Caging Grasps of Rigid and Partially Deformable 3D Objects with Double Fork and Neck Features

Anastasiia Varava, Danica Kragic, and Florian T. Pokorny

***Abstract*—Caging provides an alternative approach to point-contact based rigid grasping which relies on reasoning about the global free configuration space of an object under consideration. While substantial progress has been made towards the analysis, verification and synthesis of cages of polygonal objects in the plane, the use of caging as a tool for manipulating general complex objects in 3D remains challenging. Furthermore, many objects are naturally partially deformable, making classical rigid caging methods inapplicable. In this work, we formally generalize the caging condition in 3D to caging of objects up to a defined set of deformations and propose a novel framework for the synthesis and verification of caging grasps on a specific class of 3D objects which exhibit geometric features we call double forks and necks. We consider the synthesis and verification of a caging grasps on 3D objects by means of one or more caging tools that can be arranged to form an approximate closed loop around the identified neck or double fork features of the object. As a key theoretical tool, we utilize an augmentation of the classical topological invariant – the linking number, allowing us to prove sufficient conditions for such cages to exist, even in the case when the object under consideration is partially deformable. We present and evaluate an algorithm for synthesizing and verifying such caging grasps on triangular surface meshes in 3D.**

***Index Terms*—Cage, grasping, shape features**

I. Introduction

To manipulate an object in its environment, a robot requires the ability to synthesize manipulator configuration that will enable it to control the pose of an object up to some degree. For a rigid object in particular, one may consider an immobilizing grasp of an object. However, most analytical grasp synthesis frameworks [3], [19], require a knowledge of physical properties such as friction coefficients and center of mass, while current data-driven learning-based methods [4] require large corpora of prior training data with similar sets of objects, physical parameters and robot embodiment to infer grasps. Furthermore, many objects as depicted in Fig. 1 naturally allow some degree of deformability, making well-established frameworks for rigid grasping inapplicable. Similarly, even when the object under consideration is rigid, new generations of robots relying on soft hands [6], require new paradigms for manipulation not based on the rigid object assumption.

An alternative to grasping, is the notion of a cage, where the robot generates a joint-configuration which prohibits the object from moving arbitrarily far away, while not necessarily immobilizing the object completely. Cages can furthermore serve as a waypoint to a form or force closure grasp [24]. While substantial progress to the synthesis of caging grasps on polygons in 2D has been made in recent years [20], [23], [21], the generation of cages in 3D has remained a difficult challenge, with recent works making progress on specific basic shape primitives [17], [18]. A reason why the synthesis of caging grasps on general 3D objects has remained elusive is that the caging condition requires reasoning about the global topology of the configuration space of an object, rather than just a local geometric analysis as in the case of form and force closure. Additionally, many objects in the environment exhibit some degree of deformability. Consider the examples in Fig. 1. Each of these human hand configurations intuitively cages the object under consideration, because both extremities of the caged object are larger than the loop formed by the human hand.

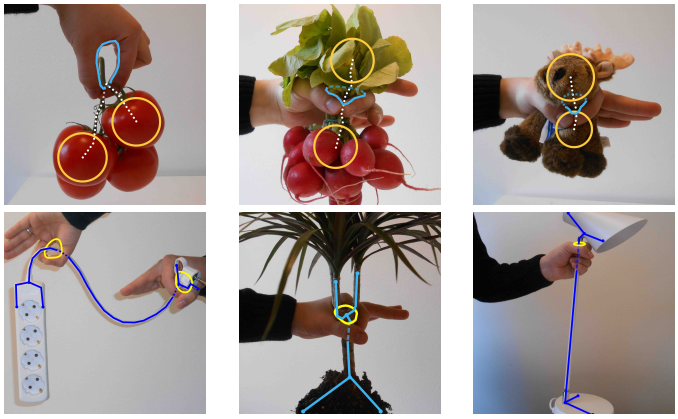

Fig. 1. Illustration of human manipulations exhibiting two types of cages of rigid and partialy deformable objects studied and formalized in this work. The objects in the first row have a neck feature - a thin object 'neck' with thicker ends at both sides. The second row showcases deformable objects that contain an inscribed double fork, intuitively consisting of a pair of inscribed curves whose ends are sufficiently spread spacially to prevent the loop formed by the human hand from moving arbitrarily far away from the object.

The main goal of this paper is to provide a formal framework allowing us to formalize, algorithmically verify and synthesize caging configurations similar to the ones displayed in the figure above. For this purpose, we define the notions of a 'double fork' and 'neck' in an object and provide an algorithm to both determine these features and to verify the caging condition for a given loop representing the manipulator and a mesh representation of the object. For this purpose we introduce an approach formalizing when a loop encloses a 'thin' part of an

A. Varava and D. Kragic are with the Computer Vision and Active Perception Lab, Centre for Autonomous Systems, School of Computer Science and Communication, Royal Institute of Technology (KTH), SE-100 44 Stockholm, Sweden. E-mail: {varava, dani}@kth.se

F. T. Pokorny is with AMPLab and Automation Lab, University of California, Berkeley, CA 94720 Berkeley, USA. E-mail: ftpokorny@berkeley.edu

object. This requires the use of several ideas from topology, and in particular linking numbers. Our approach allows us to discuss caging under partial deformations in 3D, thereby extending the types of objects that can be analyzed from prior works that required a restricted set of shape primitives [18] or the existence of holes [22], [29] in an object. The second goal of the paper is to provide algorithms for shape analysis and cage verification. Finally, we also provide simple algorithms generating caging configuration for a PR2 robot, simulated in OpenRave [9]. However, providing and evaluating a complete framework for this is beyond the scope of the paper, as the main purpose of the proposed algorithms is to illustrate how one can use our theoretical results.

The rest of the paper is organized as follows: in the next section, we briefly review the history of caging in the robotic manipulation context. Section III contains a general overview of our system. Section IV recalls some notions from topology, used in Section V. Section V covers the theoretical background of our approach: we define the necessary constructions and derive the sufficient conditions for the caging tool to bound the mobility of the object, represented as a graph. In Section VI, we describe our methodology and discuss practical ways to extract a graph representation from an object. Finally, we summarize our results and discuss possible directions of future work in Section VII.

II. Motivation and Related Work

Here, we briefly review key aspects of related prior works on caging in robotics, and the relationship between caging and grasping research in particular.

A. Caging

Following early work of [2] who considered sufficient conditions for a sphere to be caged by a net, the notion of a planar cage was introduced in 1990 by Kuperberg [15] as a set of n coplanar points lying in the complement to the interior of a given polygon and preventing it from moving arbitrarily far from its initial position. In the robotics context, these points can be considered as fingertip positions of a robot manipulating an object in the plane. In general, the caging problem in 2D or 3D consists of determining a configuration of one or several robotic grippers or caging tools such that an object of interest cannot escape arbitrarily far, and additional obstacles in the environment can also be considered in this approach.

The derivation of necessary and sufficient caging conditions poses a highly challenging problem due to the complex geometric structure of the free configuration space of real world objects. Several works were devoted to the case of 2D object representations, as even in this simplified case the exact computation of caging configurations poses a challenging problem. Early work of [23] proposed an algorithm computing a set of configurations of a two-fingered hand to cage planar non-convex objects. Subsequently, [20], [31] addressed closely related problems and [20] proposed an algorithm reporting all two-finger caging sets for a given concave polygon. This result was extended by [31], where an algorithm that returns all caging placements of a third finger is also described when a polygonal object and a placement of two other fingers is provided.

The caging problem becomes even more challenging when we move to the 3D-objects. The work [21] proposed an algorithm that computes all two-finger caging configurations for non-convex polytopes, where the fingertips are approximated by points or spheres. In [24], a notion of a *pregrasping cage* is introduced: a caging configuration from which the object can be grasped without breaking the cage. The fingers are again approximated as points, and pregrasping cages are studied via *grasping functions*: scalar functions defined on the finger formation that control the process of going from a cage to a grasp. However, in these works the fingertips are approximated by points or spheres, and more complex hand representations are not taken into account. In [17], [18], the authors proposed intuitively determined conditions for caging a small collection of parametrically defined simple objects in 3D, such as discs, tori and objects with 'waists'. In [18], this approach is generalised to objects for which geometric sub-structure is pre-determined and stored in a database. For general 3D objects and manipulators, there currently does not exist a provably correct practical caging algorithm to the best of our knowledge.

