Frontiers of Information Technology & Electronic Engineering
www.zju.edu.cn/jzus; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Building a dense surface map incrementally from semi-dense point cloud and RGB images*

Qian-shan LI[1,3], Rong XIONG‡[1,3], Shoudong HUANG[2,3], Yi-ming HUANG[4]

([1]*State Key Laboratory of Industrial Control Technology, Zhejiang University, Hangzhou 310027, China*)

([2]*Faculty of Engineering and Information Technology, The University of Technology, Sydney, NSW 2007, Australia*)

([3]*ZJU-UTS Joint Center on Robotics, Zhejiang University, Hangzhou 310027, China*)

([4]*Department of Control Science and Engineering, Zhejiang University, Hangzhou 310027, China*)

E-mail: liqianshan@zju.edu.cn; rxiong@iipc.zju.edu.cn; shoudong.huang@uts.edu.au; ymhuang@zju.edu.cn

Received Sept. 3, 2014; Revision accepted Jan. 24, 2015; Crosschecked June 9, 2015

Abstract: Building and using maps is a fundamental issue for bionic robots in field applications. A dense surface map, which offers rich visual and geometric information, is an ideal representation of the environment for indoor/outdoor localization, navigation, and recognition tasks of these robots. Since most bionic robots can use only small light-weight laser scanners and cameras to acquire semi-dense point cloud and RGB images, we propose a method to generate a consistent and dense surface map from this kind of semi-dense point cloud and RGB images. The method contains two main steps: (1) generate a dense surface for every single scan of point cloud and its corresponding image(s) and (2) incrementally fuse the dense surface of a new scan into the whole map. In step (1) edge-aware resampling is realized by segmenting the scan of a point cloud in advance and resampling each sub-cloud separately. Noise within the scan is reduced and a dense surface is generated. In step (2) the average surface is estimated probabilistically and the non-coincidence of different scans is eliminated. Experiments demonstrate that our method works well in both indoor and outdoor semi-structured environments where there are regularly shaped objects.

Key words: Bionic robot, Robotic mapping, Surface fusion
doi:10.1631/FITEE.14a0260 **Document code:** A **CLC number:** TP242.6

1 Introduction

Building and using maps is a fundamental issue for bionic robots in field applications like military surveillance, disaster relief, and urban reconstruction, where the maps are expected to carry detailed information. The map should not only show general environment shape, but also provide detailed context of the surface, which is important for navigating autonomously and implementing semantic tasks, such as object recognition and scene understanding. Most bionic robots use the configuration of a rotating 2D laser scanner and RGB cameras to observe the environment. However, such a configuration can only lead to a relatively sparse map, i.e., a semi-dense point cloud. A sparse map contains very limited information which reflects only the general shape of the environment. Using this kind of map, a bionic robot can hardly localize itself or read subtle cues in the environment. Therefore, it is necessary to find a way to build a detailed map from a semi-dense point cloud and its corresponding RGB images.

In the past decade, researchers in the field of robotic mapping often used laser scanners with a pan-tilt platform to collect 3D data (Cole and Newman, 2006; Maurelli *et al.*, 2009). The point cloud

‡ Corresponding author

* Project supported by the National Natural Science Foundation of China (Nos. 61075078 and 61473258)

ORCID: Qian-shan LI, http://orcid.org/0000-0003-0370-7100

obtained is said to be semi-dense because the angular step between two consecutive laser scan lines is often more than 1° and the laser scan range can be up to dozens of meters. In comparison, a point cloud collected by RGB-D cameras like Microsoft Kinect is said to be dense because the angular step between two neighboring pixels is only about 0.1° and the camera measurement range is just several meters. KinectFusion (Newcombe *et al.*, 2011) and Kintinuous (Whelan *et al.*, 2012) are good solutions for these RGB-D cameras to align and fuse dense data. But for large scenes, it is necessary to find a way to build a dense map using ordinary laser scanners and RGB cameras. The reason is two-fold: first, using Kintinuous with dense devices in large scenes is tedious because data should be collected up close in every corner of the environment; second, it is impossible to use the combination of KinectFusion/Kintinuous with semi-dense devices because KinectFusion/Kintinuous uses dense voxels to statistically find the average surface and it does not support semi-dense data.

Although there are long range 3D lidars like Velodyne HDL-64E (Velodyne, USA) that can obtain a dense point cloud, they are either too expensive or of too narrow a field of view in some direction.

In this paper, we propose an incremental method to generate a dense surface model from a semi-dense point cloud. The method first generates a dense surface mesh for every single scan which consists of a 3D point cloud and its corresponding image(s) by doing edge-aware resampling and triangulating the surface points. Then we incrementally fuse the dense surface mesh of a new scan into the whole map by estimating an optimal surface for the overlapping surface mesh to avoid non-coincidence of different scans. Figs. 1a and 1b are the semi-dense point cloud obtained by our self-made laser-camera system and the dense point cloud generated with our method, respectively. Fig. 1c shows the generated dense surface map represented by a textured triangular mesh. We choose triangular mesh as the representation of the surface because it has low cost in storage and it is easy to access the neighboring vertices and triangular faces. In addition, it is easy to attach the detailed texture information from an image on the surface.

The main contribution of this work is that a method is proposed to fuse a semi-dense point cloud and RGB images and generate a dense surface map, so that a bionic robot can use small light-weight 2D laser scanners and RGB cameras to build and to use a dense textured map.

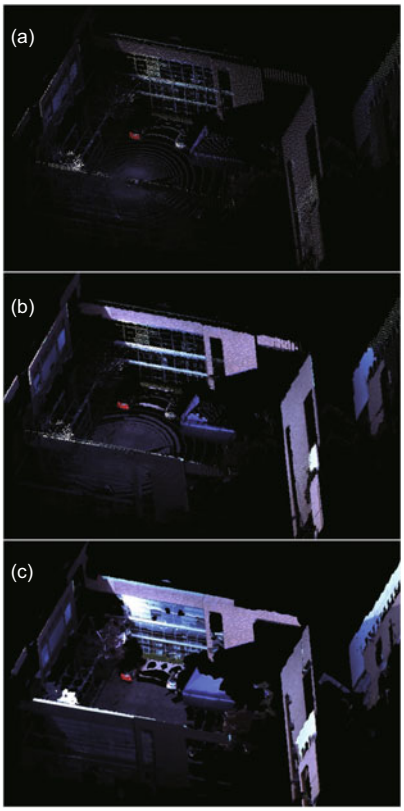

Fig. 1 Examples of semi-dense models and dense models: (a) colored point cloud gathered by a 2D laser scanner, which is semi-dense; (b) dense colored point cloud generated from (a) by our method; (c) triangular surface mesh generated by our method, where texture is mapped

1. A segmentation-based resampling method is presented. To denoise the data and resample the surface, the point cloud is first segmented into components and then every regularly shaped component is resampled individually. In segmentation, four rules on smoothness, distance, reflectivity, and color respectively are tested to insure that every segmented component is regular in shape.

2. A probabilistic vertex relocation method is proposed for mesh fusion. After aligning two mesh frames into the same coordinate, vertices on the substantial surface are generated to replace the old ones by probabilistically averaging two observations on the same local surface around each vertex. The proposed method integrated with the re-triangulation operation ensures that the fusion result is consistent

in both geometry and data structure.

In addition, we give a triangulation method on organized point cloud data, which uses a mask on the data and then deletes unqualified triangles, and a re-triangulation method by detecting overlapping border vertices.

2 Related work

There are some practical instances where rotating 1D/2D laser scanners are used to generate a 3D point cloud. Schadler *et al.* (2014) used a continuously rotating 2D laser scanner to realize mapping and navigation in rough terrain. Wulf and Wagner (2003) introduced a 3D scanning system with laser time-of-flight measurement devices and some methods for fast scanning. Básaca-Preciado *et al.* (2014) used their 3D laser scanning technical vision system, which is also based on a rotating laser scanner, to navigate autonomous mobile robots. Lopez *et al.* (2010) introduced the design of an optical scanner-based geodetic device for structural health monitoring. These works focus mainly on acquiring a 3D point cloud map which is mostly semi-dense, while the proposed method focuses mainly on building a dense surface model from this kind of semi-dense point cloud generated by 2D laser scanners.

