A K-Nearest Neighbours Based Inverse Sensor Model for Occupancy Mapping

Yu Miao (✉), Ioannis Georgilas, and Alan Hunter

University of Bath, Claverton Down, Bath BA2 7AY, United Kingdom
y.miao@bath.ac.uk

Abstract. OctoMap is a popular 3D mapping framework which can model the data consistently and keep the 3D models compact with the octree. However, the occupancy map derived by OctoMap can be incorrect when the input point clouds are with noisy measurements. Point cloud filters can reduce the noisy data, but it is unreasonable to apply filters in a sparse point cloud. In this paper, we present a k-nearest neighbours (k-NN) based inverse sensor model for occupancy mapping. This method represents the occupancy information of one point with the average distance from the point to its k-NN in the point cloud. The average distances derived by all the points and their corresponding k-NN are assumed to be normally distributed. Our inverse sensor model is presented based on this normal distribution. The proposed approach is able to deal with sparse and noisy point clouds. We implement the model in the OctoMap to carry out experiments in the real environment. The experimental results show that the 3D occupancy map generated by our approach is more reliable than that generated by the inverse sensor model in OctoMap.

Keywords: K-nearest neighbours · Inverse sensor model · Occupancy mapping.

1 Introduction

Autonomous vehicles are widely used in military, agricultural, industrial and commercial areas. Robotic mapping as an essential part of the autonomous navigation system is often required in these application scenarios. The process of building a map of the environment while locating a robot itself is referred to as simultaneous localization and mapping (SLAM) [1].

In robotics, point clouds can be obtained by low-cost sensors and are effective in representing the environment [2]. Point clouds usually consist of a great number of points which contain the location data of the environment. The point cloud is a popular data type since it is easily accessible to researchers. Depth sensors such as the LIDAR, stereo cameras and RGB-D cameras can be used to produce point clouds. Point clouds are used for a wide range of research and commercial applications, including mapping, navigation and autonomous driving. With 3D point clouds, a mapping approach for household environments is

proposed in [3]. Some SLAM systems are also based on point clouds. In [4], a 3D SLAM system for outdoor scenes is presented using laser range data.

The Point Cloud Library (PCL) has been developed for processing point clouds [5]. Many functions such as filtering, registration and segmentation are included in this library. Recently, a popular occupancy mapping approach to map the environment with point clouds is OctoMap [6]. It is an efficient framework for 3D occupancy mapping. However, the map generated by OctoMap can be inaccurate when the point clouds are noisy. Although PCL provides several point cloud filters to remove noisy data, it is unreasonable to apply filters in a sparse point cloud with only a few points.

To solve these problems, we propose a k-nearest neighbours (k-NN) based inverse sensor model for occupancy mapping. We use the average distance from a point to its k-NN to represent the occupancy information of the point. This distance is assumed to be follow the normal distribution. Two neighbouring point clouds are compared to obtain a dynamic normal distribution (Section 3.2). With this distribution, a k-NN based inverse sensor model is presented to update the occupancy map (Section 3.3). The proposed model is implemented in the OctoMap to illustrate its effectiveness. We test it in a semi-structured environment by generating a 3D occupancy map with the point clouds derived by ORB-SLAM [7] (Section 4). The experimental results show that the map generated by our method is more accurate than that derived by the model in OctoMap.

2 Related Work

The processing and mapping methods for point clouds have been investigated in many studies.

The octree is an tree-based data structure in which each node is a cube and has eight child nodes [8]. Due to this hierarchical structure, an octree can quickly expand a large number of nodes by recursively subdividing a node into eight children. Based on the octree structure, PCL is able to index 3D point cloud data efficiently and provide processing algorithms, including point cloud compression, spatial partitioning, neighbour search and spatial change detection [9–11].

OctoMap is a 3D occupancy mapping method, which is based on octrees and probabilistic estimation [6]. OctoMap models the environment by volumetric representation and can keep models compact by implementing octrees. With a clamping update policy to limit the occupancy estimates, OctoMap can rapidly adapt to environment changes.

