages are done both for `cam0_data` and `cam1_data`.
Now that the `proj_mat0` and `proj_mat1` are completed, we needed to retrieve the 2D points corresponding to the same 3D point for both camera positions.
We did this by accessing the observations_ vector.

```
points0.emplace_back
(observations_[cam_observation_[new_cam_pose_idx][pt_idx] * 2,
observations_[cam_observation_[new_cam_pose_idx][pt_idx] * 2 + 1]);
```

We had to multiply by 2 because, in observations_ vector, both the x and y coordinates have been stored for each point.
Next, we've computed the triangulation of the 3D point, by using the `cv::triangulatePoints()` function, which reconstructs the 3D point homogeneous coordinates by using the `proj_mat0` and `proj_mat1` matrices and the set of observations taken at the previous step.
Finally, we only had to check if the cheirality constraint was valid, so if the 3D point's z coordinate was greater than 0.
If so, we increment the 3D points counter, transform the coordinates from homogeneous to euclidean and save the found 3D point coordinates into the pt parameter block.

Task 5 - Giuseppe Labate

In this task, we had to fill the `ReprojectionError` struct to create an auto-differentiable cost function for the Ceres solver.
To accomplish this task, we followed the Ceres bundle adjustment tutorial.
While following it, we had to change only some details because we were dealing with a normalized canonical camera.
In fact, in the `*Create()` function we've changed the camera parameters from 9 to 6
In the `operator()` function we've rotated the 3D point using camera angle-axis rotation parameters and then translated it accordingly with camera translation parameters.
With this new roto-translated position, we calculated the predicted 3D point pose and the residual.

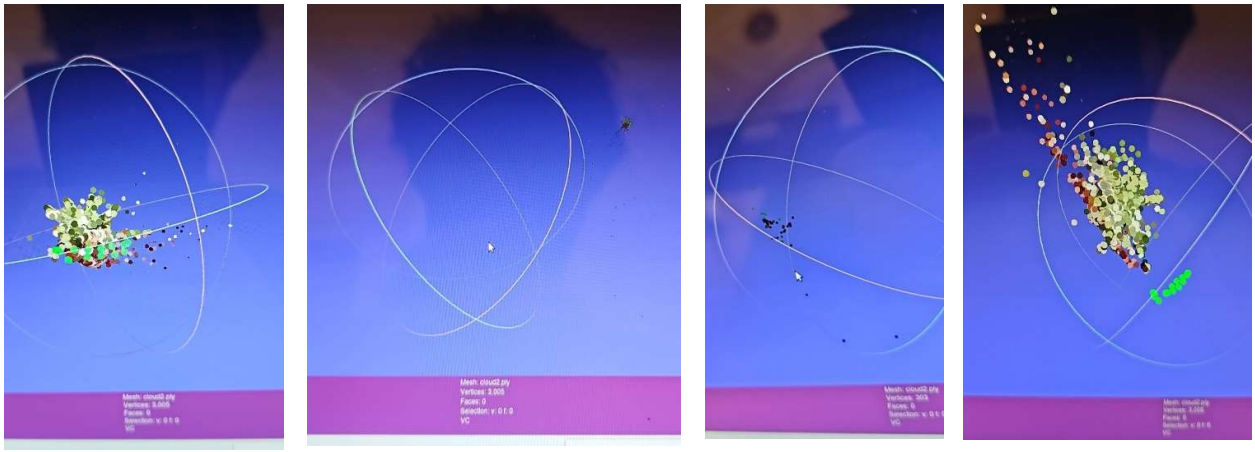
Task 6 - Giuseppe Labate, Mattia Toffanin

In this task, we've simply utilized the `AddResidualBlock()` function to solve the bundle adjustment problem.
We employed the `ReprojectionError` of the i_obs observation as a cost function and a Cauchy Loss as a loss function.
Additionally, the camera and point parameter blocks corresponding to our i_obs were provided as parameters to `AddResidualBlock()`, which handles the computation.

```
ceres::CostFunction *cost_function = ReprojectionError::Create(observations_[i_obs * 2],
                                                               observations_[i_obs * 2 + 1]);
problem.AddResidualBlock(cost_function, new ceres::CauchyLoss(2 * max_reproj_err_),
                         cameraBlockPtr(cam_pose_index_[i_obs]), pointBlockPtr(point_index_[i_obs]));
```

Task 7 - Giuseppe Labate

In this task, it was requested to check when the reconstruction diverges by looking at how previous camera and point positions were updated during the current iteration.
When the reconstruction diverges, these points are, in fact, scattered away from the origin, producing a poor result.
Below we're reporting some of the common bad results.



Here is where we encountered the most difficulties.
This task stopped us for an entire week because all the methods we used didn't provide a good result in detecting when the reconstruction diverged.
The only dataset that posed significant challenges during the reconstruction process was the aloe, primarily due to the presence of outliers.
Below we are reporting all the methods we used.

