Real-time surface of revolution reconstruction on dense SLAM

Liming Yang
Ecole Centrale de Nantes

Hideaki Uchiyama
Kyushu University

Jean-Marie Normand
Ecole Centrale de Nantes

Guillaume Moreau
Ecole Centrale de Nantes

Hajime Nagahara
Kyushu University

Rin-ichiro Taniguchi
Kyushu University

Abstract

We present a fast and accurate method for reconstructing surfaces of revolution (SoR) on 3D data and its application to structural modeling of a cluttered scene in real-time. To estimate a SoR axis, we derive an approximately linear cost function for fast convergence. Also, we design a framework for reconstructing SoR on dense SLAM. In the experiment results, we show our method is accurate, robust to noise and runs in real-time.

1. Introduction

Scene understanding from 3D data is an important research topic in computer vision and robotic perception because it can be applied to various types of systems such as robotic grasping, model simplification, augmented reality, etc. Existing methods on this topic can be classified into two main families: top-down and bottom-up approaches [6]. Top-down approaches are often referred to as object detection, since they aim at detecting pre-known models. Bottom-up approaches are sometimes called geometric reconstruction methods. They detect primitive shapes in a scene and establish relationships between these shapes whenever possible.

Primitive detection and localization is a crucial step in bottom-up approaches: it represents the first step of those methods and determines their application range. However, existing approaches can only deal with a limited type of primitive shapes such as: planes only [6], cylinders only [9] or thin-structures [11]. More complex primitives such as spheres, cones or tori [10] can be detected but those techniques still remain limited to pre-defined parametric shapes. It is important for applications in reverse engineering or in the construction field to deal with various types of geometric primitive shapes. Also, on-line scene understanding is desirable as dense SLAM runs in real-time [14].

In this paper, we aim at detecting and reconstructing surfaces of revolution (SoR) in a cluttered scene in real-time. SoR can express the majority of primitives and are quite common in man-made objects due to their ease of production. Since the most crucial step in detecting a SoR is to estimate its rotation axis, we propose a fast and accurate method, which boils down the estimation of rotation axis to one dimension search. The main contributions of this paper are summarized as follows:

- a fast and accurate estimation of SoR axes on 3D data
- a framework for detecting, localizing and reconstructing SoR in a cluttered scene in real-time

The rest of the paper is organized as follows: after introducing related work in Section 2, we present a fast and accurate method for SoR axis estimation in Section 3, its implementation details in Section 4 and real-time SoR reconstruction in Section 5. Finally, we present our results with both synthetic and real data in Section 6.

2. Related work

For semantic 3D understanding, structural modeling using geometric primitives has been investigated in the literature. Qiu et al. [9] use parallel cylinder detection to reconstruct a pipeline system. Song et al. [11] use beams and planes to reconstruct thin structural systems, such as chairs. Thanh et al. [6] use planes to find semantic structures of a scene. In order to model the structure of more complex objects, Schnabel et al. [10] propose a RANSAC-like method which can detect cones, cylinders, spheres and tori to reconstruct more types of primitive shapes. Taguchi et al. [12] improved this method by introducing distance field functions for efficient computation. Drost et al. [2] propose to use Hough Transform to detect cylinders and spheres. All these methods can detect limited and parametric shapes, and are not designed to work in real-time.

A surface of revolution (SoR), or rotational object, is formed by rotating a 2D curve (i.e. the profile) in 3D space with respect to a 3D line (i.e. the axis), thus it is axis-symmetric and is a more general representation of geometric primitives. Once the position of its rotation axis is

978-1-5090-5407-7/16 $31.00 © 2016 IEEE
DOI 10.1109/3DV.2016.13

determined, the profile can be easily calculated. Existing approaches are dedicated to estimate the rotation axis of a SoR, and can be classified as direct, iterative and brute force methods. For direct methods, Pottmann et al. [8] use 3D line geometry to classify surfaces and find the rotation axis of a SoR by solving a 6D eigenvalue problem. Lou et al. [4] derive the same formulation by using distance functions. For iterative methods, [4],[15] and [7] use ICP-like method [1]. They first estimate the profile function of a SoR by using a presumed rotation axis, thus creating an initial model of the SoR. Then, they try to align this model with the point cloud, which results in a better estimation of the axis for the next iteration. Brute force methods use the fact that the intersection between the SoR and a plane perpendicular to its rotation axis is a circle. Han et al. [3] use planes containing the normal of a surface point to cut a SoR and evaluate the intersecting curve. The plane which gives the least curvature variation is used to calculate the rotation axis. Mara et al. [5] use a set of parallel planes to cut a SoR. Then circles are fitted on each plane and the variation of circle centers across all planes is calculated. This set of planes is then rotated in a 2-dimensional search space so that the set which gives the least variation of circle centers is used to calculate the rotation axis.

Although direct methods are quite fast, they cannot find accurate results if input data contains even moderate noise (cf. Section 6). Iterative methods use the whole point cloud, fit the SoR profile at each step, and are thus very time consuming. Brute force is time consuming as well. To reconstruct SoR in real-time, we propose a method to make the iterative step faster but still accurate. It is inspired by the method of Mara et al. [5] that uses parallel planes to cut a SoR. However, [5] uses a 2-dimensional brute force search, while our method finds the axis by solving a 1-dimensional objective function. The same cutting operation (cf. Section 4.2) is repeated near a hundred times in [5] but is only required at most 5 times in our method.