In the field of robotics, building a dense map incrementally in a frame-by-frame mode is more the concern than operating in a post-processing mode. KinectFusion (Newcombe *et al.*, 2011) is a famous system that incrementally builds a dense map in small-scale scenes from a dense point cloud. Its extension, Kintinuous (Whelan *et al.*, 2012), has been proved effective in larger scenes. For mesh fusion, Lin *et al.* (2008) introduced a mesh composition method and Lou *et al.* (2010) proposed a triangular mesh merging method. These works focus mainly on grafting one specific shape onto another. The most related forerunner in this field is the work of Marton *et al.* (2009), which proposed a triangulation method that incrementally adds input points to the triangular mesh. It is time efficient and a good option for robotic applications.

In the field of graphics, researchers focus more on how to construct the surface model given a completed point cloud. Various triangulation methods have been introduced, either for organized point cloud data (Crossno and Angel, 1999; Holz and Behnke, 2013) or for unorganized data (Bajaj *et al.*, 1997; Wang *et al.*, 2007). For unorganized data, the Crust family (Amenta and Bern, 1999; Amenta *et al.*, 2001) and Cocone family (Dey *et al.*, 2001; 2011) of algorithms were proposed. In these methods, point data gathered at different positions are integrated in a set, on which triangulation is implemented by batch. These methods are widely used and have good performance.

3 Segmentation-based resampling

To generate a dense surface mesh for every single scan of point cloud and its corresponding image(s), edge-aware resampling is executed first to denoise the data and then the resampled point cloud is triangulated into a triangular surface mesh. We select triangular mesh as the representation of the surface because it has low cost in storage and it is easy to access the neighboring vertices and triangular faces. In addition, it is easy to attach the detailed texture information from an image on the surface. However, one can also choose to generate a dense point cloud as the RGB-D camera output.

3.1 Edge-aware resampling

Noise in the measured point cloud is inevitable. Fusing surfaces of multiple scans could reduce the error to some extent but it may fail when the noise in the measurement is too large. Thus, resampling the data is essential for every scan of a point cloud.

Moving least squares (MLS) is a popular surface fitting method (Rusu *et al.*, 2008; Huang *et al.*, 2013). Benefitting from the idea of estimating the surface shape in a piecewise way, MLS works well in denoising the data. However, this piecewise idea makes it easy to lose some true corners and edges, which will heavily influence the result of triangulation and fusion. To prevent this, we segment the point cloud data into components before using MLS. Some of the components may each represent a smooth surface patch, and they are resampled separately by MLS. With the proposed method, sharp corners and edges are well preserved and the result model can be more accurate. Compared with methods that detect and protect features like corners and edges before resampling outside the protected area (Dey *et al.*, 2012; Dey and Wang, 2013), our method first finds

the patches that are regular enough to be resampled, and then resamples them separately. In some cluttered areas like tree areas, the data arrangement is out of order. It is impossible to resample or interpolate the data, and there is no need to do so. Our method picks out patches that can be resampled and resamples them separately, leaving the features and the cluttered areas unchanged.

For segmenting the point cloud into components, region growing is a common algorithm. It recursively searches the neighborhood for new points which belong to the same region. In each region, points as a whole represent a smooth surface patch. The main effort is to define criteria to evaluate whether a new neighbor point belongs to the same surface patch. We have designed the following four rules. Once a new neighboring point, $\boldsymbol{P}_i$, is found around a member point, $\boldsymbol{P}_j$, it will be rejected if any of the four rules fails.

1. Smoothness

Neighboring points in a component should be smoothly connected. Otherwise, they might belong to different components. As mentioned above, resampling on a surface patch that contains points actually belonging to other surface patches will weaken the boundary between patches. Thus, this criterion is the most important. Denote $\boldsymbol{n}_i$ and $\boldsymbol{n}_j$ the unit normals at $\boldsymbol{P}_i$ and $\boldsymbol{P}_j$, respectively, $\boldsymbol{v}_{ij}$ the vector from $\boldsymbol{P}_i$ to $\boldsymbol{P}_j$, and d_{ij} the distance between $\boldsymbol{P}_i$ and $\boldsymbol{P}_j$. The local surface connecting $\boldsymbol{P}_i$ and $\boldsymbol{P}_j$ is considered smooth if and only if the following three equations are all satisfied:

$$\frac{2}{d_{ij}}\sin\left(\frac{\arccos|\boldsymbol{n}_i^{\mathrm{T}}\boldsymbol{n}_j|}{2}\right)<C_{\mathrm{th}}, \tag{1}$$

$$\arccos\left(\frac{|\boldsymbol{n}_i^{\mathrm{T}}\boldsymbol{v}_{ij}|}{d_{ij}}\right)<\theta_{\mathrm{th}}, \tag{2}$$

$$\arccos\left(\frac{|\boldsymbol{n}_j^{\mathrm{T}}\boldsymbol{v}_{ij}|}{d_{ij}}\right)<\theta_{\mathrm{th}}. \tag{3}$$

Condition (1) limits the local curvature lower than a threshold C_{th}, while (2) and (3) ensure that the two sample points locate in the same local plane, by constraining the angles between each normal and line segment ij. θ_{th} is the threshold for angle.

2. Distance

It is possible that a true gap exists between two neighboring points, especially when the data are sparse. The farther the distance between the two points, the higher the possibility. Thus, the distance between two sample points, d_{ij}, should be small, to ensure that the two sample points are indeed in the same surface patch. Considering the fact that the distance between two neighboring points is proportional to the measure distance of the laser scanner, points farther away from the sensor should be permitted to have a larger distance. Thus, this criterion is described as

$$\frac{d_{ij}}{d_i+d_j}<\alpha, \tag{4}$$

where d_i and d_j are the distances from the sensor to $\boldsymbol{P}_i$ and $\boldsymbol{P}_j$, respectively, and α is the threshold for point distance.

3. Reflectivity

Significant difference in reflectivity indicates different materials. Although they might well be connected in geometry, separating them up reserves more context for further semantic use, and it also supports using different resampling parameters for surface patches of different materials. Denoting the laser reflectivity values at $\boldsymbol{P}_i$ and $\boldsymbol{P}_j$ as r_i and r_j respectively, the criterion is

$$|r_i-r_j|<\beta, \tag{5}$$

where β is the threshold for reflectivity.

4. Color

The criterion on color is used if and only if more semantic context is needed. In this case, the colors of the two points c_i and c_j in the CIELAB color space are compared, with the threshold γ, i.e.,

$$|c_i-c_j|<\gamma. \tag{6}$$

After the above verifications, points are segmented into many components. The member points of each component share similar surface properties, and thus the re-prediction of each point's location through its neighbors is more reliable. The second-order MLS is executed for each component, avoiding the influence of other components near the boundary.

It is worth mentioning that if the input point cloud is organized, the region-growing based segmentation can be implemented in a more efficient way. The organized point cloud forms an undirected graph which takes each pixel in the 'image' as a vertex and constructs an edge between every two adjacent pixels. A graph-based region-growing scheme proposed by Felzenszwalb and Huttenlocher (2004) can be used in tandem with the four rules defined above. An edge connecting two adjacent points $\boldsymbol{P}_i$ and $\boldsymbol{P}_j$

in the 'image' would be deleted if any of the rules fails.

3.2 Triangulation

Since in many cases, each scan of a point cloud is gathered at a fixed location, the resampled points can be organized easily. Thus, we provide a method to triangulate these organized data. Owing to the ordered feature of the data, neighbors of a point can be determined directly as well as the triangulation. Unlike the methods in Gopi and Krishnan (2002) and Marton *et al.* (2009), which operate in a local neighborhood and gradually extend the triangulation boundary, our method directly uses a triangulation mask on the organized data and then deletes unqualified triangles (also called 'faces' in this paper).

