Camera Calibration
In camera calibration, a vector of C3DPoints (src3D) is given along with a vector of C2DPoints (src2D) containing the corresponding 2D points of the 3D points specified by the user in the image.

STEP1: Classifying points
From src3D, I first classified the 3D points in the vector into those in xz planes and those in the yz planes, placing each point in the newly created vectors of src3D_xz or src3D_yz (or src3D_others, if they didn't belong to either of the two planes). I put the corresponding 2D points into src2D_xz or src2D_yz (or src2D_others).

STEP2: Estimating plane-to-plane projectivity
Then, I initialized a D matrix for each plane (D_xz and D_yz) using the known 3D and 2D values of the points from the corresponding vectors (src3D_xz, src3D_yz, src2D_xz and src2D_yz). The number of rows in each matrix is 2*number of points in the vector, and the number of columns in the matrix is 8. For each plane, I also created a matrix f (f_xz and f_yz) which are both of dimension (2*number of points, 1), consisting of x and y value from the 2D points stacked in one column. Then, I calculated the pseudoinverse of the two D matrices and multiplied them with corresponding f matrices to get the plane-to-plane projectivity for both planes(stored in p_xz and p_yz). I converted the p_xz and p_yz to a 3 by 3 matrix (stored in p_xz_33 and p_yz_33).

STEP3:Using plane-to-plane projectivities to assign 2D coordinates for corners
I then used the given real-world dimension of the grid in the image to algebraically calculate the real-world coordinates of all corners in the grid in the image. I converted these 3D points into corresponding 2D points by making use of the p_xz_33 and p_yz_33 matrices.

For the 2D points converted from the 3D world coordinates using the aforementioned method, I found the closest point in the corners array. The closest point was only stored if the distance between the two points was smaller than 3, and the closest 2D point from the corners array were stored in closest2D, and its corresponding 3D point was stored in closest3D.

STEP4: Estimating 3*4 camera projection matrix
I then created a matrix A using the values from closest2D vector and closest3D vector. I used linear least squares method to find the values of the 3*4 camera projection matrix by performing SVD on A and taking the last column of V. I divided the values of the retrieved column by its last value and converted it to a 3*4 matrix (stored in matPrj).

Decompose
STEP1:QR decomposition
I took the 3*3 submatrix from prjMatrix and stored it in Temp. I then performed QR decomposition on its inverse. I stored the inverse of the resulting R into calibration matrix K and transpose of Q into rotation matrix R.

STEP2: Normalize K
I normalized the matrix K. I divided K by its last value (K(2,2)).

STEP3: Translation vector T
Using the last columns of prjMatrix, I calculated the translation vector. prjRt was constructed by combining the rotation matrix and the translation vector.

Triangulate
STEP1: number of points & number of images
I identified the number of points & the number of images by making use of src2Ds[0]'s size and prjMats' size.

STEP2: for each point, calculate matrix A and solve for corresponding 3D coordinates.
I set up two equations from each point in one image that correlate 2D point to 3D point, and I expressed the collective set of equations from all images on that point into a matrix equation Asol= b where sol is a vector of the point's unknown 3D coordinate. For each point, I set up a matrix A using the values in given projection matrices of corresponding images. I also set up a matrix of one column (b) using values from the given projection matrices. I found the pseudoinverse of the matrix A and multiplied it by matrix b to retrieve the 3D point in sol vector.