Probabilistic Collision Detection Between Noisy Point Clouds Using Robust Classification

Jia Pan, Sachin Chitta and Dinesh Manocha

Abstract We present a new collision detection algorithm to perform contact computations between noisy point cloud data. Our approach takes into account the uncertainty that arises due to discretization error and noise, and formulates collision checking as a two-class classification problem. We use techniques from machine learning to compute the collision probability for each point in the input data and accelerate the computation using stochastic traversal of bounding volume hierarchies. We highlight the performance of our algorithm on point clouds captured using PR2 sensors as well as synthetic data sets, and show that our approach can provide a fast and robust solution for handling uncertainty in contact computations.

1 Introduction

The problems of collision detection and proximity computation are widely studied in different areas, including robotics, physically-based modeling, haptics and virtual environments. In particular, reliable and fast collision detection algorithms are required for robot motion planning, grasping and dynamics simulation to enforce the non-penetration constraints with the environment.

Most of the prior work on collision detection assumes an exact geometric description of the objects in the scene, typically represented as a polygon mesh. However, these methods may not work well for robots operating in real-world environments, where only partial observations of the environment are possible based on robot sensors. For example, inaccurate motor control makes a robot

J. Pan (✉) · D. Manocha
The Department of Computer Science, UNC, Chapel Hill, USA
e-mail: panj@cs.unc.edu

D. Manocha
e-mail: dm@cs.unc.edu

S. Chitta
Willow Garage Inc., Menlo Park CA 94025, USA
e-mail: sachinc@willowgarage.com

deviate from its exact configuration and the sensors tend to add noise to the environment measurements. Current robot sensors including cameras and LIDAR and new devices such as Kinect can easily generate detailed point cloud data of real-world environments. However, it is hard to directly use prior collision detection algorithms which perform a boolean query and compute a yes/no answer. Moreover, exact collision checking may not be suitable in terms of handling uncertainty in perception and control, which also causes uncertainty in collision results. For many robotics applications, such as grasping or motion planning, we need to reduce the risk of physical contacts between the robot and the environment that may result in damages. Hence, we need to develop methods that tend to minimize the probability of collisions. Our point cloud collision and proximity algorithm can also be used to improve many methods' feasibility and robustness in real world. For example, algorithms in tactile manipulation usually require an exact or approximated mesh model of manipulated objects (e.g., [21]) and our method can extend them to directly handle the point clouds provided by sensors (See Fig. 1).

Main Results: In this paper, we present a probabilistic collision detection algorithm that can handle environments with uncertainty. Our approach can handle noisy or inexact point data representations that are gathered using sensors. In order to handle point cloud data with noise, we reformulate the collision detection problem as a two-class classification problem, where points of different objects belong to different classes. The collision probability is directly related to the separability of the corresponding two-class problem, which can be elegantly and efficiently solved using support vector machines (SVMs). We accelerate the computation using bounding volume hierarchies and perform a stochastic traversal of the hierarchies that takes into account noise and uncertainty. These hierarchies are updated for dynamic scenes or when the robot head or the gripper moves. Our probabilistic collision algorithm also estimates the contact points and contact normals. We test
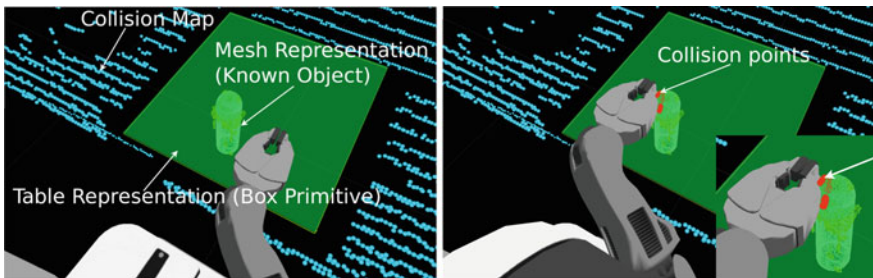

Fig. 1 A visual representation of the collision information generated by the sensors on the PR2 robot. (*Left*) The environment includes the points in a collision map (in *light blue*), mesh representations for known objects detected through visual sensing (*green cylindrical object* on table), and an exact geometric representation of the table surface (*green flat surface*). A detailed mesh model for the robot is also seen in the picture. (*Right*) A representation of the collision points (shown by *red spheres*) between the gripper and the object on the table

our algorithm on point clouds generated from PR2 sensors and synthetic data sets. Our method can provide robust results for probabilistic collision detection and its run-time performance is similar to that of hierarchy-based collision detection algorithms for triangle meshes (e.g., 500–1000 ms for 10 K points on a single CPU core).

The rest of the paper is organized as follows. We survey related work in Sect. 2. We introduce our notation and give an overview of the approach in Sect. 3. Section 4 shows how probabilistic collision detection computation is reduced to robust classification. We highlight the performance of our algorithm on different benchmarks in Sect. 5.

2 Previous Work

The field of probabilistic robotics provides a mathematical framework to handle the uncertainty that exists in the physical world [30]. It deals with representing uncertainty explicitly using the calculus of probability distribution and obtains robust control choices relative to the uncertainty in the robot system. Probabilistic robotics can handle perception uncertainty (or environment uncertainty) due to sensor and action errors. However, previous approaches tend to use simple methods to model environment uncertainty, such as feature-based methods or occupancy grid based methods [30]. These models can only provide a rough description of the environment while many robot actions (e.g., grasping) require more detailed information for robust computation.

2.1 Uncertainty of Point Cloud Data

Raw point cloud data obtained from sensor data can have a high degree of uncertainty, which results mainly from discretization error and noise. As a result, it is difficult to obtain robust estimation of high-order features like surface normals. This causes difficulty for many applications that require precise estimates of normal vectors at the boundary, such as grasping.

Many approaches consider uncertainty of point clouds implicitly. For example, [25, 28] encode surface uncertainty as a parameter tolerance for learning algorithms as they apply geometric operations (e.g., reconstruction) on the point clouds. However, without an explicit model of uncertainty, we can only consider a single uncertainty formulation for the overall surface, but may not be able to model varying uncertainty at different parts of the surface for local control.

There is recent work on explicitly modeling the uncertainty of point cloud data for different applications. Bae et al. [1] present a closed-form expression for the positional uncertainty of point clouds. Pauly et al. [20] propose two methods, confidence map and likelihood map, to analyze shape uncertainty in point clouds

for resampling and reconstruction applications. Jenke et al. [11] describe a Bayesian model for point cloud uncertainty for surface reconstruction.

2.2 *Collision Detection*

Prior collision detection methods mainly focus on performing efficient and accurate contact computations between objects represented by triangulated primitives [17].