ORB-SLAM is considered to be the most complete feature-based monocular visual SLAM system and has been extended to stereo visual SLAM [7, 12]. It uses ORB features in the mapping process. ORB-SLAM contains three threads, tracking, local mapping and loop closing. They are parallel threads, so ORB-SLAM can run efficiently on a CPU without using a GPU. Bundle adjustment (BA) is performed in both local mapping and loop closing to reduce the projection error of the vision sensor. The map will be updated when the loop is detected.

We implement ORB-SLAM to get point clouds. In ORB-SLAM, keyframes are selected frames to map the environment. Each keyframe contains the camera pose, camera intrinsics and ORB features extracted from the image. Each ORB feature point corresponds to a point on the objects in the environment, so we can get a point cloud by accumulating the feature points in each keyframe. In this paper, the point cloud obtained by ORB features in the keyframe is referred to as the keyframe point cloud.

3 Methodology

The point cloud which contains the points from all the observations is referred to as the map point cloud. In the map point cloud, the average distances from all the points to their corresponding k-NN are assumed to follow the normal distribution. We use these average distances to represent the occupancy information of the points. Our mapping approach first updates the normal distribution and then updates the occupancy map with the k-NN based inverse sensor model. We implement our inverse sensor model in the OctoMap to generate a 3D occupancy map. This section will explain our method in detail.

3.1 Background of OctoMap

OctoMap is an efficient probabilistic framework for 3D occupancy mapping. It has been explained in detail in [6]. The probability $P(n|z_{1:t})$ of a node is modelled by Bayes rule and can be written as

$$P(n|z_{1:t}) = \left[1 + \frac{1 - P(n|z_t)}{P(n|z_t)} \frac{1 - P(n|z_{1:t-1})}{P(n|z_{1:t-1})} \frac{P(n)}{1 - P(n)}\right]^{-1}, \tag{1}$$

where n is the node, $z_{1:t}$ denotes the set of all sensor measurements acquired from time 1 to time t, $P(n|z_t)$ is the occupied probability given the measurement z_t and $P(n)$ is the prior probability.

In OctoMap, the prior probability $P(n) = 0.5$ and (1) can be simplified by the log-odds notation

$$L(n) = ln\left[\frac{P(n)}{1 - P(n)}\right]. \tag{2}$$

Then the probability of a node given measurements to time t is

$$L(n|z_{1:t}) = L(n|z_{1:t-1}) + L(n|z_t), \tag{3}$$

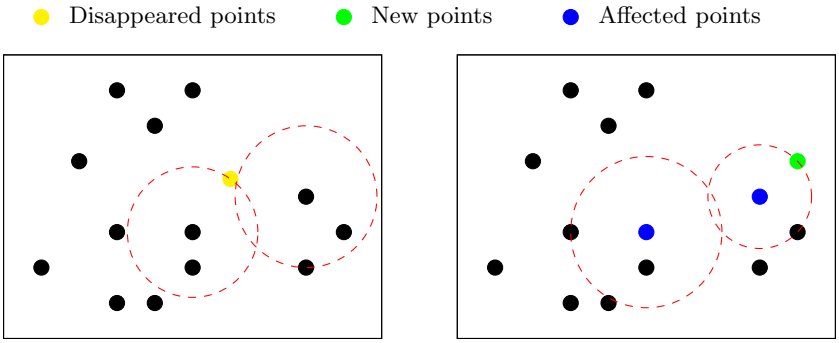
where $L(n|z_t)$ is the inverse sensor model. For any kind of ranging senor, a ray-cast operation will be performed from the sensor origin to the end points. The inverse sensor model in OctoMap is defined as

$$L(n|z_t) = \begin{cases} l_{occ} & \text{if the endpoint is in the node } n \\ l_{free} & \text{if a ray traverses the node } n \end{cases}, \tag{4}$$

where l_{occ} and l_{free} are the probabilities set to the nodes. A clamping update policy is applied in OctoMap to set lower and upper bounds on the log-odds value.