3. Surface of revolution axis estimation

The procedure of reconstructing a SoR in our method is summarized as follows:

1. Estimate initial rotation axis l_0 using Pottmann's method [8] (denoted as *P-method* afterwards)
2. Improve rotation axis estimation (cf. Algorithm 1)
   (a) Use parallel planes to cut the SoR along l_0
   (b) For each plane: find the center of the intersecting curve (cf. Figure 1)
   (c) Fit a line to all centers
   (d) Rotate the cutting planes and goto step 2a if the line is not perpendicular to the planes

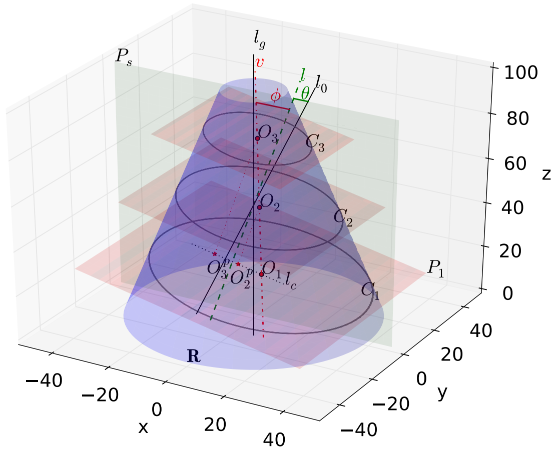

Figure 1. Symmetry of SoR and planes

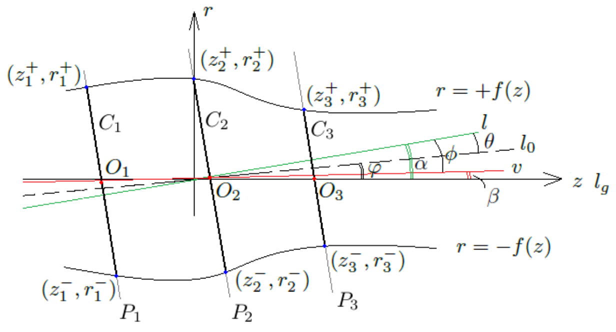

Figure 2. Sketch of P_s: All useful quantities are inside P_s

3. Reconstruct the SoR by finding its profile (cf. Section 4.1)

A fast and accurate iterative step 2 is our main contribution. Usually, the direction of a rotation axis l can be determined by two Euler angles and many existing methods use two dimensional search, such as Mara et al. [5]. We show that the search space can be reduced to one dimension, and the objective function is simple to solve. It makes the algorithm more efficient. Steps 1 and 3 can be seen as pre-process and post-process respectively.

In this section, we first explain the idea of step 2 and then mathematically prove it. The implementation of step 2 and step 3 are then presented in Section 4.

3.1. Basic idea

We first show that, given an initial guess l_0 obtained from the P-method, the search space of the rotation axis direction is unidimensional. In Figure 1, we use a cone **R** to represent a general SoR for the sake of simplicity. The solid line l_g is the ground truth of its rotation axis. Line l_0 represents the initial guess. A plane P_s containing l_g can be defined by the normal $\vec{l_g} \times \vec{l_0}$, where unit vector $\vec{\cdot}$ represents the line's direction. By varying $\vec{l_0}$ inside plane P_s, it is possible to find $\vec{l_g}$, thus the search space is reduced to one dimension. Note that we use a 2D sketch of P_s in Figure 2 to represent Figure 1 because all useful information is contained in P_s.

P_s can be determined without knowing l_g. We first create parallel planes P_j $(j = 1..n)$ with normal $\vec{l_0}$ to cut $\mathbf{R}$, which results in a set of 2D curves C_j. Since C_j are symmetric curves with respect to P_s, their centers O_j should lie on P_s. We then project centers of C_j on an arbitrary plane, e.g. P_1. These projections O_j^p $(O_1^p = O_1)$ will form a line l_c (cf. Figure 1), which is also the intersection between P_s and P_1. By finding $\vec{l_c}$, one can determine P_s.

Next, we derive the objective function of our method. Let us assume that $\vec{l}$ is rotated from $\vec{l_0}$ inside P_s by angle θ (cf. Figures 1 and 2). The set of parallel planes P_j is also rotated so that their normal is $\vec{l}$. A line v is fitted to all points O_j. Intuitively, if $\vec{l} = \vec{l_g}$, all C_j should be circles because of symmetry, and their centers O_j should lie on l_g. It means that $\vec{l} = \vec{l_g} \Rightarrow \vec{v} = \vec{l_g} = \vec{l}$. In the next section, we show that the converse is true if $\mathbf{R}$ is not a sphere. That is to say $\vec{v} = \vec{l} \Leftrightarrow \vec{l} = \vec{l_g}$ for non-spheres. In Figure 2, unit vectors inside P_s can be represented by 2D vectors. By abusing notations, we still use $\vec{\cdot}$ to refer to these 2D vectors, so their cross products are real numbers. Finally, the objective function is:

$$\phi(\theta) = \sin^{-1}(\vec{l} \times \vec{v}) \tag{1}$$

with signed angle $\theta = \sin^{-1}(\vec{l_0} \times \vec{l})$. Our objective is to find θ as the signed angle $\phi(\theta)$ is 0.

Once $\vec{l_g}$ is found, we can use the average of O_j as the point on the axis. This gives us the final rotation axis as:

$$l_g = (\vec{l_g}, \frac{1}{n}\sum_{j=1}^{n} O_j) \tag{2}$$

3.2. Approximately linear objective function

In this section, we prove that the objective function ϕ (equation (1)) is approximately linear. This is the core of our method since it guarantees its fast convergence.