Since the point cloud data gathered by the laser scanners are in spherical coordinates, a triangular face can be generated only from three adjacent points. This means, for a vertex of a triangular face of the surface, denoted as $\boldsymbol{P}$, the other two vertices of this triangular face are sure to be located in the neighborhood of $\boldsymbol{P}$. Thus, a mask shown in Fig. 2 can offer candidates of triangular faces. Each rectangle in the grid is split into two triangles by either of its diagonals, NW-SE (northwest-southeast) or NE-SW (northeast-southwest), and the diagonal which produces the best triangle will be adopted. Knowing the four vertices of a rectangle, $\boldsymbol{P}_A$, $\boldsymbol{P}_B$, $\boldsymbol{P}_C$, $\boldsymbol{P}_D$, and their normals, $\boldsymbol{n}_A$, $\boldsymbol{n}_B$, $\boldsymbol{n}_C$, $\boldsymbol{n}_D$, the weight of a diagonal (here take AC as an example) is specified by

$$w_{AC}=\max\left\{\frac{(\boldsymbol{n}_B+\boldsymbol{n}_A+\boldsymbol{n}_C)^{\mathrm{T}}[(\boldsymbol{P}_A-\boldsymbol{P}_B)\times(\boldsymbol{P}_C-\boldsymbol{P}_B)]}{\|(\boldsymbol{P}_A-\boldsymbol{P}_B)\times(\boldsymbol{P}_C-\boldsymbol{P}_B)\|_2},\right.$$
$$\left.\frac{(\boldsymbol{n}_D+\boldsymbol{n}_A+\boldsymbol{n}_C)^{\mathrm{T}}[(\boldsymbol{P}_A-\boldsymbol{P}_D)\times(\boldsymbol{P}_C-\boldsymbol{P}_D)]}{\|(\boldsymbol{P}_A-\boldsymbol{P}_D)\times(\boldsymbol{P}_C-\boldsymbol{P}_D)\|_2}\right\}. \tag{7}$$

Here the maximization operator is used, which ensures that the selected diagonal can generate the best triangle. Although a poor triangle may be generated simultaneously by the selected diagonal, there is a high possibility that the poor triangle does not exist. This is why we adopt the maximization operator rather than the average.

Because we use a full mask to generate the faces, some faces may be false, e.g., the face that contains the edge connecting foreground and background, and some faces may be very uncertain, e.g., the face with a normal direction that is nearly perpendicular to the sight line (laser beam). To filter out those false or uncertain faces, the following criterion is defined: the angle between the face normal and the sight line (laser beam) should be less than a threshold θ_{th}:

$$\max\left(\arccos\left(\frac{\boldsymbol{n}_{\Delta_{ijk}}\cdot\boldsymbol{P}}{\|\boldsymbol{P}\|_2}\right)\right)<\theta_{\mathrm{th}},\ \boldsymbol{P}\in\{\boldsymbol{P}_i,\boldsymbol{P}_j,\boldsymbol{P}_k\}. \tag{8}$$

Here Δ_{ijk} denotes the triangular face formed by vertices $\boldsymbol{P}_i$, $\boldsymbol{P}_j$, $\boldsymbol{P}_k$, and $\boldsymbol{n}_{\Delta_{ijk}}$ denotes its unit normal. The sensor locates at the origin.

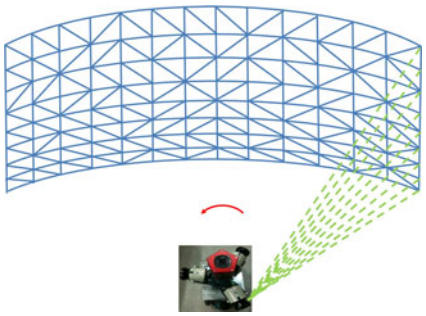

Fig. 2 Triangulation mask

Actually, $\boldsymbol{P}_i$, $\boldsymbol{P}_j$, $\boldsymbol{P}_k$ can be substituted by their range values r_i, r_j, r_k, respectively, and a simpler criterion can be used:

$$\frac{\max(r)-\min(r)}{\min(r)}<d_{\mathrm{th}},\ r\in\{r_i,r_j,r_k\}, \tag{9}$$

where d_{th} is the threshold. Faces that are rejected by this criterion are all deleted. Then camera image fragments are pasted onto the remaining faces.

4 Probabilistic vertex relocation

As the core of this paper, this section describes how the dense surfaces of multiple scans are fused together. The goal of fusion is to construct a surface model that is consistent not only in geometry but also in data structure. The word 'fusion' in this section does not include the alignment of triangular meshes, but refers to fusing the denoised and aligned triangular mesh only. We use scan-matching and SLAM (simultaneous localization and mapping) to align the surface meshes in advance. However, no matter how well the pose estimation performs, errors always exist, which means there are non-coincidences, i.e., gaps and intersections, among

multiple overlapping surfaces. If the vertices of the overlapped region are re-triangulated directly after pose alignment, the resultant surface mesh would be very rough. To solve this problem, our solution is to relocate the vertices of the overlapped region onto a new 'average' surface, and then to re-triangulate these relocated vertices.

Since point cloud data are often obtained one scan at a time, our fusion works in an incremental way. Once a new scan is available, it is fused into the previous constructed map. Thus, the problem is equivalent to fusing two meshes. One is the surface mesh that has been constructed from the beginning up to the current time. The other is the newest one. First, we relocate the vertices of both frames according to each vertex's corresponding faces in the two frames and their uncertainty. Second, the relocated vertices are relinked to form a whole consistent mesh. The first step glues different layers of surface together by generating vertices on the substantial surface, while the second integrates two meshes into one. After that, image fragments with higher resolution are reserved and tailored for new faces.

4.1 Vertex relocation

The vertex of a triangular face is actually a sample point on the real surface. Therefore, all the neighboring faces of a vertex in one frame can be viewed as an observation of this vertex's local surface. Meanwhile, faces from another frame, which locate near this vertex after alignment, act as another observation of the local surface. Due to the inevitable errors, the two observations do not coincide. To relocate a vertex, our idea is to fuse the two observations by taking into account all the neighboring faces in the frame that the vertex belongs to and the nearest face in the other frame. A constraint on the nearest face in the other frame is that the difference between the normal of the discussed vertex and that of the nearest face in the other frame should be smaller than a threshold. In the fusion process, the uncertainty of each face is used as its weight. A face with lower uncertainty plays a more important role in determining the new location, while a face with higher uncertainty plays a lesser role. Obviously, taking the uncertainty into account can ensure the certainty of the fusion result. Overall, the uncertainty of a vertex can be reduced by relocation.

In the following part of this subsection, the calculation of the vertices' new locations is introduced. First, given the neighboring vertices/faces of a vertex and the uncertainties of these faces, how a vertex's new location is calculated is introduced. Then the calculation of the uncertainties of these faces is discussed in detail.

Let TM_1 be one triangular mesh frame and TM_2 the other frame. Assume the two meshes have been transformed into the same coordinate via scan-matching techniques. Denote the vertex that is needed to be relocated in TM_1 by $\boldsymbol{S}$, and its n neighbors by $\boldsymbol{P}_1, \boldsymbol{P}_2, \ldots, \boldsymbol{P}_n$. $\boldsymbol{S}$ and each two of its adjacent neighbors form a triangular face. The n triangular faces are denoted by $\Delta_{P_1SP_2}, \Delta_{P_2SP_3}, \ldots, \Delta_{P_nSP_1}$. Note that $\boldsymbol{S}$ must locate in the overlapping area of TM_1 and TM_2. Denote $\boldsymbol{S}$'s nearest triangular face in TM_2 by $\Delta_{Q_1Q_2Q_3}$, where $\boldsymbol{Q}_1$, $\boldsymbol{Q}_2$, and $\boldsymbol{Q}_3$ are its vertices. $\Delta_{Q_1Q_2Q_3}$ can be determined by finding $\boldsymbol{S}$'s nearest vertex from the kd-tree. Then we pick out the nearest face that the vertex belongs to. An example is shown in Fig. 3.