In this work, we propose sufficient conditions for caging objects in 3D with particular well-defined shape properties. A core contribution of our approach is the use of a graph-based shape representation in conjunction with a technique for rigorously proving that an object is caged. Our approach is flexible with respect to the choice of the manipulator type: it is applicable for a set of fingertips, an entire hand, an arm, or a system of cooperating mobile robots as long as these manipulators can form an approximate closed *caging loop*.

The present work constitutes a further evolution and extension of a sequence of previous works by our group [22], [29], [30] which introduced an approach to caging 3D objects that exhibit holes (non-trivial first homology) with complex multifingered robotic hands. We previously considered both control strategies to guide the robot arm/hand towards a caging configuration and an augmented task-space RRT planner for planning hand and arm motions to achieve caging configurations. In the current work, we focus in particular on the novel theoretical aspects of our contribution and the proposed shape representation.

B. Grasping

The caging problem has been considered as a complement to the traditional force-closure grasping approach [3], [19] which is based on the calculation of the forces exerted by the robot and the friction between the contacting surfaces, and hence relies on the determination of contact points and normals to the object surface at these points. Therefore, a detailed knowledge of the manipulated object's local geometry and the robot's pose is crucial in this context.

An important advantage of caging is that it relies on global geometric information about the object's shape and manipulator's configuration which may be estimated more

robustly than local friction coefficients and contact normals. Furthermore, while in force and form closure grasping, the manipulator is required to tightly establish contact with the object, thereby introducing an additional hard constraint, this is often not necessary when a caging grasp is considered instead.

One of the disadvantages of our approach is that it requires a three-dimensional model of the object. For instance, in [26] the authors compute grasping points using raw vision input in a data-driven manner without requiring a three-dimensional object model. While data-driven approaches to grasping provide an exciting avenue of recent research [4][16], training a data-driven model requires large corpora of training data, which is especially challenging for objects that exhibit deformability. Our object-model based approach, while being applicable only to objects with neck or double fork features, instead allows us to perform a complete geometric analysis of previously unseen models with these features, and enables us to synthesize cages and study their properties formally.

In this paper, we work with deformable objects and, in particular, generalize the notion of caging to this case. However, we do not attempt to model the deformability in the sense of predicting the deformations of the object, like it has been done in [12]. We do not consider physical properties of objects and simply allow them to be partially deformable under certain constraints.

III. Overview of our approach

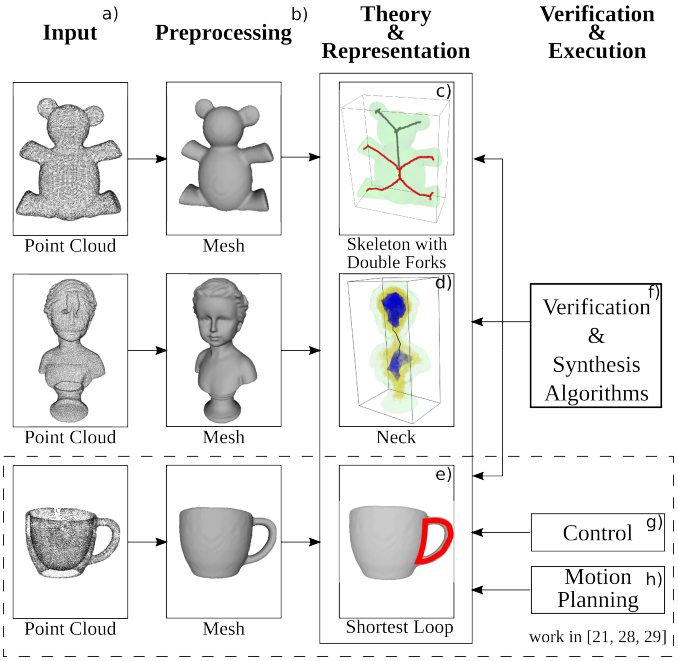

Fig. 2. Overview of our approach to caging.

The focus of this work is on the shape representation, theory and algorithmic synthesis and verification of caging configurations. Fig. 2 outlines our approach for the two novel shape representations we propose. The first is based on 'skeletonization and double forks' (first row) and the second on object features we call 'necks' (second row). Intuitively,

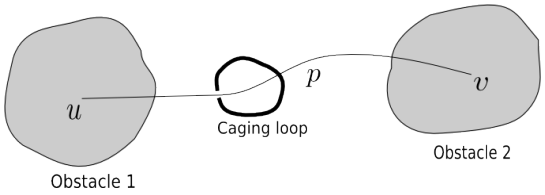

Fig. 3. Consider two 'obstacles', containing points u and v respectively which are large enough to block the displayed loop from escaping. In this case, the loop cages the two obstacles connected via p. In this paper, these 'obstacles' correspond to wide parts of an object; the presence of such parts makes the object suitable for caging with our approach.

both of these shape features can be considered as narrow parts of the object, around which the caging tool is placed.

The third row outlines previous work [22], [29], [30] by some of the coauthors and collaborators. As input to our pipeline, we consider a point-cloud or a mesh of an object, see Fig. 2a,b). In our previous work (third row), we considered objects that exhibit tunnels/voids as formalized by a non-trivial first homology group. We previously extracted a shortest homology basis resulting in a collection of loops that furnished an object representation suitable for caging. In the case of the depicted tea cup, this basis consisted of a single tight loop winding around the handle. Using such loops, control and motion planning algorithms for caging were developed [22], [29], [30].

In the present paper, we consider the initial (either point cloud or mesh) representation of the object. We define and extract shape features from the object, place the manipulator around the narrow part, derive sufficient caging conditions, and prove that the resulting configuration yields a valid cage. We abstract the caging tool/manipulator by means of a closed loop $l \in \mathbb{R}^3$ in the complement of the interior of the object. We call l the *caging loop*. Examples of caging loops include loops formed by two arms with locked hands that are closed around an object, two touching fingers, etc.

To bound the mobility of the object such as the depicted bear and bust our caging approach considers placing a loop around a narrow part of the object as illustrated in Fig. 3. Intuitively, this representation allows us to work with partially deformable objects, provided the 'wide parts' preserve a lower bound on their size under deformation. Double forks and necks are two examples of this geometric structure, proposed in this paper. While, at first sight, this approach appears to be intuitive and rather simple, a key difficulty we resolve is to algorithmically determine these features and to algorithmically verify weather a caging loop 'surrounds' the finite narrow part in a way that is invariant under a set of continuous deformations of the object. Section VI reviews the required technical background on topology that may be skipped by experienced reader.

Section V presents the core formalizations of our approach, including:

- A generalization of caging for partially deformable objects (Def. 11);
- Definition of neck and double fork features;
- Formalization of when the manipulator is placed 'around' the object;

- Proof of sufficient conditions for caging objects with necks and double forks.

Section VI provides algorithms and implementation results. The contributions of this section are:

- algorithms for detection of double forks and necks in a mesh, representing the object;
- an algorithm constructing the caging loop candidates around the computed shape features;
- a simple algorithm computing caging configurations for a PR2 robot, simulated in OpenRAVE [8];
- an algorithm verifying a given caging configuration, based on the sufficient conditions derived in Section V.

IV. Topological background

Here we provide underlying notions from topology required in Section V. This section can be skipped by a reader familiar with general and algebraic topology. Further details can be found in [13].

A. Deformations: homotopy and isotopy

To work with deformable objects, we require a formal definition of a continuous deformation. Intuitively, a continuous deformation allows stretching and bending, but disallows actions such as cutting and gluing. In topology, a continuous deformation between two maps $f, g : X \to Y$ can be formalized by means of the notion of a homotopy. Using the notation $I = [0, 1]$, we have:

Definition 1. *Let X, Y be topological spaces, and let $f, g : X \to Y$ be continuous functions. A continuous map $H : X \times I \to Y$, such that for each $x \in X$: $H(x, 0) = h_0(x) = f(x)$, $H(x, 1) = h_1(x) = g(x)$, is called a* **homotopy** *between f and g. Respectively, the functions f and g are called* **homotopic** *if there exists a homotopy between them.*

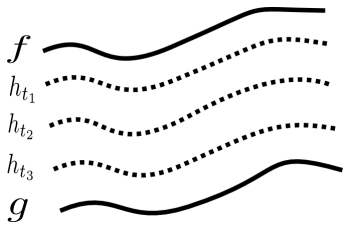

Fig. 4. Let $f : I \to X$, $g : I \to X$ be paths in the space X. Then f can be deformed to g by means of a homotopy $H : I \times I \to X$, where $H(s, t) = h_t(s)$ for each $s, t \in I$.