The unknown l_g is chosen to be the z-axis in Figure 2 for easier deduction. The profile of the SoR in this plane can be expressed as $r = \pm f(z)$. Parallel planes intersect with z-axis at z_j. The 2D projection of C_js on P_s can be represented by segments between (z_j^+, r_j^+) and (z_j^-, r_j^-). φ is defined as the offset angle of initial guess $\vec{l_0}$ from $\vec{l_g}$:

$$\varphi = \sin^{-1}(\vec{l_g} \times \vec{l_0}) \tag{3}$$

Please notice that when $\theta = -\varphi$, we have $\vec{l} = \vec{l_g}$ and thus the objective function $\phi(\theta) = \phi(-\varphi) = 0$. To derive the formula of $\phi(\theta)$, two intermediate angles are used:

$$\begin{aligned} \alpha &= \sin^{-1}(\vec{l_g} \times \vec{l}) \\ \beta &= \sin^{-1}(\vec{l_g} \times \vec{v}) \end{aligned} \tag{4}$$

It is easy to derive that:

$$\begin{aligned} \theta &= \alpha - \varphi \\ \phi &= \beta - \alpha \end{aligned} \tag{5}$$

By assuming that α is small, the profile near z_j can be approximated by its first order Taylor expansion:

$$r = \pm f(z) \approx \pm\left(f(z_j) + f'(z_j)(z - z_j)\right) \tag{6}$$

The equation of plane P_j can be expressed as:

$$r = \tan(\frac{\pi}{2} + \alpha)(z - z_j) \tag{7}$$

The intersection points (z_j^+, r_j^+) can be calculated by solving equation (7) and using $r = +f(z)$ from equation (6). Similarly, (z_j^-, r_j^-) can be calculated from (7) and using $r = -f(z)$ from (6). The center point $O_j(z_j^*, r_j^*)$ lies in the middle of the $[(z_j^-, r_j^-), (z_j^+, r_j^+)]$ line segment and thus can be derived as:

$$\begin{aligned} z_j^* &= \frac{1}{2}(z_j^- + z_j^+) = z_j + \frac{f(z_j)f'(z_j)}{\cot^2\alpha - f'^2(z_j)} \\ r_j^* &= \frac{1}{2}(r_j^- + r_j^+) = -\frac{\cot\alpha f(z_j)f'(z_j)}{\cot^2\alpha - f'^2(z_j)} \end{aligned} \tag{8}$$

Since v is an interpolation of all O_j, its slope k in the plane can be expressed as follows:

$$k = \tan\beta = -\frac{\sum_1^n (z_j^*)^2 - n\left(\sum_1^n \frac{z_j^*}{n}\right)^2}{\sum_1^n z_j^* r_j^* - n\left(\sum_1^n \frac{z_j^*}{n}\right)\left(\sum_1^n \frac{r_j^*}{n}\right)} \tan\alpha \tag{9}$$

As α is small, the equation above can be simplified by Taylor expansion and $\tan\alpha \approx \alpha$, by using equation (8):

$$k \approx -\frac{n\sum_1^n z_j f(z_j)f'(z_j) - \left(\sum_1^n z_j\right)\left(\sum_1^n f(z_j)f'(z_j)\right)}{n\sum_1^n z_j^2 - \left(\sum_1^n z_j\right)^2}\alpha \tag{10}$$

When β is small, $k = \tan\beta \approx \beta$. Let $\xi_j = z_j - \frac{1}{n}\sum_1^n z_j$, we have $\sum_1^n \xi_j = 0$. Equation (10) can be simplified as:

$$\beta \approx -\frac{\sum_1^n \xi_j f(\xi_j + \bar{z})f'(\xi_j + \bar{z})}{\sum_1^n \xi_j^2}\alpha = -(A - 1)\alpha \tag{11}$$

where

$$A = \frac{\sum_1^n \xi_j f(\xi_j + \bar{z}) f'(\xi_j + \bar{z})}{\sum_1^n \xi_j^2} + 1 \tag{12}$$

Thus, by using equation (5) and (11), the objective function $\phi(\theta)$ can be written as:

$$\phi(\theta) = \beta - \alpha = -A\alpha = -A(\theta + \varphi) \tag{13}$$

When $\vec{l}$ is rotated to the ground-truth $\vec{l_g}$, then $\theta = -\varphi$ and thus $\phi(\theta) = 0$, which corresponds to our intuitive knowledge. In equation (13), φ is an unknown variable depending on l_0. Although A depends on ξ_j (cf. equation (12)), we can show that when $n \to \infty$:

$$\begin{aligned} A &= \frac{\sum_1^n \xi_j f(\xi_j + \bar{z}) f'(\xi_j + \bar{z}) \Delta\xi}{\sum_1^n \xi_j^2 \Delta\xi} + 1 \\ &\approx \frac{\int_{\xi_{min}}^{\xi_{max}} \xi_j f(\xi_j + \bar{z}) f'(\xi_j + \bar{z}) d\xi}{\int_{\xi_{min}}^{\xi_{max}} \xi_j^2 d\xi} + 1 \end{aligned} \tag{14}$$

It shows that A is a discrete approximation of an integral. Thus although A depends on ξ_j, it does not vary much when n is big. In other words, the objective function $\phi(\theta)$ is approximately linear.

We then show that the solution $\phi(\theta) = 0$ is unique except for spheres and that it corresponds to the rotation axis. From equation (13), we can see that $A = 0 \Rightarrow \phi(\theta) \equiv 0$. By solving $A = 0$ from equation (14), we find the profile function is $f(z) = \pm\sqrt{(C - z^2)}$, where C is an arbitrary constant. This equation is that of a sphere. $\phi \equiv 0$ means a sphere can have its rotation axis in any direction, which is obvious. But for other surfaces, since $A \neq 0$, the solution of $\phi(\theta) = 0$ is uniquely $\theta = -\varphi$ (cf. equation 13). It means that by finding the unique solution of $\phi(\theta) = 0$, one can find its unique rotation axis.