In terms of collision checking with point clouds, there are several simple methods. For example, we can first reconstruct triangle meshes from point clouds and then perform exact collision checking between the reconstructed surfaces. However, this approach suffers from inefficiency (>10 s for 10 K points) and robustness issues that arise in terms of using reconstruction algorithms (e.g., reconstruction quality, sensitiveness to parameter and noise, etc.). We can also simply expand every point as a sphere with suitable radius and approximate the object as a union of spheres [10] for collision checking. The main difficulty is in terms of automatically choosing different sphere radii for different points. Other direct collision checking methods for point cloud data are based on using bounding volume hierarchies [13, 27] and reconstructing implicit functions at the leaf nodes, which are prone to robustness issues. Minkowski sums of point clouds have also been used for collision queries [16]. Sucan et al. [29] describe a collision map data structure, which uses axis aligned cubes to model the point cloud and to perform collisions with a robot. Some applications, including virtual reality and haptics, need real-time collision checking, and use probabilistic criteria based on minimum distance computation between the point sets [15]. However, these methods do not take into account point cloud data's inherent shape uncertainty that arises from discretization or sampling [20].

There has been relatively little work in terms of handling uncertainty in collision detection. A special type of collision uncertainty is discussed in [7], which projects objects onto different image planes to perform collision culling using GPU-based computation. Guibas et al. [8] propose a method to compute the collision probability between 2D objects composed of line segments in a 2D environment with uncertainty. In order to estimate the collision uncertainty, this method models the endpoints of a line segment as probability distributions with a rectangular support region. Missiuro et al. [18] also try to model uncertainty in probabilistic roadmaps by using the collision probability of a configuration to bias the sampling process for roadmap computation.

3 Overview

In this section we introduce the notation used in the rest of the paper and give an overview of our approach.

The main pipeline of our system consists of three steps: (1) Obtain raw data from sensors and filter the point clouds to remove points on the robot and reduce the shadow effect [29]; (2) Compute the separating surface between two point clouds by estimating the noise from sensor parameters (Sects. 4.1–4.3); (3) Estimate the collision probability for each point and the overall collision probability between two point clouds (Sect. 4.4). Moreover, we use bounding volume hierarchies to accelerate the computation and recompute the hierarchies for dynamic environments (Sect. 4.5).

The inputs to our collision detection algorithm are the point clouds. In some cases, we need to perform the collision query between two different point clouds or between a point cloud and a polygonal object (e.g., when the mesh representation of a robot hand or gripper is available). We first present our approach for two different point clouds, and later show how it can be applied to a point cloud and a polygonal object.

Let the two point clouds be denoted as C_1 and C_2. We assume that each point cloud C is obtained from sensors and is a partial and noisy representation of the underlying exact surface S. There are two kinds of errors introduced in the generation of point clouds: *discretization errors* and *position errors* or *noise uncertainty*. Intuitively, the discretization error refers to how these point samples are distributed on the boundary of the surface and the position error measures the imprecision in the coordinates of each point. Formally, we assume C is generated from S according to the following process: first a series of n sample points $\mathbf{x}_i'$ is generated according to some sampling process and we use the symbol $p(\mathbf{x}_i'|S)$ to represent the distribution of coordinates for a random point $\mathbf{x}_i'$, i.e., it models the *discretization error*. Next, $\mathbf{x}_i$ is generated from $\mathbf{x}_i'$ according to some noise distribution $p(\mathbf{x}_i|\mathbf{x}_i'; \Sigma_i)$, i.e., it models the *position error*. Generally $p(\mathbf{x}_i'|S)$ is not given, but we can estimate it based on the observed point-cloud data with some assumptions about surface smoothness and sampling density. The symbol Σ_i is used to model point cloud's uncertainty due to noise, and is typically computed based on the sensor characteristics. For example, Σ_i may measure the level of noise that is a combination of sensing noise, motion uncertainty and deformation error. Then the overall uncertainty of a point $\mathbf{x}_i$ can be modeled as

$$\mathbf{x}_i|S \sim p(\mathbf{x}_i|S) = \int p(\mathbf{x}_i'|S)p(\mathbf{x}_i|\mathbf{x}_i'; \Sigma_i)d\mathbf{x}_i'. \tag{1}$$

In this formulation, we have an implicit assumption that the sensor is able to capture the features of the underlying surface. For example, more sample points $\mathbf{x}_i'$ are generated near the sharp features so that we can reconstruct the necessary features of the original model.

The output of the collision detection algorithm is a probability $\mathbb{P}_{C_1,C_2}$ that estimates whether two point clouds C_1 and C_2 are in-collision.

3.1 Separating Surface

Given a point cloud, we can possibly reconstruct a triangulated surface representation using Bayesian optimization. That is, the underlying surface should be the one with the maximum probability:

$$\hat{S} = \underset{S}{\operatorname{argmax}} \, p(S|\{\mathbf{x}_i\}_{i=1}^n) = \underset{S}{\operatorname{argmax}} \, p(S) \prod_i p(\mathbf{x}_i|S). \qquad (2)$$

Next, we can perform collision checking based on reconstructed models. However, reconstruction process is only an estimation and the collision computation based reconstruction can be rather inaccurate. Our formulation is based on the theory of convex sets: two convex sets are non-intersecting if there exists an oriented *separating plane P* so that one set is completely in the positive (open) halfspace P^+ and the other completely in the negative (open) halfspace P^- [19]. For non-convex sets, we extend the concept of separating plane to the *separating surface*: two sets are non-intersecting (or *separable*) if and only if there exists a separating surface P between them. Previous work in collision detection [19, 22] is limited to the special case when P is composed of multiple planes.

We extend the idea of separating surfaces to handle point clouds. Given two point clouds $C_1 = \{\mathbf{x}_i^1\}_{i=1}^{n_1}$ and $C_2 = \{\mathbf{x}_i^2\}_{i=1}^{n_2}$ with n_1 and n_2 elements, respectively, a separating surface P is a surface that can separate the two sets completely with C_1 in P^+ and C_2 in P^-. In this case, P^+ and P^- represent a partition of the space $\mathscr{R}^3$ into two parts. Notice that here P should not be an arbitrary surface, i.e., it should not be a very complex function in terms of acting as a valid separating surface. Otherwise, even if P can completely separate the point clouds, it may not be able to separate the underlying surfaces. Such a problem is called *overfitting* in machine learning literature, i.e., the statistical model biases too much on the observed data and may not be able to predict the underlying model correctly. In order to avoid overfitting, we need to assume *regularity conditions* for P, which intuitively impose suitable smoothness constraints on the separating surface. For example, we represent P as a parameterized implicit surface $\{\mathbf{x} : f(\mathbf{x}; \theta) = 0\}$ with θ as its parameters. In this case, the regularity condition can limit the value of $f'(\mathbf{x}; \theta)$. Moreover, P^+ and P^- can be represented as $\{\mathbf{x} : f(\mathbf{x}; \theta) > 0\}$ and $\{\mathbf{x} : f(\mathbf{x}; \theta) < 0\}$, respectively. As a result, collision detection problem is reduced to finding the separating surface, i.e., deciding the parameter set θ, that can separate C_1 and C_2.

