Efficient Global Penetration Depth Computation for Articulated Models

Hao Tian^a, Xinyu Zhang^a, Changbo Wang^a, Jia Pan^b, and Dinesh Manocha^c

^a*School of Software, East China Normal University, Shanghai, China.*

^b*The University of Hongkong.*

^c*The University of North Carolina at Chapel Hill.*

Abstract

We present an algorithm for computing the global penetration depth between an articulated model and an obstacle or between the distinctive links of an articulated model. In so doing, we use a formulation of penetration depth derived in configuration space. We first compute an approximation of the boundary of the obstacle regions using a support vector machine in a learning stage. Then, we employ a nearest neighbor search to perform a runtime query for penetration depth. The computational complexity of the runtime query depends on the number of support vectors, and its computational time varies from 0.03 to 3 milliseconds in our benchmarks. We can guarantee that the configuration realizing the penetration depth is penetration free, and the algorithm can handle general articulated models. We tested our algorithm in robot motion planning and grasping simulations using many high degree of freedom (DOF) articulated models. Our algorithm is the first to efficiently compute global penetration depth for high-DOF articulated models.

Keywords: Configuration Space, Articulated Models, Penetration Depth, Support Vector Machine

---

1. Introduction

Computing the magnitude of inter-penetration between two overlapping rigid/articulated objects is a fundamental problem in computational geometry. One metric that is widely used to measure the extent of inter-penetration is penetration depth (PD), which requires computing a minimum transformation (translation and rotation) to separate two overlapping objects. The resulting transformation motion can be used to compute the contact force for penalty-based methods, valid poses in grasping simulations, force/torque feedback in haptic rendering, sample generation in narrow passages for motion planning, etc.

The exact computation for PD, particularly the so-called generalized PD that involves both translation and rotation [1], can be reduced to arrangement computation in a high-dimensional configuration space that has high computational complexity. For instance, the combinatorial complexity of exact PD is as high as $O(n^{12})$ [2] for two models with n triangles in three-dimensional space. As a result, all practical algorithms tend to compute an approximate solution. A wide variety of algorithms have been proposed in the literature for rigid models (e.g [3, 4, 2, 5, 6, 7, 8]). For articulated models, the resulting configuration spaces are high-dimensional non-Euclidean spaces. For instance, the configuration space for a stationary obstacle and a six degrees of freedom (DOFs) robot arm fixed on the ground, is a six-dimensional non-Euclidean space. If we allow the arm base to move in space, its configuration space becomes nine-dimensional and non-Euclidean. As the number of joints

increases, the complexity of a configuration space can become very high. In particular, if self-collisions between distinctive links must be considered, the complexity can increase rapidly. These self-collisions may correspond to many small and isolated components in the high-dimensional configuration space. Due to its high complexity, it is challenging to produce an exact representation of a space with such high dimensionality. To the best of our knowledge, only one recent study [9] has attempted to compute PD between articulated models, but its solution yielded only locally optimal PD.

Main Result: We present an efficient algorithm to approximate the global PD in high-dimensional spaces for articulated models. Built on the early framework proposed in [8, 10], we use a machine learning method to approximate the boundary of the obstacle regions in the configuration space for an articulated model and its surrounding obstacles. We generate a set of configuration samples to densely cover the boundary of obstacle regions. Given a query configuration for computing PD, the closest configuration can be found quickly using approximate k -nearest-neighbor search. The magnitude of PD can be computed using non-Euclidean distance metrics between the query configuration and the closest configuration. Compared with existing methods, our method has the following advantages:

Novelty: Our algorithm is the first global PD approach for high-DOF articulated models.

Generality: Our algorithm can handle hybrid joints and links represented using polygonal models.

Conservativeness: Our algorithm can guarantee that the configuration realizing PD is penetration free. This feature is particularly important for enforcing non-penetration constraints in certain applications. such as dynamic simulation, motion plan-

---

Email address: xyzhang@sei.ecnu.edu.cn. (Xinyu Zhang)

ning, and grasping simulation.

Efficiency: Our algorithm takes about 0.03~3 milliseconds per runtime PD query on single core. The computational complexity of runtime query depends only on the number of support vectors used in learned obstacle regions.

The rest of the paper is organized as follows. In Section 2, we provide a review of the related work on PD computation. In Section 3, we introduce the notation that we use in the paper and present the algorithm for approximating obstacle regions for articulated models. In Section 4, we describe our approach to compute PD by using approximate obstacle regions and a solution for computing conservative PD. Section 5 describes the implementation details and some basic experimental results. Section 6 highlights the results on complex scenarios.

2. Related Work

There are two types of PD: translational PD and generalized PD. Translational PD corresponds to a translational motion to separate two objects, whereas generalized PD corresponds to both translational and rotational motions. Various work on PD computation has been reported in computer graphics, geometric modeling, haptics, and robotics, and most of the associated algorithms address rigid models. In the following, we offer a brief overview of these algorithms.

Translational Penetration Depth: Exact translational PD can be formulated using the Minkowski sum; it is obtained by computing the closest distance from the origin to the boundary of the Minkowski sum [11, 12]. The worst-case complexities for these approaches is $O(n^2)$ for convex polytopes and $O(n^6)$ for non-convex polytopes, where n is the number of features in the polytopes [11]. Because of the high computational cost involved in computing exact translational PD, most practical approaches compute an approximation instead. For convex polytopes, many methods simply compute an approximate Minkowski sum, which is then used as an approximate translational PD calculation [13, 14]. Translational PD between non-convex objects is typically computed using convex decomposition, which is based on the fact that an exact Minkowski sum can be computed based on convex decomposition and the union of all the pairwise Minkowski sums [15]. As it is expensive and not particularly robust to compute the union explicitly, many approximate solutions have been proposed, including GPU-based approaches [2, 16] and methods that are based on reduced convolution and filtering [17, 18]. Other methods avoid the expensive Minkowski union entirely by computing only translational PD between each pair of convex components. These approaches are called local methods because the resulting PD only depends locally on the closest point on the penetrated surfaces and may not result in a globally consistent solution. Most local PD methods are based on local features [19, 20, 18, 21, 22], i.e., each convex piece generated by the convex decomposition is a mesh triangle. Some recent methods [7] uses iterative optimization to compute approximate translational PD.

