Free Space of Rigid Objects: Caging, Path Non-Existence, and Narrow Passage Detection

Anastasiia Varava[⋆], J. Frederico Carvalho[⋆], Florian T. Pokorny, Danica Kragic

KTH Royal Institute of Technology
`{varava,jfpbdc,fpokorny,dani}@kth.se`

Abstract. In this paper, we present an approach towards approximating configuration spaces of 2D and 3D rigid objects. The approximation can be used to identify caging configurations and establish path non-existence between given pairs of configurations. We prove correctness and analyse completeness of our approach. Using dual diagrams of unions of balls and uniform grids on $SO(3)$, we provide a way to approximate a 6D configuration space of a rigid object. Depending on the desired level of guaranteed approximation accuracy, the experiments with our single core implementation show runtime between $5 - 21$ s. and $463 - 1558$ s. Finally, we establish a connection between robotic caging and molecular caging from organic chemistry, and demonstrate that our approach is applicable to 3D molecular models.

Keywords: Caging · Path non-existence · Computational Geometry

1 Introduction

Understanding the global topological and geometric properties of the free space is a fundamental problem in several fields. In robotics, it can be applied to manipulation and motion planning. In computational chemistry and biology, it may be used to predict how molecules can restrict each other's mobility, which is important for sorting molecules, drug delivery, etc. [14, 22].

In robotic manipulation, the mobility of an object may be constrained by manipulators and/or obstacles. When the object cannot escape arbitrarily far from an initial configuration, we say that the object is *caged*. Formally, this means that it is located in a compact path-connected component of its free space. One of the biggest challenges in caging is verification – i.e., designing efficient algorithms which are able to provide theoretical guarantees that an object in a given configuration is caged. One approach towards caging verification relies on particular geometric and topological features of the object under consideration. This approach can be computationally efficient, but is limited to objects with certain shape properties, such as the existence of narrow parts or handles.

To deal with objects and obstacles of arbitrary shape, one needs to show that an object is located in a bounded path-connected component of its free space.

[⋆] The first two authors contributed equally.

Direct reconstruction of a 6D configuration space is computationally expensive, and to the best of our knowledge no fast and rigorous approach has been proposed so far. In this paper, we address this problem.

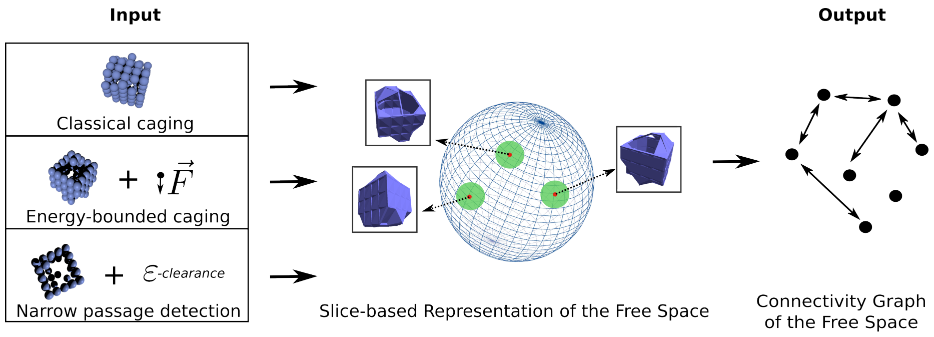
Input
Output
Classical caging
Energy-bounded caging
Narrow passage detection
$+ \vec{F}$
$+ \varepsilon$-clearance
Slice-based Representation of the Free Space
Connectivity Graph of the Free Space

Fig. 1: Diagram of our method for different application scenarios. We approximate the collision space of an object by choosing a finite set of fixed object's orientations and considering the corresponding projections of the collision space to $\mathbb{R}^n$ $(n \in \{2, 3\})$.

We consider caging a special case of the problem of proving path non-existence between a pair of configurations. To show that two configurations are disconnected, we construct an approximation of the object's collision space. Intuitively, we construct a set of projections of a subset of the object's configuration space to subspaces corresponding to fixed orientations of the object, see Fig. 1. By construction, our approximation is a proper subset of the object's collision space, which implies that when our algorithm reports that the two configurations are disconnected, then there is no path between them in the real free space. However, for the same reason, our algorithm is not guaranteed to find all possible caging configurations, since we do not reconstruct the entire collision space.

In [27], we presented our initial work on this approach without analyzing its theoretical properties and discussing practical details of its implementation. The core contributions of the present paper with respect to [27] can be summarized as follows: *(i)* we present a way to implement our algorithm using geometric constructions from [4] and [34], and run experiments in a 2D and 3D workspace; *(ii)* we provide an accurate bound for the displacement of a 3D object induced by a given rotation, and discuss how to practically discretize $SO(3)$ using a technique from [33]; *(iii)* we generalize our approach to narrow passage detection and construct δ-clearance configuration space approximations; *(iv)* we provide a correctness proof of our algorithm by showing that if in our approximation of the free space two configurations are disconnected, then they are disconnected in the actual free space; *(v)* we prove δ-completeness of the algorithm: if two configurations are disconnected, we can construct a good enough approximation of the free space to show that these configurations are either disconnected or connected by a narrow passage; *(vi)* we establish a connection between robotic caging and molecular caging in organic chemistry and discuss possible future research in this direction; as a proof of concept, we run our algorithm on a pair of molecules studied in [14] and obtain the same result as the authors in [14] obtained in a chemical laboratory.

2 Related Work

In manipulation, caging can be considered as an alternative to a force-closure grasp [9, 10, 18, 26], as well as an intermediate step on the way towards a form-closure grasp [21]. Unlike classical grasping, caging can be formulated as a purely geometric problem, and therefore one can derive sufficient conditions for an object to be caged. To prove that a rigid object is caged, it is enough to prove this for any subset (part) of the object. This allows one to consider caging a subset of the object instead of the whole object, and makes caging robust to noise and uncertainties arising from shape reconstruction and position detection.

The notion of a planar cage was initially introduced by Kuperberg in 1990 [5] as a set of n points lying in the complement of a polygon and preventing it from escaping arbitrarily far from its initial position. In robotics, it was subsequently studied in the context of point-based caging in 2D by Rimon and Blake [19], Pipattanasomporn and Sudsang [16], Vahedi and van der Stappen [28], and others. A similar approach has also been adopted for caging 3D objects. For instance, Pipattanasomporn and Sudsang [17] proposed an algorithm for computing all two-finger cages for non-convex polytopes. In [25, 29] the authors present a set of 2D caging-based algorithms enabling a group of mobile robots to cooperatively drag a trapped object to the desired goal.