If there are n ($n \geq 3$) non-coplanar faces in the neighborhood of $\boldsymbol{S}$, e.g., $\Delta_{P_1SP_2}$, $\Delta_{P_2SP_3}$, $\Delta_{P_3SP_4}, \ldots$, and a nearest face can be found in TM_2, the new location of $\boldsymbol{S}$, denoted as $\boldsymbol{S}_{\mathrm{new}}$, can be determined by

$$\boldsymbol{S}_{\mathrm{new}} = \underset{\boldsymbol{X}}{\operatorname{argmin}} \left(\frac{1}{n} \left(\frac{|(\boldsymbol{X}-\boldsymbol{S})^{\mathrm{T}} \boldsymbol{n}_{P_1SP_2}|^2}{\sigma^2_{P_1SP_2}} + \frac{|(\boldsymbol{X}-\boldsymbol{S})^{\mathrm{T}} \boldsymbol{n}_{P_2SP_3}|^2}{\sigma^2_{P_2SP_3}} + \frac{|(\boldsymbol{X}-\boldsymbol{S})^{\mathrm{T}} \boldsymbol{n}_{P_3SP_4}|^2}{\sigma^2_{P_3SP_4}} + \cdots \right) + \frac{|(\boldsymbol{X}-\boldsymbol{Q}_1)^{\mathrm{T}} \boldsymbol{n}_{Q_1Q_2Q_3}|^2}{\sigma^2_{Q_1Q_2Q_3}} \right), \tag{10}$$

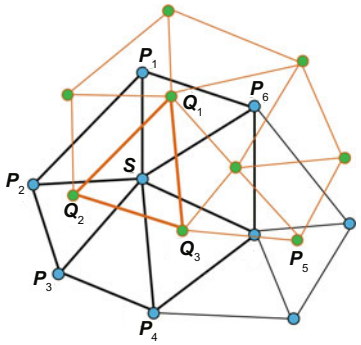

Fig. 3 An example of vertex relocation. For vertex $\boldsymbol{S}, \boldsymbol{P}_1, \boldsymbol{P}_2, \boldsymbol{P}_3, \boldsymbol{P}_4, \boldsymbol{P}_5$, and $\boldsymbol{P}_6$ are its neighbors in TM_1, while $\Delta_{Q_1Q_2Q_3}$ is its nearest triangular face in TM_2 and $\boldsymbol{Q}_1, \boldsymbol{Q}_2, \boldsymbol{Q}_3$ are the three vertices

where $\boldsymbol{n}_{P_1SP_2}, \boldsymbol{n}_{P_2SP_3}, \boldsymbol{n}_{P_3SP_4}, \ldots$ and $\boldsymbol{n}_{Q_1Q_2Q_3}$ are the unit normals of corresponding triangular faces, and $\sigma^2_{P_1SP_2}, \sigma^2_{P_2SP_3}, \sigma^2_{P_3SP_4}, \ldots$ and $\sigma^2_{Q_1Q_2Q_3}$ are the variances (the details will be described later in this section). $1/n$ means the influence of the faces from TM_1 is averaged, because it should be the two observations but not all the faces that play the same important role in calculating the new location.

Problem (10) can be solved in weighted least square form:

$$\boldsymbol{S}_{\mathrm{new}} = (\boldsymbol{A}^{\mathrm{T}}\boldsymbol{W}^2\boldsymbol{A})^{-1}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{W}^2\boldsymbol{b}, \tag{11}$$

where $\boldsymbol{W}$ is the weight matrix, i.e.,

$$\boldsymbol{W} = \mathrm{diag}\left(\frac{\sigma_{P_1SP_2}}{\sqrt{n}}, \frac{\sigma_{P_2SP_3}}{\sqrt{n}}, \frac{\sigma_{P_3SP_4}}{\sqrt{n}}, \ldots, \sigma_{Q_1Q_2Q_3}\right), \tag{12}$$

and

$$\boldsymbol{A} = (\boldsymbol{n}_{P_1SP_2}\ \boldsymbol{n}_{P_2SP_3}\ \boldsymbol{n}_{P_3SP_4}\ \ldots\ \boldsymbol{n}_{Q_1Q_2Q_3})^{\mathrm{T}}, \tag{13}$$

$$\boldsymbol{b} = \begin{bmatrix} \boldsymbol{P}_S^{\mathrm{T}}\boldsymbol{n}_{P_1SP_2} \\ \boldsymbol{P}_S^{\mathrm{T}}\boldsymbol{n}_{P_2SP_3} \\ \boldsymbol{P}_S^{\mathrm{T}}\boldsymbol{n}_{P_3SP_4} \\ \vdots \\ \boldsymbol{Q}_1^{\mathrm{T}}\boldsymbol{n}_{Q_1Q_2Q_3} \end{bmatrix}. \tag{14}$$

The covariance of the solution $\boldsymbol{S}_{\mathrm{new}}$ can be computed by

$$\Sigma_{\boldsymbol{S}_{\mathrm{new}}} = (\boldsymbol{A}^{\mathrm{T}}\boldsymbol{W}^2\boldsymbol{A})^{-1}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{W}^{\mathrm{T}}\boldsymbol{W}\Sigma_{\boldsymbol{b}}\boldsymbol{W}^{\mathrm{T}}\boldsymbol{W}\boldsymbol{A}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{W}^2\boldsymbol{A})^{-1}. \tag{15}$$

If a nearest face of $\boldsymbol{S}$ can be found in TM_2, but there are fewer than three non-coplanar faces in the neighborhood of $\boldsymbol{S}$, new constrains should be added. In the case of two non-coplanar faces intersecting at a line l, the new location is constrained to be the projection of $\boldsymbol{S}$ on l. In the case of only one face in the neighborhood, the new location is constrained to be the projection of $\boldsymbol{S}$ on the target plane.

If a nearest face of $\boldsymbol{S}$ cannot be found in TM_2 (the distance exceeds a certain value), nothing needs to be done because all the faces come from the same observation and the data have already been resampled (smoothed).

So far, the idea of vertex relocation has been presented, and the only question left is how to determine the variance $\sigma^2_{P_1SP_2}, \sigma^2_{P_2SP_3}, \sigma^2_{P_3SP_4}, \ldots$ and $\sigma^2_{Q_1Q_2Q_3}$ in Eq. (10), by which the observation uncertainties take effect. Since these variances indicate the uncertainty of point-plane distance, they can be quantized by investigating the uncertainty propagation process.

When a point $\boldsymbol{P}$ is sampled by a general 3D laser scanner, three values about its location are obtained, including range r, vertical angle of observation ϕ, and horizontal angle of observation θ. The variances of the three values are $\sigma_r^2(r)$, σ_ϕ^2, and σ_θ^2, respectively. $\sigma_r^2(r)$ is a function of range r. σ_ϕ^2 and σ_θ^2 indicate the angular accuracy. Then the point $\boldsymbol{P}$ can be expressed in local Cartesian coordinates by

$${}_{\mathrm{L}}\boldsymbol{P} = \begin{bmatrix} r\cos\phi\cos\theta \\ r\cos\phi\sin\theta \\ r\sin\phi \end{bmatrix}, \tag{16}$$

with the covariance matrix

$$\Sigma_{{}_{\mathrm{L}}\boldsymbol{P}} = \boldsymbol{J}_{\mathrm{P2C}} \begin{bmatrix} \sigma_r^2(r) & & \\ & \sigma_\phi^2 & \\ & & \sigma_\theta^2 \end{bmatrix} \boldsymbol{J}_{\mathrm{P2C}}^{\mathrm{T}}, \tag{17}$$

where $\boldsymbol{J}_{\mathrm{P2C}}$ is the Jacobian matrix of the Polar-Cartesian transformation.