Generalized Penetration Depth: Few algorithms can compute generalized PD, which considers both translation and ro-

tation. [1] estimate the upper and lower bounds for generalized PD between two general polyhedral models by decomposing the models into convex components. Most practical algorithms for generalized PD computation follow the iterative, constrained optimization framework, which generates a series of configurations on the contact space with decreasing distances to the given in-collision query. [5] generates such a series of configurations by moving a small step size from a configuration along the gradient direction. [6] first compute an approximate local contact space near a configuration and then perform random sampling within the approximate contact space to find a suitable following configuration. [23] calculate a linearized contact space in the neighborhood of a configuration and then obtain an optimal following configuration by solving a linear complementary problem (LCP). Most of these approaches [1, 6, 5] are slow for interactive applications and do not have the necessary guarantees for a global solution.

Penetration Depth for Articulated Models: Recently, [9] present an algorithm to approximate PD for articulated models. Their algorithm approximated a local configuration space using iterative and constrained optimization techniques and provided a locally optimal PD.

Other Penetration Depth Metrics: In addition to the related work above, there are other definitions of PD. Intersection volume and its derivative can also be used for volume-based repulsion [24]. Distance fields are also used for local translational PD computation [25] and can be computed in realtime using GPUs. Point-based Minkowski sum approximation [26] can also be used to compute translational PD.

3. Configuration Space Approximation

3.1. Notation

We denote an articulated model as $\mathcal{A}$ and an obstacle as O , as shown in Fig. 1. We assume that $\mathcal{A}$ has n links. The i -th link of $\mathcal{A}$ is denoted as $\mathcal{L}_i$, and the link attached to $\mathcal{L}_i$ is denoted as $\mathcal{L}_{i+1}$. The joint connecting two links, $\mathcal{L}_i$ and $\mathcal{L}_{i+1}$, is denoted as $\mathcal{J}_i$. Joint $\mathcal{J}_i$'s parameter is denoted as θ_i . We denote the configuration space composed of $\mathcal{A}$ and O as C and each configuration $\mathbf{q}$ in C corresponds to the relative configuration (i.e., translation and rotation) of $\mathcal{A}$ with respect to O . The i -th component of $\mathbf{q}$ is $q_i = \theta_i$. C has two regions: the obstacle region $C_{obs} = \{\mathbf{q} : \mathcal{A}(\mathbf{q}) \cap O \neq \emptyset\}$ and free space $C_{free} = C \setminus C_{obs}$. Intuitively speaking, C_{obs} corresponds to configurations in which $\mathcal{A}$ and O collide, and C_{free} corresponds to configurations in which $\mathcal{A}$ and O do not. If self-collisions between distinctive articulated links are considered and added to the definition of C_{obs} , we can rewrite $C_{obs} = \{\mathbf{q} : \mathcal{A}(\mathbf{q}) \cap O \neq \emptyset\} \cup \{\mathbf{q} : \mathcal{L}_i(\mathbf{q}) \cap \mathcal{L}_j(\mathbf{q}) \neq \emptyset, |i - j| > 1\}$. Note that consecutive links are excluded from the set of collision pairs because they are connected by a joint and are in contact all the time. The boundary of C_{obs} is denoted as ∂C_{obs} . The approximation of C_{obs} is denoted as $\bar{C}_{obs}$ and its boundary is $\partial \bar{C}_{obs}$. Fig. 1-(b) shows the obstacle region C_{obs} for the models given in Fig. 1-(a) in which two revolute joints are active.

Figure 1: Articulated model and obstacle region. (a) An articulated model, $\mathcal{A}$, and an obstacle, O . $\mathcal{A}$ consists of three links. The first link, $\mathcal{L}_0$, is fixed to the ground. Every other link, $\mathcal{L}_{i+1}$, is attached to $\mathcal{L}_i$ by a revolute joint, $\mathcal{J}_i$. (b) The figure shows the obstacle region C_{obs} (red) between $\mathcal{A}$ and O . Each joint parameter (coordinate) θ_i ranges from $-\pi$ to π .

3.2. Approximating Obstacle Regions

Uniform Sampling: For an articulated model, $\mathcal{A}$, and an obstacle, O , we use a machine learning method to compute an approximate model of their obstacle regions. We first perform uniform sampling in their configuration space, $\mathcal{C}$, to obtain a set of configuration points. These samples belong either to $\mathcal{C}_{free}$ or to $\mathcal{C}_{obs}$, and their collision states are determined using discrete collision detection algorithms, such as [27]. For the sake of simplicity, we only account for collisions between articulated links and obstacles in this section. Self-collisions are considered further below (in Section 5).

