The goal of this assignment is to implement a stereo reconstruction algorithm on two given view images, $\mathcal{I}_1$ and $\mathcal{I}_2$, shown in Figure 1.

Figure 1: *Two view images $\mathcal{I}_1$ and $\mathcal{I}_2$.*

SIFT Feature Matching: Function `find_match()` uses `OpenCV` SIFT to extract keypoints and match between two views based on k-nearest neighbor search. The matches are filtered using the ratio test with threshold 0.7 and bidirectional consistency check. This function is the same as that in Homework 2. The matched points between images $\mathcal{I}_1$ and $\mathcal{I}_2$ are shown in Figure 2.

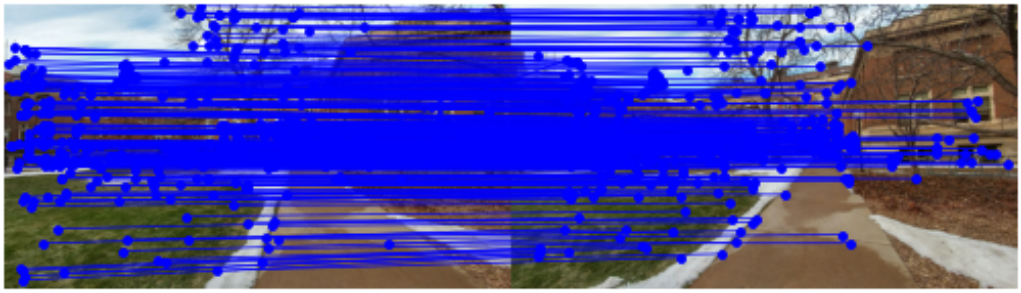

Figure 2: *Match points between $\mathcal{I}_1$ and $\mathcal{I}_2$ using SIFT features.*

Fundamental Matrix Computation: Function `compute_F()` computes a fundamental matrix by the 8-point algorithm within RANSAC. When implementing RANSAC, the threshold is set to be 0.01 and the number of iteration is set to be 1000. SVD clean-up is used to make the rank of the fundamental matrix to be 2. The validity of the fundamental matrix is verified by visualizing epipolar lines as shown in Figure 3.

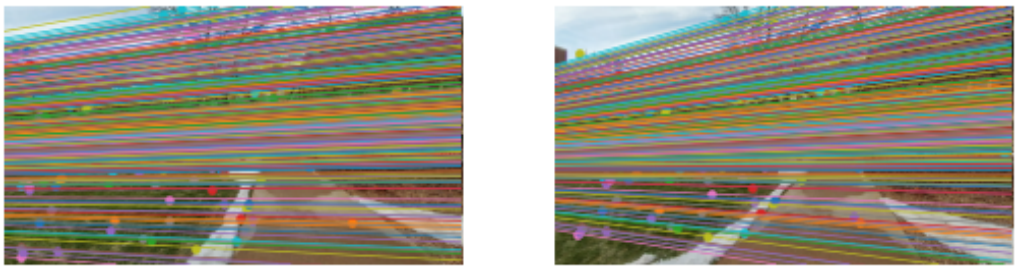

Figure 3: *The epipolar lines based on the fundamental matrix.*

Triangulation and Pose Disambiguation: Function `triangulation()` uses linear triangulation method to reconstruct 3D points given camera pose and correspondences. Then given four configurations of relative camera pose and reconstructed points, function `disambiguate_pose()` finds the best camera pose by verifying through 3D point triangulation. We find the 3D points that are in front of both cameras by checking the Cheirality condition. Four configurations produce 93, 11, 309 and 391 valid points respectively and the fourth one gives the maximum number of valid points (391), and therefore is the best configuration. The camera pose configurations with point cloud are shown in Figure 4.