In the above mentioned works fingertips are represented as points or spheres. Later, more complex shapes of caging tools were taken into account by Pokorny et al. [18], Stork et al. [23, 24], Varava et al. [26], Makita et al. [9, 10]. In these works, sufficient conditions for caging were derived for objects with particular shape features. Makapunyo et al. [12] proposed a heuristic metric for partial caging based on the length and curvature of escape paths generated by a motion planner. The authors suggested that configurations that allow only rare escape motions may be successful in practice.

We address caging as a special case of the path non-existence problem: an object is caged if there is no path leading it to an unbounded path-connected component. The problem of proving path non-existence has been addressed by Basch et al. [2] in the context of motion planning, motivated by the fact that most modern sampling-based planning algorithms do not guarantee that two configurations are disconnected, and rely on stopping heuristics in such situations [6]. Basch et al. prove that two configurations are disconnected when the object is 'too big' or 'too long' to pass through a 'gate' between them. There are also some related results on approximating configuration spaces of 2D objects. In [31], Zhang et al. use approximate cell decomposition and prove path non-existence for 2D rigid objects. They decompose a configuration space into a set of cells and for each cell decide if it lies in the collision space. In [13] McCarthy et al. propose a related approach. There, they randomly sample the configuration space of a planar rigid object and reconstruct its approximation as an alpha complex. They later use it to check the connectivity between pairs of configurations. This approach has been later extended to planar energy-bounded caging [8].

The problem of explicit construction (either exact or approximate) of configuration spaces has been studied for several decades in the context of motion planning, and a summary of early results can be found in [30]. In [7] the idea of slicing along the rotational axis was introduced. To connect two consecutive slices, the authors proposed to use the area swept by the robot rotating between two consecutive orientation values. In [32], this idea was extended to using both outer and inner swept areas to construct a subset and a superset of the collision space of polygonal robots. The outer and inner swept areas are represented as generalized polygons defined as the union and intersection of all polygons representing robot's shape rotating in a certain interval of orientation values, respectively. Several recent works propose methods for exact computation of configuration spaces of planar objects [3, 15]. In [3], a method towards exact computation of the boundary of the collision space is proposed. In [15], the authors explicitly compute the free space for complete motion planning.

As we can see, several approaches to representing configuration spaces of 2D objects, both exact and approximate, have been proposed and successfully implemented in the past. The problem however is more difficult if we consider a 3D object, as its configuration space is 6-dimensional. In the recent survey on caging [11], the authors hypothesise that recovering a 6D configuration space and understanding caged subspaces is computationally infeasible. We present a provably-correct algorithm for approximating 3D and 6D configuration spaces, which has a reasonable runtime on a single core of Intel Core i7 processor.

We study path-connectivity of the free space of 2D and 3D rigid objects. We do this by decomposing the configuration space into a finite set of lower dimensional slices. Although the idea of slicing is not novel and appears in the literature as early as in 1983 [7], recent advances in computational geometry and topology, as well as a significant increase in processing power, have made it easier to approximate a 6D configuration space on a common laptop. When dealing with slicing a 6D configuration space, we identify two main challenges: how to quickly compute 3D slices, and how to efficiently discretize the orientation space. For slice approximation, our method relies on simple geometric primitives — unions of balls, which makes our algorithm easy to implement and generalizable to 6D configuration spaces. Given a subset of an object, we determine how densely the set of orientations should be sampled in order to approximate the collision space of the object, and use dual diagrams of a union of balls [4] to approximate the free space. This way, we do not need to use generalized polygons, which makes previous approaches more difficult in 2D and very hard to generalize to 3D workspaces. For $SO(3)$ discretization, we use the method from [33], which provides a uniform grid representation of the space. The confluence of these factors result in overcoming the dimensionality problem without losing necessary information about the topology of the configuration space, and achieving both practical feasibility and theoretical guarantees at the same time.

Finally, our method does not require precise information about the shape of objects and obstacles, and the only requirement is that balls must be located strictly inside them, which makes our approach robust to noisy and incomplete sensor data. The focus of this paper is on proving path non-existence rather than complete motion planning. In practice, it is often enough to have a rough approximation of the free space to find its principal connected components, which is easier to compute than its exact representation.

3 Free Space Decomposition

Since our work is related both to object manipulation and motion planning, we use the general term 'object' without loss of generality when talking about objects and autonomous rigid robots (e.g., disc robots) moving in n-dimensional workspaces[1]

Definition 1. *A* rigid object *is a compact connected non-empty subset of* $\mathbb{R}^n$. *A* set of obstacles *is a compact non-empty subset of* $\mathbb{R}^n$.

We want to represent both the object and the obstacles as a set of $n-$dimensional balls. Therefore, we do not allow them to have 'thin parts'. Formally, we assume that they can be represented as regular sets [20]:

Definition 2. *A set U is* regular *if it is equal to the closure of its interior:* $U = \mathrm{cl}(\mathrm{int}(U))$.

We approximate both the obstacles and the object as unions of balls lying in their interior, $\mathcal{S} = \{B_{R_1}(X_1), \ldots, B_{R_n}(X_n)\}$ and $\mathcal{O} = \{B_{r_1}(Y_1), \ldots, B_{r_m}(Y_m)\}$ of radii $R_1, \ldots, R_n$ and $r_1, \ldots, r_m$ respectively.

Let $\mathcal{C}(\mathcal{O}) = SE(n)$ denote the configuration space of the object. We define its *collision space*[2] $\mathcal{C}^{col}(\mathcal{O})$ as the set of the objects configurations in which the object penetrates the obstacles:

Definition 3. $\mathcal{C}^{col}(\mathcal{O}) = \{c \in \mathcal{C} \mid [\mathrm{int}\, c(\mathcal{O})] \cap [\mathrm{int}\, \mathcal{S}] \neq \emptyset\}$, *where $c(\mathcal{O})$ denotes the object in a configuration c. The* free space $\mathcal{C}^{free}(\mathcal{O})$ *is the complement of the collision space:* $\mathcal{C}^{free}(\mathcal{O}) = \mathcal{C}(\mathcal{O}) - \mathcal{C}^{col}(\mathcal{O})$.

In [27], we suggested that configuration space decomposition might be a more computationally efficient alternative to its direct construction. Namely, we represent the configuration of an object space as a product $\mathcal{C}(\mathcal{O}) = \mathbb{R}^n \times SO(n)$, and consider a finite covering of $SO(n)$ by open sets (this is always possible, since $SO(n)$ is compact): $SO(n) = \bigcup_{i \in \{1,\ldots,s\}} U_i$. We recall the notion of a slice, introduced in [27]:

Definition 4. *A* slice *of the configuration space $\mathcal{C}(\mathcal{O})$ of a rigid object, is a subset of $\mathcal{C}(\mathcal{O})$ defined as follows: $Sl_U(\mathcal{O}) = \mathbb{R}^n \times U$, where U is an subset of $SO(n)$.*

We denote a slice of the collision (free) space by $Sl_U^{col}(\mathcal{O})$ ($Sl_U^{free}(\mathcal{O})$). For each slice we construct an approximation $aSl_U^{col}(\mathcal{O})$ of its collision space in such a way that our approximation lies inside the real collision space of the slice, $aSl_U^{col}(\mathcal{O}) \subset Sl_U^{col}(\mathcal{O})$.