Machine Learning: With these labeled data (configuration samples and their collision states), a support vector machine (SVM) technique is used to train a binary classifier to separate $\mathcal{C}_{obs}$ and $\mathcal{C}_{free}$. Briefly speaking, an SVM maps the given original data into a feature space ($\mathcal{H}$) by a function ϕ , to reduce a nonlinear classification to a linearly separable problem. The function ϕ computes a mapping from an *input space* onto a *feature space*. A point in the feature space is the image of an input configuration. Any point in the input space corresponding to a point in the feature space is called its pre-image. Then, an optimal separating hyperplane in the feature space can be mapped back into input space via its inverse mapping. Let $K(\mathbf{q}_i, \mathbf{q}_j) = \phi(\mathbf{q}_i)^T \phi(\mathbf{q}_j)$ represent the kernel function, which is used to calculate inner products in the feature space. The radial basis function (RBF) serves as the kernel.

$$K(\mathbf{q}_i, \mathbf{q}_j) = \exp(-\gamma \|\mathbf{q}_i - \mathbf{q}_j\|^2), \quad (1)$$

where γ is a positive parameter. Thus, the classifier can be modeled as

$$f(\mathbf{q}) = \mathbf{w} \cdot \phi(\mathbf{q}) + \mathbf{b} = \sum_{i=1}^m \alpha_i K(\mathbf{q}_i, \mathbf{q}) + \mathbf{b}, \quad (2)$$

where $\mathbf{w} \in \mathcal{H}$ and $\mathbf{b} \in \mathbb{R}$. Most of α_i are non-zero and their corresponding $\mathbf{q}_i$ are the support vectors. $f(\mathbf{q}) = 0$ is referred to as a decision boundary, which corresponds to an implicit function that can be used to predict the collision state for an input configuration. This implicit function can also be used to approximate the boundary of the obstacle region, $\partial \bar{\mathcal{C}}_{obs}$.

In general, better classifiers can be trained by an SVM technique when more samples are given. However, due to the high computational cost associated with the training method, the training samples must be limited. Therefore, [8] suggests an active learning strategy to accelerate the training process in an iterative manner. In short, a relatively coarse SVM classifier is first obtained using a small set of configuration samples. Then, it is refined iteratively by adding more samples into the training set. The main difficulty is to select *good* samples that can help improve the approximation quickly and lead to rapid convergence in machine learning. Here, we present a new approach to quickly select good samples for purposes of refining the trained boundary, $\partial \bar{\mathcal{C}}_{obs}$. This approach is based on the following two key observations regarding support vectors. One observation is that support vectors fully determine the decision boundary, $f(\mathbf{q}) = 0$ ($\partial \bar{\mathcal{C}}_{obs}$), and the other observation is that the pre-images of these support vectors are distributed near the exact boundary, $\partial \mathcal{C}_{obs}$ (i.e. the margin boundaries). Based on these two observations, our goal is to add more samples near $\partial \mathcal{C}_{obs}$ into the training data.

Figure 2: (a) A few new samples are generated on the sphere with center $\mathbf{s}$ and radius r . Some of these new samples will be added to the training set. (b) This strategy can be used to find narrow passages and disconnected components.

Refinement: Uniform sampling can explore unknown spaces effectively but disregards the particular space, such as obstacle regions, $\partial \mathcal{C}_{obs}$. Based on the observation discussed above, the samples around $\partial \mathcal{C}_{obs}$ play an important role in determining the decision boundary, $f(\mathbf{q}) = 0$ ($\partial \bar{\mathcal{C}}_{obs}$). Here, we propose a new strategy to generate more samples around $\partial \mathcal{C}_{obs}$, which will help refine the coarse approximation of $\partial \mathcal{C}_{obs}$. The basic idea is to generate more samples between the margin boundaries. As shown in Fig. 2-(a), margin boundaries are highlighted using two dotted curves, which can be achieved to generate more samples around support vectors. More specifically, for any support vector, $\mathbf{s}$, we uniformly generate some samples on the sphere with center $\mathbf{s}$ and radius r . r is determined by

$$r = \max_{\mathbf{q} \in SV^+} \min_{\mathbf{q}' \in SV^-} \text{dist}(\mathbf{q}, \mathbf{q}'), \quad (3)$$

where SV^+ and SV^- are the support vectors that are labeled as collision-free and in-collision, respectively. This equation is to find an appropriate radius that is used to generate more samples within the margin boundaries. Intuitively, this equation implies that, for each SV , we find its nearest neighbor among

the opposite-labeled SV s. Then we choose the pair of SV s that has the maximum distance to compute radius r . Since most of SV s lie in the maximal margin boundary, radius $r \geq \frac{1}{\|w\|}$, where $\frac{1}{\|w\|}$ is the half of margin of margin boundaries. Sampling using r , will increase the probability of generating new sample pairs that have opposite collision states. Therefore, it will help generate more new samples within the margin boundaries in the next step. If the new sample has a different collision state, we generate one more sample between the new sample and the given support vector (e.g., the middle point). However, if the new sample has the same collision state as the given support vector, this sample is discarded. As shown in Fig. 2-(a), the black dot indicates a support vector. Four samples are generated on the sphere with radius r . Using our approach, the four new samples on the left will be added into the training set, and the two samples on the right will be discarded. Then, a new and better classifier can be constructed incrementally by using the old decision boundary and the new samples. Our approach can also achieve better performance when there are narrow passages, as shown in Fig. 2-(b).

To gain an understanding of our refinement process, a training process is shown in Fig. 3. The blue regions correspond to obstacle regions for the models given on the right. After a few iterations, the approximation $\bar{C}_{obs}$ finds more isolated obstacle regions.

4. Penetration Depth Computation

In this section, we first define distance metric for articulated models used in our algorithm and then present our approach to compute penetration depth.

4.1. Distance Metric

In this paper, we consider both translational and rotational motion. For an articulated model $\mathcal{A}$, when its configuration changes from $\mathbf{q}$ to $\mathbf{q}'$, we use the following displacement to calculate distance metric [9].

$$\text{dist}(\mathbf{q}, \mathbf{q}') = d_0 + d_1 + \dots + d_{n-1}, \quad (4)$$

where d_i is the distance component contributed by joint i and its associated links.