3.2 Update Normal Distribution

Description of Point Cloud Normally, the map point cloud can be different at different times. We use new points, disappeared points and affected points to describe the changes in the point cloud. As shown in Fig. 1, the points in the point cloud at time t but not in the point cloud at time $t-1$ are called new points. While the points in the point cloud at time $t-1$ but not in the point cloud at time t are called disappeared points. Due to the changes in the point cloud, some points may have different k-NN or average distances at time $t-1$ and t. These points are called affected points. In Fig. 1, the points selected by the dashed circle are the k-NN of the point in the center.



(a) Point cloud at time $t-1$ (b) Point cloud at time t

Fig. 1. Different points in the point cloud

To detect the changes in the map point cloud, the spatial change detection in the PCL is implemented. The points in the point cloud are indexed by the octree structure in PCL. By comparing the octree structures at time $t-1$ and time t, we can get new points and disappeared points. The detection of affected points will be discussed later.

Normal Distribution To describe how informative one point is, we assume that the average distance from any point to its k-NN is subject to a normal distribution

$$f(s) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(- \frac{(s-\mu)^2}{2\sigma^2} \right), \tag{5}$$

where s is the average distance from the point to its k-NN, μ is the mean of average distances derived by all the points and σ is the standard deviation.

To mitigate the impact from noisy or wrong measurements, a limit, s_{limit}, for the average distance is set to remove unreasonable data. The remaining points, called valid points, will be used to calculate the mean and the standard

deviation. When the map point cloud changes, the most common way to update the normal distribution is to calculate the mean with all the average distances and then sum the squares of all the deviations from the mean. To avoid this brute force calculation, the method for calculating corrected sums of squares is implemented [13]. This method does not need to store prior data and can reduce rounding errors in the computer implementation [14]. A series of means and standard deviations will be generated when the number of input values grows. Based on the description of the point cloud and the method in [13], we will illustrate how to update the normal distribution.

Add Point to Normal Distribution At time t, if a point p is a new point or an affected point, we will compare the average distance $s_{p,t}$ of this point with the limit s_{limit}. If $s_{p,t} < s_{limit}$, the following process will be performed

$$N \leftarrow N + 1, \tag{6}$$

$$S \leftarrow S + \frac{N-1}{N}(s_{p,t} - \mu)^2, \tag{7}$$

$$\mu \leftarrow \frac{N-1}{N}\mu + \frac{1}{N}s_{p,t}, \tag{8}$$

where N is the number of valid points and S is the sum of the squares of the deviations of valid points. The sum of squares and the mean are updated after this process. The average distance of the point will be included in the normal distribution.

Erase Point from Normal Distribution This process is the inverse process of adding a point to the normal distribution. At time t, if a point p is a disappeared point or an affected point, we will compare the average distance $s_{p,t-1}$ of this point with the limit s_{limit}. If $s_{p,t-1} < s_{limit}$, the following process will be performed

$$\mu \leftarrow \frac{N}{N-1}\mu - \frac{1}{N-1}s_{p,t-1}, \tag{9}$$

$$S \leftarrow S - \frac{N-1}{N}(s_{p,t-1} - \mu)^2, \tag{10}$$

$$N \leftarrow N - 1, \tag{11}$$

After this process, the sum of squares and the mean are updated. The average distance of this point will be removed from the normal distribution.