[1] Throughout the paper $n \in \{2, 3\}$.
[2] A theoretical analysis of different ways to define the free space can be found in [20].

This way, we approximate the entire collision space by a subset $a\mathcal{C}^{col}(\mathcal{O})$:

$$a\mathcal{C}^{col}(\mathcal{O}) = \left(\bigcup_{i \in \{1,\dots,s\}} aSl^{col}_{U_i}(\mathcal{O}) \right) \subset \mathcal{C}^{col}(\mathcal{O})$$

An object's $\varepsilon-$core. First of all observe that by Def. 3, if a subset $\mathcal{O}'$ of an object $\mathcal{O}$ placed in configuration $c \in \mathcal{C}(\mathcal{O})$ is in collision, then the entire object $\mathcal{O}$ is in collision. Therefore, the collision space of $\mathcal{O}$ is completely contained within the collision space of $\mathcal{O}'$. This allows us to make the following observation:

Observation 1 *Consider an object $\mathcal{O}$ and a set of obstacles $\mathcal{S}$. Let $c_1, c_2 \in \mathcal{C}^{free}(\mathcal{O})$ be two collision-free configurations of the object. If there is no collision-free path between these configurations for its subset $\mathcal{O}' \subset \mathcal{O}$, then there is no collision-free path connecting these configurations for $\mathcal{O}$.*

Therefore, if some subset $\mathcal{O}'$ of $\mathcal{O}$ in configuration c is caged, then the entire object $\mathcal{O}$ in the same configuration c is caged. This means that if we construct aSl^{col}_U in such a way that for any configuration $c \in aSl^{col}_U$ there exists a subset $\mathcal{O}'$ of $c(\mathcal{O})$ such that $\mathcal{O}'$ is in collision, then $c(\mathcal{O})$ is also in collision. In [27] we defined an $\varepsilon-$core of an object as follows:

Definition 5. *The ε-core of an object $\mathcal{O}$ is the set $\mathcal{O}_\varepsilon$ comprising the points of $\mathcal{O}$ which lie at a distance*[3] *of at least ε from the boundary of $\mathcal{O}$: $\mathcal{O}_\varepsilon = \{p \in \mathcal{O} \mid d(p, \partial\mathcal{O}) \geq \varepsilon\}$.*

Now, for an object $\mathcal{O}$ and its ε-core $\mathcal{O}_\varepsilon$, we write $\mathcal{O}^\phi$ and $\mathcal{O}^\phi_\varepsilon$ respectively to mean that their orientation is fixed at $\phi \in SO(n)$. So, let $\mathcal{C}^{col}(\mathcal{O}^\phi_\varepsilon)$ denote the collision space of $\mathcal{O}_\varepsilon$ with a fixed orientation ϕ. Note that since the orientation is fixed, we can identify $\mathcal{C}^{col}(\mathcal{O}^\phi_\varepsilon)$ with a subset of $\mathbb{R}^n$.

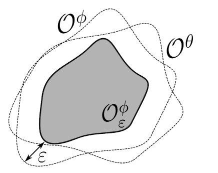

Fig. 2: An $\varepsilon-$core remains inside the object when we slightly rotate it

In [27], we showed that for an object $\mathcal{O}$, $\varepsilon > 0$ and a fixed orientation $\phi \in SO(n)$ there exists a non-empty neighbourhood $U(\phi, \varepsilon)$ of ϕ such that for any $\theta \in U(\phi, \varepsilon)$, $\mathcal{O}^\phi_\varepsilon$ is contained in $\mathcal{O}^\theta$, see Fig. 2.

We approximate the collision space as follows: we pick an $\varepsilon > 0$ and a set of orientation values $\{\phi_1, \dots, \phi_s\}$ so that $U(\phi_1, \varepsilon), \dots, U(\phi_s, \varepsilon)$ cover $SO(n)$, and so that for any $\theta \in U(\phi_i, \varepsilon)$ we have $\mathcal{O}^{\phi_i}_\varepsilon \subset \mathcal{O}^\theta$.

Finally, for each $\phi_i \in \{\phi_1, \dots, \phi_s\}$, we compute collision space slice approximations $aSl^{col}_{U(\phi_i,\varepsilon)}$ as the collision space of $\mathcal{O}^{\phi_i}_\varepsilon$, $aSl^{col}_{U(\phi_i,\varepsilon)} = \mathcal{C}^{col}(\mathcal{O}^{\phi_i}_\varepsilon) \times U(\phi_i, \varepsilon)$ which results in our approximation of the collision space: $a\mathcal{C}^{col}_\varepsilon(\mathcal{O}) = \bigcup_{\phi_i} aSl^{col}_{U(\phi_i,\varepsilon)}$.

Discretization of $SO(n)$: Elements of $SO(n)$ can be seen as parametrizing rotations in $\mathbb{R}^n$, so for any $q \in SO(n)$ we define R_q as the associated rotation. Let $D(\mathrm{R}_q)$ denote the maximal *displacement* of any point $p \in \mathcal{O}$ after applying R_q, i.e. $D(\mathrm{R}_q) = \max_{p \in \mathcal{O}}(d(p, \mathrm{R}_q(p)))$, then $\mathcal{O}_\varepsilon \subset \mathrm{R}_q(\mathcal{O})$ if $D(\mathrm{R}_q) < \varepsilon$.

[3] By distance here we mean Euclidean distance in $\mathbb{R}^n$

In [27] we have derived the following upper bound for the displacement of a two-dimensional object: $D(\mathrm{R}_q) \leq 2|\sin(q/2)| \cdot \mathrm{rad}(\mathcal{O})$ assuming that we rotate the object around its geometric center and $\mathrm{rad}(\mathcal{O})$ denotes the maximum distance from it to any point of the object, and q is the rotation angle. In the two-dimensional case, discretization of the rotation space is trivial: given an ε we pick $q \in SO(2)$ such that $D(\mathrm{R}_q) < \varepsilon$, and compute a set of orientation samples $\{\phi_1 = 0, \phi_2 = 2q, \dots, \phi_s = 2(s-1)q\}$, where $s = \lceil \pi/q \rceil$. This gives us a covering $\{U(\phi_1, \varepsilon), \dots, U(\phi_s, \varepsilon)\}$ of $SO(2)$, where for each $i \in \{1, \dots, s\}$ we define $U(\phi_i, \varepsilon) = [\phi_i - q, \phi_i + q]$[4].

