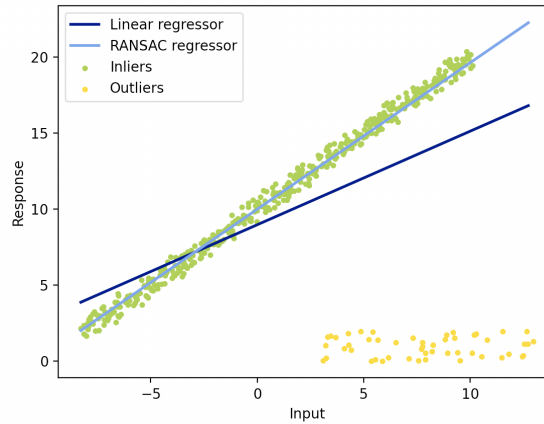


CV Assignment 3, RANSAC & MVS

Jingyu Liu

RANSAC

2.1.3. Write down the ground truth, estimation from least-squares and estimation from RANSAC:



	True	Least-Squares	RANSAC
k	1	0.6159656	0.96438949
b	10	8.9617271	9.98292129

MVS

3.2.2. Write down the equation of corresponding pixel $p_{i,j} := p_i(d_j)$, first we convert the 2d coordinate into 2d homogeneous coordinate $p_{homo} = (p(x), p(y), 1)$. Compute the relative transformation from the source projection and reference projection, which is given by the code. Then we decompose the homography into rotation and translation. The final equation is:

$$p_{i,j,unhomo} = R_i(p_{homo} * d_j) + t_i, p_{i,j} = homogenize(p_{i,j,unhomo})$$

3.3. Screenshot of training:



Due to the computational complexity and my hardware problem, I only trained for one epoch but it is also obvious that the loss is decreasing very fast and steadily. Each training iteration takes too long and training the whole 4 epochs on my machine will take days.