Update Standard Deviation At time t, after all the new points, disappeared points and affected points have been processed, we are able to update the standard deviation. The standard deviation is

$$\sigma = \sqrt{\frac{S}{N}}. \tag{12}$$

Update Affected Points This process will search the affected points and update the normal distribution with affected points. Algorithm 1 illustrates the process of searching affected points. Let C_{new}, $C_{disappeared}$ and $C_{affected}$ denote the collections of new points, disappeared points and affected points, respectively. s_{max} is the maximum limit for the affected points searching process. The searching process increases the radius of the circle centred on any point $p \in (C_{new} \bigcup C_{diappeared})$ by step Δr. Lines 10 through 11 exclude new points and affected points in the annulus with inner radius $r - \Delta r$ and outer radius r in the map point cloud at time t. Line 12 updates the average distance from each remaining point to its k-NN. If the distance are different at time $t-1$ and t, the searching point will be marked as an affected point. Lines 16 and 17 update the normal distribution with affected points. The searching process will stop when the average distance of each searching point in the annulus remains same at time $t-1$ and t, or $r \geq s_{max}$.

Algorithm 1: Update Affected Points

Input: p, Δr, s_{max}, C_{new}, $\bigcup s_{p,t-1}$, $MapPointCloud_t$, s_{limit}, N, S, μ, σ
Output: N, S, μ, σ

1 $r \leftarrow \Delta r$
2 $C_{previous} \leftarrow \emptyset$
3 $C_{affected} \leftarrow \emptyset$
4 **do**
5 $\quad flagIncreaseRadius \leftarrow false$
6 $\quad C_{temp} \leftarrow SearchPoints(MapPointCloud_t, p, r)$
7 $\quad$ **if** $(C_{temp} - C_{previous} - C_{temp} \bigcap C_{new}) = \emptyset$ **then**
8 $\quad\quad flagIncreaseRadius \leftarrow true$
9 $\quad$ **else**
10 $\quad\quad$ **for** $q \in (C_{temp} - C_{previous} - C_{temp} \bigcap C_{new})$ **do**
11 $\quad\quad\quad$ **if** $q \notin C_{affected}$ **then**
12 $\quad\quad\quad\quad s_{q,t} \leftarrow UpdateAverageDistance(MapPointCloud_t, q)$
13 $\quad\quad\quad\quad$ **if** $s_{q,t} \neq s_{q,t-1}$ **then**
14 $\quad\quad\quad\quad\quad flagSameKNN \leftarrow false$
15 $\quad\quad\quad\quad\quad C_{affected}.insert(q)$
16 $\quad\quad\quad\quad\quad ErasePointFromNormalDistribution(s_{q,t-1})$
17 $\quad\quad\quad\quad\quad AddPointFromNormalDistribution(s_{q,t})$
18 $\quad\quad\quad\quad\quad$ **if** $flagIncreaseRadius = false$ **and** $flagSameKNN = false$ **then**
19 $\quad\quad\quad\quad\quad\quad flagIncreaseRadius \leftarrow true$
20 $\quad$ **if** *flagIncreaseRadius=true* **then**
21 $\quad\quad r \leftarrow r + \Delta r$
22 $\quad\quad C_{previous} \leftarrow C_{temp}$
23 **while** $flagIncreaseRadius = true$ **and** $r < s_{max}$

Update Normal Distribution Based on the process of updating affected points, we can update the normal distribution completely. Algorithm 2 presents the complete process of updating the normal distribution. Lines 1 through 4 update the normal distribution with new points and affected points caused by new points. Lines 5 to 7 update the normal distribution with disappeared points and affected points caused by disappeared points. Line 8 calculates the standard deviation.

Algorithm 2: Update Normal Distribution

Input: C_{new}, $C_{disappeared}$, $\bigcup s_{p,t-1}$, $MapPointCloud_t$, s_{limit}, N, S, μ, σ
Output: N, S, μ, σ

1 **for** $p \in C_{new}$ **do**
2 $\quad s_{p,t} \leftarrow UpdateAverageDistance(MapPointCloud_t, p)$
3 $\quad AddPointToNormalDistribution(s_{p,t})$
4 $\quad UpdateAffectedPoints(p)$
5 **for** $p \in C_{disappeared}$ **do**
6 $\quad ErasePointFromNormalDistribution(s_{p,t-1})$
7 $\quad UpdateAffectedPoints(p)$
8 $\sigma \leftarrow \sqrt{S/N}$

