

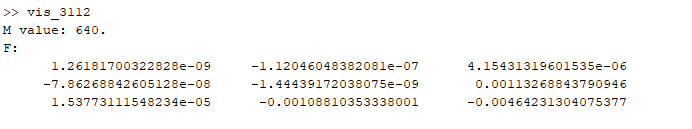
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | |
|  | 3D Reconstruction | | | | | |  |
|  |  | | | | |  | |
|  | | |  |  | | | |
|  | | | Utsav Anantbhat |  | | | |
|  | | | 4/4/2023 |  | | | |
|  | |  | | |  | | |

3.1. Sparse Reconstruction:

In this section, we wrote a set of functions to compute the sparse reconstruction from two sample images of a temple. We first estimated the Fundamental matrix, computed point correspondences, then plotted the results in 3D.

3.1.1. Implement the Eight Point Algorithm:

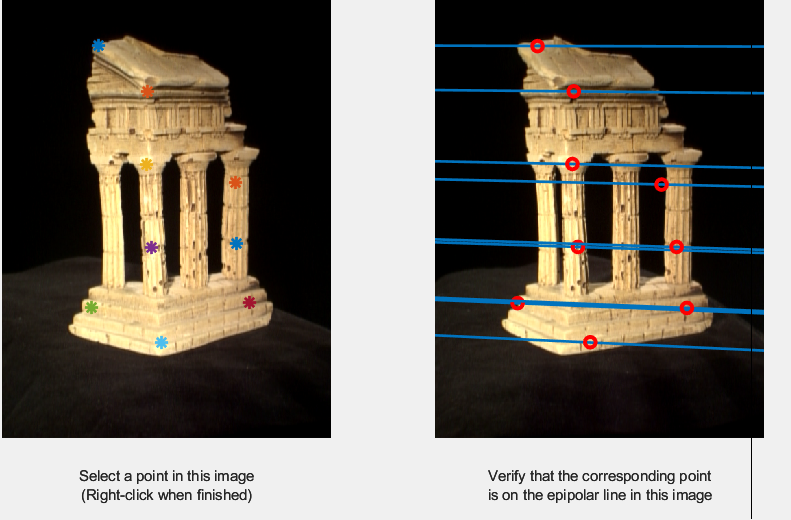
According to the handout, we used the eight point algorithm to estimate the fundamental matrix. We recovered F and visualized some epipolar lines as required:

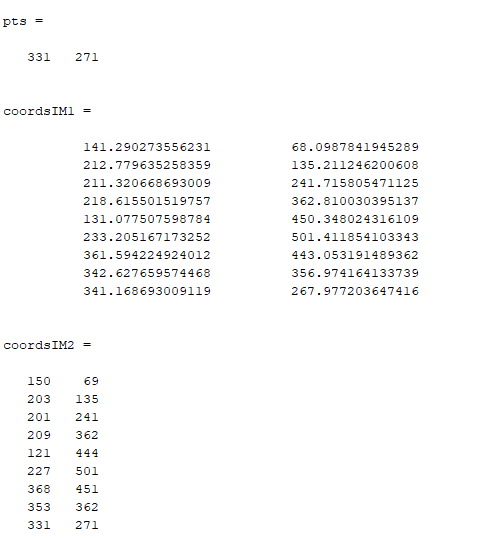




3.1.2. Find Epipolar Correspondences:

As stated in the handout, in this section, we found many point pairs to acquire a rough outline of the 3D object, in this case a temple. We wrote a function called epipolarCorrespondence which takes as arguments the pair of temple images (stereo pair), fundamental matrix F (from eightpoint), and pts1, a Nx2 matrix containing the (x,y) points in the first image. As required, we used the epipolarMatchGUI.m file to visually test the epipolarCorrespondence function:

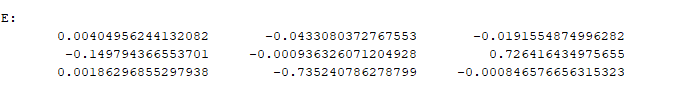




For the similarity metric, we decided to match the windows of the two images. In the second image, we search features in a range respective to the x value from the first image. This avoids the selection of similar features.

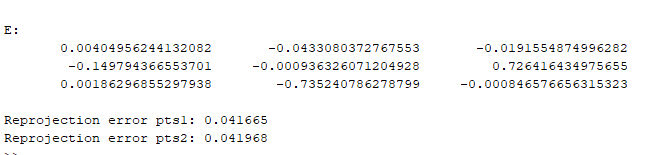
3.1.3. Write a Function to Compute the Essential Matrix:

As stated in the handout, here we computed the Essential matrix, E. While F is the Fundamental matrix computed between two images, K1 and K2 are the intrinsic camera matrices for the first and second image respectively (contained in intrinsics.mat), E is the computed essential matrix. We estimated E for the temple image pair provided:



3.1.4. Implement Triangulation:

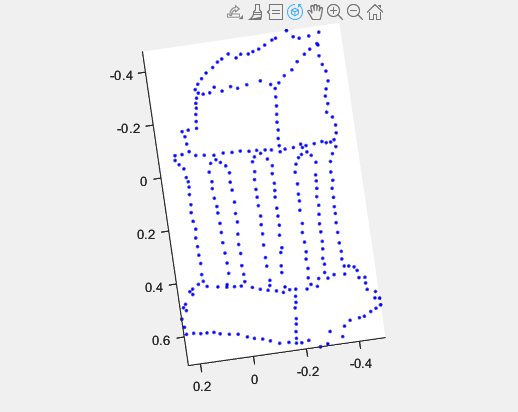
Here, we wrote a function to triangulate pairs of 2D points in the images to a set of 3D points with the form “function pts3d = triangulate(P1, pts1, P2, pts2).” Upon implementation, we checked the performance by looking at the re-projection error. To compute the re-projection error, we projected the estimated 3D points back to the image 1(2) and computed the mean Euclidean error between projected 2D points and pts1(2):

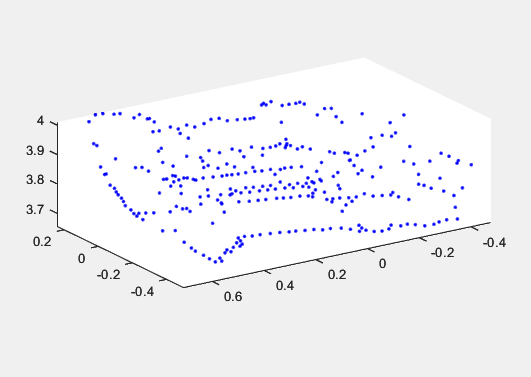


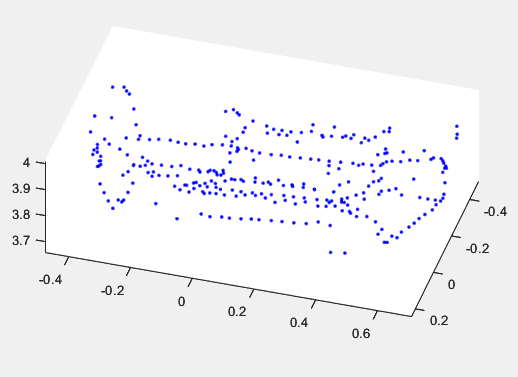
The reprojection errors are less than 1, meaning that the implementation was correct.

3.1.4. Write a Test Script That Uses templeCoords:

Here, we created a test script as outlined in the handout to generate a full 3D reconstruction. Here are images of the reconstruction from different angles:







3.2. Dense Reconstruction

In this section, we wrote a set of functions to perform a dense reconstruction on our temple examples. Given the provided intrinsic and computed extrinsic parameters, we wrote a function to compute the rectification parameters of the two images. The rectified images are such that the epipolar lines are horizontal, so searching for correspondences becomes a simple linear. This was done for every point. Finally, we computed the depth map.

3.2.1. Image Rectification

Here, we wrote a function that computes rectification matrices. Here is the output of testRectify.m as stated in the handout:



3.2.2/3. Dense Window Matching to Find Per Pixel Density + Depth Map

Here, we wrote a function to create a disparity map from a pair of rectified images (im1 and im2). We then wrote a function that uses the obtained disparity and creates a depth map. Here is the output of the test scripts as desired by the handout:

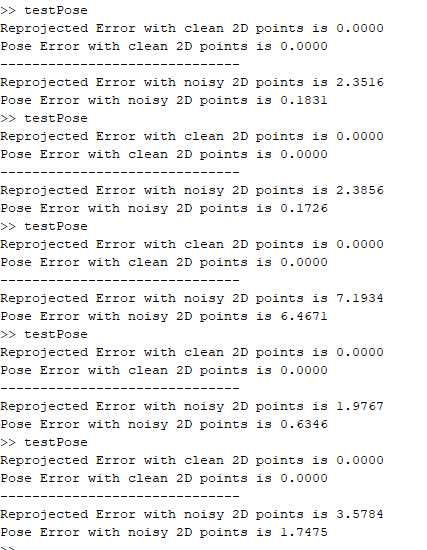


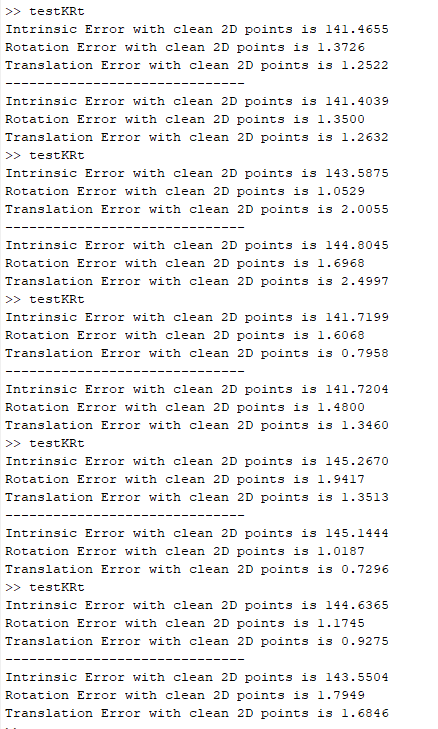
3.3. Pose Estimation

In this section, we estimated both the intrinsic and extrinsic parameters of camera given 2D point x on image and their corresponding 3D points X.

3.3.1/2. Estimate Camera Matrix + Estimate Intrinsic/Extrinsic Parameters

Here, we wrote a function that estimates the camera matrix P given 2D and 3D points x, X. We then wrote a function that estimates both intrinsic and extrinsic parameters from camera matrix. Here is the output of the test scripts as requested by the handout:





Each test was run 5 times to get a decent estimate of the values.