1. Get_plane_sweep_homographies():
   a. Given the intrinsic matrix, and relative pose, retrieve the rotation, translation and K^{-1}.
```
R = (relative_pose[:, :-1]).reshape((3, 3))
C = relative_pose[:, -1].reshape((3, 1))
n_tm = np.array([0, 0, 1]).reshape((1, 3))
K_inv = np.linalg.inv(K)
```
   b. For each inverse depth value, calculate the homography relating the 2 planes, where homography = K @ (R + ((C @ n_tm) * inv_depth)) @ k^{-1}

2. Compute_plane_sweep_volume():
   a. Assuming image[4] as reference image, calculate relative pose of each image to the reference image.
   b. For each depth value,
      i. calculate the homography of each image w.r.t reference image and wrap each image to reference frame.
      ii. Count how many valid pixels has been mapped successfully and update accum_count and mask accordingly.
      iii. Calculate the variance by subtracting reference image by the warpped image and average the variance across the RGB channels.
      iv. Set any invalid pixels variance to 0 using the mask made in step (c).
      v. Update ps_volume with this variance.
3. Unproject_depth_map():
   a. Create a meshgrid according to height and width of image
   b. Unproject each x and y value in meshgrid.
      i. For x values, 3d pt x = (xx – c_x) / inv_depth_image / f_x.
      ii. For y values, 3d pt y = (yy – c_y) / inv_depth_image / f_y.
      iii. Where c_* and f_* is center and focal point retrieved from intrinsic matrix.
   c. Check for mask
      i. If have, filter the meshgrid values with the mask.
      ii. If not, pass.
   d. Using np.dstack(), stack the x, y and 1/inv_depth_image and reshape to (N, 3)
   e. Reshape image to (N, 3) for the rgb values
4. Post_process():
   a. Using compute_depths, calculate the inv_depth_image.
   b. Create a mask
      i. Where pixel is valid if is < mean(inv_depth_image) + 2.5 * std(inv_depth_image)
      ii. Filter the inv_depth_image using gaussian filter.
5. Results:

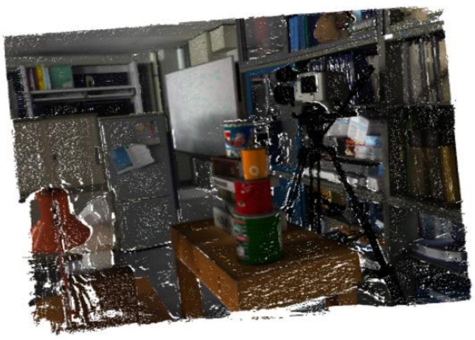

Figure 1 Without post_process

Figure 2 With post_process