Method 1 – The bounding box (BB)

In this method, we calculated the max and min coordinates of the point-cloud during iteration *iter.*
Subsequentially, we computed the distance between them, to find the 3D diagonal of the BB containing all the point-cloud.

Then at each iteration, we compared the current diagonal to the previous one.
If their absolute difference was bigger than a finely-tuned threshold, we returned false, because reconstruction has diverged.

The problem lies in determining an appropriate threshold, because sometimes iterations are small even when the reconstruction has diverged.

Method 2 – Bounding box position

To address this, we introduced another check on the origin of the BB.
This origin was calculated by picking the max vertex position and subtracting half of the diagonal from it.

By keeping track of the origin along with the BB dimension, we thought that we would have a great result, but unfortunately, the aloe reconstruction was too unstable to find an effective threshold.

Method 3 – Parameters check

At this point, we tried to check how much the camera and point parameters changed between one iteration and the other.
The objective was to compare current parameters with previous ones to see if we could find a threshold to determine when the reconstruction was diverging.

The problem here was similar to before, sometimes they significantly changed from one iteration to the next, yet the reconstruction remained accurate. Conversely, there were instances where even slight parameter variations led to divergence in the reconstruction.

Method 4 – Eigenvalues check

In this last attempt, we tried to keep track of eigenvalue changes across iterations.
This approach revealed a significant deviation from previous methods, particularly when the z-axis eigenvalue spiked to 34 upon reconstruction divergence.
This discovery was crucial because usually, the aloe eigenvalues span around (0.020, 0.05, 1.2 or sometimes 0.6).

So first our idea was to check the relative differences between iterations for each eigenvalue component.
We've noticed that if the reconstruction was good, the relative differences were certainly under 100%, so we've found a loose threshold.

Then this idea was almost discarded because when the reconstruction diverged, the changes could sometimes exceed 100% in either direction, or vary continuously but less drastically.
So the threshold we've found is a loose one.

By combining this threshold with the fact that the third eigenvalue couldn't exceed 30 or fall below 0.2 (as observed in other problematic reconstructions), we derived a more reliable final threshold.

Notes:

- For all these methods we've tried to find the difference from one iteration to the other, but sometimes we've tried to compute the one before and after bundle adjustment.
- All datasets exhibited stable behavior, maintaining nearly constant differences between iterations.
- To be sure, we put a threshold on the relative difference of 150%
- Because of the different point-cloud shapes, the eigenvalues changed a lot from one dataset to the other, complicating the task of establishing a suitable threshold specifically for aloe reconstructions without adversely affecting other datasets.
  For example for dataset 1, the third eigenvalue goes to 14, which is significantly higher than the good values for dataset 2.
- The bad reconstruction occurred only for the aloe 10% of the time.
- If the result appears bad, please execute again.
- If it still has problems switch from the eigenvalues method to the BB one.
- The very good results seen in this report are achieved by using no task 7 or BB method with a loose threshold.
  But the problem was that 20% of the time we had bad results because reconstruction diverged.
  So task 7 makes the execution more stable by sacrificing the number of points in the point-cloud.
  It's a trade-off between stability and splendid results.
- Please read the README file

2) Encountered problems

During the development of the project, we encountered some issues.
In the matcher application, the main problem was to find the right feature extraction model and to set all the parameters as best as possible so that the matches would work correctly for all datasets, especially for the aloe one, which produces the highest number of outliers.
So, we tried with all possible feature extractors and all possible parameters until we found the best ones, which, for us was the AKAZE feature extractor.

```
// create feature extractor
cv::Ptr<cv::AKAZE> akaze = cv::AKAZE::create();
akaze->setThreshold(0.0001);
akaze->setNOctaves(8);
akaze->setNOctaveLayers(6);

// cv::Ptr <cv::ORB> orb = cv::ORB::create(10000, 1.2, 8);
// cv::Ptr <cv::BRISK> brisk = cv::BRISK::create(60, 4, 1.5);
// cv::Ptr <cv::KAZE> kaze = cv::KAZE::create(true, false, 0.0005, 8, 4);
```

Here there are AKAZE parameters followed by other feature extractor parameters we've tried.

The biggest problem we've encountered was finding a good condition to determine if the reconstruction has diverged (task 7).
We've described this problem and how we addressed it in the task 7 paragraph.

3) Acquired dataset - Mattia Toffanin

The third dataset was acquired using an iPhone 12 Pro.
The camera has been accurately calibrated and its parameters have been stored in the our_camera.yml file.
For this custom dataset, we've collected 10 images of a shoe from different angles.
We then rescaled the images and formatted them in `.jpg`.
For the feature matcher algorithm, it was considered a focal length of `1.1`.

4) Qualitative results

Here are some qualitative results of our project (left zoom in, right with number of vertices):