Here, the parameter $t \in [0, 1]$ can be considered as a time parameter deforming f to g as t varies from 0 to 1. One of the simplest examples of a homotopy is a homotopy of paths, see Fig. 4. In our approach, it is convenient to consider the object as an abstract topological space $\mathcal{O}$, embedded in $\mathbb{R}^3$ whose geometric shape and position are specified by an embedding:

Definition 2. *A map $f : X \to Y$ between topological spaces X and Y is called an* **embedding** *if f is a homeomorphism* [1] *between X and its image $f(X)$.*

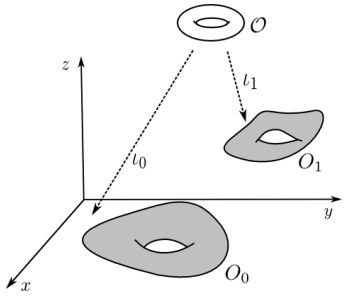

Fig. 5. Two embeddings of the space $\mathcal{O}$ into $\mathbb{R}^3$

Furthermore, we recall the notion of an isotopy:

Definition 3. *A homotopy H between two embeddings f and g from X to Y is called an* **isotopy** *if, for each $t \in I$, the map $f_t : X \to Y$ defined by $f_t(x) = H(x, t)$ is an embedding.*

We consider the deformable object's shape and position at a particular time instance to be formalized by a corresponding embedding $\iota : \mathcal{O} \to \mathbb{R}^3$, see Fig 5, and represent object deformations and position changes by means of special homotopies of its embedding called isotopies:

Given an initial initial shape and position of a deformable object $O_0 \subset \mathbb{R}^3$, and model deformations and movement of the object towards a final configuration $O_1 \subset \mathbb{R}^3$, as a corresponding isotopy between the initial embedding $\iota_0 : \mathcal{O} \to \mathbb{R}^3$, $\iota_0(\mathcal{O}) = O_0$, and the final embedding, $\iota_1 : \mathcal{O} \to \mathbb{R}^3$, $\iota_1(\mathcal{O}) = O_1$.

B. The fundamental group and linking numbers

In order to formally describe when the manipulator is placed *around* a narrow part of an object, we build on the notion of the linking number, an integer quantity that can be computed for two closed curves in $\mathbb{R}^3$. When the linking number of two curves is non-zero, these curves are linked and can in particular not be 'pulled apart' under continuous deformations of the curves (see Fig.6). We will extend the notion of linking numbers to define a linkability condition for neck and double fork features of an object which themselves do not necessarily contain closed curves. This will allow us to formally verify caging configurations.

For this purpose, we recall the notion of the fundamental group. A loop in a topological space X is given by a curve $f : I \to X$ whose end-points coincide: $f(0) = f(1)$. A loop that can be continuously contracted to a point is called trivial. For $x_0 \in X$, let $L(x_0) = \{f : I \to X : \ f(0) = x_0 = f(1)\}$ be the set of loops in X with base point x_0. We consider two loops $f, g \in L(x_0)$ equivalent if there exists a homotopy between f and g that keeps x_0 fixed. The equivalence class of a loop $f \in L(x_0)$ is denoted by $[f]$. Depending on the topological properties of the space X, two loops may or may not be continuously deformable into to each other, see Fig 7.

Definition 4. *The* **fundamental group** *of X with a base point x_0 consists of the set of the equivalence classes of loops with base point x_0 and is denoted $\pi_1(X, x_0)$.*

[1] A homeomorphism is a continuous bijection with a continuous inverse.

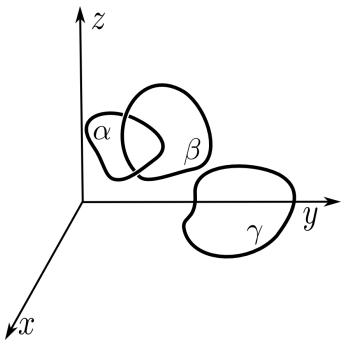

Fig. 6. The linking number $lk(\alpha, \beta)$ of curves α and β is equal to 1, while $lk(\alpha, \gamma) = lk(\beta, \gamma) = 0$. The reason for that is α and β 'pass through' each other, while γ is completely disjoint from both of them.

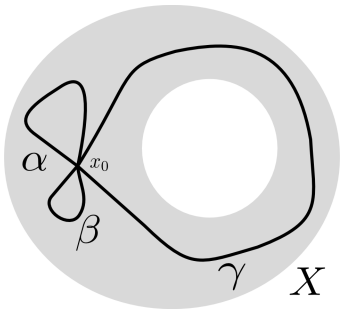

Fig. 7. The displayed space X (grey region) has a hole, thus the loop γ cannot be deformed into α, so that $[\gamma] \neq [\alpha]$, while α and β are homotopy equivalent, so that $[\alpha] = [\beta]$. In this case X is hence not simply connected.

We denote by $-f$ the direction-reversed loop $-f(t) = f(t-1)$ for $t \in I$ and by $f \circ g$ the concatenated loop following first g at twice the speed and then f at twice the speed. The fundamental group in fact forms an algebraic group under concatenation and direction-reversal of loops and allows us to study properties of a topological space X. When X is path-connected (i.e., if there is a path joining any two points in X), the choice of base point is in fact unimportant for our purposes since the groups $\pi_1(X, x) \cong \pi_1(X, x')$ are isomorphic for any $x, x' \in X$. In particular, the fundamental group is called trivial when every loop can be contracted to a point. We define:

Definition 5. *A topological space X is called* **simply-connected** *if X is path-connected and has trivial fundamental group.*

Fig. 7 visualizes a space that is not simply connected, while e.g. $X = \mathbb{R}^3$ provides an example of a simply connected space.

We now can provide two equivalent (up to sign) definitions of the linking number. We refer the reader to [25] for more details. The first definition is convenient for our theoretical framework, while the second serves for computational purposes.

Definition 6. *Consider two loops α, β in $\mathbb{R}^3$ both homeomorphic to a circle in $\mathbb{R}^3$ and such that $\alpha \cap \beta = \emptyset$. Then β represents an element in the fundamental group $\pi_1(\mathbb{R}^3 - \alpha)$, which is isomorphic to $\mathbb{Z}$. The loop β is hence mapped to an integer, called the* **linking number** *$lk(\alpha, \beta)$. In particular, if $lk(\alpha, \beta) \neq 0$, then β is non-trivial in $\pi_1(\mathbb{R}^3 - \alpha)$.*

The above definition is useful for our theoretical framework in Section V. Namely, we use it to work with the linkability relation, between the object and the manipulator.

Definition 7. *The* **linking number** *between two differentiable loops $\alpha, \beta : \mathbb{S}^1 \to \mathbb{R}^3$ that are each homeomorphic to a circle in $\mathbb{R}^3$ can be computed directly by means of the* **Gauss Linking Integral***:*

$$lk(\alpha, \beta) = \frac{1}{4\pi} \oint_{\mathbb{S}^1} \oint_{\mathbb{S}^1} \langle \frac{\alpha(s) - \beta(t)}{\|\alpha(s) - \beta(t)\|^3}, \alpha'(s) \times \beta'(t) \rangle \mathrm{d}s\mathrm{d}t.$$

A discrete version of the above formula, for the case of piecewise linear loops $\alpha, \beta : \mathbb{S}^1 \to \mathbb{R}^3$ is discussed in [14] and can be used to efficiently determine the linking number in the piecewise linear case. Thus, the above definition provides a practical approach to compute $lk(\alpha, \beta)$ in Section VI.

V. Theoretical Contribution

In this section, we first generalize the notion of caging to the case of deformable objects and show that our definition is consistent with the classical case for rigid objects. Further, we formally define double fork and neck features, as well as classes of object deformations preserving these features. This allows us to prove sufficient caging conditions for partially deformable objects exhibiting the neck and double-fork features.

A. Caging under deformability assumption

We start our formalization with the definition of the object. If we assume the object to be rigid, it can be defined simply as a subset of $\mathbb{R}^3$. However, as we want to allow object deformations, we need to distinguish between the concept of the object and its shape. For this purpose, we represent the object as an abstract topological space, and introduce the notion of its shape – the way it is placed into $\mathbb{R}^3$ by means of an embedding:

Definition 8. *Let $\mathcal{O}$ be a compact topological space. Assume that it can be embedded into $\mathbb{R}^3$: let $\iota : \mathcal{O} \to \mathbb{R}^3$ be an embedding. Then we call $\mathcal{O}$ the* **object***, and the image of the embedding $O = \iota(\mathcal{O})$ its* **shape***.*

Mathematically, the above definition allows degenerate cases when the object has no thickness. For example, a curve in $\mathbb{R}^3$ satisfies the above definition of the shape. To be realistic, we avoid these cases by adding an assumption about the shape of the object:

Assumption 1. *Assume that for any shape O of the object $\mathcal{O}$ there exists a non-empty open set $U \subset \mathbb{R}^3$, such that $O = cl(U)$.*

In the above $cl(U)$ denotes the closure of U. We distinguish between a situation when the manipulator is in contact with the object, but does not penetrate the object itself, and when the manipulator penetrates the object.