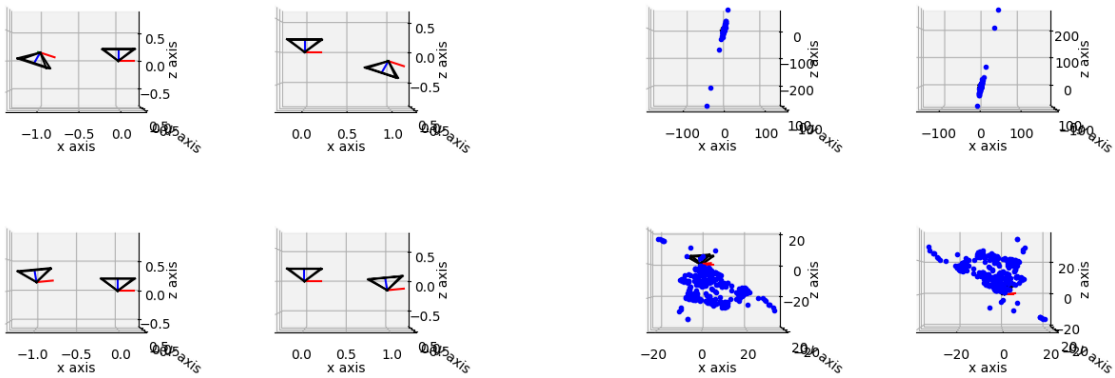

Figure 4: *Four camera pose configurations with point cloud. The number of validation points are 93, 11, 309, and 391 respectively.*

Stereo Rectification and Match: Given the disambiguated camera pose, function `compute_rectification()` find the rectification rotation matrix R_{rect} and computes the rectification homographies $H_1 = KR_{\text{rect}}K^{-1}$ and $H_2 = KR_{\text{rect}}R^{\intercal}K^{-1}$. The rectified images $\tilde{\mathcal{I}}_1$ and $\tilde{\mathcal{I}}_2$ are shown in Figure 5. This rectification sends the epipoles to infinity and makes the epipolar line to be the horizontal line.

Figure 5: *The rectified images $\tilde{\mathcal{I}}_1$ and $\tilde{\mathcal{I}}_2$.*

To visualize the stereo match, function `dense_match()` computes the dense matches across all pixels, i.e., the disparity matrix. Given a pixel, $\mathbf{u}$, we sweep along its epipolar line (which is the horizontal line after rectification) and find the disparity d that produces the best match, i.e.,

$$d = \arg\min_i \left\{ d^1_{\mathbf{u}} - d^2_{\mathbf{u}-(i,0)} : i = 0, 1, \cdots, u_x - 1 \right\},$$

where $d^1_{\mathbf{u}}$ is the dense SIFT descriptor at $\mathbf{u}$ on image $\tilde{\mathcal{I}}_1$ and $d^2_{\mathbf{u}-(i,0)}$ is the SIFT descriptor at $\mathbf{u}-(i,0)$ on image $\tilde{\mathcal{I}}_2$ and u_x is the x-coordinate of pixel $\mathbf{u}$. For pixels out of the range of the image $\tilde{\mathcal{I}}_1$, we simply set the disparity as $d = 0$. The visualization of the stereo match is shown in Figure 6.

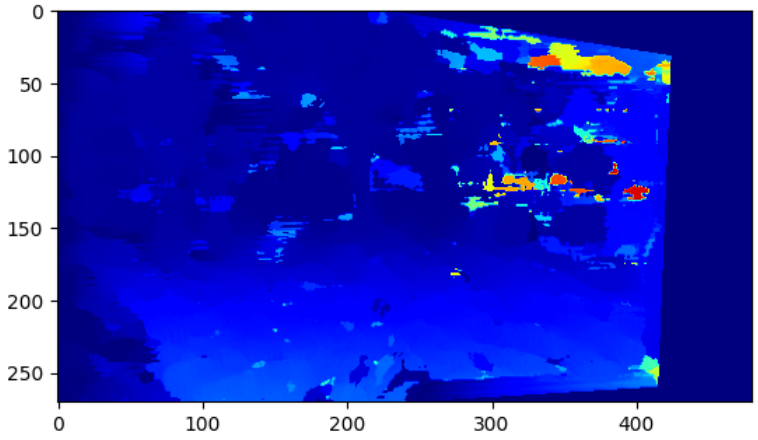

Figure 6: *Visualization of stereo match.*