CSE 152A (FA21, Su)

Final Project

Jiahong Long
[redacted]

12 December 2021

Contents

1 Approach **1**
1.1 Point cloud generation 1
1.2 Basic data cleaning 1
1.3 Higher-order data cleaning 1

2 Results **1**
2.1 Hyperparameter tuning 1
2.2 Presentation 2

Abstract. We aimed to find, then tune, a methodology for cleaning a point cloud generated from $n = 48$ images of a physical object. We discovered that by analysing the distribution of the k nearest neighbours for each point in the point cloud, we could remove outliers based on the mean and standard deviation of the k-NN distribution per-point. The results are qualitatively satisfactory.

1 Approach

We outline the computer vision methodology used to generate the point cloud, followed by the multistage process of cleaning the data generated by the computer vision tooling.

1.1 Point cloud generation

Each image was loaded into a Python environment using `OpenCV`. The images were passed through the `cv2` implementation of the SIFT corner detection algorithm, and the output descriptors were matched using the `cv2` implementation of the brute-force descriptor matcher with basic filtering done with a threshold ratio between the first and second-best matches. From these correspondences, the `cv2` triangulation logic was used to reconstruct the points in 3D. Following this, the correspondences were filtered by calculating the fundamental matrix between every pair of two adjacent views, and removing those correspondences (p_1, p_2) whose image under the fundamental matrix transformation were not within 1% of $\frac{\text{image height}+\text{image width}}{2}$ of p_2.

1.2 Basic data cleaning

To improve the initial visualisations, we removed points whose x-coordinates were not within some IQR multiple of the lower and upper quartiles of all x-coordinates. The same was then done for the y- and z-coordinates. This step filters out points which are egregiously incorrect.

1.3 Higher-order data cleaning

For each point, the point distribution of the k-nearest neighbours was found. In particular, we constructed per-point distributions of the distances to its k-nearest neighbours and found μ (mean) and σ (standard deviation) of these distributions. We then filtered out those points whose μ did not satisfy $\mu < t_\mu$, where t_μ is some mean threshold value chosen to reduce the number of points located in sparse areas. Finally, we filtered out points whose σ did not satisfy $\sigma < t_\sigma$, where t_σ is some standard deviation threshold value chosen to reduce the number of points located in areas of uneven point distribution as a heuristic to remove points whose neighbours are unevenly distributed.

2 Results

We now present the motivation for the hyperparameter values, followed by qualitative result analysis.

2.1 Hyperparameter tuning

There were *four* hyperparameters to tune: the first was the basic outlier filter, and the remaining three relate to constructing and then using k-nearest neighbours fo filter the remaining points. In particular, the remaining three are k, the number of neighbours to consider, t_μ, the threshold for mean neighbour distance, and t_σ, the threshold for the standard deviation of neighbour distances.

To remove outliers, we removed those points whose x-coordinate was not within 2 IQR of the lower and upper quartiles of the global distribution of x-coordinates. The same was then done for the y- and z-coordinates. We used 2 instead of

the conventional 1.5 because the point distribution within the point cloud is heavily uneven. Experimentally, a factor of 1.5 removes more points than is required at this step.

In tuning the k-nearest neighbours filters, we chose $k = 10$ by subjective heuristics: it was experimentally found that greater values resulted in globally inflated mean-neighbour-distances, and lower lower resulted in a failure to capture enough of the local point distribution about each point. In particular, we eliminated $k \leq 5$ and $k \geq 15$ for these reasons: we then chose $k = 10 = (5 + 15)/2$.

We then chose the mean filter threshold to be $t_\mu = 4$. We analysed the resultant point cloud with low values of t_μ, and found that values less than 2 resulted in high areas of sparseness where points are less densely clustered. Conversely, values greater than 8 cause large amounts of noise on surfaces to be retained; a final value of 4 was determined by qualitatively examining the values in between these two extremes.

The standard deviation filter threshold was chosen to be $t_\sigma = 1.5$ using exploratory data analysis (Figure 1). A histogram of the standard deviations of the per-point nearest neighbours' distances illustrates that much of the long right tail can be eliminated without loss of resolution by picking a threshold greater than 1.0. 1.5 was chosen subjectively by iteratively checking thresholds between 1.0 and 5; we selected 1.5 as it was effective in removing surface imperfections without removing too much other detail.