4. Implementation details

3D data of real objects are often obtained by using RGB-D sensors (i.e. Kinect, RealSense). They can have various representations, such as point cloud and Signed Distance Fields (SDF). However, the captured data may be noisy. In this section, we explain the implementation details by considering noisy data. The outline of the algorithm is first introduced in Section 4.1. Then we explain a basic and important operation in Section 4.2 and solve the objective function numerically in Section 4.3.

4.1. Algorithm

The outline of rotation axis estimation is shown in Algorithm 1. It takes the 3D representation (e.g. point cloud, SDF) of a SoR and an initial axis estimation l_0 from [8] as input and returns the rotation axis l_e.

Algorithm 1 Rotation axis estimation

```
Input: 3D data of SoR, initial axis estimation $l_0$
Output: SoR axis $l_e$
Parameter: number of cutting planes $n_1$, $i_{max}$
$i = 0, \theta_0 = 0, valuePair = \emptyset$
while $i < i_{max}$ do
    if $i \neq 0$ then
        $\theta_i = \theta_{i-1} - \phi_{i-1}/A_{i-1}$
        $\vec{l_i} \leftarrow rotate(\vec{l_0}, P_s, \theta_i)$
    end if
    $\mathbf{O}, \mathbf{w} \leftarrow findCircleCenters(l_i, n_1)$ or fail
    if $i = 0$ then
        $P_s, \delta \leftarrow findSymmetricPlane(\mathbf{O})$
    end if
    $\vec{v_i} \leftarrow findCenterLineDirection(\mathbf{O}, \mathbf{w})$
    $\phi_i = \sin^{-1}(\vec{l_i} \times \vec{v_i})$
    $valuePair.append(\theta_i, \phi_i)$
    if $|\phi_i| < \epsilon$ then
        return $\vec{l_e} = \vec{l_i}$
    else
        if $i = 0$ then
            Estimate $A_0$ with $(\mathbf{O}, \mathbf{r})$ from equation (19)
        else
            $A_i \leftarrow calculateSlope(valuePair)$
        end if
    end if
    $i = i + 1$
end while
Find smallest $|\phi_i|$ in $valuePair$ and return $\vec{l_e} = \vec{l_i}$
```

The process of cutting the SoR by parallel planes and finding the centers of intersecting curves is frequently used in our method. Since the angle between the presumed axis l_i and the ground-truth l_g is usually smaller than 10°, we use circle fitting to find the centers O_j. We call this process *findCircleCenters*.

$$\mathbf{O}, \mathbf{w}, \mathbf{R}, \bar{\mathbf{e}} \leftarrow findCircleCenters(l, n) \tag{15}$$

This process takes the presumed axis l and the number of planes n as input and gives n_v valid circles as output, with circle centers $\mathbf{O} = \{O_1, O_2, ..., O_{n_v}\}$, circle radii $\mathbf{R} = \{R_1, R_2, ..., R_{n_v}\}$, weights $\mathbf{w} = \{w_1, w_2, ..., w_{n_v}\}$ and average geometric errors $\bar{\mathbf{e}} = \{\bar{e}_1, \bar{e}_2, ..., \bar{e}_{n_v}\}$. n_v is the number of valid circles, which may be smaller than n. A detailed explanation is provided in Section 4.2.

In Algorithm 1, *findSymmetricPlane* is the process of finding P_s. It takes all valid circle centers $\mathbf{O}$ as input and gives P_s and δ as output. δ represents the incertitude of circle centers detection, which is explained in Section 4.3. *findCenterLineDirection* is the process of finding the direction of v in Figure 2. It takes circle centers $\mathbf{O}$ and their weights $\mathbf{w}$ as input. It fits a line to points $\mathbf{O}$ by using different weights $\mathbf{w}$ and returns $\vec{v}$ as output. *calculateSlope* is the process of finding the slope A of $\phi(\theta)$. It takes as input the previous $i+1$ value pairs $valuePair = \{(\theta_0, \phi_0), (\theta_1, \phi_1), .., (\theta_i, \phi_i)\}$, fits a line to them and returns the slope A of this fitted line.

After finding $\vec{l}_e$, $findCircleCenters(\vec{l}_e, n_2)$ is used on the whole object for reconstruction. n_2 can be larger than n_1 for a finer reconstruction result. A surface's intrinsic coordinate system is constructed, with origin O_e being the average of O_j and the direction of z-axis $\hat{z}_e = \vec{l}_e$. Within this coordinate system, the profile function $f(z_e)$ is fitted by splines, thus finishing the SoR reconstruction.

The global reconstruction error e, is defined as the average of geometrical error $\bar{\mathbf{e}}$ (cf. Section 4.2).

4.2. Plane cutting and circle fitting

We use parallel planes separated by an equal interval in *findCircleCenters*. Since scanned 3D data contain lots of defects, such as incomplete scanning and noise, some planes may contain data with better quality than others. To compensate those limitations, a circle is considered as valid only when its inlier ratio of circle fitting is more than 80%, and each valid circle C_j is associated with a weight w_j according to its fitting result. Additionally, planes near the two ends of a SoR are more influenced by inaccurate axis direction estimation, so we only use the middle 70% region of the surface for axis estimation. The weight w_j for a valid circle is calculated by:

$$w_j = \frac{\Omega_j}{\bar{e}_j} \tag{16}$$

where Ω_j is the angular spans of data points on the fitted circle as shown in Figure. 3, $\bar{e}_j$ is the average *geometric error* of all inliers in the current plane. We define the *geometric error* as the distance between a point and the fitted circle.