Definition 9. *Let $O \subset \mathbb{R}^3$ be the shape of the object $\mathcal{O}$. A point $x \in (O - int(O))$ is said to* **be in contact** *with the object, while a point $x \in int(O)$ is said to* **penetrate** *the object.*

Now we can define the notion of caging. Let us start with the classical assumption that the object is rigid. In this case, the object can escape only by means of rigid body motions. The following definition formalizes caging under this assumption.

Definition 10. *Consider an object of the shape O and let $M \subset \mathbb{R}^3$ be a set. We define the free configuration space of O with respect to M to be the set of rigid transformations not causing penetrations with the object:*

$$C_O = \{g \in SE(3) : M \cap g.int(O) = \emptyset\}.$$

If the identity element $e \in C_O$ corresponding to an initial configuration of O relative to M lies in a bounded path-component of C_O, we say that **the set M cages the object** *O. Here $g.x \in \mathbb{R}^3$ denotes the result of applying the rigid transformation $g \in SE(3)$ to $x \in \mathbb{R}^3$, and similarly $g.int(O) = \{g.x \in \mathbb{R}^3 : x \in int(O)\}$.*

Note that with the above notion of caging, we allow contacts between M and O but disallow penetrations. Note further that M can be an arbitrary set, such as a discrete number of points, a curve, etc.

We want to generalize the notion of caging by allowing the object to be deformable. In this case, the object can escape from the manipulator not only by rigid body motions, but also by changing its shape. Thus, an escaping path can be represented by an isotopy, as defined in Section IV. Since we assume that the caging tool is rigid, we can without loss of generality center our coordinate system at the manipulator and assume that its position is fixed.

Definition 11. *Consider an object $\mathcal{O}$ with shape O given by the embedding $i : \mathcal{O} \to O$ and let $M \subset \mathbb{R}^3$ be a set. We say that an isotopy $H : \mathcal{O} \times I \to \mathbb{R}^3$ with $H(x, 0) = i(x)$ for all $x \in \mathcal{O}$ is* **a non-penetrating isotopy** *of the object with respect to M if $int(H(\mathcal{O}, t)) \cap M = \emptyset$ for all $t \in I$. Let D be a subset containing all the non-penetrating isotopies of $O \subset \mathbb{R}^3$. D specifies a set of allowable deformations. Assume that M is fixed in space. We say that M* **cages** *O* **under** *D-***deformations** *if any non-penetrating isotopy $H \in D$ satisfies $\mathrm{dist}(M, H(\mathcal{O}, t)) \leq C$ for all $t \in I$ and for some fixed $C \geq 0$.*

By the distance between M and $H(\mathcal{O}, t)$ in the above definition we mean the infimum of the distances between any two of their respective points,

$$\mathrm{dist}(M, H(\mathcal{O}, t)) = \inf_{x \in M, y \in H(\mathcal{O}, t)} \|x - y\|$$

Remark 1. *Observe that the previous notion of caging under rigid transformations in Def. 10 corresponds to a particular class of isotopies induced by rigid transformations. See Lemma 1 in the Appendix for the detailed explanation.*

B. Double fork and neck features

Let us now consider our first shape feature, which we refer to as a *double fork*.

We consider a graph embedded into the object we want to cage. Such a graph G can be obtained using a skeletonization technique. We in particular rely on the technique proposed in [7] where a skeleton of the object is obtained by retracting its medial axis to a graph. This skeleton yields a geometric graph $G = (V, E)$ that lies in the object and a function, reflecting the distance from each point on the graph to the object's surface is also provided. Note though that our approach is general and does not depend on a particular skeletonization technique. We introduce the following definition, see Fig. 8.

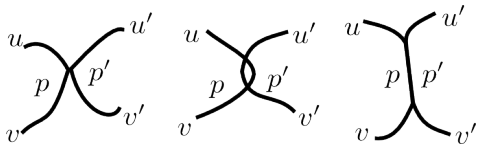

Fig. 8. Different examples of double forks. The set of central vertices can be either finite or infinite.

Definition 12. *Let $u, u', v, v' \in \mathbb{R}^3$ and let $p : I \to \mathbb{R}^3$ and $p' : I \to \mathbb{R}^3$ be two embeddings, such that p is a path in $\mathbb{R}^3$ from u to v, while p' is a path in $\mathbb{R}^3$ from u' to v'. If $\mathrm{dist}(p, \{u', v'\}) > d$, $\mathrm{dist}(p', \{u, v\}) > d$ and $p \cap p' \neq \emptyset$ we say that p, p' form a* **double fork with diameter** *d. The set of vertices $C = \{v \in V : v \in p \cap p'\}$ is called the* **set of central vertices**.

Since, according to Assumption 1, our object always has some thickness, it is convenient to consider the double fork together with some neighbourhood:

Definition 13. *Given a double fork consisting of $p, p' : I \to \mathbb{R}^3$ and with parameter d and $\varepsilon > 0$, we define a* **double fork of a thickness** *ε as $DF_\varepsilon = DF_\varepsilon(p(I), p'(I)) = \{x \in \mathbb{R}^3 : \mathrm{dist}(x, p(I)) \leq \varepsilon$ or $\mathrm{dist}(x, p'(I)) \leq \varepsilon\}$.*

Let us now consider objects with another geometric feature which we refer to as *a neck*.

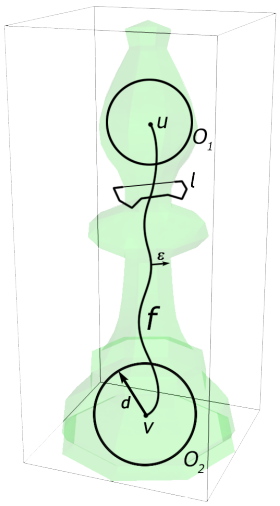

Fig. 9. A neck $N_{\varepsilon,d}(f(I))$ and a caging loop l

Intuitively, a neck of the object is a constriction which is situated between two wider parts. We use this feature to extract a simplified geometrical representation of the object, which can be considered as its 'core'. Formally, we define this as follows:

Definition 14. *Let $f : I \to \mathbb{R}^3$ be an embedding. For $\varepsilon > 0$ and $d > \varepsilon$, we define* **a neck $N_{\varepsilon,d}(f(I))$ of diameter d and thickness** *ε as follows: $N_{\varepsilon,d}(f(I)) = \{x \in \mathbb{R}^3 : \|x - f(s)\| \leq \varepsilon$ for some $s \in I$ or $\|x - f(0)\| \leq d$ or $\|x - f(1)\| \leq d\}$.*

Intuitively, we can imagine two spheres (note that, in degenerate cases, they can intersect) of radius d, inscribed into the object O (see Fig.9) and connected by an ε-neighbourhood of the defining curve f. The term 'neck' is inspired by 'choking loops' proposed in [11].

One can notice that in fact, any object with a neck contains a double fork as well. Indeed, assume we have a neck consisting of two spheres and a path between them, and let u and u' be diametrically opposed on one sphere, and v and v' be diametrically opposed on the other sphere. Let p connect u and v, p' connect u' and v'. Now each vertex is located at a distance to the other path of at least the smaller sphere's radius. However, note that the diameter of the corresponding double fork is half the diameter of the initial neck. Since the diameter of the feature will be directly related to the cagability of the object, we consider a neck as a separate shape feature here.

C. The linkability relation

We now define the 'linkability' relation between the caging loop and a curve inside the object. We address this by introducing a superset of the loop which distinguishes curves that go through the loop from those that go around the loop. Algorithmically, we then compute linkability with the help of the linking number, recalled in Section IV.

Let a loop l be located in $\mathbb{R}^3$. We want to split all the possible curves in $\mathbb{R}^3$ into two classes: those which 'pass through l', and those which do not. Given a query qurve f, we approach this problem by first constructing a special curve g with the same end-points as f that avoids a superset $S(l)$ of the loop l entirely - we call this curve an $S(l)$-avoiding augmentation. We then show how an investigation of homotopy classes then leads to a classification of 'linkability' generalizing the classical linking number concept in topology. To reach a formally correct derivation, we will require that $\mathbb{R}^3 - S(l)$ is simply connected. This is the case in particular, when $S(l)$ is given by the convex hull of l.