There is one major difference between point clouds and convex/non-convex sets.

In particular, for point cloud data, the existence of a separating surface is *not* a necessary or sufficient condition for non-intersection between the two sets. If two point clouds are noise-free and separable, their underlying surfaces may still be collision-free or in-collision, as shown in Fig. 2a, b. This is due to the discretization error from point-cloud sampling. The issue becomes more complicated when point clouds have position errors, as shown in Fig. 2c, d. This property of point cloud sets makes it difficult to perform exact collision checking, but is suitable for statistical
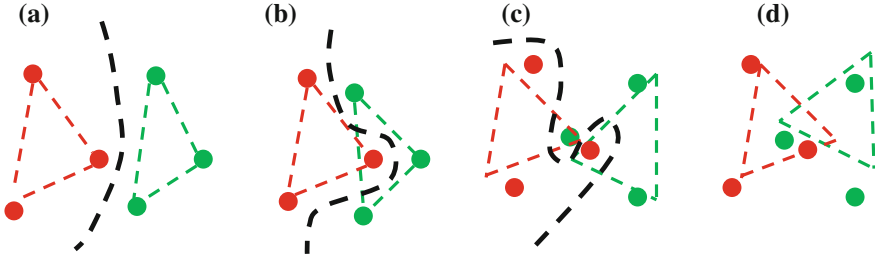
Fig. 2 Separating surface for point cloud sets. Point clouds in (**a**) and (**b**) are noise-free and are separable. However, due to discretization uncertainty, the underlying surfaces can be collision-free (**a**) or in-collision (**b**). Point clouds in (**c**) and (**d**) have some noise and may not be separable. And the underlying surfaces can be collision-free (**c**) or in-collision (**d**). Notice that we require suitable regularity or smoothness on the separating surface to avoid overfitting. For example, the separating surface provided in (**c**) has too large curvature and therefore is not valid. It in fact does not provide a good estimation for how to separate the underlying clouds. Collision result based on reconstructed meshes may not be reliable in all four cases due to discretization error (**a**, **b**) or position noise (**c**, **d**) or unsuitable parameters

learning approaches like SVM [3]. As a result, the probabilistic collision detection problem can be reduced to computing the optimal separating surface that minimizes the *separating error* for underlying surfaces: i.e., find θ that minimizes

$$\int_{\mathbf{x} \in S_1} 1_{\{\mathbf{x} \in P(\theta)^-\}} d\mathbf{x} + \int_{\mathbf{x} \in S_2} 1_{\{\mathbf{x} \in P(\theta)^+\}} d\mathbf{x}, \tag{3}$$

where S_1 and S_2 are the underlying surfaces for point clouds C_1 and C_2, respectively.

3.2 Probabilistic Model for Point Cloud Collision

We now present the probabilistic model for point cloud collision checking to compute the optimal separating surface. We rewrite $\mathbf{x}_i^l$ with $l \in \{1, 2\}$ as $(\mathbf{x}_i, c_i)$, where $\mathbf{x}_i = \mathbf{x}_i^l$ and $c_i = (-1)^{l+1} \in \{-1, 1\}$ denotes which object the point $\mathbf{x}_i$ belongs to. As a result, we have $n_1 + n_2$ elements in $\{(\mathbf{x}_i, c_i)\}$. As discussed in Sect. 3.1, collision checking between two point sets reduces to finding an optimal separating surface P. In machine learning terminology, this corresponds to finding an optimal classifier that can minimize the expected risk on the classification problem whose data is drawn from $\{\mathbf{x} : \mathbf{x} \in S_1 \bigcup S_2\}$ and its *training set* is $\{(\mathbf{x}_i, c_i)\}$. As a result, the collision detection problem is reduced to a machine learning problem. However, unlike typical machine learning algorithms which only deal with cases where $(\mathbf{x}_i, c_i)$ are specified exactly, we also need to take into account the noise in $\mathbf{x}_i$. Our solution

is based on the maximum-likelihood (ML) scheme, i.e., the optimal surface should maximize the probability on the observed inputs $\{(\mathbf{x}_i, c_i)\}$.

Similar to Eq. (1), the joint probability for $(\mathbf{x}_i, c_i)$ can be expressed as

$$p(\mathbf{x}_i, c_i) = \int p(\mathbf{x}_i', c_i; \theta)p(\mathbf{x}_i|\mathbf{x}_i'; \Sigma_i)d\mathbf{x}_i'. \tag{4}$$

Here θ is the parameter set used to represent the separating surface P. For example, P is $\{\mathbf{x} : \mathbf{w}^T \mathbf{x} + b = 0\}$ if P is a plane and $\theta = \{\mathbf{w}, b\}$. Or P is $\{\mathbf{x} : \mathbf{w}^T \Phi (\mathbf{x}) + b = 0\}$ if P is a hyper-plane in some high-dimensional inner product space $\mathscr{H}$ and Φ is the mapping $\Phi : \mathscr{R}^3 \mapsto \mathscr{H}$. The unknown surface parameter θ can be estimated from the point cloud data using ML:

$$\theta^* = \operatorname*{argmax}_{\theta} \sum_{\theta} \ln \int p(\mathbf{x}_i', c_i; \theta)p(\mathbf{x}_i|\mathbf{x}_i'; \Sigma_i)d\mathbf{x}_i' \tag{5}$$

In practice, the integration over the unknown underlying surface sample $\mathbf{x}_i'$ makes it hard to compute the surface parameter. As a result, we consider an alternative form that is computationally more efficient. Specifically, we use an approximation to Eq. (5) based on a widely used heuristic for mixture estimation: we simply regard $\mathbf{x}_i'$ as a parameter of the model instead of a random variable. Then Eq. (5) reduces to:

$$\theta^* = \operatorname*{argmax}_{\theta} \sum_{i} \ln \sup_{\mathbf{x}_i'} p(\mathbf{x}_i', c_i; \theta)p(\mathbf{x}_i|\mathbf{x}_i'; \Sigma_i). \tag{6}$$

We present an algorithm to solve Eq. (6) in Sect. 4.