When point clouds are used as raw data with resolution (i.e. average inter-point distance) τ, a point is considered inside a plane if its distance to the plane is less than 0.5τ. If a SDF representation (with voxel size τ) is used, a ray casting method can be used to find points inside a specified plane. A point is considered as an inlier if its geometric error is less than τ.

If *findCircleCenters* finds less than 2 valid circles, it means that the target surface is not a true SoR or the data is too noisy to process. The algorithm fails in this case.

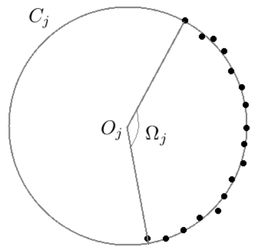

Figure 3. Angular spans of data points on a fitted circle: black points represent data points in plane P_j, C_j is the fitted circle, Ω_j is the angular spans of data points.

4.3. Determining P_s and solving $\phi(\theta)$

The basic idea of calculating P_s has been explained in Section 3.1. Assuming that the incertitude of center detection is δ, the center projections O_j^p in Figure 1 will lie in a thick line of width 2δ. By using Principle Component Analysis (PCA) on these center projections, the major eigenvector can be regarded as good approximation of $\vec{l}_c$ (cf. Figure 1) while the minor eigenvalue λ_{min} is the sum of squared distance between detected circle centers and P_s. Thus, δ can be approximated by:

$$\delta \approx \sqrt{\lambda_{min}/n_v} \tag{17}$$

where n_v is the number of valid circles.

Although $\phi(\theta)$ has a simple form, but since the scanned 3D data is noisy and $\phi(\theta)$ is not exactly linear (cf. Section 3.2), we still have to rely on an iterative approach to solve $\phi(\theta) = 0$. Recall Newton's method:

$$\theta_{i+1} = \theta_i - \frac{\phi(\theta_i)}{\phi'(\theta_i)} = \theta_i - \frac{\phi(\theta_i)}{A_i} \tag{18}$$

For the *i*-th step, θ_i is known and $\phi(\theta_i)$ is evaluated. Owing to its linearity, $\phi'(\theta_i) = A_i$ can be approximated by fitting previous $i+1$ data by using *calculateSlope*. For the first step $i = 0, \theta_0 = 0$, we solve the primitive function of equation (12) to find an approximation of A_0:

$$\begin{aligned} A_0 &\approx 3\frac{f^2(z_{max}) + f^2(z_{min}) - \frac{2}{n_v}\sum_1^{n_v} f^2(z_j)}{(z_{max} - z_{min})^2} + 1 \\ &\approx 3\frac{R_{max}^2 + R_{min}^2 - \frac{2}{n_v}\sum_1^{n_v} R_j^2}{D^2} + 1 \end{aligned} \tag{19}$$

where D is the distance between the two most distant planes. In the second approximation, $f^2(z_j) \approx R_j^2$ is used. It is because that when α is small, P_j is almost perpendicular to l_g, thus we can use the the radius of fitted circle R_j to approximate $f(z_j)$ (cf. Figure 2).

The iteration can be terminated when $|\phi(\theta)|$ is small enough, e.g. $10^{-3}rad$. However, since circle center detection cannot be more precise than δ (cf. equation (17)), the estimation of rotation axis can not be more precise than $\tan^{-1}(\delta/D)$. Since $\delta \ll D$, δ/D can be used instead of $\tan^{-1}(\delta/D)$. Therefore, the iteration is terminated when $|\phi(\theta_i)|$ is less than $\epsilon = \max(\delta/D, 10^{-3})$. Practically, we found that this method usually converges in 2-3 steps. But if a point cloud is used, $\phi(\theta_i)$ may not be continuous and $|\phi(\theta_i)| < \epsilon$ may never be achieved. This leads the process to oscillate around $\phi(\theta_i) = 0$. In order to avoid this, we set the maximum iteration number $i_{max} = 5$. If i_{max} is reached but $|\phi(\theta_{i_{max}})| \geq \epsilon$, the l_i which gives smallest $|\phi|$ will be chosen as l_e.

5. Real-time SoR reconstruction

We propose a workflow where the reconstruction of SoR is carried out in a depth SLAM framework. It uses real-time segmentation result from the work of [13], where surfel-based representation is used. A surfel is a representation of a small surface patch at position $\vec{p}(x, y, z)$ in 3D space, with its normal direction $\vec{n}$, its confidence radius r and at most one segment label. Surfels which have the same segment label make up a segment. Usually, each segment represents a single object in the scene.

Although the workflow can deal with SoRs, planes and spheres, we only focus on SoRs to highlight our contribution. A method to distinguish different types of surfaces is presented in Section 5.1. It allows us to find segments which represent SoR. Detailed algorithm to reconstruct SoR with [13] at each frame is presented in Section 5.2.