On the other hand, the uncertainty of device pose is added to that of sample points. Given the 6D pose of the device

$$\mathbf{Pos}_{\mathrm{d}} = (\ x_{\mathrm{d}}\ y_{\mathrm{d}}\ z_{\mathrm{d}}\ \alpha_{\mathrm{d}}\ \beta_{\mathrm{d}}\ \gamma_{\mathrm{d}}\ )^{\mathrm{T}}, \tag{18}$$

with its covariance matrix $\Sigma_{\mathrm{Pos_d}}$, the global position of point $\boldsymbol{P}$ can be computed by

$$\boldsymbol{P} = f(\mathbf{Pos}_{\mathrm{d}},\ {}_{\mathrm{L}}\boldsymbol{P}) = \begin{bmatrix} x_{\mathrm{d}} \\ y_{\mathrm{d}} \\ z_{\mathrm{d}} \end{bmatrix} + \begin{bmatrix} R_{11}\ R_{12}\ R_{13} \\ R_{21}\ R_{22}\ R_{23} \\ R_{31}\ R_{32}\ R_{33} \end{bmatrix} {}_{\mathrm{L}}\boldsymbol{P}, \tag{19}$$

where

$$\begin{cases} R_{11} = \cos\gamma_{\mathrm{d}}\cos\beta_{\mathrm{d}}, \\ R_{12} = \cos\gamma_{\mathrm{d}}\sin\beta_{\mathrm{d}}\sin\alpha_{\mathrm{d}} - \sin\gamma_{\mathrm{d}}\cos\alpha_{\mathrm{d}}, \\ R_{13} = \cos\gamma_{\mathrm{d}}\sin\beta_{\mathrm{d}}\sin\alpha_{\mathrm{d}} + \sin\gamma_{\mathrm{d}}\sin\alpha_{\mathrm{d}}, \\ R_{21} = \sin\gamma_{\mathrm{d}}\cos\beta_{\mathrm{d}}, \\ R_{22} = \sin\gamma_{\mathrm{d}}\sin\beta_{\mathrm{d}}\sin\alpha_{\mathrm{d}} + \cos\gamma_{\mathrm{d}}\cos\alpha_{\mathrm{d}}, \\ R_{23} = \sin\gamma_{\mathrm{d}}\sin\beta_{\mathrm{d}}\cos\alpha_{\mathrm{d}} - \cos\gamma_{\mathrm{d}}\sin\alpha_{\mathrm{d}}, \\ R_{31} = -\sin\beta_{\mathrm{d}}, \\ R_{32} = \cos\beta_{\mathrm{d}}\sin\alpha_{\mathrm{d}}, \\ R_{33} = \cos\beta_{\mathrm{d}}\cos\alpha_{\mathrm{d}}. \end{cases} \tag{20}$$

For any of the faces in Fig. 3, e.g., $\Delta_{P_1SP_2}$, the

distance between $\boldsymbol{S}_{\text{new}}$ and the plane is derived by

$$\begin{aligned} d_{S_{\text{new}}-P_1SP_2} &= (\boldsymbol{S}_{\text{new}}-\boldsymbol{S})^{\mathrm{T}}\boldsymbol{n}_{P_1SP_2} \\ &= (\boldsymbol{S}_{\text{new}}-\boldsymbol{S})^{\mathrm{T}}\frac{\boldsymbol{N}_{P_1SP_2}}{\|\boldsymbol{N}_{P_1SP_2}\|_2} \\ &= (\boldsymbol{S}_{\text{new}}-\boldsymbol{S})^{\mathrm{T}}\frac{(\boldsymbol{P}_1-\boldsymbol{S})\times(\boldsymbol{P}_2-\boldsymbol{S})}{\|(\boldsymbol{P}_1-\boldsymbol{S})\times(\boldsymbol{P}_2-\boldsymbol{S})\|_2}. \end{aligned} \tag{21}$$

Let

$$\boldsymbol{P_1} = (x_{P_1}\ y_{P_1}\ z_{P_1})^{\mathrm{T}}, \tag{22}$$

$$\boldsymbol{P_2} = (x_{P_2}\ y_{P_2}\ z_{P_2})^{\mathrm{T}}, \tag{23}$$

$$\boldsymbol{S} = (x_S\ y_S\ z_S)^{\mathrm{T}}, \tag{24}$$

$$\boldsymbol{S}_{\text{new}} = (x\ y\ z)^{\mathrm{T}}. \tag{25}$$

Eq. (21) can be rewritten as

$$\begin{aligned} d_{S_{\text{new}}-P_1SP_2} = &\begin{bmatrix} x-x_S \\ y-y_S \\ z-z_S \end{bmatrix}^{\mathrm{T}} \\ &\cdot\begin{bmatrix} (y_{p_1}-y_S)(z_{p_2}-z_S)-(z_{p_1}-z_S)(y_{p_2}-y_S) \\ (z_{p_1}-z_S)(x_{p_2}-x_S)-(x_{p_1}-x_S)(z_{p_2}-z_S) \\ (x_{p_1}-x_S)(y_{p_2}-y_S)-(y_{p_1}-y_S)(x_{p_2}-x_S) \end{bmatrix} \\ &\cdot\left\| \begin{matrix} (y_{p_1}-y_S)(z_{p_2}-z_S)-(z_{p_1}-z_S)(y_{p_2}-y_S) \\ (z_{p_1}-z_S)(x_{p_2}-x_S)-(x_{p_1}-x_S)(z_{p_2}-z_S) \\ (x_{p_1}-x_S)(y_{p_2}-y_S)-(y_{p_1}-y_S)(x_{p_2}-x_S) \end{matrix} \right\|_2^{-1}. \end{aligned} \tag{26}$$

By denoting $\boldsymbol{J}_d$ as the Jacobian matrix of $d_{S_{\text{new}}-P_1SP_2}$ with respect to x_{P_1}, y_{P_1}, z_{P_1}, x_{P_2}, y_{P_2}, z_{P_2}, x_S, y_S, and z_S, the variance of $d_{S_{\text{new}}-P_1SP_2}$ can be derived as

$$\sigma_{d^2_{S_{\text{new}}-P_1SP_2}} = \boldsymbol{J}_d\Sigma_{P_1,S,P_2}\boldsymbol{J}_d^{\mathrm{T}}, \tag{27}$$

where Σ_{P_1,S,P_2} is the covariance matrix of the three points $\boldsymbol{P}_1$, $\boldsymbol{S}$, and $\boldsymbol{P}_2$. According to Eq. (19), we can denote the vector of the combination of $\boldsymbol{P}_1$, $\boldsymbol{S}$, and $\boldsymbol{P}_2$ as

$$\begin{bmatrix} \boldsymbol{P}_1 \\ \boldsymbol{S} \\ \boldsymbol{P}_2 \end{bmatrix} = f(\mathbf{Pos}_{\mathrm{d}},\ {}_{\mathrm{L}}\boldsymbol{P}_1,\ {}_{\mathrm{L}}\boldsymbol{S},\ {}_{\mathrm{L}}\boldsymbol{P}_2), \tag{28}$$

where ${}_{\mathrm{L}}\boldsymbol{P}_1$, ${}_{\mathrm{L}}\boldsymbol{S}$, and ${}_{\mathrm{L}}\boldsymbol{P}_2$ represent the locations of $\boldsymbol{P}_1$, $\boldsymbol{S}$, and $\boldsymbol{P}_2$ in the local coordinates of the mesh frame they belong to, respectively. The covariance matrix $\Sigma_{P_1,P_2,S}$ in Eq. (27) is calculated by

$$\Sigma_{P_1,S,P_2} = \boldsymbol{J}_f\begin{bmatrix} \Sigma_{\mathbf{Pos}_{\mathrm{d}}} & & & \\ & \Sigma_{{}_{\mathrm{L}}\boldsymbol{P}_1} & & \\ & & \Sigma_{{}_{\mathrm{L}}\boldsymbol{S}} & \\ & & & \Sigma_{{}_{\mathrm{L}}\boldsymbol{P}_2} \end{bmatrix}\boldsymbol{J}_f^{\mathrm{T}}, \tag{29}$$

where $\boldsymbol{J}_f$ is the Jacobian of $f(\mathbf{Pos}_d,\ {}_{\mathrm{L}}\boldsymbol{P}_1,\ {}_{\mathrm{L}}\boldsymbol{S},\ {}_{\mathrm{L}}\boldsymbol{P}_2)$ in Eq. (28), and $\Sigma_{\mathbf{Pos}_{\mathrm{d}}}$, $\Sigma_{{}_{\mathrm{L}}\boldsymbol{P}_1}$, $\Sigma_{{}_{\mathrm{L}}\boldsymbol{S}}$, and $\Sigma_{{}_{\mathrm{L}}\boldsymbol{P}_2}$ are covariance matrices which can be calculated according to Eq. (17).