4 Probabilistic Collision Checking Between Point Clouds

In this section, we present our probabilistic algorithm for collision checking between point clouds using two-class classification. This reduces to computing the optimal separating surface that minimizes the function in Eq. (6).

4.1 Basic Formulation

For convenience, we first assume that the separating surface is a plane, i.e., $P = \{\mathbf{x} : \mathbf{w}^T \mathbf{x} + b = 0\}$. We also assume that the uncertainty due to noise can be described by a Gaussian distribution. We will relax these assumptions later. Based on these two assumptions, we have

$$p(\mathbf{x}'_i, c_i; \theta) \sim p(\mathbf{x}'_i) \exp(-\frac{(\mathbf{w}^T \mathbf{x}'_i + b - c_i)^2}{\sigma^2}) \quad \text{and}$$
$$p(\mathbf{x}_i | \mathbf{x}'_i; \Sigma_i) \sim \exp(-(\mathbf{x}_i | \mathbf{x}'_i)^T \Sigma_i^{-1} (\mathbf{x}_i - \mathbf{x}'_i)), \tag{7}$$

where σ and Σ_i are the covariance parameters of a Gaussian distribution.

As we will show in Sect. 5, the discretization uncertainty at $\mathbf{x}'_i$ can also be estimated as a Gaussian distribution with the observation $\mathbf{x}_i$ as mean. That is $p(\mathbf{x}'_i) \sim \exp(-(\mathbf{x}'_i - \mathbf{x}_i)^T \Psi_i^{-1} (\mathbf{x}'_i - \mathbf{x}_i))$, where Ψ_i is the covariance parameter for discretization uncertainty. Here we assume that the observed data $\mathbf{x}_i$ is fixed and the true value $\mathbf{x}'_i$ is subject to random errors. This is equivalent to the so-called *Berkson's model* in statistics literature [2]. Then Eq. (6) becomes

$$\theta^* = \underset{\theta}{\mathrm{argmax}} \sum_i \inf_{\mathbf{x}'_i} \left[\frac{(\mathbf{w}^T \mathbf{x}'_i + b - c_i)^2}{\sigma^2} + (\mathbf{x}_i - \mathbf{x}'_i)^T \widetilde{\Sigma}_i^{-1} (\mathbf{x}_i - \mathbf{x}'_i) \right], \tag{8}$$

where $\theta = \{\mathbf{w}, b\}$ and $\widetilde{\Sigma}_i^{-1} = \Sigma_i^{-1} + \Psi_i^{-1}$.

Moreover, notice that if $(\mathbf{x}_i - \mathbf{x}'_i)^T \widetilde{\Sigma}_i^{-1} (\mathbf{x}_i - \mathbf{x}'_i)$ is large, then $p(\mathbf{x}'_i, c_i; \theta)$ term will have a small value and can be ignored in the integration for $p(\mathbf{x}i, ci)$. As a result, we can constrain $\mathbf{x}_i$ to lie within the ellipsoid $\mathscr{E}_i = \{\mathbf{x}'_i : (\mathbf{x}_i - \mathbf{x}'_i)^T \widetilde{\Sigma}_i^{-1} (\mathbf{x}_i - \mathbf{x}'_i) \leq r_i^2\}$ and this will not influence the final result considerably. Also considering the regularity of separating surfaces, Eq. (8) can be approximated by an optimization formulation that is similar to support vector machine (SVM):

$$
\begin{aligned}
&\underset{\mathbf{w},b,\xi_i}{\text{minimize}} \quad \frac{1}{2}\|\mathbf{w}\|^2 + \lambda \sum_{i=1}^{n} \xi_i \\
&\text{subject to} \quad c_i(\mathbf{w}^T \mathbf{x}'_i + b) \geq 1 - \xi_i, \quad \forall \mathbf{x}'_i \in \mathscr{E}_i \forall 1 \leq i \leq n; \\
&\qquad\qquad\quad \xi_i \geq 0, \qquad\qquad\qquad \forall 1 \leq i \leq n.
\end{aligned}
\tag{9}
$$

The above formulation minimizes the upper bound on the classification error, which is equivalent to separating error in Eq. (3). Errors occur when $\xi_i \geq 1$, as $\mathbf{x}'_i$ lies on the wrong side of P. The quantity λ is the penalty for any data point $\mathbf{x}'_i$ that either lies within the margin on the correct side of P ($0 < \xi_i \leq 1$) or on the wrong side of P ($\xi_i > 1$). $\|\mathbf{w}\|$ is the regularization term which controls the smoothness of the separating surface.

It is easy to verify that $c_i(\mathbf{w}^T \mathbf{x}'_i + b)$ reaches its minimum at point $\mathbf{x}_i - r_i(\mathbf{w}^T \widetilde{\Sigma}_i \mathbf{w})^{1/2} \widetilde{\Sigma}_i \mathbf{w}$ and the minimum value is $c_i(\mathbf{w}^T \mathbf{x}_i + b) - r_i(\mathbf{w}^T \widetilde{\Sigma}_i \mathbf{w})^{1/2}$. As a result, Eq. (9) can be further written as:

$$
\begin{aligned}
&\underset{\mathbf{w},b,\xi_i}{\text{minimize}} \quad \frac{1}{2}\|\mathbf{w}\|^2 + \lambda \sum_{i=1}^{n} \xi_i \\
&\text{subject to} \quad c_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i + r_i \left\| \widetilde{\Sigma}_i^{1/2} \mathbf{w} \right\|, \quad \forall 1 \leq i \leq n; \\
&\qquad\qquad\quad \xi_i \geq 0, \qquad\qquad\qquad\qquad\qquad\quad \forall 1 \leq i \leq n.
\end{aligned}
\tag{10}
$$

Such optimization problems have been studied in the literature [26] and can be solved using second order cone programming (SOCP) methods. Once $\mathbf{w}$ and b are computed, we can compute $\xi_i = \max(0, 1 - c_i(\mathbf{w}^T\mathbf{x}_i + b) + r_i\left\|\widetilde{\Sigma}_i^{1/2}\mathbf{w}\right\|)$.

4.2 Non-Gaussian Uncertainty

The uncertainty of real-world sensors may not be accurately modeled using a Gaussian distribution. Our approach can also handle non-Gaussian uncertainty.