For an articulated model, translational motion is contributed by prismatic joints. When a prismatic joint parameter of $\mathcal{J}_i$ changes from q_i to q'_i , the distance contributed by a prismatic joint and its associated link $\mathcal{L}_{i+1}$ is calculated by

$$d_i = \frac{V_{i+1}}{V} |q'_i - q_i|^2, \quad (5)$$

where V_{i+1} is the volume of $\mathcal{L}_{i+1}$ and V is the volume of $\mathcal{A}$.

For a revolute joint $\mathcal{J}_i$, the displacement distance by its associated link $\mathcal{L}_{i+1}$ can be represented as

$$d_i = \frac{4}{V} \sin^2 \left(\frac{\theta}{2} \right) \omega_i^T I_i \omega_i, \quad (6)$$

where V is the volume of $\mathcal{A}$, θ is rotation angle calculated by $\theta = q'_i - q_i$, ω_i is the rotational axis of $\mathcal{J}_i$ and I_i is the inertia tensor of $\mathcal{L}_{i+1}$ in reference frame. For a spherical joint, its distance metric can be defined analogously.

4.2. PD Computation Using Discretized $\partial\bar{C}_{obs}$

Given an approximation of obstacle region $\bar{C}_{obs}$ and a query $\mathbf{q}_0 \in C_{obs}$, we can compute penetration depth by looking for the nearest point on $\partial\bar{C}_{obs}$.

$$\overline{\text{PD}}(\mathcal{A}(\mathbf{q}_0), \mathcal{O}) = \min_{\mathbf{q} \in \partial\bar{C}_{obs}} \text{dist}(\mathbf{q}_0, \mathbf{q}), \quad (7)$$

where $\mathcal{A}(\mathbf{q}_0)$ corresponds to $\mathcal{A}$ located at the configuration $\mathbf{q}_0$, $\overline{\text{PD}}$ denotes penetration depth computed using an approximate obstacle region $\bar{C}_{obs}$.

Solving Equation 7 leads to an optimization problem [8, 9]. The solution proposed in [9], depicted in Fig. 4-(a), may lead to a local nearest configuration $\mathbf{q}_2$ if a random initial guess $\mathbf{q}_g$ is used. The approach suggested in [8], shown in Fig. 4-(b) uses a nearest support vector as the initial guess and then a constrained optimization is used to search for the closest configuration. The latter may suffer from the problem of slow convergence.

Here, we extend [8] and propose a new approach to perform fast PD computation, described as follows.

Discretized Representation: A new discretized approximation is generated from the final trained $\partial\bar{C}_{obs}$. This will be a discrete representation of $\partial\bar{C}_{obs}$ with very dense samples. Here, we denote this discretized representation as $D[\bar{C}_{obs}]$, where $D[\bar{C}_{obs}]$ consists of two components: collision-free configurations $D^+[\bar{C}_{obs}]$ and in-collision configurations $D^-[\bar{C}_{obs}]$. This step is illustrated in Fig. 5. To generate a discretized representation, we choose any red in-collision configuration from SV^- and use k -NN search to find a few green collision-free neighbors from SV^+ . Then we move $\mathcal{A}$ from a green configuration to a red in-collision configuration until a contact is found. This contact corresponds to a configuration near $\partial\bar{C}_{obs}$. We add this configuration into $D^+[\bar{C}_{obs}]$. As shown in Fig. 5-(b), the black points denote $D^+[\bar{C}_{obs}]$ after applying our approach. A dotted line depicts a motion trajectory from a green point to a red point. In fact, these trajectories can be very complicated, especially for an articulated model. This step is relatively expensive and is performed in a pre-processing stage after $\bar{C}_{obs}$ is obtained.

Figure 5: Discretized representation model for $\partial\bar{C}_{obs}$. (a) Exact boundary of C_{obs} (grey) and its approximation using SVM-based machine learning (blue). Red and green points are support vectors. (b) A discretized representation can be obtained by generating more samples near the exact boundary ∂C_{obs} . $D^+[\bar{C}_{obs}]$ represents the collision-free configuration of discretized boundary.

Approximate k -Nearest Neighbor: To compute penetration depth, an approximate k -nearest neighbor algorithm is used to

Figure 3: Training process for an articulated model composed of three active revolute joints. (a)~(d) are four iterations of the learning phase, where $|SV|$ s are the numbers of their support vectors. The images on the right show three different robot poses and their corresponding configurations are highlighted in obstacle regions given in (d).

first quickly determine k_1 configurations nearest to the query configuration $\mathbf{q}_0$ for $D[\bar{\mathcal{C}}_{obs}]$. For configuration spaces, we use a hierarchical clustering algorithm to perform k -NN search [28]. As shown in Fig. 4-(c), $D[\bar{\mathcal{C}}_{obs}]$ is depicted using a set of green and red configurations. In our experiment, we choose $k_1 = 50 \sim 100$. Then the nearest point is found among these k_1 configurations. As shown in Fig. 4, these k_1 -nearest neighbor configurations are enclosed in eclipses. Then, the closest configuration is determined in a brute force manner among these k_1 configurations.

Conservative Penetration Depth: For a given query $\mathbf{q}_0$ and the closest $\mathbf{q}_1$, if $\mathbf{q}_1$ is a collision-free configuration, conservative penetration depth can be readily calculated by $\text{dist}(\mathbf{q}_0, \mathbf{q}_1)$ using Equation 4. If $\mathbf{q}_1$ is an in-collision configuration, we perform k -nearest neighbor algorithm again to find k_2 contiguous configurations around $\mathbf{q}_1$. Assume that there exists at least one collision-free configuration among these k_2 neighbors, we search for the nearest one $\bar{\mathbf{q}}_1$. This process is shown in Figs. 4-(c) and (d). We choose $k_2 = 5 \sim 10$. An alternative strategy is to directly search for the nearest neighbor for $\mathbf{q}_0$ from $D^+[\bar{\mathcal{C}}_{obs}]$ using the following query.

$$\overline{\text{PD}} = \min_{\mathbf{q} \in D^+[\bar{\mathcal{C}}_{obs}]} \text{dist}(\mathbf{q}_0, \mathbf{q}), \quad (8)$$

$\mathbf{q}$ is the closest point in $D^+[\bar{\mathcal{C}}_{obs}]$, see Algorithm 2. Conservative penetration depth and its corresponding configuration can guarantee separating two objects.