3.3 Mapping with k-NN based Inverse Sensor Model

k-NN Based Inverse Sensor Model With (5) that the average distance is subject to a normal distribution, we can define a probability function to represent the occupancy information of a point in the point cloud

$$Prob(s_{p,t}) = P_u - \frac{\int_{-\infty}^{s_{p,t}} f(s)ds}{\int_{-\infty}^{w\sigma+\mu} f(s)ds}(P_u - P_w), \tag{13}$$

where w is a scale factor, P_u is the upper limit of the probability and P_w is the corresponding probability when $s_{p,t} = w$. w, P_u and P_w are changeable parameters. The probability will be smaller than P_u since the lower limit of the integral is negative infinity. A point is more relevant to points nearby when it has a higher probability. The noisy data and wrong measurements usually have lower probabilities. In this way, noisy measurements are given lower weights and thus can be filtered out to some extent. We use a similar method as the free nodes update strategy in OctoMap to update free nodes. However, before a ray is cast from the sensor position to the endpoint p, the endpoint must satisfy

$$ln\left[\frac{Prob(s_{p,t})}{1 - Prob(s_{p,t})}\right] > ln\left[\frac{P_w}{1 - P_w}\right]. \tag{14}$$

Then the inverse sensor model can be defined as

$$L(n|z_t) = \begin{cases} \sum_p ln\left[\frac{Prob(s_{p,t})}{1-Prob(s_{p,t})}\right] & \text{if the end point } p \text{ is in the node } n \\ l_{free} & \text{if a ray traverses the node } n \end{cases}, \quad (15)$$

where $p \in C_{view}$ and p is in node n. C_{view} is the measurement z_t, the collection of points in the sensor field of view (FOV) at time t. This model considers both the occupancy probability of each point and the number of points in a node. $L(n|z_t)$ tends to be bigger when the points in the node are with higher probabilities and the node contains more points. This also accords with the common sense.

Update Occupancy Map Algorithm 3 shows the mapping process with the proposed inverse sensor model. x_{sensor} denotes the position of the vision sensor. Lines 3 through 7 update the occupied nodes and prepare for free nodes updates. Line 8 updates the free nodes. Now the map has been updated with the latest observation z_t.

Algorithm 3: Mapping with k-NN Based Inverse Sensor Model

Input: C_{view}, x_{sensor}, $\bigcup L(n|z_{1:t-1})$, $\bigcup s_{p,t}$, w, P_u, P_w
Output: $\bigcup L(n|z_{1:t})$

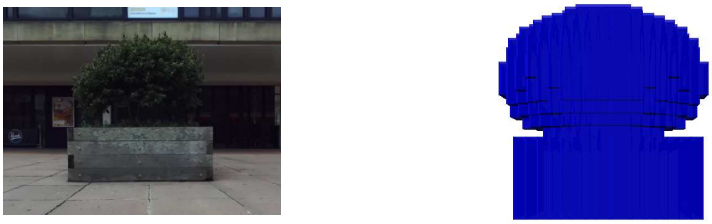
1 $L(n|z_{1:t}) \leftarrow L(n|z_{1:t-1})$
2 $C_{end} \leftarrow \emptyset$
3 **for** $p \in C_{view}$ **do**
4 $\quad n \leftarrow SearchNode(p)$
5 $\quad L(n|z_{1:t}) \leftarrow L(n|z_{1:t}) + ln\big[Prob(s_{p,t})/(1-Prob(s_{p,t}))\big]$
6 $\quad$ **if** $ln\big[Prob(s_{p,t})/(1-Prob(s_{p,t}))\big] > ln\big[P_w/(1-P_w))\big]$ **then**
7 $\quad\quad C_{end}.insert(p)$
8 $UpdateFreeNodes(x_{sensor}, C_{end})$