4.2 Re-triangulation

After all the vertices in the overlapping part are relocated onto a new consistent surface, the only operation left is to relink them. The method of Marton *et al.* (2009) provides a convenient option because it is simple and fast. In the cases when frames of data are obtained at sparse locations and the overlapping part between two frames is small, a simpler re-triangulation method by detecting the overlapping border is introduced here.

Instead of destroying all the triangular faces in the overlapping region, we use the following strategy: set one frame as the reference denoted as RF, the link pattern of which is preserved, and then incrementally relink each vertex in the overlapping part of another frame (denoted as AF) into RF. The relink process is demonstrated in Fig. 4.

1. Label the vertices in AF. In the previous relocation step, each vertex in the overlapping region of RF has found a nearest face in AF. Thus, we label the vertices of this face in AF by the corresponding vertex in RF.

2. Delete the edges in AF that connect vertices with different labels, as shown in the middle figure of Fig. 4. Along with edge deleting, the faces that these edges belong to are also deleted. We call the vertices that these deleted edges connect as 'border vertices' and the other vertices as 'non-border vertices'.

3. Insert the 'non-border vertices' into their corresponding nearest triangular face in RF, where the original edges among them are reserved.

4. Link the 'border vertices' in AF with RF. It can be regarded as linking between two polylines. First each vertex on one polyline (denoted by PL1) is linked to the nearest vertex on the other polyline (denoted by PL2), and then each unlinked vertex on PL2 is linked to the nearest vertex on PL1.

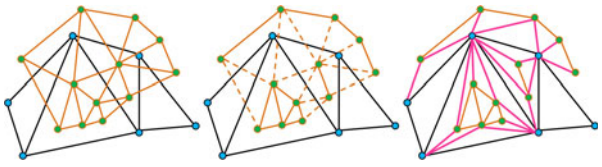

Fig. 4 Relinking of vertices

5 Experiments and results

5.1 Experimental device

The device used for experiments is shown in Fig. 5. Three SICK LMS-151 laser scanners (SICK, Germany) are integrated with a PointGrey Ladybug 3 panoramic camera (Point Grey, Canada). It is an extension of the design introduced by Sheehan *et al.* (2012). We design this prototype to verify the feasibility of our method, and further a light-weight edition will be developed and deployed on a bionic robot. The method in Sheehan *et al.* (2012) is used for the intrinsic calibration of the three-laser-scanner, and that in Pandey *et al.* (2010) is for the cross-calibration between the lasers and the camera. The RGB-D data obtained have a nearly full spherical field of view but have two black areas in the laser scanner data. One is between $-90°$ and $-45°$, close to the bottom area of the device, where the data contain the bracing components of the device, e.g., bracket and car roof. The other is the zone where the panoramic camera is mounted. The laser scanners can hardly block the view of the camera. With a scanning rate of 50 lines/s of each SICK LMS-151 and the rotational speed of $60°$/s of the disc, the device achieves a horizontal angular resolution of $1.2°$. Using this device, we can not only build the surface but also attach high-resolution textures to the surface.

Fig. 5 Three SICK LMS-151 laser scanners and a PointGrey Ladybug 3 panoramic camera integrated in an RGB-D unit

5.2 General results

To demonstrate the effectiveness of our method, several experiments in both indoor and outdoor scenes of Zhejiang University have been conducted. Fig. 6 (on the next page) demonstrates two results: one is from four frames of point cloud collected in the yard of our institute, and the other is the model built in a corner of our lab with 10 frames of data.

Fig. 7 demonstrates the results in a semi-structured environment. When the robot traverses in the wild, potential targets like buildings and vehicles can be recorded.

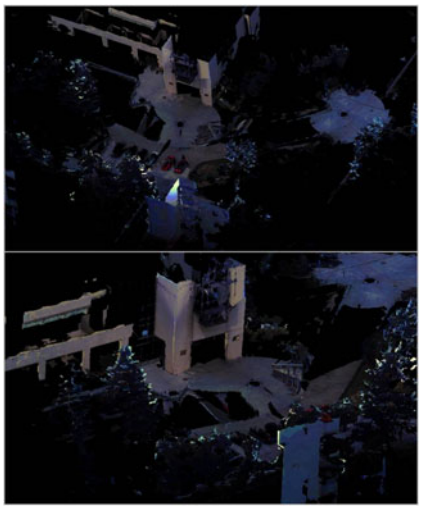

Fig. 7 Results in a semi-structured environment where there are regularly shaped objects like buildings and irregularly shaped objects like trees

Compared with the results of Cole and Newman (2006) and Maurelli *et al.* (2009), which generate only semi-dense point clouds of the environment, our results are dense and textured on the regularly shaped parts of the environment. This is important for a bionic robot in inspecting the environment. Compared with the results of KinectFusion (Newcombe *et al.*, 2011) and Kintinuous (Whelan *et al.*, 2012), which use a dense point cloud as input and use a dense voxel grid to estimate the 'average' surface, our results are not that exquisite. However, while they take dozens of scans to reconstruct a small scene, we reconstruct a large scene using just a few scans. Compared with the results of Marton *et al.* (2009), which use robust moving least square (RMLS) to resample the noisy data and gradually triangulate new points into the existing surface mesh, our results are better when fusing surface meshes of significantly different uncertainties because the probabilistic fusion method gives more weight to observations with higher certainties.