5. Results

We implemented our algorithm under the framework of [8], using C++ language. We used Motion Planning Kit [29] for modeling and simulating articulated models and their surrounding obstacles. We used GrasPlt! [30] to perform grasping simulation using hand models. We used LIBSVM [31] to perform SVM-based machine learning. We used k -nearest neighbor search multiple times. In this section, we test our algorithm using a few simple articulated models. Some complex articulated models will be given in the next section.

5.1. Two-Active-Joint Models

In Fig.1-(a), we show an articulated model and an obstacle. If we only allow two revolute joints $\mathcal{J}_0$ and $\mathcal{J}_1$ to be active, we can obtain their obstacle region shown in Fig.1-(b). We will use this benchmark for comparison between three different penetration results (i.e. analytical, approximate and conservative penetration depths).

Let D be the distance from the first joint J_0 to obstacle O . If we increase D , it is clear that the corresponding exact penetration depth will decrease. As presented in [9], penetration depth can be analytically calculated for a query configuration $\mathbf{q}_0$ in the scenario given in Fig.1-(a). As shown in Fig. 6, the red curve shows the exact penetration depth obtained using an analytical method. The green curve is for the approximate results. However, at some locations (e.g. distance is 1.3), the approximate

Figure 4: Penetration depth query process. (a) A method suggested in [9]. Given a random initial guess $\mathbf{q}_g$, an optimization-based algorithm may lead to a local nearest configuration $\mathbf{q}_2$, rather than the global nearest configuration $\mathbf{q}_1$. (b) A method suggested in [8]. A nearest support vector serves as the initial guess and then a constrained optimization is used to search for the closest configuration. (c)-(d) A discrete representation is generated from (b), using similar technique described in Fig. 5. An approximate k -nearest neighbor algorithm may find k_1 configurations, among which the nearest configuration is identified (denoted by the holly red point $\mathbf{q}_1$). Finally, the average of all contiguous configurations around $\mathbf{q}_1$ is used to calculate penetration depth.

magnitude of penetration depth is smaller than the exact penetration depth. The configuration that realizes this penetration depth does not separate two objects. The conservative penetration depth calculated by our algorithm, shown as a blue curve, is always greater than the exact penetration depth (red).

Figure 6: Analytical, approximate and conservative penetration depths.

Computational Cost: In pre-processing, our algorithm takes nearly 40 seconds for training a binary classifier and obtaining the approximate boundary of obstacle region. There are more than 3000 support vectors in the classifier. It takes additional 293 seconds in pre-processing stage to generate a discretized representation. The runtime query of penetration depth takes 0.047 ms only.

5.2. Self-Colliding Models

In this section, we consider the scenarios with self-collisions. Fig. 7-(a) shows an example of self-collision between two links of an articulated model, where link $\mathcal{L}_3$ collides with the base link $\mathcal{L}_0$. If we only allow $\mathcal{J}_0$ and $\mathcal{J}_1$ to rotate, it is obvious that the exact penetration depth is constant for the query configuration shown in Fig. 7-(a). To compare with our results, we first rotate $\mathcal{J}_0$ only by a specified angle and compute its conservative penetration depth. The results are shown in 7-(b), where the horizontal axis is the angle of rotation and the vertical axis is the magnitudes of penetration depth.

In addition, we generate different query configurations by rotating $\mathcal{J}_1$. For a query configuration, we compute its conservative penetration depth. When $\mathcal{L}_2$ and its attached links move from one side of $\mathcal{L}_0$ to the other side, the penetration depth gradually increases and then decreases, as shown in Fig. 7-(c).

Figure 8: Self-collision experiment 2. (a) 4-DOF articulated model and self-collision configuration. (b) Penetration depth results for query configurations generated by rotating $\mathcal{J}_2$.

Fig. 8-(a) shows a more complicated scenario. We let three revolute joints $\mathcal{J}_0$, $\mathcal{J}_2$ and $\mathcal{J}_3$ rotate. In order to test our algorithm, we start from the configuration shown in Fig. 8-(a) where $\mathcal{L}_4$ collide with $\mathcal{L}_2$. We rotate $\mathcal{J}_2$ to generate some query configurations in a counterclockwise manner. Then we calculate penetration depth for these inter-penetration configurations. The results are shown in Fig.8-(b).

Computational Cost: In the first two scenarios, it takes 42 seconds for training approximate obstacle region and 355 seconds to generate a discretized representation. It takes 0.058 ms on average for each penetration depth query. In the three dimensional scenario, training approximate obstacle region takes 39 seconds and generating a discretized representation takes 138 seconds. Computing penetration depth takes 0.046 ms on average for each query.

5.3. Hybrid Joints

We also test our algorithm for an articulated model composed of hybrid joints. Here we consider both revolute and prismatic joints. In our experiment, we add a prismatic joint between

Figure 7: Self-collision experiment 1. (a) 3-DOF articulated model and self-collision configuration. J_0 and J_1 are active revolute joints. (b) Only rotating J_0 to generate query configurations and their corresponding penetration depth. (c) Only rotating J_1 to generate query configurations and their corresponding penetration depth.

$\mathcal{L}_0$ and $\mathcal{L}_1$ for the articulated model shown in Fig.1-(a). After adding one degree of freedom, its obstacle region becomes three dimensional space, two dimensions for rotation and one dimension for translation. We generate some query configurations by placing obstacle O at different locations. When obstacle O moves away from $\mathcal{A}$ along horizontal axis, penetration depth magnitude decreases, as shown in Fig. 9. To test the influence of translation component in penetration depth computation, we assign a weight factor to the prismatic joint (refer to Equation 5). Fig.9 shows penetration depth results under three different weights for prismatic joint.

