PROJECT 2B: CAMERA MODEL AND STEREO DEPTH SENSING

Tanmay Khandait
Student ID : 1219385830
CIDSE, Arizona State University
tkhandai@asu.edu

23 April, 2021

ABSTRACT

This project is a programming project about camera motion and structure reconstruction. This report is divided into three parts, where every part talks about the three task as per the project document.

1 Planar Homography (Task 5)

If there is a pinhole camera that takes an image of objects on a place like the ground, the image plane and world plane can be related to each other by a planar homography. This homography can be estimated by at least four 2D-2D point correspondences in general placement. The homography is also invertible, which allows us to map image pixels to points on the world plane, or reconstruct the world plane from images.

1.1 Theory

If we have the image and world coordinates of a certain object, they can be related by a planar homography as follows. The corresponding diagram is shown in figure 1

$$w \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{21} & H_{21} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} * \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

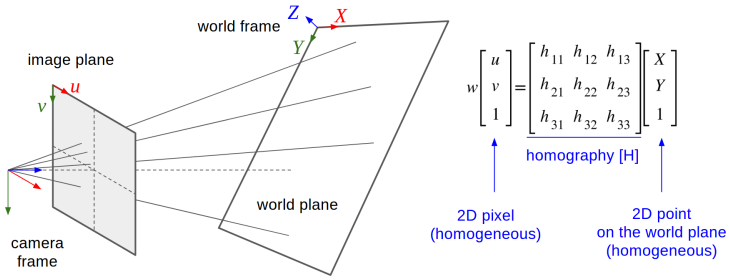


Figure 1: The image and world plane can be related to each other by a planar homography

This homography matrix H can be with objective of minimizing the back-projection error. This can be solved using a least-squares method as discussed in the class. This is done by cv2.findHomography() function. Once this homography is obtained, the idea is to apply this perspective transformation on the input image. This is done by the cv2.warpPerspective() function.

The cv2.warpPerspective() function roughly works by first transforming the image by applying the homography. It then tries to estimate the padding needed in terms of the background. Why? The transformation (rotation and/or translation)

can lead to the transformed pixels going out of the source image bounds. Hence, this motivates the need to ensure that we have padded the image. The original image is then transformed to new image on the padding.

The homography matrix that we obtain are as follows (rounded off to two decimal places):

$$H_{\text{Left image 0}} = \begin{bmatrix} 0.22 & 1.91 & 87.09 \\ -0.85 & 3.84 & 69.89 \\ -0. & 0.01 & 1. \end{bmatrix}$$

$$H_{\text{Left image 1}} = \begin{bmatrix} 0.6 & 0.77 & 124.6 \\ 0.08 & 1.87 & 193.75 \\ 0. & 0. & 1. \end{bmatrix}$$

$$H_{\text{Right image 0}} = \begin{bmatrix} 0.19 & 1.65 & 109.24 \\ -0.74 & 3.2 & 111. \\ -0. & 0.01 & 1. \end{bmatrix}$$

$$H_{\text{Right image 1}} = \begin{bmatrix} 0.66 & 0.88 & 124.45 \\ 0.11 & 2. & 194.21 \\ 0. & 0. & 1. \end{bmatrix}$$

1.2 Practical

The original images and undistorted images have been shown in figure 2. The corresponding warped perspectives have been shown in figure 3. Camera pose for one of the images (left image 0) has been shown in figure 4.