4 Experimental Results

4.1 Experiment Setup

We run ORB-SLAM with a ZED stereo camera to capture the feature points on a tree (See Fig. 2 (a)). To collect enough points to shape the tree, we move the camera around the tree. The trajectory of the camera is approximately a circle of radius 3.5 meters. In the end, we use a set of keyframes derived after loop correction to extract a series of keyframe point clouds and poses. With the pose information, the map point clouds can be recovered by accumulating keyframe point clouds. Then the map point clouds are used to update the normal distribution in section 3.2. Each keyframe point cloud can be regarded as

an measurement z_t. So the keyframe point cloud can be used to update the occupancy map with the k-NN based inverse sensor model in section 3.3. When the objects are far away from the camera, the depth measurements can be unreliable. To remove those unreasonable but avoid impacts on the occupancy map, we keep those points whose distances to the camera are within 4 meters.



(a) Tree (b) Reference

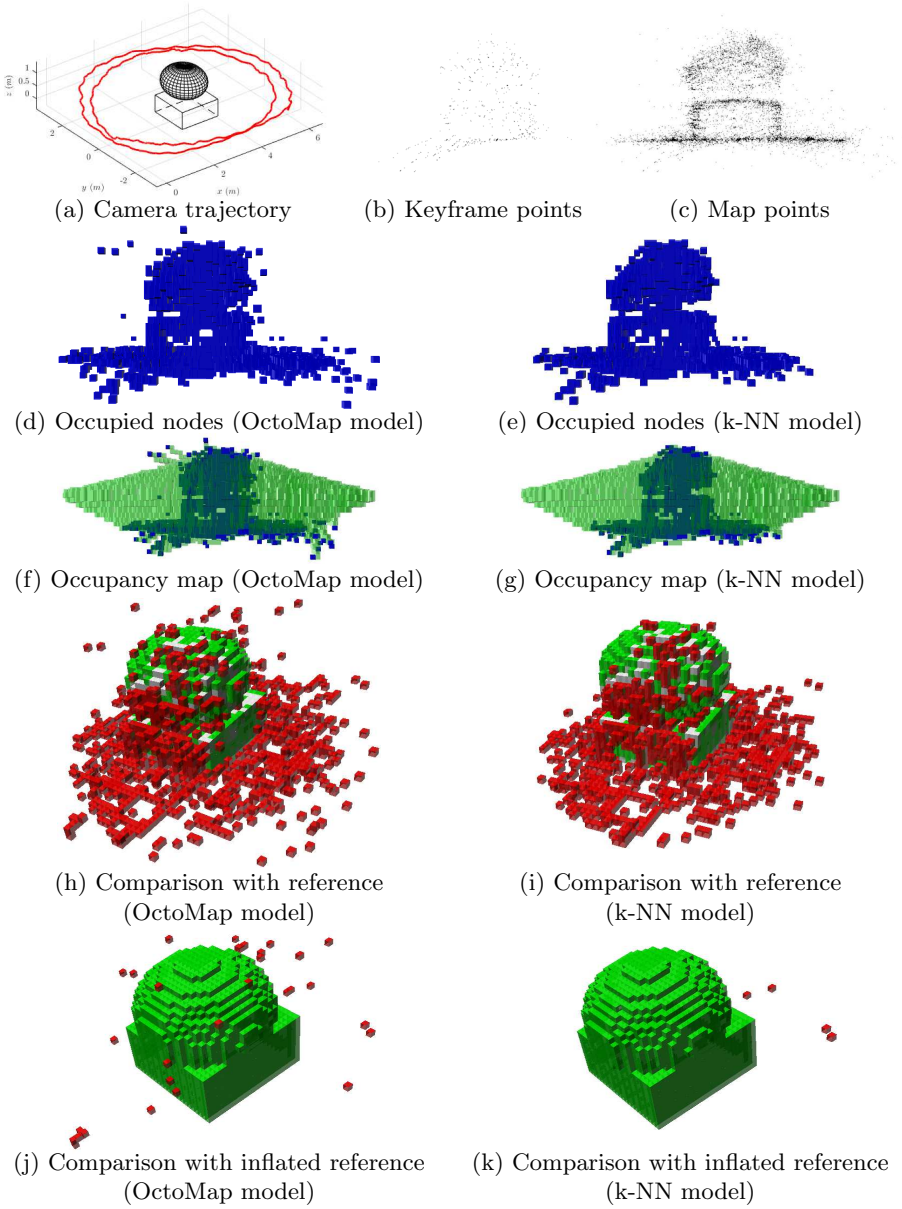
Fig. 2. Tree and reference

The ground truth is described by a reference map in which the foliage and the pot are represented by an ellipsoid and an cuboid (See Fig. 2 (b)). In this paper, the resolution of the occupancy map is set to 0.1 meter. Since the trunks in Fig. 2 (a) are partly blocked by the foliage and the grass in the pot, we ignore the trunks in the reference map. The grass is treated as a part of the pot, so the height of the pot in the reference map is greater than the real height. Although the reference is different from the ground truth, we can compare different inverse sensor models by comparing their relative mapping results to the reference.

4.2 Experiment Results

Fig. 3 (a) shows the camera trajectory derived by ORB-SLAM. Fig. 3 (b) is the last keyframe point cloud and Fig. 3 (c) is the corresponding map point cloud. Fig. 3 (d) and (e) present the occupied nodes derived by the OctoMap inverse sensor model and the proposed k-NN model, respectively. Fig. 3 (f) and (g) show the occupancy maps with both free and occupied nodes. The mapping results of the inverse sensor model in OctoMap and the proposed model are compared with the reference map. The occupied nodes in an occupancy map is a shell since the camera cannot observe the inside of an object. Fig. 3 (h) and (i) are the comparison results of the mapping results and the