The red line is obtained using the smallest weight. In this case, two objects tend to be separated by translation (prismatic joint) rather than rotation (revolute joints) because it consumes less energy. These penetration depth magnitudes are mainly contributed by translational component and they are smaller than the results given in Fig.6.

The blue line is obtained using the largest translational weight, so rotate joints tend to contribute more in penetration depth computation. In our experiment, the penetration depth magnitudes are similar to those results given in Fig.6.

Figure 9: Penetration depth for an articulated model with hybrid joints. The prismatic joint is assigned three different weight, ranging from 0.01 to 1.0. The smaller the weight is, the more the translational component contributes.

The green line is for the case using a medium translational weight. When the articulated model and obstacle have a deeper inter-penetration (i.e. larger penetration depth), their penetration depth is determined by both rotation and translation motion. In this case, their penetration depth magnitudes are greater than the translation dominated case (red line), but less than the rotation dominated case (blue line). However, for marginal inter-penetration, a translational component tends to contribute more than a rotational component. The resulting penetration depth tends to be equivalent to the one given in red, as shown in the portion where distance is greater than 1.6.

Computational Cost: In our experiment, it takes 26 seconds to train approximate obstacle region and 68 seconds to generate discretized obstacle region. The average query time for penetration depth is 0.03 ms.

6. Complex Articulated Models

In this section, we applied our algorithm to different complex articulated models.

6.1. Benchmarking Scenarios

PUMA and Cage: As shown in Fig.10-(a), a PUMA industrial robot sweeps into a cage and the two models penetrate each other during the motion. In-collision configurations and collision-free configurations are highlighted in light blue and green, respectively. The PUMA robot consists of 868 triangles and the cage consists of 432 triangles. The left image in Fig.10-(a) shows three active revolute joints. In our experiment, it takes 135 seconds to generate the approximation of obstacle regions and the number of support vectors $|SV| = 2746$. Computing penetration depth takes 1.09 ms on average per runtime query. For six active joints (shown in the right image of Fig.10-(a)), it takes 350 seconds to generate the approximation of obstacle regions and $|SV| = 8233$. It takes 2.25 ms on average to compute penetration depth.

Arc Welding Robot and Car: As shown in Fig.10-(b), an arc welding industrial robot works beside a car and the two models penetrate each other during the motion, where light blue highlights an in-collision configuration and green represents a

Figure 10: Conservative penetration depth for high-DOF articulated models. (a)-(c) The benchmarks on the left are 3-DOF experiments and on the right are 6-DOF experiments. (d) The benchmark on the left is a 3-DOF experiment and on the right is a 7-DOF experiment.

collision-free configuration. The arc welding robot consists of 3.7K triangles and the car consists of 19K triangles. In our experiment, it takes 117 seconds to generate the approximation of obstacle regions for three active joints. The number of support vectors $|SV| = 1656$. Computing conservative penetration depth takes 0.77 ms. When activating six active joints, it takes 382 seconds to generate the approximation of obstacle regions and 0.85 ms to compute penetration depth. $|SV| = 7887$.

Arc Welding Robot and Drill: As shown in Fig.10-(c), an arc welding industrial robot works beside a drill and the two models penetrate each other during the motion (grey). A collision-free configuration is highlighted in pink. The arc welding robot consists of 3.7K triangles and the drill consists of 34K triangles. It takes 62 seconds to generate the approximation of obstacle regions for three active joints and $|SV| = 3379$. Computing penetration depth takes 0.61 ms. For six active joints, it takes 383 seconds to generate the approximation of obstacle regions and computing penetration depth takes 1.04 ms. $|SV| = 7259$.

Spot Welding Robot and Car Body: As shown in Fig.10-(d), a spot welding industrial robot works around a car body and the two models penetrate each other during the motion (blue). A collision-free configuration is highlighted in green. The spot welding robot consists of 5.6K triangles and the car body consists of 4K triangles. The left of Fig.10-(d) is for activating three active joints, it takes 118 seconds to generate the approximation of obstacle regions and $|SV| = 3079$. Computing penetration depth takes 0.62 ms. The right of Fig.10-(d) is for activating seven active joints, it takes 497 seconds to generate the approximation of obstacle regions. Computing penetration depth takes 1.91 ms. This scenario is very complex, so it has more support vectors ($|SV| = 9254$) than others.

Hand Grasping: A robot hand grasps a glass (Figs.11). The hand has twelve joints, consisting of 22K triangles. The glass

consists of 3.2K triangles. A sequence of inter-penetration configurations are generated during the simulation and our algorithm is used to separate two objects. Our algorithm takes 403 seconds to generate the approximation of obstacle regions and 1.99 ms to compute penetration depth. $|SV| = 8245$ is fewer than the previous benchmark (spot welding robot and car body), as a glass model is not very complex. Note that we did not take account of any constraint that makes the hand hold the glass.

Motion Planning: We also apply our PD algorithm to a retraction-based probabilistic roadmap (PRM) planner[?]. We generate some samples in C-space and detect their collision states. If its state is collision-free, the sample will be added in the roadmap. If its state is in-collision, the sample will be used to find a configuration in free space based on our PD algorithm and a new collision-free configuration will be added in the roadmap. Our PD algorithm can be used to efficiently find these collision-free configurations for given collision configurations. As shown in Fig.12, planning a path for welding robot takes 127 seconds. In total, it generates 20181 samples, among which 3309 samples are computed using our PD algorithm.