5.1. Segment classification

We use a two-step method to classify each segment. A surfel at position $\vec{p}$ having a normal $\vec{n}$ is denoted by $(\vec{p}, \vec{n})$. Let $\vec{r} = (\vec{p} \times \vec{n}, \vec{n})$. For all N points belonging to one segment, the following two matrices can be calculated:

$$L = \frac{1}{N}\sum_{j=1}^{N} \vec{n}_j \vec{n}_j^T \text{ and } M = \frac{1}{N}\sum_{j=1}^{N} \vec{r}_j \vec{r}_j^T \tag{20}$$

Let $\{\lambda_{L,i} : i = 1, 2, 3\}$ be the eigenvalues of L and f_t be the number of small eigenvalues in $\{\lambda_{L,i}\}$. According to [4], the segment can be translated in f_t directions without changing the distance function values nearby. We can say that the segment has f_t translation free directions.

f_t	2	1		
f_r	-	> 2	1	0
Type	Plane	Sphere	SoR	Complex

Table 1. Different surface types according to f_t and f_r

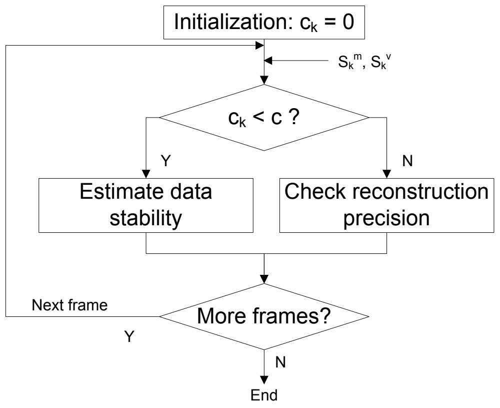

Figure 4. Workflow of real-time surface reconstruction for SoR.

Let $\{\lambda_{M,i} : i = 1, 2, 3\}$ be the eigenvalues of problem $M\vec{x} = \lambda D\vec{x}$, where $D = diag(1, 1, 1, 0, 0, 0)$, and f_r be the number of small eigenvalues in $\{\lambda_{M,i}\}$. According to [8], the segment has f_r rotation free directions. With both f_t and f_r, surfaces can be classified into 4 groups as shown in Table 1. Practically, an eigenvalue is considered to be small when following conditions are true:

$$\lambda_{N,i} < 0.05 \text{ and } \lambda_{M,i} < 0.5\tau^2 \tag{21}$$

5.2. Workflow

Let us take a SoR with segment label k (denoted as $\mathcal{B}_k$) as an example to explain the workflow. All other segments follow the same procedure.

Concerning $\mathcal{B}_k$, two sets of surfels with segment label k, i.e. S_k^m and S_k^v, can be obtained from [13] at frame t. S_k^m contains all surfels belonging to $\mathcal{B}_k$ in the model obtained from the SLAM procedure. S_k^v is the visible subset of S_k^m at frame t. When $\mathcal{B}_k$ has only appeared in front of the camera for a few frames, or when it is occluded or far from the camera, data acquired from $\mathcal{B}_k$ may be not stable. With more information coming from new frames, the data can be more stable and the reconstruction can be more reliable. We use a confidence label c_k to represent the stability of data points acquired from $\mathcal{B}_k$.

The workflow to reconstruct $\mathcal{B}_k$ is shown in Figure 4. When c_k is less than a threshold c, we check the stability by estimating the variation of S_k^v's rotation axis. Although different frames contain different sets of visible surfels S_k^v, their rotation axis should be similar. Thus, c_k is incremented by 1 if the angle between two estimations of axis is smaller than a threshold $\tilde{\gamma}$ (set to 10° in our experiment). When $c_k = c$ (set to 5 in our experiment), the data is considered to be stable. The axis of $\mathcal{B}_k$ is estimated and $\mathcal{B}_k$ is then reconstructed for the first time by applying our methods (cf. Sections 3-4) on global model S_k^m, with τ_k (cf. Section 4.2) being the average of double radius of surfels in S_k^m. More details are shown in Algorithm 2.

Algorithm 2 Estimate the stability of data

```
Input: $S_k^v$, $S_k^m$ at frame $t$
Calculate $\vec{l}_k$ from $S_k^v$ using P-method [8]
if $c_k = 0$ then
    Set $\vec{l}_k^r = \vec{l}_k$
else if $\cos^{-1}(\vec{l}_k \cdot \vec{l}_k^r) < \tilde{\gamma}$ then
    $c_k = c_k + 1$
    if $c_k \geq c$ then
        Estimate initial $\vec{l}_{k,0}^m$ from $S_k^m$ with P-method
        Improve axis estimation & reconstruct $\mathcal{B}_k$ from $S_k^m$
    end if
else
    $c_k = c_k - 1$
end if
```

In later frames, we continue to check the precision of this reconstruction. If the reconstruction is precise, the reconstructed model should fit well all visible surfels S_k^v in new frames. Otherwise, the axis of $\mathcal{B}_k$ should be re-estimated and $\mathcal{B}_k$ should be reconstructed by using the newest global model S_k^m. Details on checking reconstruction precision are shown in Algorithm 3. $enum_k$ indicates the number of surfels whose modeling error is larger than $\tilde{e}$. Empirically, $\tilde{e}$ is set to $1.5e_k$, with e_k being the global reconstruction error of $\mathcal{B}_k$ (cf. Section 4.1). When $enum_k$ is larger than a threshold $\tilde{N}$, the reconstruction $\mathcal{B}_k$ should be improved. Since consecutive frames have large overlapping area, the same surfel may be frequently recalculated. We found that $\tilde{N} = 30\%|S_k^m|$ works well in our experiment.

The *calcModelingError* process estimates the modeling error e_i on surfel s_i. e_i is computed as the distance between s_i and its approximate nearest point on the reconstructed surface. The approximate nearest point is constrained so that itself and s_i are lying on the same plane which is perpendicular to the rotation axis.