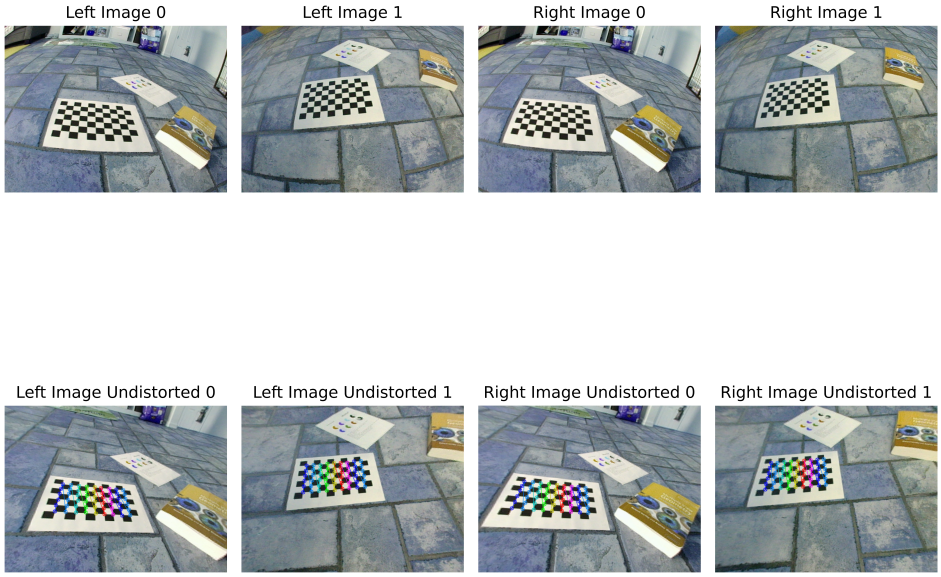


Figure 2: Original and Undistorted Images

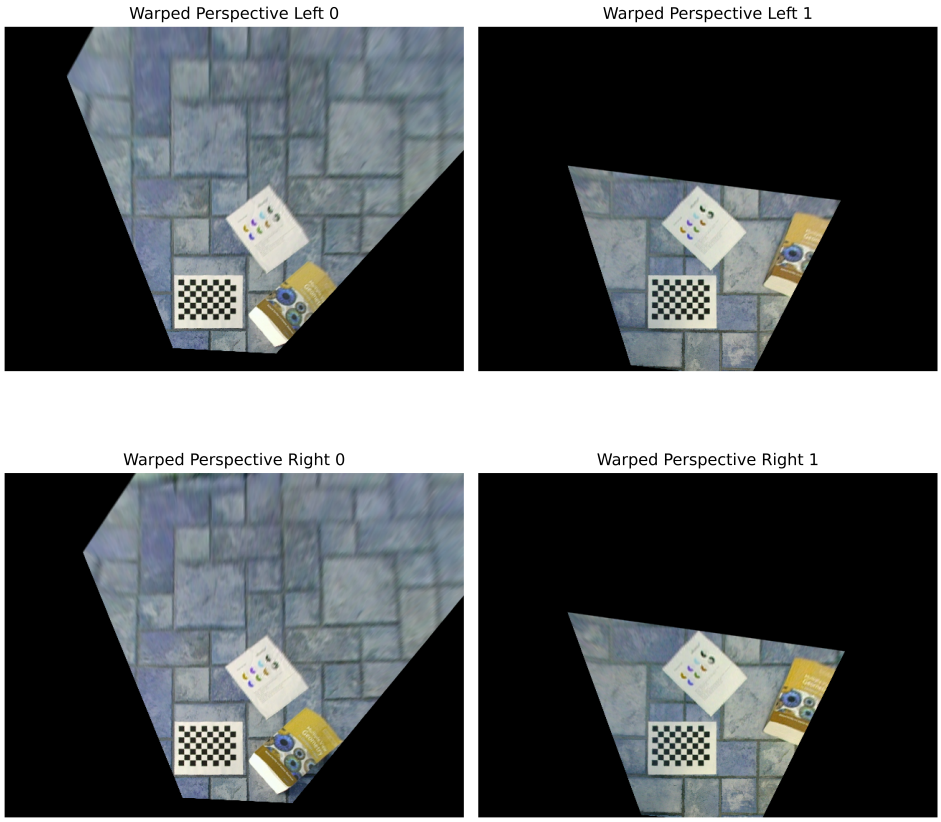


Figure 3: Warped Perspective of different images

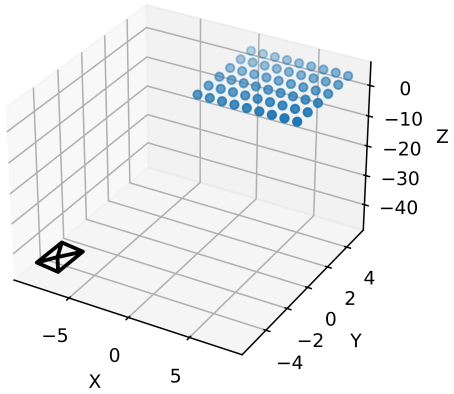
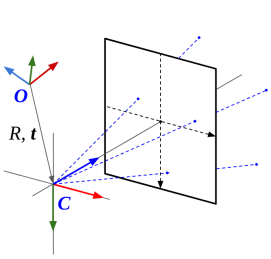