Shivaswamy et al. [26] point out that the ellipsoid radius r_i is related to the confidence of the classification result when the training data contains noise. Briefly, if we desire the underlying surface point $\mathbf{x}_i'$ with Gaussian distribution to lie on the correct side of the separating surface with a probability greater than κ_i

$$\mathbb{P}_{\mathbf{x}_i' \sim \mathcal{N}(\mathbf{x}_i, \widetilde{\Sigma}_i)}\big((c_i(\mathbf{w}^T\mathbf{x}_i' + b) \geq 1 - \xi_i\big) \geq \kappa_i, \tag{11}$$

then $r_i = \mathrm{cdf}^{-1}(\kappa_i)$, where $\mathrm{cdf}(u) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{u}\exp\left(\frac{-S^2}{2}\right)ds$. Using multivariate Chebyshev inequality, this relationship between κ_i and r_i can be further extended to the case when $\mathbf{x}_i'$ follows non-Gaussian distribution. That is, if $\mathbf{x}_i' \sim (\mathbf{x}_i, \widetilde{\Sigma}_i)$ represents a family of distributions with a common mean and covariance given by $\mathbf{x}_i$ and $\widetilde{\Sigma}_i$, and we want $\mathbf{x}_i$ to lie on the correct side of the separating surface with a probability greater than κ_i

$$\sup_{\mathbf{x}_i' \sim (\mathbf{x}_i, \widetilde{\Sigma}_i)} \mathrm{P}_{\mathbf{x}_i'}\big((c_i(\mathbf{w}^T\mathbf{x}_i' + b) \geq 1 - \xi_i\big) \geq \kappa_i, \tag{12}$$

then $r_i = \sqrt{\frac{\kappa_i}{1 - \kappa_i}}$. This formulation implies that we can perform collision detection using Eq. (10) even when the uncertainty is non-Gaussian.

4.3 Non-linear Separating Surface

Linear separating surface is mainly limited to the case when all the underlying surfaces are convex. If any one of them is non-convex, a separating plane may not exist even when the surfaces are collision-free. Therefore, we need to extend our algorithm to non-linear P. Similar to typical SVM algorithms [31], we can remove the linear separating surface assumption by applying a *kernel trick* on the dual form of Eq. (10). Briefly, kernel trick is a method that transforms the Euclidean space $\mathscr{R}^n$

into another inner space $\mathscr{H}$ using mapping Φ and then replaces the inner product $\langle \mathbf{y}, \mathbf{z} \rangle \mathscr{R}^n$ by the new inner product $K(\mathbf{y}, \mathbf{z}) = (\Phi(\mathbf{y}), \Phi(\mathbf{z})) \mathscr{H}$ in space $\mathscr{H}$. Here $K(\cdot, \cdot)$ is called the *kernel function*. Usually a hyper-plane in $\mathscr{H}$ will correspond to a non-linear surface in $\mathscr{R}^n$, which is a popular way to construct non-linear classifiers in machine learning [9]. Some of the widely used kernel functions include linear ($K(\mathbf{y}, \mathbf{z}) = \mathbf{y}^{\mathrm{T}} \mathbf{z}$) and Gaussian ($K(\mathbf{y}, \mathbf{z}) = \exp(-\gamma \|\mathbf{y} - \mathbf{z}\|^2)$).

Based on the kernel trick, the non-linear separating surface can be formulated as $P = \{\mathbf{x} : \mathbf{w}^T \Phi(\mathbf{x}) + b = 0\}$. To compute P, we first transform Eq. (10) into its dual form. Next, based on the Taylor-expansion technique [3], we replace $\mathbf{y}^{\mathrm{T}} \mathbf{z} \partial \mathbf{z}$ by kernel function $K(\mathbf{y}, \mathbf{z})$ and replace $\mathbf{y}$ by the kernel gradient $\frac{\partial K(\mathbf{y}, \mathbf{z})}{\partial \mathbf{z}}$ and finally obtain the optimization formulation in non-linear case as

$$
\begin{aligned}
\underset{\alpha_i, \mathbf{v}_i}{\text{maximize}} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \Big(\sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j c_i c_j K(\mathbf{x}_i, \mathbf{x}_j) + \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i c_i (\widetilde{\Sigma}_j^{1/2} \frac{\partial K(\mathbf{x}_i, \mathbf{x}_j)}{\partial \mathbf{x}_j})^T \mathbf{v}_j \\
& + \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_j c_j (\widetilde{\Sigma}_j^{1/2} \frac{\partial K(\mathbf{x}_i, \mathbf{x}_j)}{\partial \mathbf{x}_j})^T \mathbf{v}_j + \sum_{i=1}^{n} \sum_{j=1}^{n} \mathbf{v}_i^T (\widetilde{\Sigma}_j^{1/2} \frac{\partial^2 K(\mathbf{x}_i, \mathbf{x}_j)}{\partial \mathbf{x}_i \partial \mathbf{x}_j} \widetilde{\Sigma}_j^{T/2}) \mathbf{v}_j)
\end{aligned}
$$

$$
\text{subject to} \quad \|\mathbf{v}_i\| \leq r_i \alpha_i,\ 0 \leq \alpha_i \leq C, \forall 1 \leq i \leq n;\ \text{and} \sum_{i=1}^{n} \alpha_i c_i = 0;
$$

$$(13)$$

where C is a regularity term similar to λ in Eq. (10). Once α_i and $\mathbf{v}_i$ are computed, we can compute the formulation for the separating surface P

$$
f(\mathbf{x}) = b + \sum_{j=1}^{n} \alpha_j c_j K(\mathbf{x}_j, \mathbf{x}) + \sum_{j=1}^{n} \mathbf{v}_j^T \widetilde{\Sigma}_j^{1/2} \frac{\partial K(\mathbf{x}_j, \mathbf{x})}{\partial \mathbf{x}_j} \tag{14}
$$

and $\xi_i = \max(0, \xi_i')$, where

$$
\xi_i' = 1 - c_i f(\mathbf{x}_i) + r_i \left\| \widetilde{\Sigma}_i^{1/2} f'(\mathbf{x}_i) \right\|. \tag{15}
$$

Notice that the surface parameter b does not appear in the dual form, but it can be computed based on Karush–Kuhn–Tucker conditions [5]. We first choose i so that $0 < \alpha_i < C, \|\mathbf{v}_i\| < r_i \alpha_i$ and then set $\xi_i' = 0$ in Eq. (15) to obtain b. Moreover, notice that all the results for non-linear separating surface are consistent with those for linear separating surface, which use a linear kernel $K(\mathbf{y}, \mathbf{z}) = \mathbf{y}^T \mathbf{z}$.