Let us now discuss the three-dimensional case. Similarly to the previous case, our goal is to cover $SO(3)$ with balls of fixed radius. To parametrize $SO(3)$ we use unit quaternions. For simplicity, we extend the real and imaginary part notation from complex numbers to quaternions, where $\Re q$ and $\Im q$ denote the real and "imaginary" parts of the quaternion q. Further, we identify $\Im q = q_i i + q_j j + q_k k$ with the vector $(q_i, q_j, q_k) \in \mathbb{R}^3$; and we write $\bar{q}$, and $|q|$ to mean the conjugate $\Re q - \Im q$ and the norm $\sqrt{q\bar{q}}$, respectively. We use *angular distance* [33] to define the distance between two orientations: $\rho(x, y) = \arccos(|\langle x, y \rangle|)$, where x and y are two elements of $SO(3)$ represented as unit quaternions which are regarded as vectors in $\mathbb{R}^4$, and $\langle x, y \rangle$ is their inner product. A unit quaternion q defines a rotation R_q as the rotation of angle $\theta_q = 2\cos^{-1}(\Re q)$ around axis $w_q = \frac{\Im q}{|\Im q|}$. This allows one to calculate the displacement of the rotation $D(q) = D(R_q)$ as: $D(q) = 2\sin(\frac{\theta_q}{2}) = 2\sin(\cos^{-1}(\Re q)) = 2|\Im q|$

Define the angle distance from a point to a set $S \subseteq SO(3)$ in the usual way as $\rho(S, x) = \min_{y \in S} \rho(y, x)$. In [33], the authors provide a deterministic sampling scheme to minimize the *dispersion* $\delta(S) = \max_{x \in SO(3)} \rho(S, x)$. Intuitively, the dispersion of a set $\delta(S)$ determines how far a point can be from S, and in this way it determines how well the set S covers $SO(3)$. Now, assume we are given a set of samples $S \subseteq SO(3)$ such that $\delta(S) < \delta$. Then for any point $p \in SO(3)$ denote $D(p\bar{S}) = \max_{q \in S} D(p\bar{q})$, we want to show that in these conditions, there exists some small ε such that $\max_{p \in SO(3)} D(p\bar{S}) < \varepsilon$. This would imply that there exists some $\delta' > \delta$ such that if we take $U = \{q \in SO(3) \mid \rho(1, q) < \delta'\}$, then denoting $U_p = \{qp \mid q \in U\}$, the family $\{U_p\}_{p \in S}$ fully covers $SO(3)$ and for any $q \in U_p$ satisfies $D(q\bar{p}) < \varepsilon$.

Now, Proposition 1 allows us to establish the relation between the distance between two quaternions and displacement associated to the rotation between them.

Proposition 1. *Given two unit quaternions p, q, the following equation holds:*

$$D(p\bar{q}) = 2\sin(\rho(p, q)). \tag{1}$$

The proof of Proposition 1 can be found in the supplementary material.

This means that if we want to cover $SO(3)$ with patches U_i centered at a point p_i such that $D(R_{q\bar{p}_i})$ is smaller than some ε, we can use any deterministic

[4] This is a cover by closed sets, but given $q' > q$ satisfying $D(R_q) < \varepsilon$ we can use instead $U(\phi_i, \varepsilon) = (\phi_i - q', \phi_i + q')$ which esults on the same graph.

sampling scheme on $SO(3)$ (e.g. [33]) to obtain a set S with dispersion $\delta(S) < \arcsin(\frac{\varepsilon}{2})$.

Note that if $S \subseteq SO(3)$ has a dispersion $\delta(S) < \arcsin(\frac{\varepsilon}{2})$ then the family of patches $\{U(s,\varepsilon) \mid s \in S\}$ forms a cover of $SO(3)$.

Finally, given such a cover of $SO(3)$, recall that we want to approximately reconstruct the full free space of the object $\mathcal{C}^{free}$ as the union of slices $aSl^{free}_{U(s,\varepsilon)}$. This requires us to test the whether two slices overlap. To make this efficient, we create a *slice adjacency graph*, which is simply a graph with vertices S and edges (s,s') if $U(s,\varepsilon) \cap U(s',\varepsilon) \neq \emptyset$.

To compute this graph, we use the fact that $U(s,\varepsilon) = \{p \in SO(3) | \rho(s,p) < \delta\}$ for some δ, and so if two slices $U(s,\varepsilon), U(s',\varepsilon)$ are adjacent, there must exist some p such that $\rho(p,s), \rho(p,s') < \delta$, which implies $\rho(s,s') < \rho(s,p) + \rho(s',p) < 2\delta$. We use this fact to calculate the slice adjacency graph in Alg. 1.

Algorithm 1: ComputeAdjacentOrientations

```
   input : $S$ /* Set of points in $SO(3)$.                 */
   output: $G$ /* Patch adjacency graph.                    */
1  $V \leftarrow S$;
2  $\delta \leftarrow \delta(S)$;
3  $T \leftarrow KDTree(V)$;
4  $E \leftarrow \{\}$;
5  for $p \in V$ do
6      $P \leftarrow T.query(p, dist \leq 2\sin(\delta) + \tau) \setminus \{p\}$ ;
7      $P \leftarrow L \cup T.query(-p, dist \leq 2\sin(\delta) + \tau)$;
8      for $q \in L$ do
9          add $(p,q)$ to $E$;
10     end
11 end
12 return $G = (V, E)$;
```

The algorithm starts by setting $V = S$, as this is the set of vertices in the graph, and we put these vertices in a $KDTree$ in order to quickly perform nearest neighbor queries. Now, to compute the set of edges E, we locate for each $p \in S$ the points at an angle distance smaller than 2δ [5] in line 6. Finally, in line 7 we also add edges to the points at an angle distance smaller than 2δ of $-p$, as both p and $-p$ represent the same orientation.

One of the prerequisites of Alg. 1 is the availability of an estimate of the dispersion of $\delta(S)$. In our implementation we used the algorithm in [33] to compute S where the authors provide a provable upper bound for dispersion. However because this is an upper bound, and we want a tight estimate of the dispersion, in our implementation we employed a random sampling method to estimate the dispersion. At each step we draw a uniform sample p and compute its two nearest neighbors q, q'. We estimate the displacement at point p to be half the (angle) distance between q and q'. The final estimate is taken to be the maximum over the estimate for every sample taken.

[5] Since the $KDTree$ T uses the Euclidean distance in $\mathbb{R}^4$ we employ the formula $\|p - q\| = 2\sin(\frac{\rho(p,q)}{2})$.