Figure 4: Camera pose for Left Image 0

2 Camera Pose Estimation from a Single View (Task 6)

If we are given a calibrated camera, localizing its pose in the world coordinate system (3D) is possible using four image correspondences. These are called the landmark points, which are mostly the man-made markers. These markers are places at points whose 3D locations are known. For this project, we make use of the ArUco markers. These markers comparatively easy to detect and efficient to detect and are widely used for pose estimation from a single view. The idea is to use the 3D and 2D corresponidences and estimate the pose of the camera. This can be done using cv2.solvePnP() function. This function takes in the 3D and the 2D point correspondences and solves them using various optimization functions. For this task, we used the cv2.SOLVEPNP_ITERATIVE() function which uses the Levenberg-Marquardt



The **Perspective-N-Points (PnP) problem** is essentially calculating the camera projection matrix given N pairs of 3D-2D point correspondences (not collinear).

- A camera has 6 DOF, the minimal solution requires 3 point correspondences in general position (not collinear).
- There are a few methods to solve this problem, such as P3P, EPnP, nonlinear optimization, etc.
- Usually RANSAC is used for robust estimation. A nonlinear optimization step is also commonly used for refining results.

Figure 5: Perspective-N-Points problem

optimization, also known as the damped-least squares solution. The intuition behind this type of optimization method relates to curve fitting ideas. The rotation and translation vectors have been shown below (rounded off to two decimal places).

$$R_{\text{Left Camera 0}} = \begin{bmatrix} 0.99 & -0.01 & 0.1 \\ 0.01 & 1.0 & 0.03 \\ -0.1 & -0.03 & 0.99 \end{bmatrix}$$

$$R_{\text{Left Camera 1}} = \begin{bmatrix} 1.0 & -0.01 & 0.06 \\ -0.02 & 0.92 & 0.39 \\ -0.06 & -0.39 & 0.92 \end{bmatrix}$$

$$R_{\text{Left Camera 2}} = \begin{bmatrix} 0.86 & 0.06 & 0.51 \\ 0.21 & 0.87 & -0.45 \\ -0.47 & 0.49 & 0.73 \end{bmatrix}$$

$$R_{\text{Left Camera 3}} = \begin{bmatrix} 1.0 & -0.04 & 0.04 \\ 0.06 & 0.92 & -0.38 \\ -0.02 & 0.38 & 0.92 \end{bmatrix}$$

$$R_{\text{Left Camera 4}} = \begin{bmatrix} 0.95 & -0.0 & 0.32 \\ -0.02 & 1.0 & 0.06 \\ -0.32 & -0.06 & 0.95 \end{bmatrix}$$

$$R_{\text{Left Camera 5}} = \begin{bmatrix} 0.97 & -0.02 & -0.23 \\ 0.0 & 1.0 & -0.07 \\ 0.23 & 0.07 & 0.97 \end{bmatrix}$$

$$R_{\text{Left Camera 6}} = \begin{bmatrix} 0.97 & -0.02 & 0.23 \\ 0.03 & 1.0 & -0.05 \\ -0.23 & 0.05 & 0.97 \end{bmatrix}$$

$$R_{\text{Left Camera 7}} = \begin{bmatrix} 1.0 & 0.06 & 0.07 \\ -0.06 & 0.99 & -0.09 \\ -0.08 & 0.09 & 0.99 \end{bmatrix}$$

$$R_{\text{Left Camera 8}} = \begin{bmatrix} 0.94 & -0.07 & 0.32 \\ 0.06 & 1.0 & 0.05 \\ -0.32 & -0.03 & 0.95 \end{bmatrix}$$

$$R_{\text{Left Camera 9}} = \begin{bmatrix} 0.95 & 0.24 & -0.22 \\ -0.23 & 0.97 & 0.08 \\ 0.23 & -0.03 & 0.97 \end{bmatrix}$$

$$R_{\text{Left Camera 10}} = \begin{bmatrix} 0.92 & -0.22 & -0.33 \\ -0.0 & 0.83 & -0.56 \\ 0.39 & 0.51 & 0.76 \end{bmatrix}$$

$$T_{\text{Left Camera 0}} = \begin{bmatrix} 0.88 \\ 1.91 \\ -17.48 \end{bmatrix}$$

$$T_{\text{Left Camera 1}} = \begin{bmatrix} 0.4 \\ -4.34 \\ -17.65 \end{bmatrix}$$