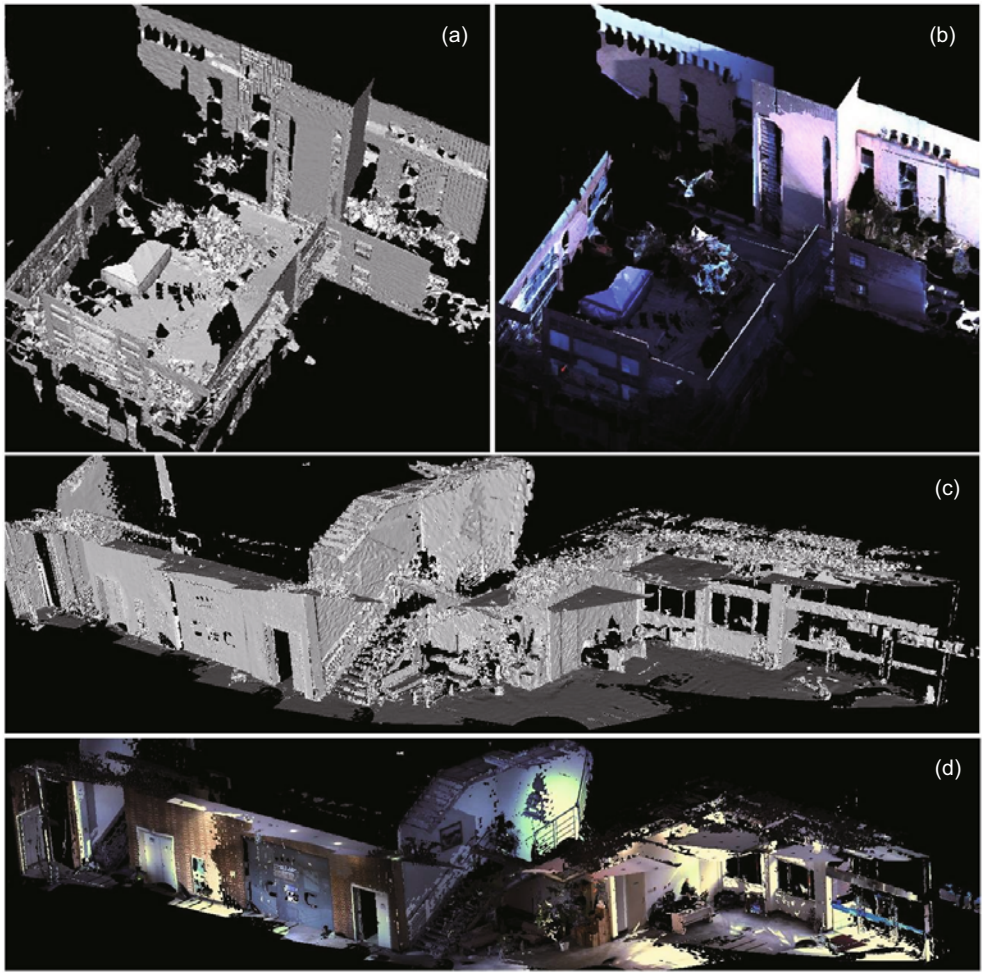

Fig. 6 General results of outdoor and indoor scenes: (a) outdoor surface; (b) outdoor surface with texture; (c) indoor surface; (d) indoor surface with texture

5.3 Resampling

Fig. 8 illustrates the importance of resampling: if the input point data are not resampled before triangulation, the triangular surface would be influenced by noise. In some serious cases, too many faces are deleted, leaving large holes on the surface, as shown in Fig. 8a. Fig. 9 compares the point clouds after resampling with the traditional MLS method and our segmentation-based MLS method. It is shown that sharp edges and corners are well preserved by our method, due to non-interference among different geometric components.

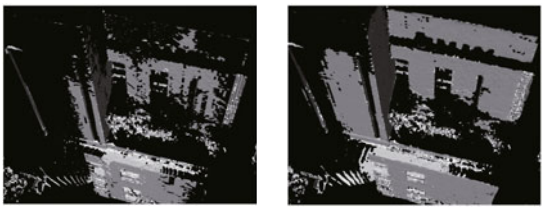

Fig. 8 Triangulation results without (a) or with (b) resampling

5.4 Vertex relocation

This experiment demonstrates the necessity of vertex relocation. As mentioned above, this operation relocates the vertices of overlapped surface mesh onto the underlying surface, which would remove the gap between two surface meshes and make the resultant mesh smooth at the joint area. Fig. 10 shows the part of a building where two frames of the surface mesh overlap. In Fig. 10a, re-triangulation is directly carried out without relocating any of the vertices, so the surface at the joint is rough. While in Fig. 10b,

vertices are first relocated onto the underlying surface, so the fused mesh is much smoother.

5.5 Surface update

Since the proposed method has a probabilistic nature, we are able to demonstrate its advantage in surface update. As indicated in Fig. 11, as soon as a closer observation (which is more certain) is made, the previous low-resolution surface is updated to be finer.

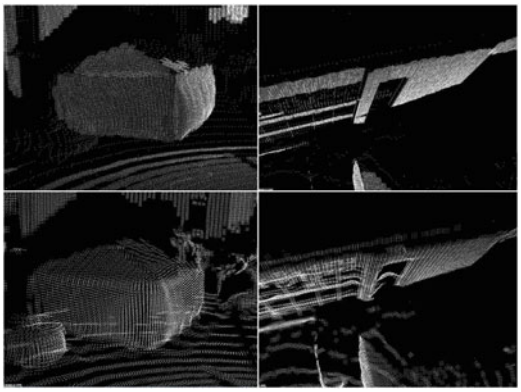

Fig. 9 Segmentation-based resampling (first row) v.s. ordinary MLS (second row). By resampling separately on each geometric component, the resulting cloud preserves better shape and is much cleaner, while ordinary MLS gets confused at the boundary (points are upsampled to better demonstrate the resampling result)

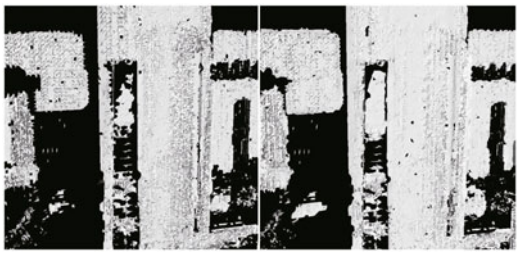

Fig. 10 Results without (a) (having a rough surface at the joint) or with (b) (much smoother at the joint) vertex relocation

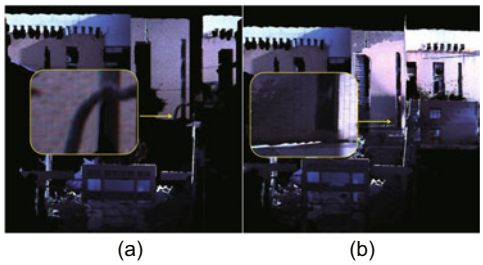

Fig. 11 Texture update: (a) observe at 10 m away; (b) observe at 3 m away

5.6 Change in vertices' covariance

Fig. 12 plots the histograms of the maximum eigenvalue of the point covariance matrix (square root) and Fig. 13 plots those of the traces. The maximum eigenvalue indicates the primary axis length of covariance of a point, while the trace indicates the total length of the three axes, which also measures the uncertainty of a point. For almost every histogram, it is true that the black shape (data after fusion) locates more on the left than the green one

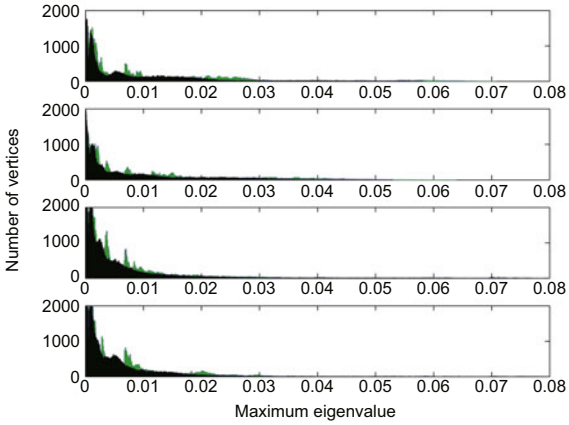

Fig. 12 Histograms of four frames of surface vertices on the maximum eigenvalue of squared covariances. Green: data before fusion; black: data after fusion. References to color refer to the online version of this figure

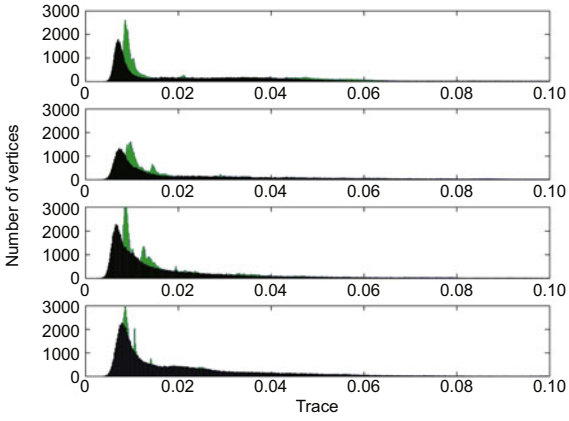

Fig. 13 Histograms of four frames of surface vertices on the trace of squared covariances. Green: data before fusion; black: data after fusion. References to color refer to the online version of this figure

(data before fusion) does. Thus, fusion indeed decreases the uncertainties of points (i.e., vertices of a surface).

