Implemented colmap for RGB images and rgb depth images.

**Instructions to load the dataset and images:**

To implement the code for RGB-D point cloud, you need to load the images first. In order to do that you need to put all RGB images along with the depth image placed next to it in a folder. Your folder should look something like this Eg: rbg\_image1.png depth\_image1.png rbg\_image2.png depth\_image2.png rbg\_image3.png depth\_image3.png....... and load the folder by

To implement the code for RGB point cloud, you just need to put all RGB images in a folder.

**Code implementation:**

Part one has the code for the creation of point clouds using rbgD and part 2 has the code for the creation of point clouds using rgb.

RGB folder has the point clouds that are been created after the code execution of part 2

RGB folder has the point clouds that are been created after the code execution of part 1

After the code execution, you can find 2 folders have been created named rgb1 and rgbd1. These folders contain the sparse and dense point clouds of RGB images and RGB-D images respectively.

**Matlab:**

This code is to create point clouds using RGB images. The execution time of this code is far less than the colmap. We have taken the code from and made some changes.

**Cloud compare**: Cloud compare is the tool used for the representation of pointclouds in 3d and to identify the findings in the point clouds. Here it is used to compare both point clouds i.e. RGB point clouds and RGB-D point clouds and make quantitative and qualitative analysis between both the point clouds. The images in the folder () contains the images of results of the analysis of point clouds that we worked on in cloud compare.

**Localization algorithm:** localization using slam with RGB-D images. Inorder to do this. we need to calculate the pose estimation i.e. Rotation and traslation.

The approach we took to perform this is :

1. Feature extraction:

In the case of 2D images, feature extraction involves analyzing the image to identify certain patterns or features that can be used to distinguish it from other images. Common feature extraction techniques for 2D images include edge detection, corner detection, and blob detection. These techniques are used to identify salient points or regions in the image that can be used for further analysis.

1. Feature matching:

In the context of the given problem, there are 1000 features extracted from the localization image and 3000 features extracted from the map image. The goal is to find at most 1000 matched features that have corresponding positions in both images. The output of feature matching is a set of pairs of matched features, where each pair consists of a feature index from the localization image and a feature index from the map image.

1. Pose estimation:

In the previous approach, 2D-3D pose estimation was performed using the p3p RANSAC algorithm. This involved using the 2D position of a feature in the localization image, its corresponding 3D position in the map, and camera information to estimate the pose of the camera relative to the map. In the new approach, 3D-3D pose estimation is performed using a new algorithm that involves randomly selecting 4-6 matched features, performing pose estimation, and using the other matched features to calculate the average error. The output of the pose estimation step is the rotation and translation of the camera relative to the map, which is used to transform the 3D positions of the matched features from the localization image to the map. The pose with the lowest average error is returned as the final estimate.

We are working on feature matching and the draft code is named pose\_estimation.ipynb. The code is for matching the source and destination points and finding the pose estimation of it. i.e. rotation and translation of it.

**Code review:**

The function 'p3p\_3d\_3d' takes two sets of 3D points as input - 'points\_3d\_src' and 'points\_3d\_dst'. 'points\_3d\_src' is the set of 3D points in the camera's view, while 'points\_3d\_dst' is the corresponding set of 3D points in the world coordinate system.

The function first normalizes the input 3D points to have zero mean and unit variance using the 'mean' and 'std' functions of numpy. This normalization is done to improve the numerical stability and convergence of the algorithm.

Next, the function computes the similarity transform between the normalized 3D points. The similarity transform is a combination of a rotation matrix 'R' and a translation vector 't'. The function first calculates the Homography matrix 'H' by taking the dot product of the normalized source and destination 3D points. The Homography matrix is then decomposed using Singular Value Decomposition (SVD) to obtain the rotation matrix 'R'. Finally, the translation vector 't' is calculated by subtracting the product of the rotation matrix 'R' and the mean of the destination 3D points from the mean of the source 3D points.

The function then returns the estimated rotation matrix 'R' and translation vector 't'. These values can be used to calculate the camera pose in the world coordinate system.