$$T_{\text{Left Camera 2}} = \begin{bmatrix} -9.87 \\ 10.21 \\ -20.02 \end{bmatrix}$$

$$T_{\text{Left Camera 3}} = \begin{bmatrix} 0.09 \\ 8.39 \\ -17.02 \end{bmatrix}$$

$$T_{\text{Left Camera 4}} = \begin{bmatrix} -6.59 \\ 2.38 \\ -17.34 \end{bmatrix}$$

$$T_{\text{Left Camera 5}} = \begin{bmatrix} 7.91 \\ 3.4 \\ -18.08 \end{bmatrix}$$

$$T_{\text{Left Camera 6}} = \begin{bmatrix} -1.76 \\ 3.85 \\ -12.51 \end{bmatrix}$$

$$T_{\text{Left Camera 7}} = \begin{bmatrix} 1.45 \\ 5.08 \\ -26.77 \end{bmatrix}$$

$$T_{\text{Left Camera 8}} = \begin{bmatrix} -6.3 \\ -0.66 \\ -18.78 \end{bmatrix}$$

$$T_{\text{Left Camera 9}} = \begin{bmatrix} 11.42 \\ -1.28 \\ -19.32 \end{bmatrix}$$

$$T_{\text{Left Camera 10}} = \begin{bmatrix} 12.13 \\ 10.23 \\ -17.78 \end{bmatrix}$$

2.1 Practical

The detected ArUco markers for the right images has been shown in figure 6 and for the left images has been shown in figure 7. The camera poses of different cameras have been shown figure 8.

Figure 6: ArUco Markers for Right images

Figure 7: ArUco Markers for Left images

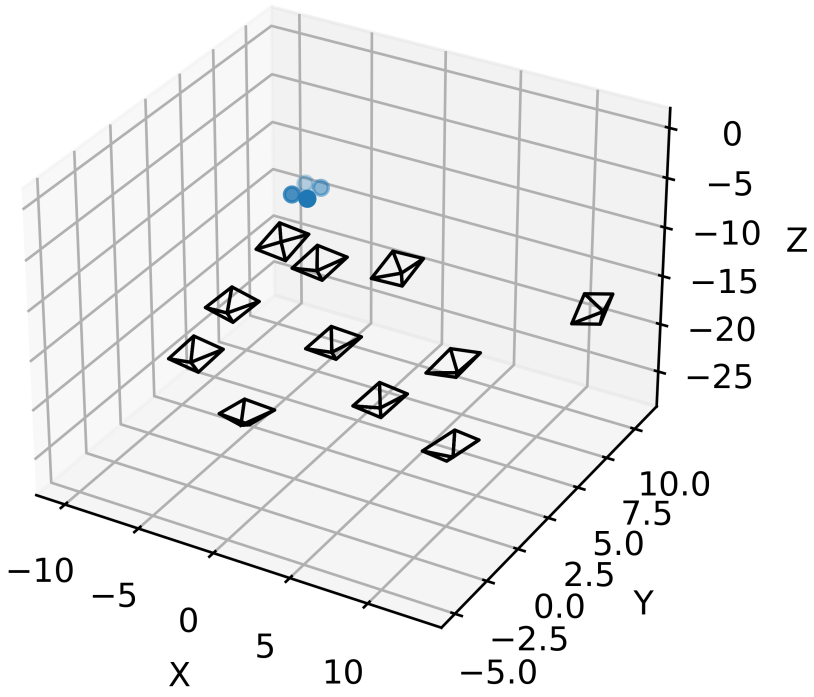


Figure 8: Camera Pose Problem

3 Camera Pose Estimation and Reconstruction from Two Views (Task 7)

Two view reconstruction is essentially creating the 3D structure (usually a point cloud) from two views taken by the same camera at two poses. In this task, we reconstruct sparse 3D points from two views and recover the camera motion.

The function cv2.findEssentialMatrix() helps in determining the essential matrix from the matches found in the two corresponding images. This function makes use of the 5-point algorithm. The idea is that we use the image point correspondences to determine the essential matrix. The idea of solving this problem can be based on the epipolar geometry. The essential matrix obtained is shown below (rounded off to two decimal places).

$$E = \begin{bmatrix} 0.01 & 0.01 & -0.7 \\ -0.3 & 0.05 & 0.07 \\ 0.63 & -0.1 & 0.04 \end{bmatrix}$$

The function cv2.recoverPose() uses the essential matrix from the cv2.findEssentialMatrix() function the two image correspondences to find the relative camera pose. It starts by decomposing the essential matrix and then uses then enforces the positive depth constraint (cherialty check) on the triangulated 3D points.