6 Conclusions

In this paper, we have presented a method to fuse a semi-dense point cloud and RGB images and generated a dense surface map. The resultant dense surface map offers rich visual and geometric information, and is an ideal representation of the environment for indoor/outdoor localization, navigation, and recognition tasks of bionic robots. With this method, a bionic robot can use small light-weight 2D laser scanners and RGB cameras to build and use a densely textured map. Experiments conducted in Zhejiang University campus have demonstrated the effectiveness of our method in the semi-structured environment.

First, the proposed method which generates a dense surface map relies on a hypothesis that the environment surface is 'smooth' enough to be represented by textured mesh. This hypothesis heavily limits the use of bionic robots. In un-structured environments, data are cluttered and no surface can be generated. Then the image, which is more dense than the sparse laser points, cannot be used to interpolate or to enrich the surface. Thus, extending this method to un-structured environments is of great significance.

Second, the current stop-and-go mode of mapping is not that intelligent nor elegant, and mapping while the robot is moving can be more practical. Considering that bionic robots do not have odometry to indicate the displacement of the robot, visual methods can be used and incorporated with laser observations to estimate the robot pose in real time.

References

Amenta, N., Bern, M., 1999. Surface reconstruction by Voronoi filtering. *Discr. Comput. Geom.*, **22**(4):481-504. [doi:10.1007/PL00009475]

Amenta, N., Choi, S., Kolluri, R.K., 2001. The power crust. Proc. 6th ACM Symp. on Solid Modeling and Applications, p.249-266. [doi:10.1145/376957.376986]

Bajaj, C.L., Bernardini, F., Xu, G., 1997. Reconstructing surfaces and functions on surfaces from unorganized three-dimensional data. *Algorithmica*, **19**(1-2):243-261. [doi:10.1007/PL00014418]

Básaca-Preciado, L.C., Sergiyenko, O.Y., Rodríguez-Quinonez, J.C., *et al.*, 2014. Optical 3D laser measurement system for navigation of autonomous mobile robot. *Opt. Lasers Eng.*, **54**:159-169. [doi:10.1016/j.optlaseng.2013.08.005]

Cole, D.M., Newman, P.M., 2006. Using laser range data for 3D SLAM in outdoor environments. Proc. IEEE Int. Conf. on Robotics and Automation, p.1556-1563. [doi:10.1109/ROBOT.2006.1641929]

Crossno, P., Angel, E., 1999. Spiraling edge: fast surface reconstruction from partially organized sample points. Proc. Conf. on Visualization, p.317-324.

Dey, T.K., Wang, L., 2013. Voronoi-based feature curves extraction for sampled singular surfaces. *Comput. Graph.*, **37**(6):659-668. [doi:10.1016/j.cag.2013.05.014]

Dey, T.K., Giesen, J., Hudson, J., 2001. Delaunay based shape reconstruction from large data. Proc. IEEE Symp. on Parallel and Large-Data Visualization and Graphics, p.19-146. [doi:10.1109/PVGS.2001.964399]

Dey, T.K., Dyer, R., Wang, L., 2011. Localized Cocone surface reconstruction. *Comput. Graph.*, **35**(3):483-491. [doi:10.1016/j.cag.2011.03.014]

Dey, T.K., Ge, X., Que, Q., *et al.*, 2012. Feature-preserving reconstruction of singular surfaces. *Comput. Graph. Forum*, **31**(5):1787-1796. [doi:10.1111/j.1467-8659.2012.03183.x]

Felzenszwalb, P.F., Huttenlocher, D.P., 2004. Efficient graph-based image segmentation. *Int. J. Comput. Vis.*, **59**(2):167-181. [doi:10.1023/B:VISI.0000022288.19776.77]

Gopi, M., Krishnan, S., 2002. A fast and efficient projection-based approach for surface reconstruction. Proc. Brazilian Symp. on Computer Graphics and Image Processing, p.179-186. [doi:10.1109/SIBGRA.2002.1167141]

Holz, D., Behnke, S., 2013. Fast range image segmentation and smoothing using approximate surface reconstruction and region growing. Proc. 12th Int. Conf. on Intelligent Autonomous Systems, p.61-73. [doi:10.1007/978-3-642-33932-5_7]

Huang, H., Wu, S., Gong, M., *et al.*, 2013. Edge-aware point set resampling. *ACM Trans. Graph.*, **32**(1):Article 9. [doi:10.1145/2421636.2421645]

Lin, J., Jin, X., Wang, C., *et al.*, 2008. Mesh composition on models with arbitrary boundary topology. *IEEE Trans. Visual. Comput. Graph.*, **14**(3):653-665. [doi:10.1109/TVCG.2007.70632]

Lopez, M.R., Sergiyenko, O.Y., Tyrsa, V.V., *et al.*, 2010. Optoelectronic method for structural health monitoring. *Struct. Health Monit.*, **9**(2):105-120. [doi:10.1177/1475921709340975]

Lou, R., Pernot, J.P., Mikchevitch, A., *et al.*, 2010. Merging enriched finite element triangle meshes for fast prototyping of alternate solutions in the context of industrial maintenance. *Comput.-Aid. Des.*, **42**(8):670-681. [doi:10.1016/j.cad.2010.01.002]

Marton, Z.C., Rusu, R.B., Beetz, M., 2009. On fast surface reconstruction methods for large and noisy point clouds. Proc. IEEE Int. Conf. on Robotics and Automation, p.3218-3223. [doi:10.1109/ROBOT.2009.5152628]

Maurelli, F., Droeschel, D., Wisspeintner, T., *et al.*, 2009. A 3D laser scanner system for autonomous vehicle navigation. Proc. Int. Conf. on Advanced Robotics, p.1-6.

Newcombe, R.A., Izadi, S., Hilliges, O., *et al.*, 2011. KinectFusion: real-time dense surface mapping and tracking. Proc. 10th IEEE Int. Symp. on Mixed and Augmented Reality, p.127-136. [doi:10.1109/ISMAR.2011.6092378]

Nüchter, A., Lingemann, K., Hertzberg, J., *et al.*, 2007. 6D SLAM—3D mapping outdoor environments. *J. Field Robot.*, **24**(8-9):699-722. [doi:10.1002/rob.20209]

Pandey, G., McBride, J., Savarese, S., *et al.*, 2010. Extrinsic calibration of a 3D laser scanner and an omnidirectional camera. Proc. 7th IFAC Symp. on Intelligent Autonomous Vehicles.

Rusu, R.B., Marton, Z.C., Blodow, N., *et al.*, 2008. Towards 3D point cloud based object maps for household environments. *Robot. Auton. Syst.*, **56**(11):927-941. [doi:10.1016/j.robot.2008.08.005]

Schadler, M., Stückler, J., Behnke, S., *et al.*, 2014. Rough terrain 3D mapping and navigation using a continuously rotating 2D laser scanner. *Künstl. Intell.*, **28**(2):93-99. [doi:10.1007/s13218-014-0301-8]

Sheehan, M., Harrison, A., Newman, P., 2012. Self-calibration for a 3D laser. *Int. J. Robot. Res.*, **31**(5):675-687. [doi:10.1177/0278364911429475]

Wang, Y.B., Sheng, Y.H., Lv, G.N., *et al.*, 2007. A Delaunay-based surface reconstrution algorithm for unorganized sampling points. *J. Image Graph.*, **12**(9):1537-1543 (in Chinese).

Whelan, T., Kaess, M., Fallon, M., *et al.*, 2012. Kintinuous: Spatially Extended KinectFusion. Technical Report No. MIT-CSAIL-TR-2012-020. Massachusetts Institute of Technology, USA.

Wulf, O., Wagner, B., 2003. Fast 3D scanning methods for laser measurement systems. Proc. Int. Conf. on Control Systems and Computer Science, p.2-5.