Algorithm 3 Check reconstruction precision

```
Input: $S_k^v$, $S_k^m$ at frame $t$
for each $s_i \in S_k^v$ do
    $e_i \leftarrow calcModelingError(s_i, S_k^m)$
    $enum_k = enum_k + 1$ if $e_i > \tilde{e}$
end for
if $enum_k > \tilde{N}$ then
    Improve axis estimation and reconstruct $\mathcal{B}_k$ from $S_k^m$
    Set $enum_k = 0$
end if
```

6. Results

In this section, we first evaluate the method for estimating the rotation axis, before using real data to evaluate reconstruction results and the efficiency of our method. $n_1 = 10$ is used for all experiments.

6.1. Synthetic study

We use synthetic truncated cones with grid size $\tau = 0.2$ to investigate the feasibility of the method. A truncated cone's properties are shown in Figure 5.

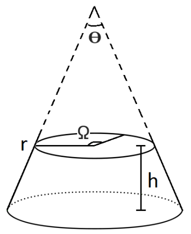

Figure 5. Synthetic truncated cones: with $r = 20, h = 10$, angular span $\Omega = 120°$. Θ is the opening angle.

To simulate noise, $\delta_r \sim N(0, 0.5\tau)$ is added to the three coordinates of each point, and $\delta_n \sim N(0, \eta)$ is added to each normal component (normalized afterwards). At last, the synthetic cone is rotated and moved by a random rigid transformation. We compare the results of our method with the one from [8] (denoted as *P-method*), since we are interested in efficient methods for real-time application.

The objective functions $\phi(\theta)$ of different objects are shown in Figure 6. For the synthetic cone, we use $\Theta = 100°$, $\eta = 0.03$. The Cup and Can are taken from Dataset 2 of Section 6.2 (cf. Figure 10). We can see that these $\phi(\theta)$s are almost linear, when θ is near the ground truth.

Figures 7 shows the rotation axis estimation error $\cos^{-1}(\vec{l}_g, \vec{l})$ on synthetic data. Our method gives better results, especially for large noise and opening angles. Moreover, the error of our method is less than $0.5°$ in most cases.

6.2. Real data

For real-time reconstruction, $n_2 = 10$ is chosen. Two datasets were used for evaluating our online reconstruction algorithm. We use downsampled image with resolution

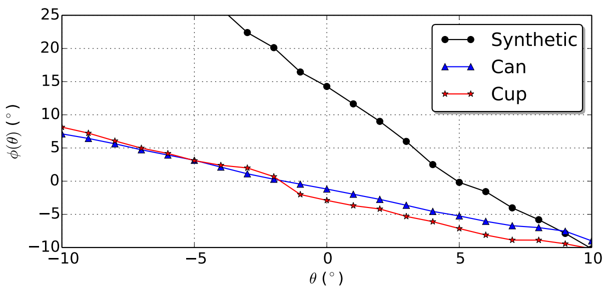

Figure 6. $\phi(\theta)$ for different objects: synthetic cone, Cup and Can. They are approximately linear.

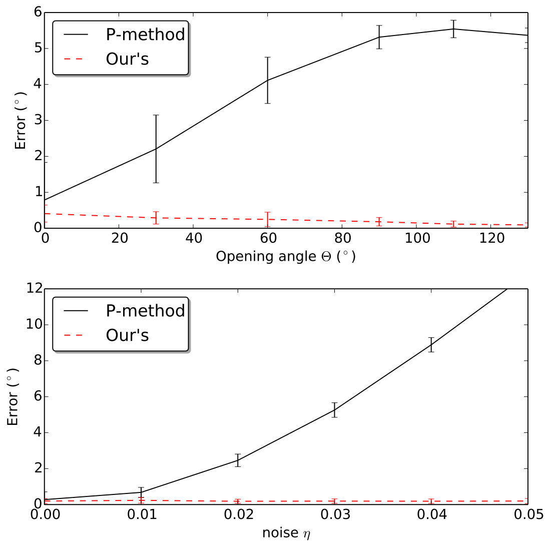

Figure 7. Synthetic result with different opening angles Θ (top), and with different amount of noise η (bottom).

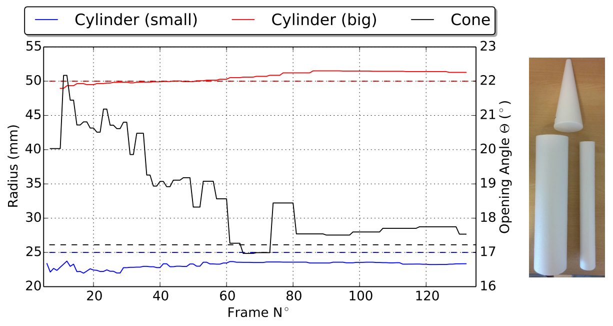

Figure 8. Dataset 1: Measuring objects of known dimension.

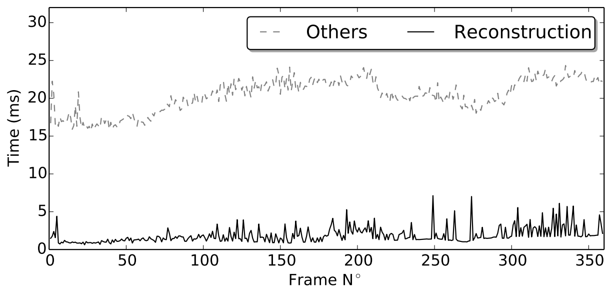

Figure 9. Dataset 2: Time consumption.