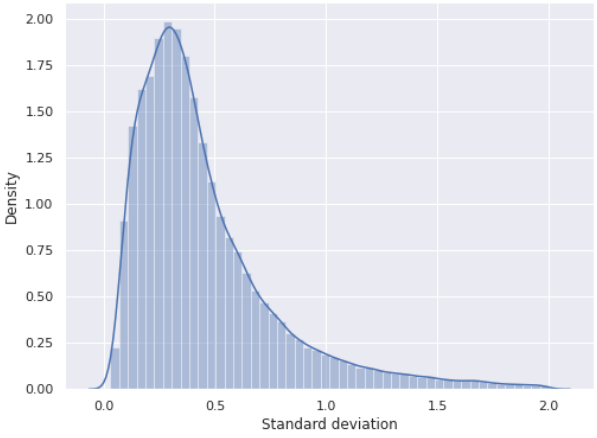

Figure 1: Distribution of standard deviations of neighbour distances per-point.

Combining all of these criteria, we arrive at the following filters:

```
# The list of reconstructed points is all_pts.
# Removal of outliers using IQR:
outlier_thresh = 2
x_hq, x_lq = np.percentile(all_pts[0, :], [75, 25])
y_hq, y_lq = np.percentile(all_pts[1, :], [75, 25])
z_hq, z_lq = np.percentile(all_pts[2, :], [75, 25])
x_bound = (x_hq - x_lq) * outlier_thresh
y_bound = (y_hq - y_lq) * outlier_thresh
z_bound = (z_hq - z_lq) * outlier_thresh
within_x = (all_pts[0, :] >= (x_lq - x_bound)) &
    (all_pts[0, :] <= (x_hq + x_bound))
within_y = (all_pts[1, :] >= (y_lq - y_bound)) &
    (all_pts[1, :] <= (y_hq + y_bound))
within_z = (all_pts[2, :] >= (z_lq - z_bound)) &
    (all_pts[2, :] <= (z_hq + z_bound))
# IQR filter:
in_bounds = within_x & within_y & within_z
# Generate k-NN tree and associated distributions:
tree = KDTree(all_pts.T)
neighbors_to_check = 10
knn_d = tree.query(all_pts.T, k=neighbors_to_check,
    return_distance=True)[0]
stds = np.std(knn_d, axis=1)
# Mean neighbour distance filter:
mean_thresh = 4
mean_condition = np.mean(knn_d[:, 1:], axis=1) <
    mean_thresh
# Neighbour distance distribution filter:
std_thresh = 1.5
std_condition = stds < std_thresh
# Final set of points in the point cloud:
filtered_points = all_points[
  std_condition & mean_condition & in_bounds
]
```

2.2 Presentation

After basic data cleaning (removal of outliers), the point cloud generated is provided in Figure 2. Note the large number of floating points that do not appear to be part of the rigid object. Usage of k-nearest neighbours to remove those points whose mean distance to its neighbours is less than a threshold was efficacious in removing these points (Figure 3). The final filter, removing points whose 10-nearest neighbours' distances had a standard deviation of more than 1.5, serves to further refine the point cloud by decreasing the bumpiness of surfaces (Figure 4). Finally, note that the points removed using the standard deviation filter do not have as much texture noise on the reconstructed brick surface (Figure 5). Thus, the qualitative appearance of the point cloud after these filters is satisfactory as it eliminates clear outliers and smooths out surface imperfections. However, we also note that there are other ways of optimising the point cloud, depending on what heuristics we choose to optimise for. In particular, during hyperparameter tuning, we discovered that certain values for the k-nearest neighbours filters is highly efficacious in resolving surface-level texture (and was able to reveal the text written on the bricks), at the cost of leaving the remaining point cloud highly sparse.