Construction of Slices. In [27], we derive the following representation for the collision space of $\mathcal{O}_{\varepsilon}^{\phi_i}$: $\mathcal{C}^{col}(\mathcal{O}_{\varepsilon}^{\phi}) = \bigcup_{i,j}(B_{R_j+r_i-\varepsilon}(X_j - \overline{GY_i}))$, where G is the origin chosen as the geometric center of the object, and $\overline{GY_i}$ denotes the vector from G to Y_i. Now, let us discuss how we construct path-connected components of $\mathcal{C}^{free}(\mathcal{O}_{\varepsilon}^{\phi})$, see Alg. 2.

For this, we first need to approximate the complement of $\mathcal{C}^{col}(\mathcal{O}_{\varepsilon}^{\phi})$, which is represented as a finite collection of balls. We do this by constructing its *dual diagram* [4]. A dual diagram $Dual(\bigcup B_i)$ of a union of balls $\bigcup B_i$ is a finite collection of balls such that the complement of its interior is contained in and is homotopy equivalent to $\bigcup B_i$. It is convenient to approximate $\mathcal{C}^{free}(\mathcal{O}_{\varepsilon}^{\phi})$ as a dual diagram of collision space balls for several reasons. First, balls are simple geometric primitives which make intersection checks trivial. Second, the complement of the dual diagram is guaranteed to lie strictly inside the collision space, which provides us with a conservative approximation of the free space. Finally, homotopy equivalence between its complement and the approximate collision space implies that our approximation preserves the connectivity of the free space that we want to approximate. Another advantage of a dual diagram is that it is easy to construct.

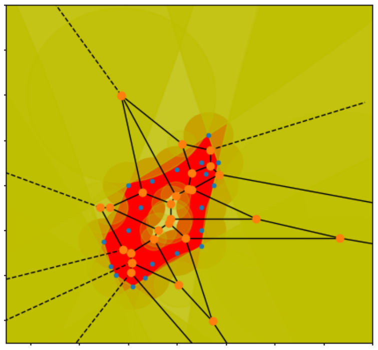
Fig. 3: Dual diagram of a set of circles. Red circles with blue centers represent the collision space, the yellow circles approximate the free space. The black edges form the associated weighted Voronoi diagram.

First, recall that a weighted Voronoi diagram is a special case of a Voronoi diagram, where instead of Euclidean distance between points a weighted distance function is used. In our case, the weighted distance of a point $x \in \mathbb{R}^n$ from $B_{r_i}(z_i)$ is equal to $d_w(x, B_{r_i}(z_i)) = ||x - z_i|| - r_i^2$. Consider a weighted Voronoi diagram of a set of balls $\bigcup_i B_{r_i}(z_i)$. For each vertex y_j of the Voronoi diagram, let $B_{q_j}(y_j)$ whose radius q_j is equal to the square root of the weighted distance of y_j to any of the four (three in the two-dimensional case) balls from $\bigcup_i B_{r_i}(z_i)$ generating y_j. Then, take each infinite edge of the Voronoi diagram, and add an degenerate "infinitely large" ball (a half-space) with center at the infinite end of this edge, see Fig. 3. After constructing the dual diagram, it is easy to compute connected components of the slice. Note that there always is one infinite component corresponding to the outside world, and caging configurations are contained in bounded components.

The Connectivity Graph. A naive algorithm described in [26] requires us to first compute all the slices and their connected components, and only after that compute the connectivity graph. This strategy leads to suboptimal usage of memory — in practice, when one needs to compute around 30000 slices, storing all their free space representations at the same time is not feasible on a common laptop. However, we notice that this is not necessary, since the representation of

Algorithm 2: Compute-Slice-Connectivity

```
   Data: A union of $d$-dimensional balls $\mathcal{C}^{col}(\mathcal{O}_\varepsilon^\phi)$
   Result: A set of connected components $aC_0, \dots, aC_n$
1  $V(\mathcal{C}^{col}(\mathcal{O}_\varepsilon^\phi)) \leftarrow$ Weighted-Voronoi-Diagram($\mathcal{C}^{col}(\mathcal{O}_\varepsilon^\phi)$)
2  $a\mathcal{C}^{free} \leftarrow$ Dual-Diagram($V(\mathcal{C}^{col}(\mathcal{O}_\varepsilon^\phi))$)
3  $aC_0 \leftarrow$ Compute-Infinite-Component()
4  $i \leftarrow 0$
5  foreach $ball \in a\mathcal{C}^{free}$ do
6      if Connected-Component(ball) $= \emptyset$ then
7          $i \leftarrow i + 1$
8          $aC_i \leftarrow$ Compute-Component($ball$)
9      end
10 end
11 return $\{aC_0, \dots, aC_n\}$
```

each slice is needed only to check whether its connected components overlap with the connected components of adjacent slices. Therefore, in our implementation each slice is deleted as soon as the connectivity check between it and the slices adjacent to it is performed.

Let us now discuss how we check whether two connected components overlap. Let $\mathcal{G}(a\mathcal{C}_\varepsilon^{col}(\mathcal{O})) = (V, E)$ be a graph approximating the free space. The vertices of $\mathcal{G}$ correspond to the connected components $\{aC_1^i, \dots, aC_{n_i}^i\}$ of each slice, $i \in \{1, \dots, s\}$, and are denoted by $v = (aC, U)$, where aC and U are the corresponding component and orientation interval. Two vertices representing components $C_p \subset aSl_{U_i}^{free}$ and $C_q \subset aSl_{U_j}^{free}$, $i \neq j$, are connected by an edge if the object can directly move between them. For that, both the sets U_i, U_j, and aC_q, aC_p must overlap: $U_i \cap U_j, aC_q \cap aC_p \neq \emptyset$. $\mathcal{G}(a\mathcal{C}_\varepsilon^{col}(\mathcal{O}))$ approximates the free space of the object: if there is no path in $\mathcal{G}(a\mathcal{C}_\varepsilon^{col}(\mathcal{O}))$ between the vertices associated to the path components of two configurations c_1, c_2, then they are disconnected in $\mathcal{C}^{free}(\mathcal{O})$. Finally, to check whether two connected components in adjacent slices intersect, we recall that they are just finite unions of balls. Instead of computing all pairwise intersections of balls, we approximate each ball by its bounding box and then use the CGAL implementation of Zomorodian's algorithm [34], which efficiently computes the intersection of two sequences of three-dimensional boxes. Every time it finds an overlapping pair of boxes, we check whether the respective balls also intersect.