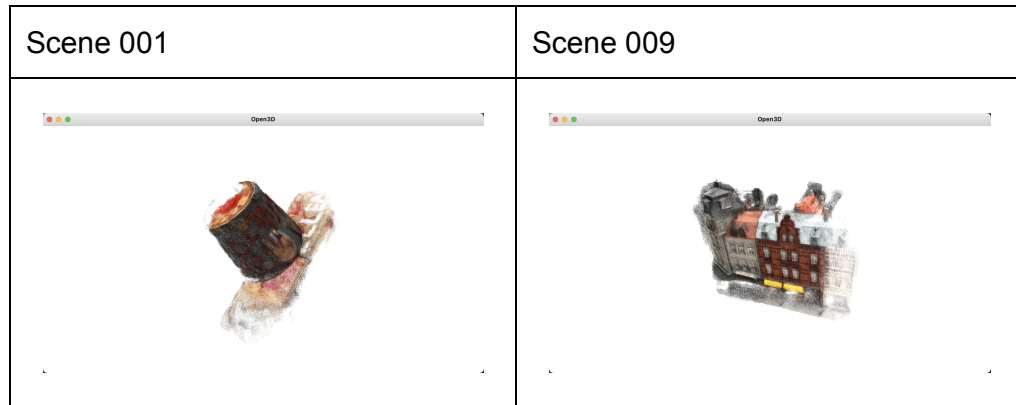
Training Log	Evaluation Log
<pre> Epoch 0/1, iter 0, loss = 1.000, time = 0.000 Epoch 0/1, iter 100, loss = 0.999, time = 0.001 Epoch 0/1, iter 200, loss = 0.998, time = 0.002 Epoch 0/1, iter 300, loss = 0.997, time = 0.003 Epoch 0/1, iter 400, loss = 0.996, time = 0.004 Epoch 0/1, iter 500, loss = 0.995, time = 0.005 Epoch 0/1, iter 600, loss = 0.994, time = 0.006 Epoch 0/1, iter 700, loss = 0.993, time = 0.007 Epoch 0/1, iter 800, loss = 0.992, time = 0.008 Epoch 0/1, iter 900, loss = 0.991, time = 0.009 Epoch 0/1, iter 1000, loss = 0.990, time = 0.010 Epoch 0/1, iter 1100, loss = 0.989, time = 0.011 Epoch 0/1, iter 1200, loss = 0.988, time = 0.012 Epoch 0/1, iter 1300, loss = 0.987, time = 0.013 Epoch 0/1, iter 1400, loss = 0.986, time = 0.014 Epoch 0/1, iter 1500, loss = 0.985, time = 0.015 Epoch 0/1, iter 1600, loss = 0.984, time = 0.016 Epoch 0/1, iter 1700, loss = 0.983, time = 0.017 Epoch 0/1, iter 1800, loss = 0.982, time = 0.018 Epoch 0/1, iter 1900, loss = 0.981, time = 0.019 Epoch 0/1, iter 2000, loss = 0.980, time = 0.020 Epoch 0/1, iter 2100, loss = 0.979, time = 0.021 Epoch 0/1, iter 2200, loss = 0.978, time = 0.022 Epoch 0/1, iter 2300, loss = 0.977, time = 0.023 Epoch 0/1, iter 2400, loss = 0.976, time = 0.024 Epoch 0/1, iter 2500, loss = 0.975, time = 0.025 Epoch 0/1, iter 2600, loss = 0.974, time = 0.026 Epoch 0/1, iter 2700, loss = 0.973, time = 0.027 Epoch 0/1, iter 2800, loss = 0.972, time = 0.028 Epoch 0/1, iter 2900, loss = 0.971, time = 0.029 Epoch 0/1, iter 3000, loss = 0.970, time = 0.030 Epoch 0/1, iter 3100, loss = 0.969, time = 0.031 Epoch 0/1, iter 3200, loss = 0.968, time = 0.032 Epoch 0/1, iter 3300, loss = 0.967, time = 0.033 Epoch 0/1, iter 3400, loss = 0.966, time = 0.034 Epoch 0/1, iter 3500, loss = 0.965, time = 0.035 Epoch 0/1, iter 3600, loss = 0.964, time = 0.036 Epoch 0/1, iter 3700, loss = 0.963, time = 0.037 Epoch 0/1, iter 3800, loss = 0.962, time = 0.038 Epoch 0/1, iter 3900, loss = 0.961, time = 0.039 Epoch 0/1, iter 4000, loss = 0.960, time = 0.040 Epoch 0/1, iter 4100, loss = 0.959, time = 0.041 Epoch 0/1, iter 4200, loss = 0.958, time = 0.042 Epoch 0/1, iter 4300, loss = 0.957, time = 0.043 Epoch 0/1, iter 4400, loss = 0.956, time = 0.044 Epoch 0/1, iter 4500, loss = 0.955, time = 0.045 Epoch 0/1, iter 4600, loss = 0.954, time = 0.046 Epoch 0/1, iter 4700, loss = 0.953, time = 0.047 Epoch 0/1, iter 4800, loss = 0.952, time = 0.048 Epoch 0/1, iter 4900, loss = 0.951, time = 0.049 Epoch 0/1, iter 5000, loss = 0.950, time = 0.050 </pre>	<pre> Epoch 0/1, iter 0, loss = 1.000, time = 0.000 Epoch 0/1, iter 100, loss = 0.999, time = 0.001 Epoch 0/1, iter 200, loss = 0.998, time = 0.002 Epoch 0/1, iter 300, loss = 0.997, time = 0.003 Epoch 0/1, iter 400, loss = 0.996, time = 0.004 Epoch 0/1, iter 500, loss = 0.995, time = 0.005 Epoch 0/1, iter 600, loss = 0.994, time = 0.006 Epoch 0/1, iter 700, loss = 0.993, time = 0.007 Epoch 0/1, iter 800, loss = 0.992, time = 0.008 Epoch 0/1, iter 900, loss = 0.991, time = 0.009 Epoch 0/1, iter 1000, loss = 0.990, time = 0.010 Epoch 0/1, iter 1100, loss = 0.989, time = 0.011 Epoch 0/1, iter 1200, loss = 0.988, time = 0.012 Epoch 0/1, iter 1300, loss = 0.987, time = 0.013 Epoch 0/1, iter 1400, loss = 0.986, time = 0.014 Epoch 0/1, iter 1500, loss = 0.985, time = 0.015 Epoch 0/1, iter 1600, loss = 0.984, time = 0.016 Epoch 0/1, iter 1700, loss = 0.983, time = 0.017 Epoch 0/1, iter 1800, loss = 0.982, time = 0.018 Epoch 0/1, iter 1900, loss = 0.981, time = 0.019 Epoch 0/1, iter 2000, loss = 0.980, time = 0.020 Epoch 0/1, iter 2100, loss = 0.979, time = 0.021 Epoch 0/1, iter 2200, loss = 0.978, time = 0.022 Epoch 0/1, iter 2300, loss = 0.977, time = 0.023 Epoch 0/1, iter 2400, loss = 0.976, time = 0.024 Epoch 0/1, iter 2500, loss = 0.975, time = 0.025 Epoch 0/1, iter 2600, loss = 0.974, time = 0.026 Epoch 0/1, iter 2700, loss = 0.973, time = 0.027 Epoch 0/1, iter 2800, loss = 0.972, time = 0.028 Epoch 0/1, iter 2900, loss = 0.971, time = 0.029 Epoch 0/1, iter 3000, loss = 0.970, time = 0.030 Epoch 0/1, iter 3100, loss = 0.969, time = 0.031 Epoch 0/1, iter 3200, loss = 0.968, time = 0.032 Epoch 0/1, iter 3300, loss = 0.967, time = 0.033 Epoch 0/1, iter 3400, loss = 0.966, time = 0.034 Epoch 0/1, iter 3500, loss = 0.965, time = 0.035 Epoch 0/1, iter 3600, loss = 0.964, time = 0.036 Epoch 0/1, iter 3700, loss = 0.963, time = 0.037 Epoch 0/1, iter 3800, loss = 0.962, time = 0.038 Epoch 0/1, iter 3900, loss = 0.961, time = 0.039 Epoch 0/1, iter 4000, loss = 0.960, time = 0.040 Epoch 0/1, iter 4100, loss = 0.959, time = 0.041 Epoch 0/1, iter 4200, loss = 0.958, time = 0.042 Epoch 0/1, iter 4300, loss = 0.957, time = 0.043 Epoch 0/1, iter 4400, loss = 0.956, time = 0.044 Epoch 0/1, iter 4500, loss = 0.955, time = 0.045 Epoch 0/1, iter 4600, loss = 0.954, time = 0.046 Epoch 0/1, iter 4700, loss = 0.953, time = 0.047 Epoch 0/1, iter 4800, loss = 0.952, time = 0.048 Epoch 0/1, iter 4900, loss = 0.951, time = 0.049 Epoch 0/1, iter 5000, loss = 0.950, time = 0.050 </pre>