6.2. Comparison and Discussion

In this section, we make some comparisons. As shown in Fig.13 and 14, we compare our conservative penetration depth against non-conservative penetration depth. Since the penetration depth queried using k -nearest neighbor does not guarantee to lie outside the obstacle regions, our conservative penetration depths are generally greater than non-conservative penetration depth. In some cases, the non-conservative penetration magnitudes are greater than conservative. That is because the query point lie far away from the configuration space boundary. We also gave the time statistics for the two algorithms, shown in

Figure 11: A 12-DOF robot hand grasps a glass. (a) An inter-penetration configuration is shown in two different views; (b) The robot hand and the glass are separated using our algorithm; (c) Conservative PD magnitude and runtime query time.

Fig.14. Both the algorithms achieved similar runtime performance.

[9] is the only work to approximate penetration depth for articulated models. Their work uses local contact space projection to iteratively find a local minimum point at the runtime. The query time includes successive perturbation to find start point and iterative contact space projection. Our query time mainly affected by the number of support vectors $|SV|$. In general, the runtime query is more efficient than [9] and does not suffer the iterative process. However, we need to spend more time in learning stage than theirs.

6.3. Time Complexity

Learning: The time spent in the learning stage can be estimated by $T = T_c \cdot N_s + T_L + T_R + T_D$, where $T_c = T_{col} \cdot N_L$ is

Figure 12: Screenshots of motion planning using PRM and our PD algorithm.

the time of collision detection (T_{col} is collision detection time for one articulated link and N_L is the number of links in an articulated model), N_s is the number of samples, T_L is the time to learn an initial approximation, T_R is the time for refinement and T_D is the time of generating discretized representation. $T_D = k \cdot N_{SV} \cdot T_{ccd}$, where k is a number specified by users (i.e. k nearest neighbors with opposite state for a given support vector), N_{SV} is the number of support vectors, T_{ccd} is the time of continuous collision detection for given two configurations.

Query: The time of runtime query is mainly determined by the number of support vectors.

7. Limitations and Conclusions

We presented an efficient algorithm to compute global penetration depth for articulated models. The performance gain is due to the use of machine learning techniques and the simplification of runtime query. The former dramatically reduces computational complexity and the latter avoids constrained optimization process. We carried out extensive experiments to demonstrate the efficiency of our algorithm.

Our algorithm has some limitations. Firstly, due to the difficulty to understand and obtain true penetration depth for high-DOF obstacle regions, we have not compared our conservative penetration with the ground truth for some complex benchmarks. Secondly, though our algorithm can guarantee the conservativeness of penetration depth, we still have difficulty deriving its upper bound. This deserves for the future research.

Besides, due to the importance of obstacle regions and penetration depth computation in many applications, it would be very interesting to characterize and visualize obstacle regions for high-DOF articulated models. Extending our algorithm to handle multiple articulated models would be another future direction. Analyzing complex configuration spaces that exhibit singularity is very difficult. However, the problem can be simplified if we treat the singular configurations as the samples in

C_{obs} . Applying machine learning techniques and configuration space approximation to these problem would make it possible, especially for high-DOF articulated models. Another direction for future research is to apply our algorithm to motion planning for narrow passages. At the moment, a tight theoretical error bound is not known for the learning algorithm used in our algorithm, but evaluating accuracy for specific applications will be another good topic for future work.

Appendix

Below are the pseudocode for training a classifier using SVM to obtain the discretized representation $D[\bar{C}_{obs}]$ (Algorithm 1) and for computing pPD (Algorithm 2).

In Algorithm 1, a training procedure takes an articulated model $\mathcal{A}$ and an obstacle O as input and produces a discrete boundary $D[\bar{C}_{obs}]$. The number of iterations in training refinement is governed by argument N . Q is a set of configurations obtained in sampling.

Algorithm 1 $\text{train}(\mathcal{A}, O, N)$

Output: [Discretized Approximation $D[\bar{C}_{obs}]$]

```

1:  $Q \leftarrow \text{sample}(\mathcal{A}, O)$ 
2: while  $N \neq 0$  do
3:    $f \leftarrow \text{svm}(Q)$ 
4:    $Q \leftarrow \text{refine}(\mathcal{A}, O, f, Q)$ 
5:    $N = N - 1$ 
6: end while
7:  $f \leftarrow \text{svm}(Q)$ 
8:  $D[\bar{C}_{obs}] \leftarrow \text{discretized}(f, Q)$ 
9: return  $D[\bar{C}_{obs}]$ 
```

Note that, in step 1, any sampling scheme can be used. In our implementation, we use uniform sampling.

In Algorithm 2, the configuration that realizes PD is determined using k -nearest neighbor search (denoted by procedure knn). Q_1 is a set of k configurations returned by the approximate knn procedure. The final penetration depth magnitude is calculated using Equations 4, 5 and 6.

Algorithm 2 $\text{computePD}(D[\bar{C}_{obs}], \mathbf{q}_0, k)$

Output: [PD magnitude]

```

1:  $Q_1 \leftarrow \text{knn}(D^+[\bar{C}_{obs}], \mathbf{q}_0, k)$ 
2:  $\mathbf{q}_1 \leftarrow \min(Q_1)$ 
3: return  $\text{PD} = \text{dist}(\mathbf{q}_0, \mathbf{q}_1)$ 
```

- [1] L. Zhang, Y. J. Kim, G. Varadhan, D. Manocha, Generalized penetration depth computation, *Computer-Aided Design* 39 (8) (2007) 625–638.
- [2] Y. J. Kim, M. C. Lin, D. Manocha, DEEP: Dual-space expansion for estimating penetration depth between convex polytopes, in: *International Conference on Robotics and Automation*, 2002, pp. 921–926.
- [3] G. van den Bergen, Proximity queries and penetration depth computation on 3D game objects, in: *Game Developers Conference*, 2001.
- [4] P. K. Agarwal, L. J. Guibas, S. Har-Peled, A. Rabinovitch, M. Sharir, Computing the penetration depth of two convex polytopes in 3d, in: *Scandinavian Workshop on Algorithm Theory*, 2000, pp. 328–338.