Remark 1 *Recall that in each slice the $a\mathcal{C}^{free}(\mathcal{O}_\varepsilon^{\phi_i})$ are constructed as the dual of the collision space $\mathcal{C}^{col}(\mathcal{O}_\varepsilon^{\phi_i})$, which entails that $a\mathcal{C}^{free}(\mathcal{O}_\varepsilon^{\phi_i})$ has the same connectivity as $\mathcal{C}^{free}(\mathcal{O}_\varepsilon^{\phi_i})$. However, it also entails that any connected component of $a\mathcal{C}^{free}(\mathcal{O}_\varepsilon^{\phi_i})$ partially overlaps with the collision space $\mathcal{C}^{col}(\mathcal{O}_\varepsilon^{\phi_i})$. This means that for two connected components $C_j^i, C_{j'}^{i'}$ of adjacent slices which do not overlap, it may occur that the corresponding approximations $aC_j^i, aC_{j'}^{i'}$ do overlap. In this case the resulting graph $\mathcal{G}(a\mathcal{C}_\varepsilon^{col}(\mathcal{O}))$ would contain an edge between the corresponding vetices. This effect can be mitigated by verifying whether the overlap between the approximations occurs within the collision space of both slices. This can be done for example by covering the intersection $aC_j^i \cap aC_{j'}^{i'}$*

with a union of balls and checking if it is contained inside the collision space $\mathcal{C}^{col}(\mathcal{O}_{\varepsilon}^{\phi_i}) \cup \mathcal{C}^{col}(\mathcal{O}_{\varepsilon}^{\phi_{i'}})$.

Existence of δ-clearance paths. For safety reasons, in path planning applications a path is often required to keep some minimum amount of clearance to obstacles. The notion of clearance of an escaping path can also be applied to caging: one can say that an object is partially caged if there exist escaping paths, but their clearance is small and therefore the object is unlikely to escape.

Consider a superset of our object $\mathcal{O}$, defined as a set of points lying at most at distance δ from $\mathcal{O}$, and let us call it a *δ-offset* of the object: $\mathcal{O}_{+\delta} = \{p \in \mathbb{R}^n \mid d(p, \mathcal{O}) \leq \delta\}$. Now observe that two configurations are δ-connected if and only if there exists a collision-free path connecting these configurations in the configuration space of the δ-offset $\mathcal{O}_{+\delta}$ of the object. This means that two configurations c_1 and c_2 are not δ-connected in $\mathcal{C}^{free}(\mathcal{O})$ if and only if they are not path-connected in $\mathcal{C}^{free}(\mathcal{O}_{+\delta})$. Therefore, to understand the $\delta-$connectivity of the free space it is enough to compute path-connected components as previously described, for the δ-offset $\mathcal{O}_{+\delta}$.

One application of this observation is narrow passage detection. One can use our free space approximation to identify narrow passages: if two configurations are disconnected in $a\mathcal{C}_{\varepsilon+\delta}^{free}(\mathcal{O}_{+\delta})$, but connected in $a\mathcal{C}_{\varepsilon}^{free}(\mathcal{O})$, then they are connected by a narrow passage of width at most δ. Our approximation then can be used to localize this passage, so that probabilistic path planning algorithm can sample in this location.

Furthermore, we can view δ as the level of accuracy of the approximation: if $\delta = 0$, and our algorithm reports path non-existence, then this result is provably correct; for $\delta > 0$, we can guarantee non-existence of paths whose clearance does not exceed δ. This means that depending on the desired level of accuracy one can construct a coarse approximation of the free space by looking at the path connectivity of $a\mathcal{C}_{\varepsilon+\delta}^{free}(\mathcal{O}_{+\delta})$. This approximation requires fewer slices, and Sec. 5 demonstrates that this significantly reduces the computation time.

4 Theoretical Properties of Our Approach

In this section, we discuss correctness, completeness and computational complexity of our approach. The proofs of Propositions 2 and 3, as well as the discussion about computational complexity, can be found in the supplementary material. First of all, let us show that our algorithm is correct: i.e., if there is no collision-free path between two configurations in our approximation of the free space, then these configurations are also disconnected in the actual free space.

Proposition 2 (correctness). *Consider an object $\mathcal{O}$ and a set of obstacles $\mathcal{S}$. Let c_1, c_2 be two collision-free configurations of the object. If c_1 and c_2 are not path-connected in $\mathcal{G}(a\mathcal{C}_{\varepsilon}^{free}(\mathcal{O}))$, then they are not path-connected in $\mathcal{C}^{free}(\mathcal{O})$.*

Now, we show that if two configurations are not path-connected in $\mathcal{C}^{free}(\mathcal{O})$, we can construct an approximation of $\mathcal{C}^{free}(\mathcal{O})$ in which these configurations are either disconnected or connected by a narrow passage.

Proposition 3 **(δ-completeness).** *Let c_1, c_2 be two configurations in $\mathcal{C}^{free}(\mathcal{O})$. If they are not path-connected in $\mathcal{C}^{free}(\mathcal{O})$, then for any $\delta > 0$ there exists $\varepsilon > 0$ such that the corresponding configurations are not path-connected in $\mathcal{G}(a\mathcal{C}_{\varepsilon}^{free}(\mathcal{O}_{+\delta}))$, where the graph is produced according to the procedure outlined in Rem. 1.*

Computational complexity. The complexity of the first part of the algorithm (slice construction) is $O(n^2m^2)$, where n and m are the number of balls in the object's and the obstacle's spherical representation; the complexity of the second part (connectivity graph construction) is $O(sl(b \log^3(b) + k))$, where s is the number of slices, l is the number of neighbours of each slice, b is the maximum number of balls in each pair of connected components, and k the number of pairwise intersections of the balls. The details can be found in the supplementary material.

5 Examples and Experiments

We implemented our algorithm for 2D and 3D objects. All experiments were run on a single core of a computer with Intel Core i7 processor and 16GB GPU.

Different Approximations of the Workspace. In this experiment, we consider how the accuracy of a workspace spherical approximation affects the output and execution time of the algorithm, see Fig. 4. For our experiments, we generate

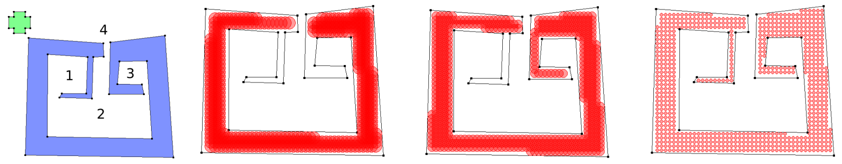

Fig. 4: In the left we present a set of 2D obstacles in blue, and an object in green. The numbers represent the connected components of the free space. In red are three approximations of the obstacles by sets of balls of radius 15, 10, and 4, respectively. Note that the smaller the radius the more features from the original configuration space are preserved.

a workspace and an object as polygons, and approximate them with unions of balls of equal radii lying strictly inside the polygons. Note that the choice of the radius is important: when it is small, we get more balls, which increases the computation time of our algorithm; on the other hand, when the radius is too large, we lose some important information about the shape of the obstacles, because narrow parts cannot be approximated by large balls, see Fig. 4. We consider a simple object whose approximation consists of 5 balls. We run our algorithm for all the 3 approximations of the workspace, and take 5 different values of ε, see Tab. 1. We can observe that as we increase the ε the computation time decreases. This happens because we are using fewer slices. However, we can also observe that when the ε is too large, our approximation of the collision space becomes too small, and we are not able to find one connected component (see the last column of Tab. 1).