reference. A number of nodes are attached to the surface of the shell in the reference map and a minor translation occurs between two maps. This is because the measurements from the stereo camera are not accurate enough. To find those nodes that are apparently irrelevant to the map, we inflate the reference above ground by 0.3 meter. The ground part and the nodes whose vertical distances to the ground are within 0.3 meter are also ignored. The nodes outside the inflated reference

map are presented in Fig. 3 (j) and (k). These nodes are apparently irrelevant to the ground truth.



(a) Camera trajectory (b) Keyframe points (c) Map points

(d) Occupied nodes (OctoMap model) (e) Occupied nodes (k-NN model)

(f) Occupancy map (OctoMap model) (g) Occupancy map (k-NN model)

(h) Comparison with reference (OctoMap model) (i) Comparison with reference (k-NN model)

(j) Comparison with inflated reference (OctoMap model) (k) Comparison with inflated reference (k-NN model)

Fig. 3. Experimental results

Table 1 shows the comparison results of the reference map and the inflated reference map. The total nodes number of the reference shell is 1441. The mapping results of the OctoMap model and the k-NN model share 399 and 387 nodes, respectively. 756 occupied nodes in the map derived by the OctoMap model are either on the reference shell or inside the shell. While the reference shell can cover 703 occupied nodes in the map generated by the k-NN model. The numbers of irrelevant nodes for two maps are 33 and 4, respectively.

Table 1. Comparison with reference

Model	Nodes in reference shell	Nodes covered by reference	Irrelevant Nodes
OctoMap model	399	756	33
K-NN model	387	703	4

4.3 Discussion

Sometimes the spatial change detection in PCL cannot detect all the new points and disappeared points. It is possible that the average distance $s_{p,t-1}$ of an affected point and the average distance $s_{p,t}$ of a point in the keyframe point cloud are unknown. In these cases, we process such points as new points.

The experiment in the semi-structured environment shows that our approach can deal with sparse and noisy point clouds. The occupied nodes derived by the k-NN based model are more accurate and with less irrelevant nodes. Due to the noise and wrong measurements in the point cloud, the OctoMap model can branch free nodes even when the measurements are highly irrelevant. While the k-NN based model only generate free nodes with reliable measurements.

