3D Vision
Homework 2

Mustafa İzzet Muştu
504211564

May 26, 2023

Task 1 - Camera Calibration

I chose 6 different images from the given dataset. Instead of manually finding the corner points in the images, I used OpenCV's built-in findChessboardCorners function. Outputs of this function for these selected images are seen in Figure 1.

Figure 1: Chessboard patterns.

For the implementation of Zhang's camera calibration, I also utilized another OpenCV function called calibrateCamera. This function returns an intrinsic camera matrix, distortion coefficients for all views, and rotation and translation vectors for each different view. By using these, I distorted the images and the result can be seen in Figure 2. It can be observed from Figure 2 that there is a small translation between the image center and the camera center. In addition, we can say that there is a small skewness between default and undistorted images. These are also can be seen in the intrinsic camera matrix. I also included the L2 norm reprojection errors of each image. It is shown in Figure 3.

Task 2 – Augmented Reality

To project the given model on the board, we need to match the axis of the mesh and the chessboard. After observing the mesh model via pyvista library, I rotated it around the x-axis

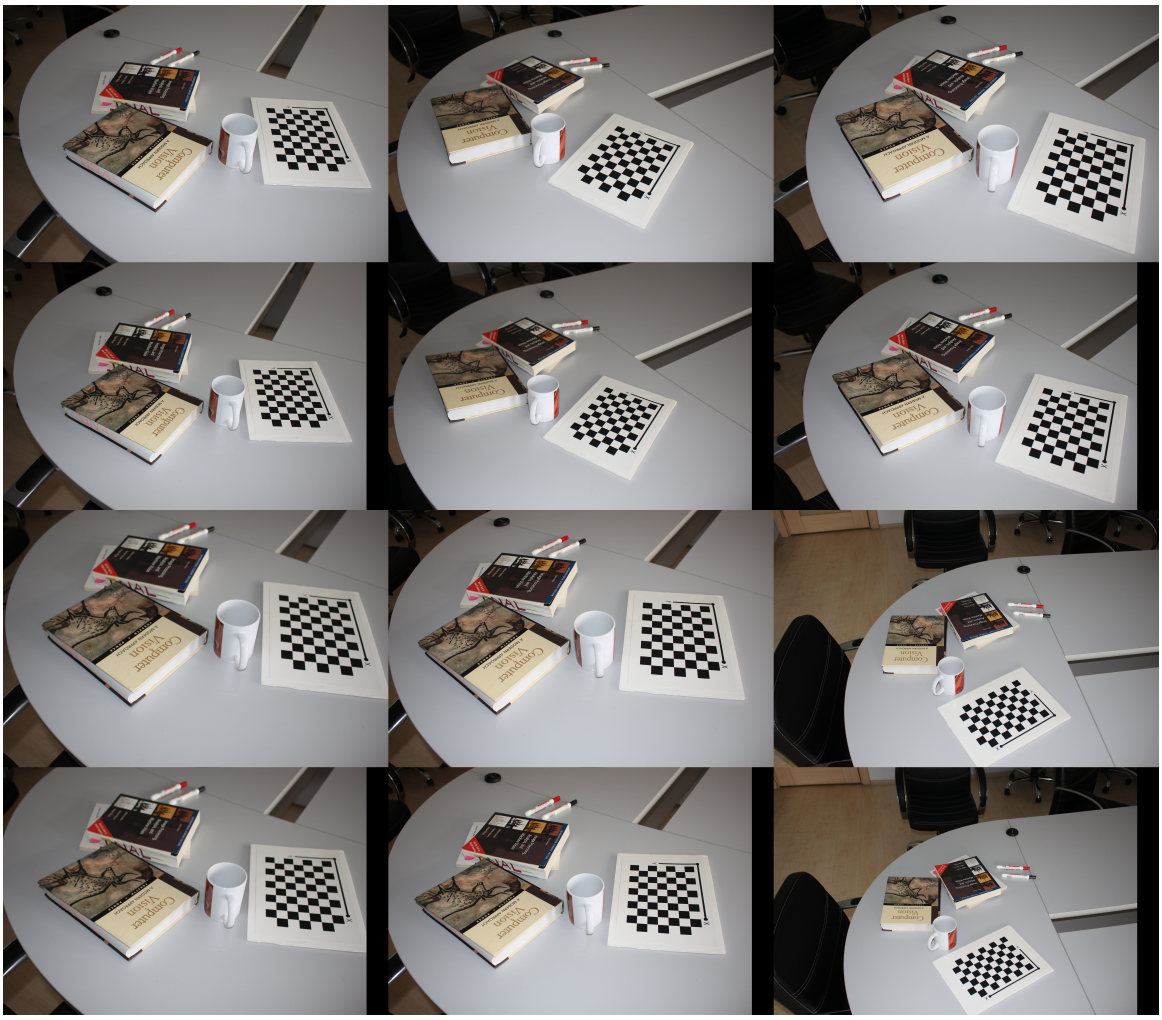

Figure 2: Original (top) and undistorted (bottom) images.

by 90 degrees and scaled it by 0.2 to fit the chessboard. 3D mesh points before and after this operation are shown in Figure 4. For each selected image, I projected the mesh structure on the chessboard. The pipeline of this operation is as follows:

1. Undistort the image.
2. Project the mesh points on the chessboard by using intrinsic (K, same for all views) and extrinsic matrix (R_i and T_i, different for each view).
3. Draw small circle around the projected points via OpenCV.

After this pipeline, I got the results that can be seen in Figure 5.

Task 3 – 8-Point Algorithm & Depth Estimation

At first, I draw green circles around corresponding points in the images so that I can better know what I should expect. These corresponding points can be seen in Figure 6. After that, I undistort the points, which means I converted the points from pixel coordinates to camera coordinates by using the intrinsic camera matrix K. The difference is shown in Figure 7. In the 8-Point algorithm, I used points in camera coordinates. The pipeline of the 8-Point algorithm with triangulation:

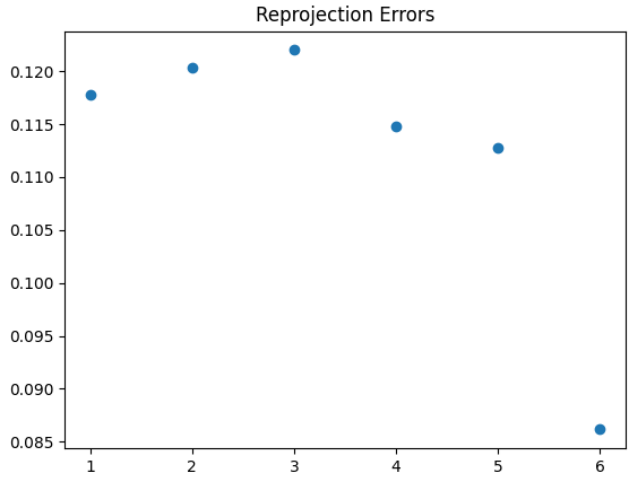

Figure 3: Reprojection errors of camera calibration.