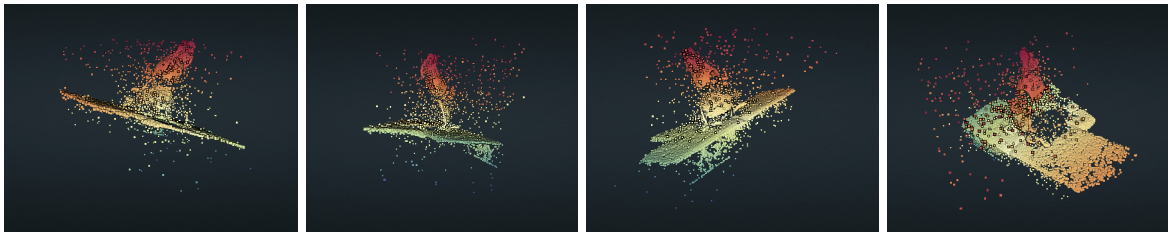

Figure 2: Point cloud without outliers.

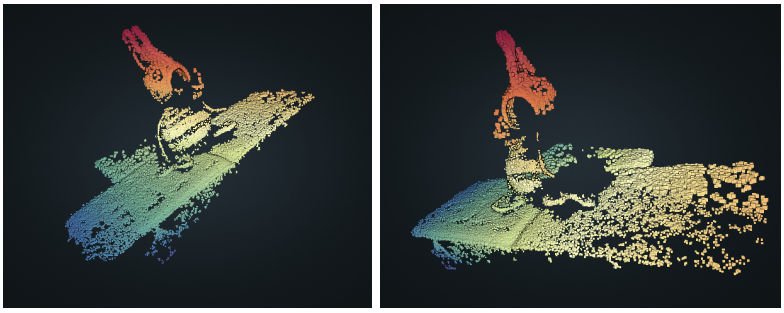

Figure 3: Point cloud without outliers, using k-NN mean filtering.

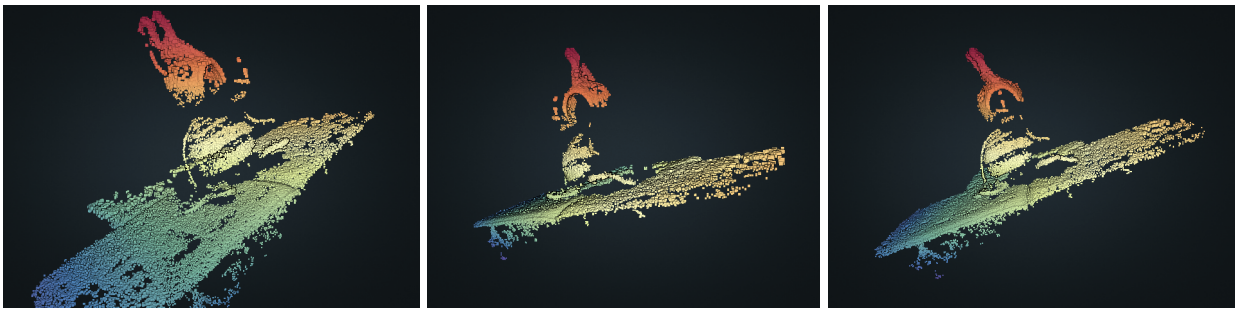

Figure 4: Point cloud without outliers, using k-NN mean and standard deviation filtering.

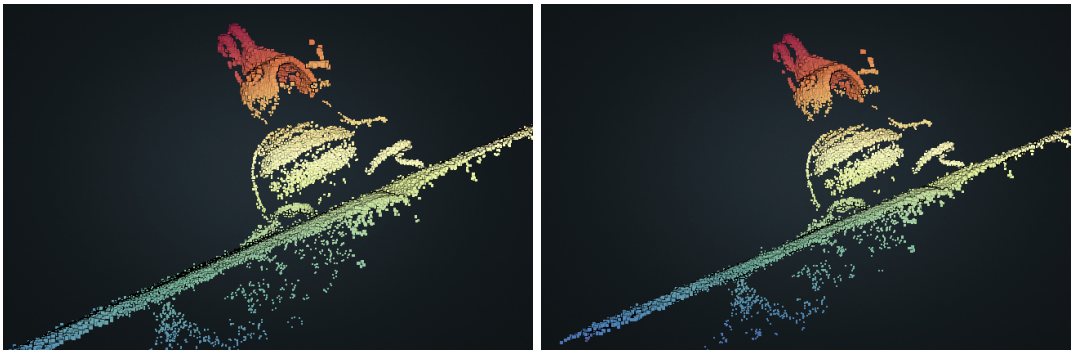

Figure 5: Left: without σ filter. Right: with σ filter.