Since the point clouds are sparse, and camera measurements and the reference map are not accurate enough, the number of nodes shared by the occupancy map and the reference shell is smaller than the number of the nodes which form the reference shell. In table 1, the numbers of the k-NN model are slightly smaller than the numbers of the OctoMap model. Because the k-NN based method can filter unreliable points, and the reference map cannot overlap the ground truth completely so that wrong matches may occur between the map derived by the OctoMap model and the reference map.

In this paper, we ignore the influence of the pruning and clamping update policy in the OctoMap when counting the number of nodes. Because it is rare to see pruning and clamping when the input point clouds are sparse.

5 Conclusion

In this paper, we present a k-NN based inverse sensor model for occupancy mapping with an illustration of the update process of the normal distribution and the inverse sensor model. Our approach is reliable even when the input

point cloud is noisy and sparse. The mapping result of the k-NN based method is with less irrelevant nodes than that of the OctoMap model. In the future, the performance of the proposed algorithms will be optimized. An map update policy will be developed so that the occupancy map can update accordingly when ORB-SLAM detects loop closure and updates the keyframes. Then the proposed model can be extended to the real-time mapping.

Acknowledgments Yu Miao thanks University of Bath grant University Research Studentship Award-Engineering and China Scholarship Council grant No. 201706120022 for financial support.

References

1. Robotic mapping, https://en.wikipedia.org/wiki/Robotic_mapping. Last accessed 23 Jan 2019
2. Kwon, Y., Kim, D. and Yoon, S.E.: Super ray based updates for occupancy maps. In: Proceedings of the 2016 IEEE International Conference on Robotics and Automation, pp. 4267–4274. IEEE, Stockholm (2016)
3. Rusu, R.B., Marton, Z.C., Blodow, N., Dolha, M. and Beetz, M.: Towards 3D point cloud based object maps for household environments. Robotics and Autonomous Systems **56**(11), 927–941 (2008)
4. Cole, D.M. and Newman, P.M.: Using laser range data for 3D SLAM in outdoor environments. In: Proceedings of the 2006 IEEE International Conference on Robotics and Automation, pp. 1556–1563. IEEE, Orlando (2006)
5. Rusu, R.B. and Cousins, S.: 3D is here: Point Cloud Library (PCL). In: Proceedings of the 2011 IEEE International Conference on Robotics and Automation, pp. 1–4. IEEE, Shanghai (2011)
6. Hornung, A., Wurm, K.M., Bennewitz, M.: OctoMap: An efficient probabilistic 3D mapping framework based on octrees. Autonomous Robots **34**(3), 189–206 (2013)
7. Mur-Artal, R., Montiel, J.M.M. and Tardos, J.D.: ORB-SLAM: a versatile and accurate monocular SLAM system. IEEE Transactions on Robotics **31**(5), 1147–1163 (2015)
8. Octree, https://en.wikipedia.org/wiki/Octree. Last accessed 24 Jan 2019
9. Point Cloud Compression, http://pointclouds.org/documentation/tutorials/compression.php#octree-compression. Last accessed 24 Jan 2019
10. Spatial Partitioning and Search Operations with Octrees, http://pointclouds.org/documentation/tutorials/octree.php#octree-search. Last accessed 24 Jan 2019
11. Spatial change detection on unorganized point cloud data, http://pointclouds.org/documentation/tutorials/octree_change.php#octree-change-detection. Last accessed 24 Jan 2019
12. Taketomi, T., Uchiyama, H. and Ikeda, S.: Visual SLAM algorithms: A survey from 2010 to 2016. IPSJ Transactions on Computer Vision and Applications **9**(1), 16 (2017)
13. Welford, B.P.: Note on a method for calculating corrected sums of squares and products. Technometrics **4**(3), 419–420 (1962)
14. Standard deviation, https://en.wikipedia.org/wiki/Standard_deviation. Last accessed 23 Jan 2019