3.1 Practical

The matches found before and after the constraint can be found in figure 9 and figure 10 correspondingly. The recovered camera pose and the reconstructed 3D points are shown in figure 11.

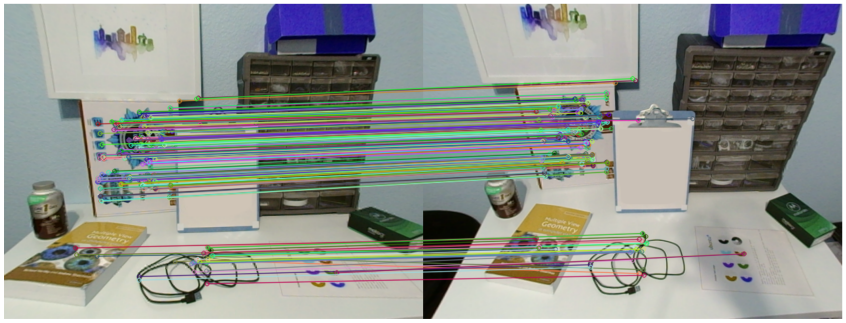


Figure 9: Matches before enforcing the constraint using the Essential Matrix

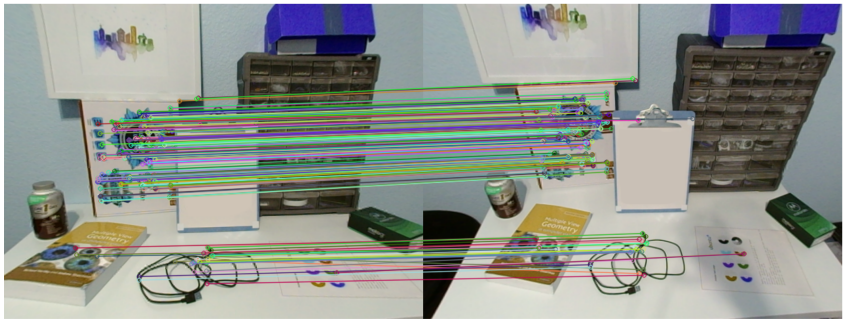


Figure 10: Matches after enforcing the constraint using the Essential Matrix

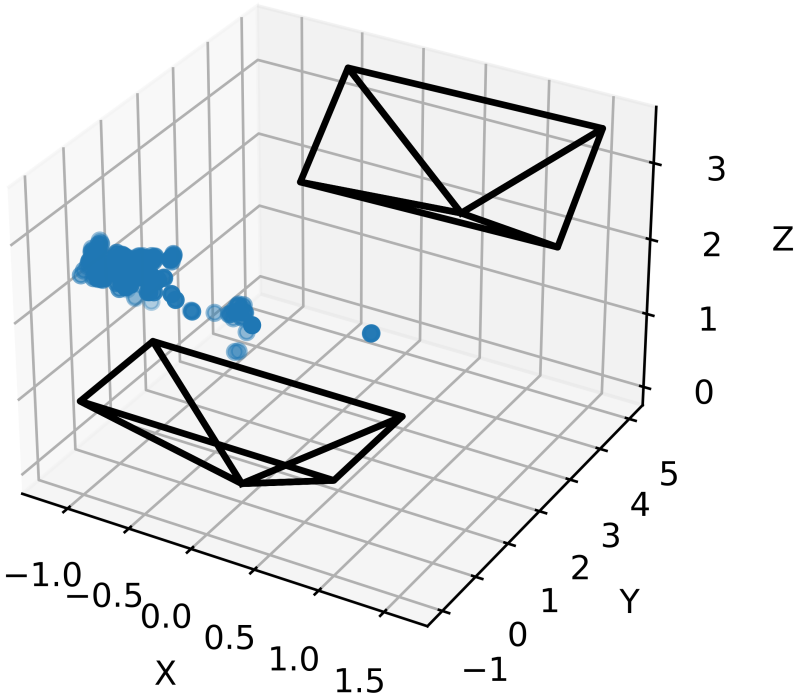


Figure 11: Recovered Camera Pose and Reconstructed 3D points