Definition 15. *Consider a path $f : I \to \mathcal{W}^{free} = \mathbb{R}^3 - l$, $f(0) = u$, $f(1) = v$ and where l denotes a loop in $\mathbb{R}^3$ homeomorphic to a circle. Let $S(l)$ denote a subset of $\mathbb{R}^3$ containing l such that $\mathbb{R}^3 - S(l)$ is simply-connected. The path f is called* **linkable to l via a $S(l)$-avoiding augmentation** *if there exists another path $g : I \to \mathcal{W}^{free}$ from u to v such that $g \cap S(l) = \emptyset$ and the loop $f \circ -g$ is not null-homotopic in $\mathcal{W}^{free}$.*

Note that any path which is linkable to l via a $S(l)$-avoiding augmentation intuitively 'passes through l'. However, $S(l)$ can be constructed in different ways, and for each particular $S(l)$ a set of all paths linkable to l is a subset of all paths passing through l.

Let us now consider a curve passing through the loop l. In order for it to escape from l via a continuous deformation, either of its endpoints must 'go through' the loop at least once. However, the process of 'going through' the loop for a point also must be adequately formalized. The following proposition makes this precise:

Proposition 1. *Let $l \subset \mathbb{R}^3$ be a loop homeomorphic to a circle. Consider a path $f : I \to \mathcal{W}^{free} = \mathbb{R}^3 - l$, $f(0) = u$, $f(1) = v$, linkable to l via a $S(l)$-avoiding augmentation. Assume f can be continuously deformed to a path $f' : I \to \mathcal{W}^{free}$, such that $f' \cap S(l) = \emptyset$, by a homotopy $F : I \times I \to \mathcal{W}^{free}$, $F(s,0) = f(s)$, $F(s,1) = f'(s)$. Then $\exists t \in I$ such that $F(0,t) \in S(l)$ or $F(1,t) \in S(l)$.*

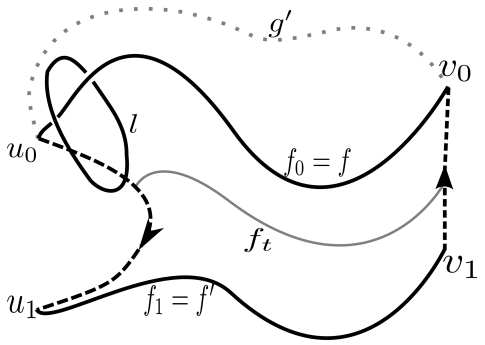

Fig. 10. A homotopy between the paths f and f'

Proof. Assume the contrary that for all $t \in I$, $F(0,t) \notin S(l)$ and $F(1,t) \notin S(l)$. Let us denote $f_t(s) = F(s,t)$, so that $f_0 = f$ and $f_1 = f'$. Similarly, let $u_s = F(s,0)$, $v_s = F(s,1)$ for all $s \in I$. Finally, let u_iu_j (v_iv_j) denote the corresponding path from u_i to u_j (from v_i to v_j) in $F(0,.)$ (respectively, in $F(1,.)$), where $i, j \in I, i \neq j$.

Let $g = u_0u_1 \circ f_1 \circ v_1v_0$, see Fig. 10. Consider a family of paths $\phi_t : I \to \mathcal{W}^{free}$ defined as $\phi_t = u_0u_{(1-t)} \circ f_{(1-t)} \circ v_{(1-t)}v_0$, $t \in I$. This defines a path homotopy deforming $\phi_0 = u_0u_1 \circ f_1 \circ v_1v_0 = g$ to $\phi_1 = f_0$. So, $g \simeq f_0$. Recall that $f_1 \cap S(l) = \emptyset$. Moreover, by our assumption, for all $t \in I$: $u_t \notin S(l)$, $v_t \notin S(l)$. So, g does not intersect with $S(l)$. Recall now that f_0 is linkable to l. So, there exists a path $g' \subset \mathcal{W}^{free}$ from u_0 to v_0 such that $g' \cap S(l) = \emptyset$ and the loop $f \circ -g'$ is not null-homotopic. Finally, note that $g \simeq g'$, since both of them can be considered as a paths in $\mathbb{R}^3 - S(l)$. Therefore, $f \circ -g \simeq f \circ -g'$. This leads us to contradiction. □

Let us now discuss how to algorithmically check whether a given path p from u to v is linkable to the loop l with respect to some $S(l)$. First of all, we have to properly construct $S(l)$. For our implementation, we choose $S(l) = \mathrm{Conv}(l)$, where $\mathrm{Conv}(l)$ denotes the convex hull of l. Note that $\mathbb{R}^3 - S(l)$ is simply-connected since $S(l)$ can be contracted to a point in $\mathbb{R}^3$ and hence $\pi_1(\mathbb{R}^3 - S(l)) \simeq \pi_1(\mathbb{R}^3 - \{0\}) = \{0\}$.

To determine if a path f from u to v such that $u, v \notin S(l) = \mathrm{Conv}(l)$ is linkable via a $\mathrm{Conv}(l)$ avoiding augmentation, we will algorithmically determine an arbitrary path p_{aux} from u to v in the complement of $\mathrm{Conv}(l)$. The relation of being linkable does not depend on the particular choice of $p_{aux} \in \mathbb{R}^3 - S(l)$, since $\mathbb{R}^3 - S(l)$ is simply-connected and therefore for any

other path $p'_{aux} \in \mathbb{R}^3 - S(l)$ with the same endpoints the loop $p \circ -p_{aux}$ can be continuously deformed to $p \circ -p'_{aux}$.

Finally, we have to check whether the resulting loop $p \circ -p_{aux}$ is non-trivial in $\mathbb{R}^3 - l$ for which we will utilize the linking number $lk(l, p \circ -p_{aux})$.

D. Sufficient caging conditions for objects with double forks and necks

Now we can prove sufficient caging conditions for objects with double fork and neck features. Since we want our objects to be partially deformable – i.e., to preserve shape features under deformations – we start with defining the classes of allowed deformations.

In the definition of a double fork of the thickness ε, we consider two paths p, p' as a subset of the object DF_ε. Since in our context this structure is crucial for an object to be suitable for caging, we consider it as a 'skeleton' of the object in the sense that its deformations induce the deformations of the entire object. To formalize this, we introduce the following definition.

Definition 16. *Let $\mathcal{O}$ be an object of a shape O, where $\iota : \mathcal{O} \to \mathbb{R}^3$, $\iota(\mathcal{O}) = O$ is the corresponding embedding, and $\iota^{-1} : O \to \mathcal{O}$ is its inverse. Let $DF = DF_\varepsilon(p(I), p'(I)) \subset O$ be double fork of diameter d and thickness ε. Consider an isotopy $H : \mathcal{O} \times I \to \mathbb{R}^3$, such that $H(x, 0) = \iota(x)$. If for each $t \in I$ a pair of paths $H(\iota^{-1}(p(I)), t)$, $H(\iota^{-1}(p'(I)), t)$ yields a double fork of a diameter d, then H is called a* **double fork preserving** *deformation of the object with respect to DF.*

Now we can prove caging conditions for objects with double forks:

Theorem 1. *Let $\mathcal{O}$ be an object of a shape $O \subset \mathbb{R}^3$,$\iota(\mathcal{O}) = O$ be the corresponding embedding, and $\iota^{-1} : O \to \mathcal{O}$ is its inverse. Let l be a loop in $\mathbb{R}^3$, representing the manipulator. Let $S(l)$ be a subset of $\mathbb{R}^3$ containing l such that $\mathbb{R}^3 - S(l)$ is simply-connected. Let $p, p' : I \to O$ be two embeddings, linkable to l via $S(l)$-avoiding augmentations and suppose that p, p' form a double fork with parameter $d > 0$. Let $\varepsilon > 0$ and assume $DF = DF_\varepsilon(p(I), p'(I))$ to be a subset of O. Finally, assume $O \cap l = \emptyset$.*

Consider the deformation class $\mathcal{FP}$ of the object O, such that any deformation H in $\mathcal{FP}$ is non-penetrating with respect to l and double fork preserving with respect to DF.

If $\operatorname{diam}(S(l)) < d$, then l cages the object O under $\mathcal{FP}$-deformations.