3.4.

1. Explain what geometric consistency filtering is doing in the report.

When we're reconstructing dense 3d models, we want to make sure that the depth information is correct, which is ensured partially by the geometric consistency. Specifically, it first unprojects the images pixels from the reference image with the depth map, and then transforms it into the source image (just like warping) using the extrinsic and intrinsic matrices of both images. Then it uses the warped pixels to sample the depths of the source image, which then gets unprojected and transformed back to the original reference image coordinate. Finally, it checks two things to make sure that the depth information is correct: 1) it measures L2 differences between the "circular" transformed pixels and the reference pixels (like triangulation verification) and 2) calculates the relative difference between the depth information from the depth map and the depth values produced by the forward-reverse transformations. So that only those pixels with both errors below certain threshold will be used for constructing dense 3d models. At the end it also makes sure that a pixel is valid for further steps if there are at least three source images that can pass the geometric consistency test.

2. For all the scenes, visualize (visualize ply.py) and take screenshots of the point clouds in Open3D



3.5.

1. Taking uniform samples from the inverse range and then taking the inverse to get the depths might be helpful for large-scale scenes.

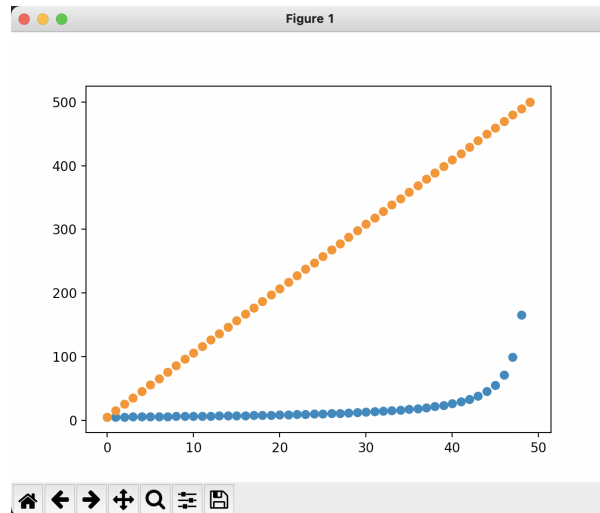


Figure 1: orange line is uniform samples in $[\text{depth_min}, \text{depth_max}]$, blue line is uniform samples in the inverse range and then taking the inverse of the samples to get depths. Horizontal axis denotes sample index, and horizontal axis means the depth sample values.

The idea is that in the stereo setting, we have better accuracies for nearby objects and coarse estimations for objects far away. Just like being illustrated in the class, the disparity becomes less informative in far away scenes (also due to numerical issues), hence in large-scale scenes, using the new sampling methods (inverse one) can help get denser samples close to the camera, which is also shown in the plot.

2. It might not be robust because we can treat the pairs of images with occlusions as outliers as the matching similarity will likely to be low. The averages will not necessarily be the best statistics for describing the quality of the true matching similarity for the population

(or batch in our case). Maybe using other statistics robust for outliers such as median or percentile would help.