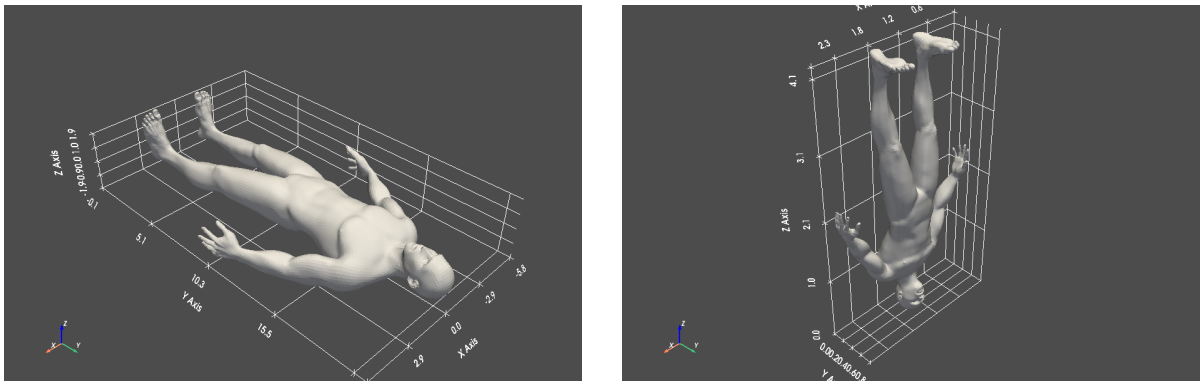

Figure 4: Default (left), rotated and scaled meshes (right).

1. Normalize the points with 0 mean and $\sqrt{2}$ (in homogeneous coordinates) variance and also store normalization matrices.
2. Find A matrix and estimate E, essential matrix, by using SVD.
3. Project estimated E matrix onto essential space.
4. Denormalize essential matrix via normalization matrices.
5. Decompose the correct combination of R and T from 4 possibilities.
6. Estimate the 3D point by using camera matrices via triangulation (direct method).

Structured 3D point cloud by using this pipeline is shown in Figure 8. For a task at this level, the created point cloud looks nice to the eye and looks realistic.

Figure 5: Mesh projected images.

Figure 6: Corresponding points from different views.

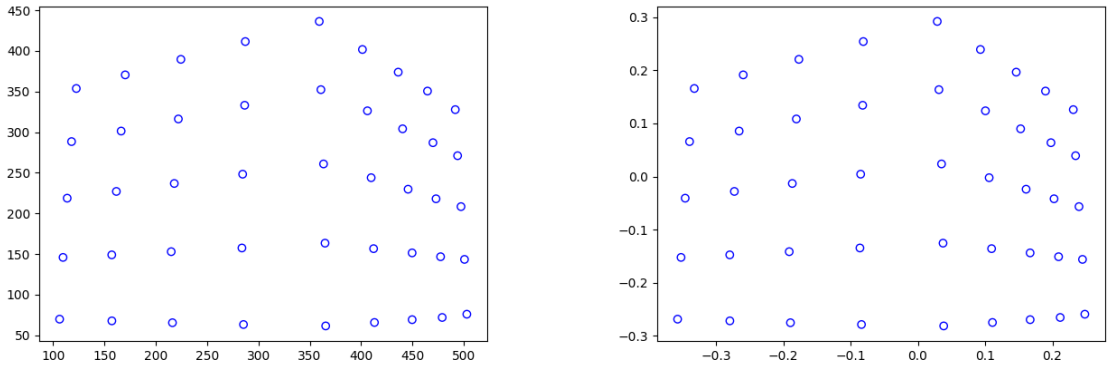

Figure 7: Difference between pixel coordinates (left) and camera coordinates (right).

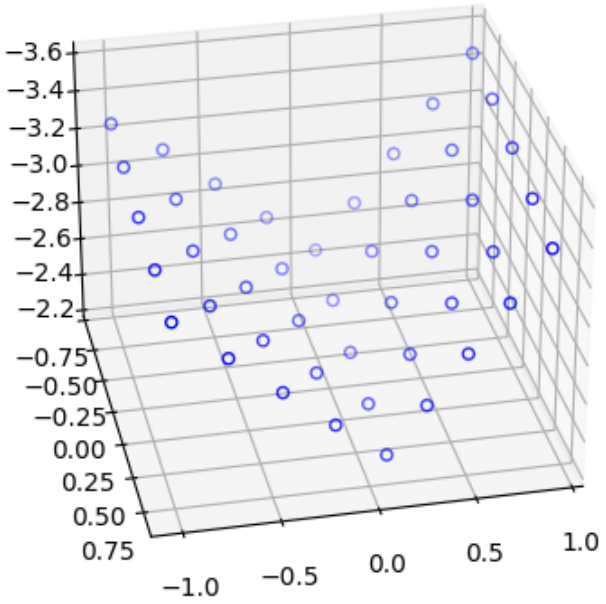

Figure 8: Structured 3D point cloud.