- [5] G. Nawratil, H. Pottmann, B. Ravani, Generalized penetration depth computation based on kinematical geometry, *Computer Aided Geometric Design* 26 (4) (2009) 425–443.
- [6] L. Zhang, Y. J. Kim, D. Manocha, A fast and practical algorithm for generalized penetration depth computation, in: *Robotics: Science and Systems*, 2007.
- [7] C. Je, M. Tang, Y. Lee, M. Lee, Y. J. Kim, Polydepth: Real-time penetration depth computation using iterative contact-space projection, *ACM Transactions on Graphics* 31 (1) (2012) 5:1–5:14.
- [8] J. Pan, X. Zhang, D. Manocha, Efficient penetration depth approximation using active learning, *ACM Transactions on Graphics (ACM SIGGRAPH ASIA 2013)* 32, article 191.
- [9] M. Tang, Y. J. Kim, Interactive generalized penetration depth computation for rigid and articulated models using object norm, *ACM Transactions on Graphics* 33 (1) (2014) Article No. 1.
- [10] X. Zhang, Y. J. Kim, D. Manocha, Continuous penetration depth, *Computer-Aided Design* 46 (2014) 3–13.
- [11] S. Cameron, R. Culley, Determining the minimum translational distance between two convex polyhedra, in: *IEEE International Conference on Robotics and Automation*, 1986, pp. 591–596, volume 3.
- [12] D. Dobkin, J. Hershberger, D. Kirkpatrick, S. Suri, Computing the intersection-depth of polyhedra, *Algorithmica* 9 (6) (1993) 518–533.
- [13] S. Cameron, Enhancing GJK: Computing minimum and penetration distance between convex polyhedra, in: *IEEE International Conference on Robotics and Automation*, 1997, pp. 3112–3117.
- [14] C. J. Ong, E. G. Gilbert, Growth distances: New measures for object separation and penetration, *IEEE Transactions on Robotics and Automation* 12 (6) (1996) 888–903.
- [15] P. Hachenberger, Exact minkowski sums of polyhedra and exact and efficient decomposition of polyhedra into convex pieces, *Algorithmica* 55 (2) (2009) 329–345.
- [16] Y. J. Kim, M. A. Otaduy, M. C. Lin, D. Manocha, Fast penetration depth computation for physically-based animation, in: *SIGGRAPH/Eurographics Symposium on Computer Animation*, 2002, pp. 23–31.
- [17] J.-M. Lien, Covering minkowski sum boundary using points with applications, *Computer Aided Geometric Design* 25 (8) (2008) 652–666.
- [18] J.-M. Lien, A simple method for computing minkowski sum boundary in 3d using collision detection, in: *Algorithmic Foundation of Robotics VIII*, Vol. 57 of Springer Tracts in Advanced Robotics, Springer Berlin / Heidelberg, 2009, pp. 401–415.
- [19] E. Guendelman, R. Bridson, R. Fedkiw, Nonconvex rigid bodies with stacking, *ACM Transactions on Graphics* 22 (3) (2003) 871–878.
- [20] S. Redon, M. C. Lin, A fast method for local penetration depth computation, *Graphical Tools* 8 (1) (2006) 63–70.
- [21] M. Tang, M. Lee, Y. J. Kim, Interactive hausdorff distance computation for general polygonal models, *ACM Transactions on Graphics* 28 (3) (2009) 74:1–74:9.
- [22] M. Tang, D. Manocha, M. A. Otaduy, R. Tong, Continuous penalty forces, *ACM Transactions on Graphics* 31 (4) (2012) 107:1–107:9.
- [23] M. Tang, Y. J. Kim, Interactive generalized penetration depth computation for rigid and articulated models using object form, Tech. rep., Department of Computer Science, Ewha Womans University (2012).
- [24] B. Wang, F. Faure, D. K. Pai, Adaptive image-based intersection volume, in: *SIGGRAPH*, 2012, pp. 97:1–9.
- [25] B. Heidelberger, M. Teschner, R. Keiser, M. Müller, M. H. Gross, Consistent penetration depth estimation for deformable collision response, in: *International Fall Workshop on Vision, Modeling and Visualization*, 2004, pp. 339–346.
- [26] J.-M. Lien, Covering minkowski sum boundary using points with applications, *Computer Aided Geometric Design* 25 (8) (2008) 652–666.
- [27] E. Larsen, S. Gottschalk, M. C. Lin, D. Manocha, Fast proximity queries with swept sphere volumes, in: *International Conference on Robotics and Automation*, 2000, pp. 3719–3726.
- [28] M. Muja, D. G. Lowe, Fast approximate nearest neighbors with automatic algorithm configuration, in: *International Conference on Computer Vision Theory and Application*, 2009, pp. 331–340.
- [29] F. Schwarzer, M. Saha, J.-C. Latombe, Motion Planning Kit, <http://ai.stanford.edu/~mitul/mpk/> (2006).
- [30] A. T. Miller, P. K. Allen, Graspit! a versatile simulator for robotic grasping, *Robotics & Automation Magazine, IEEE* 11 (4) (2004) 110–122.

Figure 13: Conservative penetration depth vs approximate penetration depth. Also refer to Fig.10 for corresponding benchmarks.

Figure 14: Computational costs for conservative penetration depth vs approximate penetration depth. Also refer to Fig.10 for corresponding benchmarks.

- [31] C.-C. Chang, C.-J. Lin, LIBSVM: A library for support vector machines, ACM Transactions on Intelligent Systems and Technology 2 (2011) 27:1–27:27, software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.