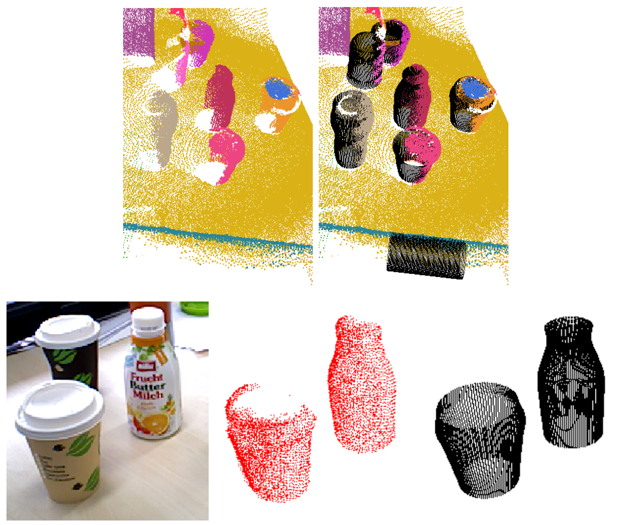

Figure 10. Dataset 2. Top: Original segmented point cloud (left) and reconstruction result (right). Bottom: visual comparison between reconstruction result, original point cloud and RGB image. (Cup and Can).

160×120 to let the whole workflow run in real-time on the CPU as indicated in [13].

We use two cylinders and a cone with known size in dataset 1. The cylinders' radii and the cone's opening angle are used to measure the precision of the algorithm. Results are shown in Figure 8. It can be seen that the accuracy of radius is about 1-2mm, while the accuracy of opening angle has a big variation at the beginning but gradually converges near the ground truth in about 80 frames.

Dataset 2 is an online dataset provided by Tateno[1]. It contains diverse daily objects lying on a desktop. We draw the time consumption for each frame in Figure 9. The solid line represents the reconstruction time, with an average of 1.87ms per frame, while the dashed line represents the SLAM+segmentation time, with an average of 20.40ms per frame. These results are produced on a laptop equipped with Intel Core i7-4510U CPU @2.00GHz and 8GB RAM.

The final reconstructed object along with the segmentation map is presented in Figure 10. The reconstructed objects correctly match the original data and are visually similar to the original objects in the RGB image.

7. Conclusion

We have developed a fast and accurate method to estimate the rotation axis of revolution surfaces as well as a real-time online geometric reconstruction workflow. Experiments show that the proposed method and workflow work well for real world objects, achieving about 1-2mm accuracy for radius estimation and $< 1^\circ$ for cone's opening angle estimation.

With these reconstructed geometries, point clouds can be simplified and other interesting work may be performed. For example, SLAM wide-baseline re-localization problems could be solved by 3D rotation axis matching. Other future work may also include improving segmentation results by using reconstructed geometries; establishing spatial relationships between different geometries, etc.

[1]http://campar.in.tum.de/Chair/ProjectInSeg

References

[1] P. J. Besl and H. D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, Feb 1992.

[2] B. Drost and S. Ilic. Local Hough Transform for 3D Primitive Detection. In *3D Vision (3DV), 2015 International Conference on*, pages 398–406. IEEE, 2015.

[3] D. Han, D. B. Cooper, and H.-s. Hahn. Fast axis estimation from a segment of rotationally symmetric object. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 1154–1161. IEEE, 2012.

[4] C. Lou, L. Zhu, and H. Ding. Identification and reconstruction of surfaces based on distance function. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 223(8):981–994, 2009.

[5] H. Mara and R. Sablatnig. Orientation of Fragments of Rotationally Symmetrical 3D-Shapes for Archaeological Documentation. In *3D Data Processing, Visualization, and Transmission, Third International Symposium on*, pages 1064–1071. IEEE, 2006.

[6] T. Nguyen, G. Reitmayr, and D. Schmalstieg. Structural Modeling from Depth Images. *IEEE Transactions on Visualization and Computer Graphics*, 21(11):1230–1240, 2015.

[7] G. Pavlakos and K. Daniilidis. Reconstruction of 3D Pose for Surfaces of Revolution from Range Data. In *3D Vision (3DV), 2015 International Conference on*, pages 648–656. IEEE, 2015.

[8] H. Pottmann, M. Peternell, and B. Ravani. An introduction to line geometry with applications. *Computer-Aided Design*, 31(1):3–16, 1999.

[9] R. Qiu, Q.-Y. Zhou, and U. Neumann. Pipe-Run Extraction and Reconstruction from Point Clouds. In *13th European Conference on Computer Vision*, ECCV, pages 17–30. Springer, 2014.

[10] R. Schnabel, R. Wahl, and R. Klein. Efficient RANSAC for Point-Cloud Shape Detection. *Computer Graphics Forum*, 26(2):214–226, June 2007.

[11] M. Song and D. Huber. Automatic Recovery of Networks of Thin Structures. In *3D Vision (3DV), 2015 International Conference on*, pages 37–45. IEEE, 2015.

[12] Y. Taguchi and S. Ramalingam. Method for fitting primitive shapes to 3d point clouds using distance fields, Dec. 2015. US Patent 9,208,609.

[13] K. Tateno, F. Tombari, and N. Navab. Real-time and scalable incremental segmentation on dense SLAM. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 4465–4472. IEEE, 2015.

[14] K. Tateno, F. Tombari, and N. Navab. When 2.5D is not enough: Simultaneous Reconstruction, Segmentation and Recognition on dense SLAM. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2295–2302. IEEE, 2016.

[15] A. Willis, X. Orriols, and D. B. Cooper. Accurately Estimating Sherd 3D Surface Geometry with Application to Pot Reconstruction. In *Computer Vision and Pattern Recognition Workshop, 2003. CVPRW'03. Conference on*, volume 1, pages 5–5. IEEE, 2003.