ε	$R = 15$	$R = 10$	$R = 4$
$0.30 \cdot r$	2 c. 741 ms	3 c. 1287 ms	4 c. 1354 ms
$0.33 \cdot r$	2 c. 688 ms	3 c. 1208 ms	4 c. 1363 ms
$0.37 \cdot r$	2 c. 647 ms	3 c. 1079 ms	4 c. 1287 ms
$0.40 \cdot r$	2 c. 571 ms	3 c. 986 ms	3 c. 1156 ms
$0.43 \cdot r$	2 c. 554 ms	3 c. 950 ms	3 c. 1203 ms

Table 1: We report the number of path-connected components we found and the computation time for each case. When we were using our first approximation of the workspace, we were able to distinguish only between components 4 and 2 (see Fig. 4), and therefore prove path non-existence between them. For a more accurate approximation, we were also able to detect component 3. Finally, the third approximation of the workspace allows us to prove path non-existence between every pair of the four components.

Different Types of 3D Caging. As we have mentioned in Sec. 2, a number of approaches towards 3D caging is based on identifying shape features of objects and deriving sufficient caging conditions based on that. By contrast, our method is not restricted to a specific class of shapes, and the aim of this section is to consider examples of objects of different types and run our algorithm on them. The examples are depicted on Fig. 5, and Tab. 2 reports execution time for different resolutions of the $SO(3)$ grid.

#slices	rings (320)		narrow part (480)		surrounding (266)		gravity (348)		molecules (2268)	
576	5 s	$\delta = 2.4$	6 s	$\delta = 5.9$	10 s	$\delta = 1.2$	4 s	$\delta = 1.2$	21 s	$\delta = 1.02$
4608	49 s	$\delta = 0.7$	70 s	$\delta = 2.4$	108 s	$\delta = 0.4$	41 s	$\delta = 0.4$	181 s	$\delta = 0.34$
36864	540 s	$\delta = 0.0$	785 s	$\delta = 0.0$	1073 s	$\delta = 0.0$	463 s	$\delta = 0.0$	1558 s	$\delta = 0.0$

Table 2: Results from running 3D experiments in the 4 different scenarios described in Fig. 5 and the molecular model in Fig. 6 using 3 different resolutions for the $SO(3)$ grid. The number of balls used to approximate the collision space of each model is indicated in parenthesis next to the model name. We report the computational time and the value of guaranteed clearance δ for each case .

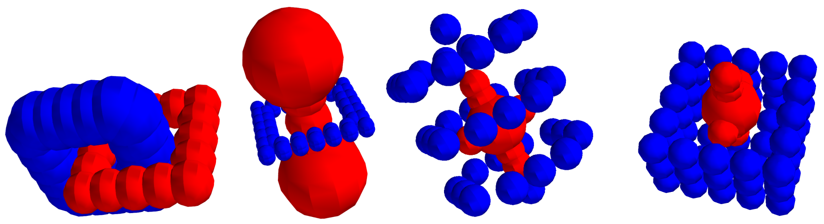

Fig. 5: Different 3D caging scenarios. From left to right: linking-based caging *linking-based caging* [18, 24, 23], *narrow part-based caging* [27, 35], *surrounding-based caging*, and *caging with gravity* [8]. In linking-based and narrow part-based caging the obstacles form a loop (not necessarily closed) around a handle or narrow part of the object. In surrounding-based the object is surrounded by obstacles so as not to escape. In caging with gravity, gravity complements the physical obstacles in restricting the object's mobility. Here, we set a fixed level of the potential energy that the object can achieve and consider every configuration with higher potential energy as "forbidden".

Observe that depending on the desired accuracy of the resulting approximation, different numbers of slices can be used. In our current implementation, we precompute grids on $SO(3)$ using the algorithm from [33], and in our experiments we consider 3 different predefined resolutions. However, given a concrete clearance value δ, one can construct a grid on $SO(3)$ with the necessary resolution based on δ.

As we see, our algorithm can be applied to different classes of shapes. It certainly cannot replace previous shape-based algorithms, as in some applications global geometric features of objects can be known a priori, in which case it might be faster and easier to make use of the available information. However, in those applications where one deals with a wide class of objects without any knowledge about shape features they might have, our algorithm provides a reasonable alternative to shape-based methods.

Molecular Cages. In organic chemistry, the notion of caging was introduced independently of the concept of [14]. Big molecules or their ensembles can form hollow structures with a cavity large enough to envelope a smaller molecule. Simple organic cages are relatively small in size and therefore can be used for selective encapsulation of tiny molecules based on their shape [14]. In particular, this property can be used for storage and delivery of small drug molecules in living systems [22]. By designing the caging molecules one is able to control the formation and breakdown of a molecular cage, in such a way as to remain intact until it reaches a specific target where it will disassemble release the drug molecule. An example on Fig. 6 shows that in principle, our algorithm can be applied to molecular models, assuming they can be treated as rigid bodies. In our future work, we aim to use our algorithm to test a set of potential molecular carriers and ligands to find those pairs which are likely to form cages. This prediction can later be used as a hypothesis for further laboratory experiments.

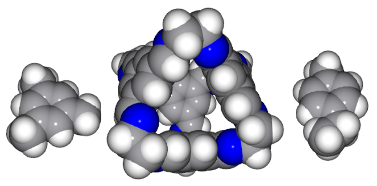
Fig. 6: In this example, we consider one big and two small molecules to see whether our algorithm can predict their ability to form a cage. Atomic coordinates and (van der Waals) radii were retrieved from the Cambridge Crystallographic Data Centre (CCDC) database. The depicted molecules are mesitylene (left, CCDC 907758), CC1 (middle, CCDC 707056), and 4-ethyltoluene (right, 1449823)[14]. Our algorithm reported that CC1 cages mesitylene, but does not cage 4-ethyltoluene. These results coincide with the experimental results reported in [14].

Possible parallel implementation. Our algorithm has a lot of potential for parallelization, and in the future we are planning on taking advantage of it. For the examples considered above, our current implementation requires between 40 and 60 milliseconds to compute each slice together with its connected components, and the connectivity check for a pair of slices is even faster. The bottleneck of the algorithm is the number of slices. However, most of the computations can be performed in parallel: indeed, we can first compute the approximation of the collision space for each orientation s in separate threads, and store them in shared memory. After that, we can again run one process per slice to compute the intersections between connected components of adjacent slices.In the future, we are planning on implementing a parallel version of our algorithm.