4.4 Probabilistic Collision Decision

Based on the computed separating surface, we present a simple scheme to perform probabilistic collision detection between the point clouds. First, we compute the

collision probability for each point, i.e., the probability that $\mathbf{x}_i'$ lies on the wrong side of separating surface:

$$\mathbb{P}_{\mathbf{x}_i' \sim \mathcal{N}(\mathbf{x}_i, \widetilde{\Sigma}_i)}(c_i f(\mathbf{x}_i') \leq 0) = \text{cdf}(-c_i f(\mathbf{x}_i)/\left\|\widetilde{\Sigma}_i^{1/2} f'(\mathbf{x}_i)\right\|). \qquad (16)$$

We denote this per-point probability as $\mathbb{P}(\mathbf{x}_i)$. Next, we need to use an appropriate metric to measure the collision probability between two point clouds. For two exact models, collision occurs if *any* subsets of them are in-collision. Therefore, for point clouds C_1 and C_2, it seems to be reasonable to define the collision probability between them as $1 - \prod_{\mathbf{x} \in \{C_1 \bigcup C_2\}}[1 - \mathbb{P}(\mathbf{x})]$. However, this metric may have some issues: when the number of points increases, its value will go to zero instead of converging to the real collision probability. The reason is that this metric does not consider the dependency between collision states of nearby points. Our approach for computing collision probability only involves far-away points with large per-point collision probability. First, we compute the maximum per-point collision probability $\max_{\mathbf{x}} \mathbb{P}(\mathbf{x})$. Next, we find all the points whose per-point collision probabilities are near the maximum value, e.g., more than $0.8 \max_{\mathbf{x}} \mathbb{P}(\mathbf{x})$. For points that are close to each other, we only use one of them in the whole body collision probability computation. The first rule filters out points whose collision probabilities are not large enough so as to improve the stability of collision results while the second rule filters out points that are closely correlated. Finally, we compute the collision probability between point clouds based on the left $m \ll n$ points $\{\tilde{\mathbf{x}}_i\} : \mathbb{P}_{C_1, C_2} = 1 - \prod_{i=1}^{m}[1 - \mathbb{P}(\tilde{\mathbf{x}}_i)]$. We can also use a simpler version of this metric which only considers the point with the maximum collision probability: $\mathbb{P}_{C_1, C_2} = \max_{\mathbf{x} \in C_1 \bigcup C_2} \mathbb{P}(\mathbf{x})$. For collision between exact models, the two metrics are equivalent, as $\mathbb{P}(\tilde{\mathbf{x}}_i) = \max_{\mathbf{x}} \mathbb{P}(\mathbf{x}) = 1$, for all i. The simpler metric can not distinguish the collision states when point clouds have one or more far-away points with large per-point collision probability, but it is more convenient to distinguish between collision-free and in-collision cases.

4.5 Acceleration Using Bounding Volume Hierarchies

We have reduced the problem of collision detection between two point clouds to a two-class classification problem and can solve it with SVM. However, performing collision detection by directly using Eq. (13) introduces some challenges. First, the timing complexity of SVM can be $\mathcal{O}(n^3)$, where $n = n_1 + n_2$ is the number of points in the two point clouds. As a result, the underlying algorithm can be slow for dense point clouds. Second, the two point clouds corresponding to different objects may have different numbers of points, which can result in unbalanced training data in terms of using machine learning algorithms. Moreover, if the two point clouds under consideration correspond to objects with different sizes (e.g., a large room

and a small robot), it will cause the optimization algorithm to have a lower separating error for the large object and higher error for the small object.

We use bounding volume hierarchies (BVH) to overcome these problems. These hierarchies provide a quick identification of objects or parts of an object that can be easily culled away and therefore perform exact collision queries on relatively few primitives.

5 Implementation and Results

In this section, we describe some details of our implementation and highlight the performance on several benchmarks.

5.1 Implementation

First, we discuss how to estimate the distribution of the underlying surface sample $p(\mathbf{x}_i')$. The mean of $p(\mathbf{x}_i')$ is $\mathbf{x}_i$ due to our unbiased assumption. We estimate the covariance $\mathbf{\Psi}_i$ based on the formulation described in [20]:

$$\Psi_i = \frac{\Sigma_{j=1}^n (\mathbf{x}_j - \mathbf{x}_i)(\mathbf{x}_j - \mathbf{x}_i)^T \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / \tau_i^2)}{\Sigma_{j=1}^n \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / \tau_i^2)}, \tag{17}$$

where n is the total number of points and τ_i is a parameter used to remove the influence of points too far away from $\mathbf{x}_i$. We set $\tau_i = \tau \cdot \eta_i.\tau$ as a global scale parameter and the variable $\eta_i = \frac{r}{\sqrt{k}}$ denotes the local sample spacing estimated from k a k-neighborhood, where r is the radius of the enclosing sphere of the k-nearest neighbors of $\mathbf{x}_i$.

Our algorithm is based on machine learning techniques and includes some parameters that need to be tuned. Fortunately, we find that our method is not sensitive to the parameter choice if we preprocess the data by scaling it to $[0, 1]^3$ volume in 3D. Scaling is considered important in terms of robustness of SVM, especially for the non-linear case. Moreover, scaling also helps us in computing the parameters that are suitable for the point clouds with different sizes or configurations. In practice, scaling also changes the uncertainty of each point, so we need to update the noise level from $\widetilde{\Sigma}_i$ to $\mathbf{S}\widetilde{\Sigma}_i\mathbf{S}^T$, where $\mathbf{S} = \mathrm{diag}(s_1, s_2, s_3)$ is the scaling matrix.

We have used our algorithm on data captured using robot sensors. Note that our method is designed for noisy environments where the ground-truth for collision detection is unknown. In this case, exact collision algorithms are not applicable as we don't have an exact representation of the environment. Therefore, it is difficult to

directly compare the quality or accuracy of our algorithm with prior methods. However, our method can guarantee: (1) For easy collision queries, i.e., when the distance between two collision-free objects is large or the two objects are in deep penetration, our method will give collision probability near 0 or 1. In this case, only very large noise can reverse the outcome of the query. However, our probabilistic algorithm would give the same result as the exact approach that first performs mesh reconstruction from the point clouds. (2) For challenging queries, i.e., when two objects are almost in-contact or have a small penetration, our method computes a collision probability near 0.5, because these configurations are more susceptible to noise. Exact collision algorithms will still provide a yes-no result, but the accuracy of the exact algorithm is governed by the underlying sampling and mesh reconstruction algorithm. If a yes-no collision answer is required, our algorithm uses two thresholds $A \geq 0.5 \geq B$: if collision probability $>A$, we report *collision-free*; if collision probability $<B$, we report *in-collision*; if collision probability is between A and B, we report *in-contact*. For example, when collision-avoidance is critical for the underlying applications, we can use large conservative value for A and small conservative value for B to achieve higher guarantees.

5.2 Results

We highlight the performance of our algorithm on real-world point clouds as well as synthetic data sets. We also compare its accuracy with prior collision detection techniques. The running time of our probabilistic algorithm is similar to that of exact collision detection algorithms and varies based on number of primitives and their relative configuration.

