Student: Li Hongguang (A0233309L)

Implementation: get_plane_sweep_homographies()

1. Retrieve relative rotation R, and other camera's center coordinates C using relative pose matrix, with formula given in lecture 11 slide 65
2. Defining plane norm as n = $(0, 0, 1)^T$, we compute the homography with formula in slide 68 for each inverse depth

Implementation: compute_plane_sweep_volume()

1. Iterate through all images to find reference image I_{ref} by comparing their pose matrix with the input reference camera pose
2. For each image I_i, compute the relative pose w.r.t I_{ref} using **concat_extrinsic_matrix()**, then compute the homography $H_{i,d}$ relating I_i and I_{ref} using **get_plane_sweep_homographies()** for the input range of depths d
3. For each depth, we warp the pixels on I_i to I_{ref} and determine the mask that corresponds to the valid boundary of the warped image, by finding pixels in the transformed image with non-zero value for all 3 channels
4. With the mask in step 3, compute the absolute difference between I_i and I_{ref} for the valid pixels averaged across RGB channels and collect the variance in ps_volume
5. With the mask in step 3, accumulate the number of images that are warped onto a certain pixel by adding 1 to the corresponding pixel where the mask is True

Implementation: compute_depths()

1. Using **np.argmin()** on ps_volume, we get the depth index that corresponds to the lowest variance
2. We apply the index matrix from step 1 to find the inverse depth image by applying it on the inv_depths array

Implementation: post_process()

1. Find the 10^{th} percentile among the accumulator count values using **np.percentile()**, and disregard depth estimates with observations less than or equals to this threshold
2. Compute the depth image after removing points with insufficient observations, for points with observations less than the threshold computed in step 1
3. Apply median filtering on the depth image from step 2 to remove salt and pepper noise
4. Compute the difference between the original and the filtered depth image, and retrieve the 95^{th} percentile of the change in depth values to be used as a threshold (for step 5)
5. As median filtering was used, it is highly likely for depth values to change. Thus, we consider depth value change that is less than the threshold value computed in step 4 to be a valid pixel

Implementation: unproject_depth_map()

1. Using **np.meshgrid()**, retrieve all pixel coordinates
2. Convert pixel coordinates to homogeneous coordinates
3. For homogeneous pixel coordinate $(u, w, 1)^T$ and depth value Z for the specific pixel, we compute unprojected point as $Z*K^{-1}(u,w,1)^T$ (Source: Piazza forum)
4. Compute unprojected point coordinates for all pixel values with step 3
5. If mask is present, we remove invalid points and their respective RGB values from the xyz and rgb array