6 Conclusion

In this paper, we provide a computationally feasible and provably-correct algorithm for 3D and 6D configuration space approximation and use it to prove caging and path non-existence. In the future, we will look into how we can make our algorithm faster and applicable to real-time scenarios. In particular, we will work on a parallel implementation, and investigate how the number of slice can be reduced. We are also interested in integrating our narrow passage detection and path non-existence techniques with probabilistic motion planners. Finally, we are going to use our approach to find molecular cages.

7 Acknowledgements

The authors are grateful to D. Devaurs, L. Kavraki, and O. Kravchenko for their insights into molecular caging. This work has been supported by the Knut and Alice Wallenberg Foundation and Swedish Research Council.

References

1. Barraquand, J., Kavraki, L., Latombe, J.-C., Motwani, R., Li, T.-Y., Raghavan, P.: A random sampling scheme for path planning.: IJRR, 16(6), 759–774 (1997).
2. Basch, J., Guibas, L. J., Hsu, D., Nguyen, A. T.: Disconnection proofs for motion planning.: IEEE ICRA (2001), 1765–1772.
3. Behar, E., Lien, J.-M.: Mapping the configuration space of polygons using reduced convolution.: IEEE/RSJ Intelligent Robots and Systems (2013), 1242–1248.
4. Edelsbrunner, H.: Deformable smooth surface design. Discr. and Comp. Geom. (1999), 21(1), 87–115.
5. Kuperberg, W.: Problems on polytopes and convex sets.: DIMACS Workshop on polytopes (1990), 584–589.
6. Latombe, J.-C.: Robot Motion Planning. Norwell, MA, USA: Kluwer Academic Publishers (1991).
7. Lozano-Perez, T.: Spatial planning: A configuration space approach.: IEEE Transactions on computers, 2, 108–120 (1983).
8. Mahler, J., Pokorny, F. T., McCarthy, Z., van der Stappen, A. F., Goldberg, K.: Energy-bounded caging: Formal definition and 2-D energy lower bound algorithm based on weighted alpha shapes.: IEEE RA-L, 1(1), 508–515 (2016).
9. Makita, S., Maeda, Y.: 3D multifingered caging: Basic formulation and planning.: IEEE IROS (2008), 2697–2702.
10. Makita, S., Okita, K., Maeda, Y.: 3D two-fingered caging for two types of objects: sufficient conditions and planning.: International Journal of Mechatronics and Automation, 3(4), 263–277 (2013).
11. Makita, S., Wan, W.: A survey of robotic caging and its applications. Advanced Robotics, 31(19-20), 2017.
12. Makapunyo, T., Phoka, T., Pipattanasomporn, P., Niparnan, N., Sudsang, A.: Measurement framework of partial cage quality based on probabilistic motion planning, IEEE ICRA (2013), 1574–1579.
13. McCarthy, Z., Bretl, T., Hutchinson, S.: Proving path non-existence using sampling and alpha shapes.: IEEE ICRA (2012), 2563–2569.
14. Mitra, T., Jelfs, K. E., Schmidtmann, M., Ahmed, A., Chong, S. Y., Adams, D. J., Cooper, A. I.: Molecular shape sorting using molecular organic cages.: Nature Chemistry, 5, 276 (2013).
15. Milenkovic, V., Sacks, E., Trac, S. Robust complete path planning in the plane.: Algorithmic Foundations of Robotics X, 37–52 (2013).
16. Pipattanasomporn, P., Sudsang, A.: Two-finger caging of concave polygon.: IEEE ICRA (2006), 2137–2142.
17. Pipattanasomporn, P., Sudsang, A.: Two-finger caging of nonconvex polytopes.: IEEE T-RO, 27(2), 324–333 (2011).
18. Pokorny, F. T., Stork, J. A., Kragic, D.: Grasping objects with holes: A topological approach.: IEEE ICRA (2013), 1100–1107.

19. Rimon, E., Blake, A.: Caging planar bodies by one-parameter two-fingered gripping systems.: IJRR, 18(3), 299–318 (1999).
20. Rodriguez, A., Mason, M. T.: Path connectivity of the free space.: IEEE T-RO, 28(5), 1177–1180 (2012).
21. Rodriguez, A., Mason, M. T., Ferry, S.: From caging to grasping.: IJRR, 31(7), 886–900 (2012).
22. Rother, M., Nussbaumer M. G., Rengglic, K., Bruns N.: Protein cages and synthetic polymers: a fruitful symbiosis for drug delivery applications, bionanotechnology and materials science.: Chemical Society Reviews, 45, 6213, (2016).
23. Stork, J. A., Pokorny, F. T., Kragic, D.: Integrated Motion and Clasp Planning with Virtual Linking.: IEEE/RSJ IROS (2013), 3007–3014.
24. Stork, J. A., Pokorny, F. T., Kragic, D.: A Topology-based Object Representation for Clasping, Latching and Hooking.: IEEE-RAS International Conference on Humanoid Robots (2013), 138–145.
25. Pereira, G. A. S., Campos, M.F.M., Kumar, V.: Decentralized algorithms for multi-robot manipulation via caging.: IJRR 23(7–8), 783–795 (2004).
26. Varava, A., Kragic, D., Pokorny, F. T.: Caging Grasps of Rigid and Partially Deformable 3-D Objects With Double Fork and Neck Features. IEEE T-RO, 32(6), 1479–1497 (2016).
27. Varava, A., Carvalho, J. F., Pokorny, F. T., Kragic, D.: Caging and Path Non-Existence: a Deterministic Sampling-Based Verification Algorithm. ISRR, 2017. Preprint: `https://www.csc.kth.se/~jfpbdc/path_non_ex.pdf`
28. Vahedi, M., van der Stappen, A. F.: Caging polygons with two and three fingers.: The IJRR, 27(11–12) 1308–1324 (2008).
29. Wang Z., Kumar, V.: Object Closure and Manipulation by Multiple Cooperating Mobile Robots.: IEEE ICRA (2002), 394–399.
30. Wise, K., Bowyer, A.: A survey of global configuration-space mapping techniques for a single robot in a static environment.: IJRR, 19(8), 762–779 (2000).
31. Zhang, L., Young, J. K., Manocha, D.: Efficient cell labelling and path non-existence computation using C-obstacle query.: IJRR, 27(11–12), 1246–1257 (2008).
32. Zhu, D., Latombe, J.: New heuristic algorithms for efficient hierarchical path planning.: IEEE T-RO, 7(1), 9–20 (1991).
33. Yershova, A., Jain S., LaValle S. M., Mitchell J. C.: Generating Uniform Incremental Grids on SO(3) Using the Hopf Fibration. IJRR, vol. 29, 801–812 (2009).
34. Zomorodian, A., Edelsbrunner H.: Fast software for box intersections, Proceedings of the 16th annual symposium on Comp. geom., 129–138 (2000).
35. Liu J., Xin S., Gao Z., Xu K., Tu C., Chen B.: Caging Loops in Shape Embedding Space: Theory and Computation. ICRA, 2018.