We evaluate the performance of our algorithm on a synthetic data set corresponding to a moving piano in a room with tables. We first generate a point cloud by sampling the polygons and adding some noise. Next, we use the PQP package to perform exact collision detection and separation distance query between the exact, triangulated model and compared the results with probabilistic collision detection on the resulting point cloud (see Fig. 3). We see a high correlation between our results and the actual separation distance, and it varies based on the level of noise. This shows that our approach is quite robust and even works well in degenerate configurations, e.g., when the two objects are barely touching or very close to each other.

Such configurations are more susceptible to noise and the exact collision detection algorithms are very sensitive to these configurations.

We have applied our probabilistic collision detection to the point cloud data generated for manipulation using the PR2 robot. Point cloud data on the PR2 robot is generated from a scanning laser range finder (*Hokuyo Top-URG(UTM-30LX)*) and a stereo camera (*WGE-100*), which is combined with an active texture projector to obtain good 3D data from untextured objects. The robot is placed in front of a table with multiple household objects (e.g., bowls, cans) on the table at a distance of
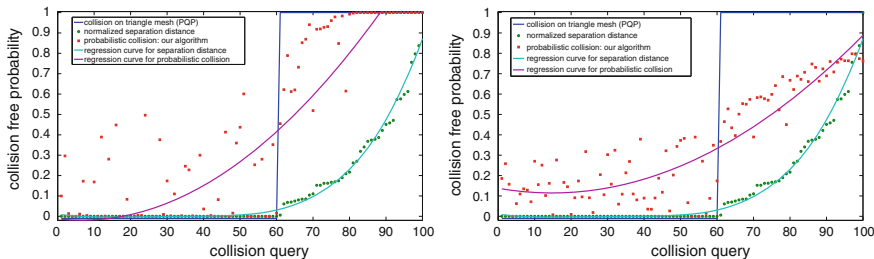
Fig. 3 Comparison between the results for 100 random queries between prior collision detection algorithms for exact triangle meshes and our algorithm on the point clouds (generated by sampling and adding noise). We show the results of exact collision detection and separation distance as well. If the noise in the point cloud is small (the *left* figure), our method returns 0 or 1 collision probability for most queries. When the queries correspond to a small separation distance or penetration depth (i.e., difficult cases), our algorithm computes collision-free probability close to 0.5. Furthermore, the collision-free probability is higher when the separation distance is large for non-overlapping objects. If the noise is large (the *right* figure), fewer queries return 0 or 1 collision probability. We see a good correlation between the regression curves computed by our algorithm and the exact queries on these synthetic datasets

about 1.5 m from the robot's sensors. The point clouds are a discretized (about ± 1.5 cm in range) representation of the real environment and are generated periodically by each sensor. The data is noisy and exhibits speckles especially in the vicinity of boundaries of objects and boundaries of the field of view of the sensor. The sensors are calibrated with respect to each other and the arms using a known calibration pattern. The known position of the arms, measured using encoders, is used to filter out the points corresponding to the arms from the point clouds obtained by the sensors. Typical point clouds generated by the stereo sensors on the PR2 robot have more than 40,000 points and are generated at 20 Hz. Point clouds generated by the laser range scanners typically have about 10,000 points. The data from the point clouds is aggregated into a *collision map* representation. The collision map is a 3-dimensional occupancy grid maintained at a fixed resolution. The resulting collision maps are at 1 cm resolution and have about 2,000 occupied cells. A complete triangulated mesh representation of the robot, including the arms and the gripper, is also available as input for the collision checker.

There are very few algorithms or systems available for collision checking between noisy point clouds. As a result, we compare our algorithm with the implementation in ROS (based on ODE) and exact collision detection on reconstructed meshes.

The collision checking procedures used in ROS are currently based on the collision checking implementation in the ODE software package. The input to the collision checker is a combination of mesh models for the robot and objects in the environment and the collision map. The points in the collision map are represented as axis-aligned box primitives whose length is equal to the resolution at which the collision map is maintained. The current representation of the collision space considers every point in the collision map to be a potential obstacle. Thus, noise in

the sensor data can frequently lead to false positives, i.e., the detection of potential collisions in parts of the environment where there are no obstacles. There is no robust criterion to compute the box size, e.g., a function of noise, so we can't compare all the features of our method with ODE collision checking. We also use a reconstruction algorithm to compute a triangle mesh from the point clouds and perform triangle-based collision as well as separation distance computation using PQP. In many ways, this formulation only provides an approximation of the ground truth and is used to evaluate the robustness of our algorithm.

As shown in Fig. 4a, our result matches well with the exact collision detection algorithm, especially with the separation distance computation. Furthermore, we notice that the collision probability of our approach changes slowly when the noise increases. It is more robust as compared to the yes-no result computed by ODE on the point clouds, which is likely to frequently switch between collision-free and in-contact configurations, when the noise level changes. We also apply our algorithm on a dataset generated using Kinect RGB-D cameras. This dataset corresponds to an indoor environment at the Intel Lab in Seattle captured using Kinect sensors. The results of our probabilistic collision detection on this dataset are shown in Fig. 4b.

Moreover, from Figs. 3 and 4, we observe that configurations with the same distances to the obstacles can have large spread in the computed collision probabilities. The reason is that distance is only a partial measurement of collision status while our collision probability is a more complete description about collision status and provides more detailed information about the relative configurations.

For one query, our method needs about 500–1000 ms for about 10,000 points on one Intel Core i7 3.2 GHz CPU, based on BVH acceleration. It is about 5–10 times slower than optimized collision packages on models with 10 K triangles (e.g., PQP can compute collisions in such situations in about 50–100 ms). However, the reconstruction algorithms take more than 10 s to compute the triangulated mesh from the point cloud. Moreover, our current implementation can be optimized in
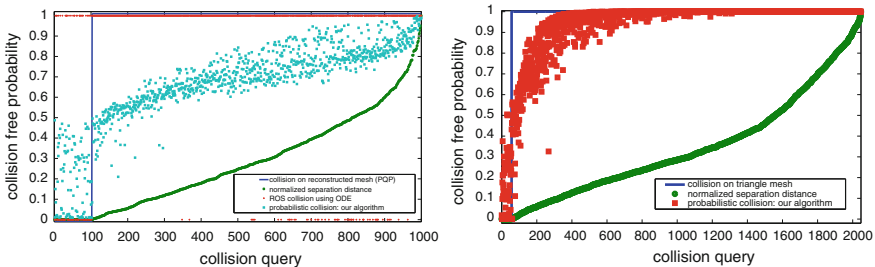

Fig. 4 **a** Comparison on point-cloud data generated by PR2 robot sensor: we use our probabilistic collision detection on the noisy point cloud versus results computed by ODE package used in ROS versus exact collision and distance queries on the reconstructed mesh model. Our results on the point cloud are more robust as compared to the ODE package. **b** Result of our algorithm on Kinect data

several ways, such as replacing the non-linear kernel by approximated linear kernel [23] and using more efficient SVM methods designed for large scale data [6]. We expect an optimized probabilistic collision method to have similar speed to the PQP algorithm. Furthermore, our approach can provide more detailed information and can be easily combined with planning/reasoning algorithms. For example, we can combine it with trajectory optimization algorithms (e.g., CHOMP [24], STOMP [12]) to find a smooth path that has a minimum probability of colliding with the obstacles.