Proof. Suppose on the contrary that for any $C \geq 0$ there exists an isotopy H in class $\mathcal{FP}$ taking the object O at distance greater than C away from l. Let $C = 2 \cdot \operatorname{diam}(S(l))$. Then, the images $H(\iota^{-1}(p(I)), 1) \subset \mathbb{R}^3$ and $H(\iota^{-1}(p'(I)), 1) \subset \mathbb{R}^3$ of p and p' are separated from l at a distance greater than $\operatorname{diam}(S(l))$. Therefore, the paths $H(\iota^{-1}(p(I)), 1)$ and $H(\iota^{-1}(p'(I)), 1)$ do not intersect $S(l)$ and therefore are not linkable to l via $S(l)$-avoiding augmentation. Hence, by Proposition 1 there exists a first time t at which one of the end-points of the deforming p, p' is in $S(l)$. Without loss of generality, assume $x = H(\iota^{-1}(p(0)), t) \in S(l)$. Since the deformed $p'(I)$ is at that point either still linkable to l, and therefore intersects $S(l)$, or an endpoint of $p'(I)$ lies also in $S(l)$, we have two points in $S(l)$, whose distance from each other is at least $d > \operatorname{diam}(S(l))$, which is a contradiction, see Fig.11. □

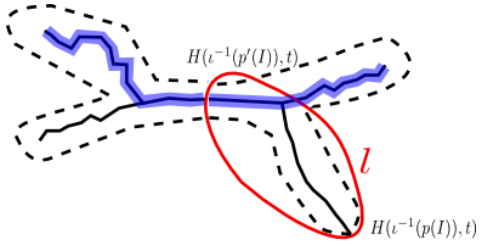

Fig. 11. The escaping process at time t. The boundary of O is depicted with dashed curve. The image of p is depicted in blue, while the caging loop is in red.

Let us now consider objects with necks. Although any neck contains a double fork of a smaller diameter, and thus can be considered as a special case of the latter feature, our caging conditions for double forks are more strict than the ones for necks, and thus we formulate them as a separate theorem.

Similarly to the case of a double fork, we consider $f(s)$ as a 'skeleton' of the object, and we hence assume that its deformations induce the corresponding deformations of the entire object.

Definition 17. *Let $\mathcal{O}$ be an object of shape $O \subset \mathbb{R}^3$, where $\iota : \mathcal{O} \to \mathbb{R}^3$, $\iota(\mathcal{O}) = O$ is the corresponding embedding, and $\iota^{-1} : O \to \mathcal{O}$ is its inverse. Let a neck $N = N_{\varepsilon,d}(f(I))$, for $f : I \to O$ be contained in O. Consider an isotopy $H : \mathcal{O} \times I \to \mathbb{R}^3$, $H(x, 0) = \iota(x)$. H is a* **neck preserving deformation with respect to** *N, if for each $t \in I$ the image $H(\iota^{-1}(f(I)), t)$ forms a neck with parameters ε, d, which is a subset of the corresponding image of O:*

$$N_{\varepsilon,d}(H(\iota^{-1}(f(I)), t)) \subset H(O, t).$$

Theorem 2. *Let $\mathcal{O}$ be an object of the shape $O \subset \mathbb{R}^3$, $\iota(\mathcal{O}) = O$ be the corresponding embedding, and $\iota^{-1} : O \to \mathcal{O}$ be its inverse. Let the neck $N = N_{\varepsilon,d}(f(I))$ be its subset, where $f : I \to \mathbb{R}^3$ is an embedding, and $\varepsilon > 0, d > 0$ are real constants. Assume that $O \cap l = \emptyset$. Let f be linkable to l via an $S(l)$-avoiding augmentation. Consider a deformation class $\mathcal{NP}$ of the object O, such that each deformation H in $\mathcal{NP}$ is non-penetrating with respect to l and neck preserving with respect to N. Then, the object O is caged by l under $\mathcal{NP}$-deformations if $\operatorname{diam}(S(l)) < d$.*

Proof. Suppose the result does not hold. Then for any real $C \geq 0$ the deformation class $\mathcal{NP}$ contains a non-penetrating isotopy $H_C : O \times I \to \mathbb{R}^3$ between of O and $H(O, 1)$ such that $\operatorname{dist}(l, H(O, 1)) > C$. Let $C = 2 \cdot \operatorname{diam}(S(l)) < d$, and consider an isotopy $H \in B$ leading the object at a distance C away from the manipulator l. Since $H(\iota^{-1}(f(I)), 1) \subset H(O, 1)$, we have $\operatorname{dist}(l, H(\iota^{-1}(f(I), 1)) > \operatorname{diam}(S(l))$. Therefore, $H(\iota^{-1}(f(I)), 1)$ does not intersect $S(l)$ and hence is not linkable to l via $S(l)$-avoiding augmentation. Then, by Proposition 1, there exists $t \in I$ such that $H(\iota^{-1}(f(0)), t) \in S(l)$ or $H(\iota^{-1}(f(1)), t) \in S(l)$. Assume, by reversing the

orientation of f if necessary, that $H(\iota^{-1}(f(0)),t) \in S(l)$. Then $\mathrm{dist}(H(\iota^{-1}(f(0)),t),l) \leq \mathrm{diam}(S(l))$. But, since the homotopy is non-penetrating, we have $dist(H(O,t),l) > d$, and hence $d < \mathrm{diam}(S(l))$ which leads to a contradiction. □

E. Approximate caging loops

In practice, we may be interested in caging objects by two or more caging tools such as fingers, arms or mobile robots placed at a small distance from each other such that these caging tools almost form a closed loop. Consider, for example the case of the PR2 gripper in Fig.12. While the fingers cannot close completely, we can still utilize such configurations as long as the distance between the fingertips is small enough. We now formalize this using the notion of an ε-augmented caging loop.

Fig. 12. A PR2 gripper caging a scoop by an approximate caging loop

Definition 18. *Let $\Gamma = (\gamma_1, \gamma_2, ..., \gamma_n)$, $\gamma_i : I \to \mathbb{R}^3$, be a tuple of piecewise linear curves contained in the caging manipulator and let and let $A = (a_1, a_2, ..., a_n)$, be a tuple of linear line segments such that a_i connects $a_i(0) \in \mathbb{R}^3$ to $a_i(1) \in \mathbb{R}^3$ and $\max_i \|a_i(0) - a_i(1)\| = \varepsilon$. Suppose furthermore that $\gamma_i(1) = a_i(0)$ and $\gamma_{i+1}(0) = a_i(1)$ for $i \in \{1, \ldots, n-1\}$ and $\gamma_1(0) = a_n(1)$, so that the loop $\widetilde{l}_\varepsilon(\Gamma, A) = \gamma_1, a_1, \gamma_2, \ldots, a_n$ forms a closed loop. Suppose further that $\widetilde{l}_\varepsilon(\Gamma, A)$ is homotopy equivalent to a circle. Then we call $\widetilde{l}_\varepsilon = \widetilde{l}_\varepsilon(\Gamma, A)$ an ε augmented caging loop.*

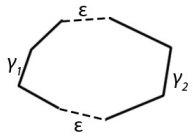

Fig. 13. A ε-augmented caging loop.

Deformations such as squeezing might change the thickness of the object, which is also important for caging by several tools. When we have an ε-augmented caging loop, we impose an additional important restriction on the class of allowed deformations of our objects. Namely, we consider objects that can change their thickness only up to some threshold. Intuitively, we can imagine this as a dense 'core' inside the object, which preserves its thickness while we squeeze the object.

Definition 19. *Let $\mathcal{O}$ be an object of the shape $O \subset \mathbb{R}^3$, and let $p : I \to O$ an embedded curve in O. Assume that $\{x \in \mathbb{R}^3 : \mathrm{dist}(x, p(I)) \leq \varepsilon\} \subset O$ for some $\varepsilon \geq 0$. Then a deformation $H : O \times I \to \mathbb{R}^3$ of the object is $\varepsilon-$***thickness preserving** *with respect to p, if for any $t \in I$ we have*

$$\{x \in \mathbb{R}^3 : \mathrm{dist}(x, H(\iota^{-1}(p(I)), t)) \leq \varepsilon\} \subset H(O,t).$$

We now can formulate a corollary of Proposition 1:

Corollary 1 (of Proposition 1)**.** *Let $\widetilde{l}_{\varepsilon'} = \widetilde{l}_\varepsilon(\Gamma, A)$ be an ε'-augmented caging loop, consisting of the set of segments Γ, representing the caging tools, and a set A of virtual segments, connecting their endpoints. Let $O \subset \mathbb{R}^3$ be an object, and let $p : I \to O$ an embedding into O, such that $\{x \in \mathbb{R}^3 : \mathrm{dist}(x, p(I)) \leq \varepsilon\} \subset O$, where $\varepsilon > \varepsilon'$. Assume that p is linkable to $\widetilde{l}_{\varepsilon'}$ via $S(\widetilde{l}_{\varepsilon'})$-avoiding augmentation, and O does not collide with the set of caging tools: $O \cap \Gamma = \emptyset$. Consider a deformation class $\mathcal{TP}$ of the object O, consisting of $\varepsilon-$thickness preserving deformations with respect to p, non-penetrating with respect to Γ. Then for any isotopy $H \in \mathcal{TP}$, such that $H(\iota^{-1}(p(I)), 1) \cap S(\widetilde{l}_{\varepsilon'}) = \emptyset$, there exists $t \in I$ for which either $H(\iota^{-1}(p(0)), t) \in S(\widetilde{l}_{\varepsilon'})$ or $H(\iota^{-1}(p(1)), t) \in S(\widetilde{l}_{\varepsilon'})$.*

Proof. By Proposition 1 we know that the statement holds for the deformation class $\mathcal{TP}' \subset \mathcal{TP}$ consisting of $\varepsilon-$thickness preserving deformations with respect to p, non-penetrating with respect to $\widetilde{l}_{\varepsilon'}$. Suppose that the statements does not hold for $\mathcal{TP}$. Let $H \in \mathcal{TP} - \mathcal{TP}'$ be such an isotopy. Since $H \in \mathcal{TP}$ but $H \notin \mathcal{TP}'$, there exists a moment of time $t \in I$, such that the image $H(\iota^{-1}(p(I)), t)$ passes through at least one of the virtual segments in A. Suppose that $y \in H(\iota^{-1}(p(I)), t) \cap A$. On the one hand, note that $\mathrm{dist}(y, \Gamma) \leq \varepsilon'$, since the length of the longest segment in A is ε'. Hence, $\{x \in \mathbb{R}^3 : \mathrm{dist}(x, y) \leq \varepsilon'\} \cap \Gamma \neq \emptyset$. On the other hand, H is $\varepsilon-$thickness preserving with respect to p, $y \in H(\iota^{-1}(p(I)), t)$, and therefore $\{x \in \mathbb{R}^3 : \mathrm{dist}(x, y) \leq \varepsilon\} \subset H(O,t)$. Recall that $\varepsilon > \varepsilon'$, and hence $\{x \in \mathbb{R}^3 : \mathrm{dist}(x, y) \leq \varepsilon'\} \subset \{x \in \mathbb{R}^3 : \mathrm{dist}(x, y) \leq \varepsilon\} \subset H(O,t)$. So, $H(O,t) \cap \Gamma \neq \emptyset$, which yields a contradiction. □

Corollary 1 implies the sufficient conditions for caging objects with double forks and necks features by several caging tools.

Proposition 2. *Let $O \subset \mathbb{R}^3$ be an object under consideration, and let $\widetilde{l}_{\varepsilon'}$ be an $\varepsilon'-$augmented caging loop, representing the set of manipulators. Let $p, p' : I \to O$ be two embeddings, linkable to $\widetilde{l}_{\varepsilon'}$ via $S(\widetilde{l}_{\varepsilon'})$-avoiding augmentations and suppose that p, p' form a double fork with parameter $d > 0$. Let $\varepsilon > \varepsilon'$ and assume $DF = DF_\varepsilon(p(I), p'(I))$ to be a subset of O. Finally, assume $O \cap \Gamma = \emptyset$.*

Consider the deformation class $\mathcal{FP}_{aug}$ of the object O, such that any deformation H in $\mathcal{FP}_{aug}$ is non-penetrating

with respect to Γ*, double fork preserving with respect to* DF*, and* $\varepsilon-$*thickness preserving with respect to both of* p *and* p'*.*

If $\operatorname{diam}(S(\widetilde{l}_{\varepsilon'})) < d$ *then* $\widetilde{l}_{\varepsilon'}$ *cages the object* O *under* $\mathcal{FP}_{aug}$*-deformations.*

Proof. Suppose on the contrary that for any $C \geq 0$ there exists an isotopy H in class $\mathcal{FP}_{aug}$ taking the object O at distance greater than C away from $\widetilde{l}_{\varepsilon'}$. Let $C = 2 \cdot \operatorname{diam}(S(\widetilde{l}_{\varepsilon'}))$. Then, the images $H(\iota^{-1}(p(I)), 1) \subset \mathbb{R}^3$ and $H(\iota^{-1}(p'(I)), 1) \subset \mathbb{R}^3$ of p and p' are separated from $\widetilde{l}_{\varepsilon'}$ at a distance greater than $\operatorname{diam}(S(\widetilde{l}_{\varepsilon'}))$. Therefore, the paths $H(\iota^{-1}(p(I)), 1)$ and $H(\iota^{-1}(p'(I)), 1)$ do not intersect $S(\widetilde{l}_{\varepsilon'})$ and therefore are not linkable to $\widetilde{l}_{\varepsilon'}$ via $S(\widetilde{l}_{\varepsilon'})$-avoiding augmentation. From Corollary 1 we know that there exists a first time t at which one of the endpoints of the deforming p, p' is in $S(\widetilde{l}_{\varepsilon'})$. Further reasoning precisely as in Theorem 1, we obtain the result. □

Let us now consider the objects with neck feature. Note that we do not have to impose any additional restrictions on the deformation class $\mathcal{NP}$ from Theorem 2, since any deformation which is neck preserving with respect to some $N_{\varepsilon,d}(f(I))$ is obviously ε-thickness preserving with respect to f. Therefore, the statement of Theorem 1 holds for any ε'-augmented caging loop, provided $\varepsilon' < \varepsilon$, where ε is the second neck's parameter.

VI. Algorithmic implementation

In this section, we provide algorithms for computing and verifying caging configurations. Our approach analyses the shape of the object and extracts double forks and necks contained in it. Since a neck feature can be considered as a special but important case of a double fork, which provides us more freedom in the choice of a manipulator, we start the shape analysis of the object with searching for necks in it. If no necks are detected, we search for double forks. We furthermore generate caging loops using the extracted information and present applications with a PR2 robot. Finally, we run our verification algorithm on the synthesized configurations to prove that the resulting configurations are valid cages.

In this section, we assume that the object's shape is represented as a mesh O. The difference between the object $\mathcal{O}$ and its shape O is important for theoretical reasoning, as it allows us to consider deformations of the object. However, we do not model deformations from the algorithmic point of view: we start with the initial shape of the object and analyse its features, assuming they are preserved by the possible deformations. Hence, in this section we do not distinguish between the object and its shape. Mathematically, this means that $\mathcal{O} = O \subset \mathbb{R}^3$ throughout this section.

A. 'Neck' recognition

Let us discuss the computation of neck features. A general approach to determine necks has been introduced in [11] based on the concept of persistent homology (see [10]). Here, we describe a simplified approach that is applicable in our setting. Let O be a tetrahedral mesh, representing the object. If we are given a surface triangle mesh, it can be tetrahedralized at a preprocessing stage, for example with the help of Tetgen([28]). We start with an empty set and then incrementally add vertices from O based on their distance to the surface ∂O of the object. We also add edges, triangles and tetrahedra as soon as all of their vertices are added. Let $\mathrm{ds} : O \to \mathbb{R}$ be a function reflecting the distance to ∂O: $\mathrm{ds}(x) = \operatorname{dist}(x, \partial O)$. At each step i of the Alg. 1, we consider a superlevel $O^+_{a_i} = \{x \in O : \mathrm{ds}(x) \geq a_i\}$, where the parameter a_i is increased at each step: $\max_{x \in O} \mathrm{ds}(x) = a_0 > a_1 > a_2 .. > a_n = 0$. As a result, we obtain a nested family of superlevel sets, called a *filtration*:

$$O^+_{a_0} \subset O^+_{a_1} \subset O^+_{a_2} \subset .. \subset O^+_{a_n} = O.$$

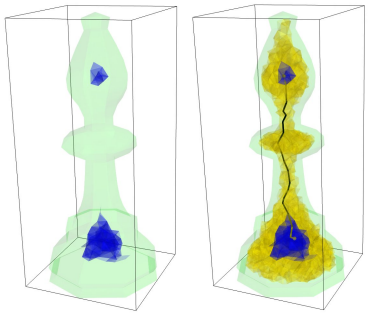

Fig. 14. Superlevels of the function reflecting the distance to the boundary

In this process, the number of connected component in the filtration changes: new components appear, while the others merge. In particular, we are interested in those points, after the addition of which different connected components in the filtration $O^+_{a_i}$ merge. Let us illustrate the example in Fig. 14, where we depict two different superlevel sets, corresponding to filtration values $a_i > a_j$. The superlevel set $O^+_{a_i}$ consists of two disjoint components and is depicted in blue. It is a subset of $O^+_{a_j}$, which is connected and depicted in yellow on the right side of the figure. A path f depicted in black connects the disjoint components of $O^+_{a_i}$ through $O^+_{a_j}$. Note that the distance from f to the boundary of the object is not less than a_j, and the distance from its endpoints to the boundary is greater or equal than a_i. Therefore, we can conclude that the shape under consideration contains a neck $N_{a_j,a_i}(f)$.