6 Conclusions and Future Work

We have presented a novel and robust method for contact computation between noisy point cloud data using machine learning methods. We reformulate collision detection as a two-classification problem and compute the collision probability at each point using support vector machines. The algorithm can be accelerated by using bounding volume hierarchies and performing a stochastic traversal. We have tested the results on synthetic and real-world data sets and the preliminary results are promising.

There are many avenues for future work. We need to test the performance on different robotic systems and evaluate its performance on tasks such as planning and grasping. It would be useful to extend this approach to continuous collision checking, which takes into account the motion of the robot between discrete intervals along the path. Similar probabilistic methods can also be developed for other queries, including separation and penetration depth computation. Finally, we are interested in improving the algorithm to handle dynamic environments where points may change position or can be added or removed from the environment due to movement, occlusion or incremental data, based on incremental SVM [4] and BVH refitting techniques [14].

References

1. K.-H. Bae, D. Belton, D.D. Lichti, A closed-form expression of the positional uncertainty for 3D point clouds. Trans. Pattern Anal. Mach. Intell. **31**, 577–590 (2009)
2. J. Berkson, Are there two regressions? J. Am. Stat. Assoc. **45**(250), 164–180 (1950)
3. J. Bi, T. Zhang, Support vector classification with input data uncertainty, in *Advances in Neural Information Processing Systems* (2005), pp. 161–168

4. G. Cauwenberghs, T. Poggio, Incremental and decremental support vector machine learning, in *Advances in Neural Information Processing Systems* (2001)
5. C.-C. Chang, C.-J. Lin, *LIBSVM: A Library for Support Vector Machines* (2001)
6. R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, C.-J. Lin, Liblinear: a library for large linear classification. J. Mach. Learn. Res. **9**, 1871–1874 (2008)
7. N. Govindaraju, M. Lin, D. Manocha, Fast and reliable collision culling using graphics hardware. Trans. Vis. Comput. Graph. **12**(2), 143–154 (2006)
8. L. Guibas, D. Hsu, H. Kurniawati, E. Rehman, Bounded uncertainty roadmaps for path planning. Algorithmic Found. Robot. VIII **57**, 199–215 (2009)
9. T. Hofmann, B. Schölkopf, A.J. Smola, Kernel methods in machine learning. Ann. Stat. **36**(3), 1171–1220 (2008)
10. P.M. Hubbard, Approximating polyhedra with spheres for time-critical collision detection. Trans. Graph. **15**, 179–210 (1996)
11. P. Jenke, M. Wand, M. Bokeloh, A. Schilling, W. Straßer, Bayesian point cloud reconstruction, in *Eurographics* (2006), pp. 379–388
12. M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, S. Schaal, Stomp: stochastic trajectory optimization for motion planning, in *International Conference on Robotics and Automation* (2011)
13. J. Klein, G. Zachmann, Point cloud collision detection, in *Eurographics* (2004), pp. 567–576
14. C. Lauterbach, Q. Mo, D. Manocha, gProximity: hierarchical gpu-based operations for collision and distance queries. Comput. Graph. Forum **29**(2), 419–428 (2010)
15. J.-K. Lee, Y.J. Kim, Haptic rendering of point set surfaces, in *EuroHaptics* (2007), pp. 513–518
16. J.-M. Lien, Point-based Minkowski sum boundary, in *Pacific Graphics* (2007), pp. 261–270
17. M. Lin, D. Manocha, Collision and proximity queries, in *Handbook of Discrete and Computational Geometry* (CRC Press, Inc., 2004), pp. 787–808
18. P.E. Missiuro, N. Roy, Adapting probabilistic roadmaps to handle uncertain maps, in *International Conference on Robotics and Automation* (2006), pp. 1261–1267
19. D.M. Mount, Geometric intersection, in *Handbook of Discrete and Computational Geometry* (CRC Press, Inc., 2004), pp. 857–876
20. M. Pauly, N.J. Mitra, L. Guibas, Uncertainty and variability in point cloud surface data, in *Symposium on Point-Based Graphics* (2004), pp. 77–84
21. A. Petrovskaya, O. Khatib, Global localization of objects via touch. Trans. Robot. **27**, 569–585 (2011)
22. M. Ponamgi, D. Manocha, M.C. Lin, Incremental algorithms for collision detection between solid models, in *Symposium on Solid Modeling and Applications* (1995), pp. 293–304
23. A. Rahimi, B. Recht, Random features for large-scale kernel machines, in *Advances in Neural Information Processing Systems* (2007)
24. N. Ratliff, M. Zucker, J.A.D. Bagnell, S. Srinivasa, Chomp: gradient optimization techniques for efficient motion planning, in *International Conference on Robotics and Automation* (2009)
25. B. Schölkopf, J. Giesen, S. Spalinger, Kernel methods for implicit surface modeling, in *Advances in Neural Information Processing Systems* (2005), pp. 1193–1200
26. P.K. Shivaswamy, C. Bhattacharyya, A.J. Smola, Second order cone programming approaches for handling missing and uncertain data. J. Mach. Learn. Res. **7**, 1283–1314 (2006)
27. D. Steinemann, M. Otaduy, M. Gross, Efficient bounds for point-based animations, in *Symposium on Point-Based Graphics* (2007), pp. 57–64
28. F. Steinke, B. Schölkopf, V. Blanz, Support vector machines for 3d shape processing, in *Eurographics* (2005), pp. 285–294
29. I.A. Sucan, M. Kalakrishnan, S. Chitta, Combining planning techniques for manipulation using realtime perception, in *International Conference on Robotics and Automation* (2010), pp. 2895–2901
30. S. Thrun, W. Burgard, D. Fox, *Probabilistic Robotics* (The MIT Press, 2005)
31. V.N. Vapnik, *The Nature of Statistical Learning Theory* (Springer, New York, 1995)