Let us now consider our technique in detail. Alg.1 computes necks with parameter d, given a tetrahedral mesh O and a parameter d. A mesh O consists of four sets: V, E, F and T, representing vertices, edges, faces and tetrahedra respectively. The subset $\partial O \subset O$ contains those vertices which are located at the boundary of the object. For each vertex, edge, face and tetrahedron we store a binary flag which has value 'True' if the object is located at the boundary ∂O, and 'False' otherwise. We represent a neck as a path f inside the object O such that its endpoints are located at a distance d (or greater) from the boundary ∂O of the object, while the distance from it to the path f itself is not less than ε. We therefore store a path as a list of edges from O as well as the parameters d and ε for each neck. At each step, we decrease a and add to the filtration O^+_a points from O located at a distance not less than a to the surface ∂O. Once both of the endpoints of an edge are in the filtration, we add this edge to the filtration as well. Analogously, we add faces (tetrahedra) once

all of their edges (faces) are in the filtration. We do this by means of function **Update-Filtration**. Once all vertices, edges, triangles and tetrahedra are added, we compute the connected components of the resulting mesh. At each moment of time, each connected component has its unique ID number. For each vertex, we store the ID number of the connected component containing it in the array $ConComp$. ID of a connected component is an integer number. When we add a new point v to the filtration, we consider all the vertices adjacent to it in O. If none of them is in the filtration, then v forms a new connected component in O_a^+. We let $ConComp[v]$ be equal to the minimal integer number which has not yet been used to denote any connected component. Otherwise, we add v to an already existing connected component by letting $ConComp[v]$ to be equal to the ID of the corresponding component. If in the filtration there are more than one connected component containing vertices adjacent to v in O, then this means that adding v results in the merging of these components. Assume that $C_1, C_2, .., C_m$ are the connected components in the filtration containing at least one vertex, adjacent to v in O. Then, once we add v, we modify $ConComp$ so that all the vertices from $C_1, C_2, .., C_m$ and v have the same value in it. For this purpose, we select the component with the minimal ID number among the components $C_1, C_2, .., C_m$, and assign $ConComp[u]$ to be equal to it for all the vertices u from $C_1 \cup C_2 \cup .. \cup C_m \cup \{v\}$. This mechanism is implemented in the function **Merge-Comp**.

Suppose $C_1, C_2 \subset O_a^+$ are two connected components, merged into one after the addition of a point v together with the corresponding edges, faces and tetrahedra to the filtration. Let $x_1 = \arg\max_{x \in C_1} \mathrm{ds}(x)$, $x_2 = \mathrm{argmax}_{x \in C_2} \mathrm{ds}(x)$. We can determine a path f connecting x_1 and x_2 in O_a^+. This path is computed by **Find-Path**, resulting in a neck, whose first parameter is equal to $\min\{\mathrm{dist}(x_1, \partial O), \mathrm{dist}(x_2, \partial O)\}$. The values of their second parameter, reflecting the thickness of the object, may vary and are not computed at this stage. Instead, we compute them further if necessary.

Algorithm 1: Compute-Necks(mesh O, mesh ∂O, float d)

input : a tetrahedral mesh $O = (V, E, F, T)$;
a surface mesh representing the boundary of the object ∂O;
neck parameter d
output: a set of necks with parameter d

```
nextID ← 0
N_d ← ∅
foreach v ∈ O do
    D[v] ← Compute-Dist-To-Surface(v, O)
end
V ← Sort-Vertices-by-Distance(V, D)
a =← Max-Element(D)
while a > 0 do
    NewVert ← Vertices-at-Distance(V, d, D)
    O_a^+ ← Update-Filtration(O, O_a^+, NewVert)
    foreach v ∈ NewVert do
        ConComp[v] ← ∞
        size ← 0
        f ← ∅
        foreach u ∈ Adjacent(v) do
            if u ∈ O_a^+ and ConComp[u] ≠ ConComp[v] then
                if ConComp[v] ≠ ∞ then
                    uCentre ← ConComCentre(V, ConComp, u)
                    vCentre ← ConComCentre(V, ConComp, v)
                    size ← Max(size, Min(D[uCentre], D[vCentre]))
                    Merge-Comp(u, v, ConComp)
                    if size ≥ d and f = ∅ then
                        f ← Find-Path(O_a^+, uCentre, vCentre)
                    end
                end
                else
                    ConComp[v] ← ConComp[u]
                end
            end
        end
        if ConComp[v] = ∞ then
            ConComp[v] ← nextID
            nextID ← nextID + 1
        end
        if size ≥ d then
            N_d = N_d ∪ f
        end
    end
    a ← Next-Distance-To-Surface(O, O_a^+, D, a)
end
return N_d
```

B. 'Double fork' recognition

We start by generating a skeleton $G = (V, E)$ of an object using the algorithm of [7]. This algorithm generates a skeleton as a subset of the medial axis. More precisely, the authors consider balls of maximal radius defined by the medial axis. These balls touch the object's surface and the authors define the **medial geodesic function (MGF)** to be the length of the shortest surface path connecting these points. The skeleton is then defined as the singular set of MGF. Unlike the medial axis, which might be a two-dimensional surface, this skeleton is always one-dimensional and represented as a graph. The MGF values are provided for each edge of the skeleton graph as a part of the output of their software and provide size information about the object.

We denote the MGF value by $\mathrm{geod}(e)$ for $e \in E$. We also extend this function to vertices: $\mathrm{geod}(v) = \min_{i \in \{1,..,n\}} \mathrm{geod}(e_i)$, where $\{e_1, .., e_n\} \subset E$ is the set of edges incident to $v \in V$. In our current implementation, we assume the skeleton graph G to be a tree. In the future, we plan to consider more general skeletons as well, for example, by working with spanning trees.

Our goal is to find double forks in the skeleton graph. We are interested in double forks with a parameter greater than or equal to some predefined threshold d. We stop the algorithm once a double fork with a required parameter is found. Otherwise, the algorithm terminates after considering all the promising sets of vertices of the graph.

All the graphs in this section are assumed to be undirected. For convenience, we introduce a special notation for paths in graphs. Let $p : u \to v$ denote a path between the vertices u and v in a graph. We will also use the notation (u, v) for an edge between vertices u and v.

Definition 20. *Consider a skeleton as a graph $G = (V, E)$. Let $G_v^d \subset G$ be the subgraph, defined by $G_v^d = (\{u \in V : \mathrm{dist}(u, v) \geq d\}, \{e \in E : \mathrm{dist}(e, v) \geq d\})$ for some $v \in V$ and $d > 0$. Then the subgraph G_v^d is called d-***separated from*** v.*

In the above definition, $\mathrm{dist}(.,.)$ denotes the Euclidean distance in $\mathbb{R}^3$. Intuitively, G_v^d is just a subgraph of G separated from v by a distance not less than d. Note that G_v^d is likely do consist of several connected components.

Recall the definition of a double fork. It imposes two conditions on the distances between the parts of the double fork: $\mathrm{dist}(p, \{u', v'\}) > d$ and $\mathrm{dist}(p', \{u, v\}) > d$. These conditions can be rewritten as $p \subset G_{u'}^d \cap G_{v'}^d$ and $p' \subset G_u^d \cap G_v^d$. Consequently, the pairs $\{u, v\}$ and $\{u', v'\}$ are also separated from each other:

$$u, v \in G_{u'}^d \cap G_{v'}^d, \qquad u', v' \in G_u^d \cap G_v^d.$$

Moreover, any double fork is obviously connected due to condition $p \cap p' \neq \emptyset$. From this we conclude that the centre C of a double fork must lie at a distance not less than d from all the vertices u, v, u', v', and therefore we have the following condition:

$$C \subset G_u^d \cap G_v^d \cap G_{u'}^d \cap G_{v'}^d \neq \emptyset$$

These observations give us a way to compute double forks in G, see Alg. 2.

Let d be a threshold determining the minimal acceptable parameter of a double fork. As a preprocessing stage, we compute a real valued symmetric $|V| \times |V|$ matrix D, containing pairwise distances between vertices in G: $D_{v_i, v_j} = \mathrm{dist}(v_i, v_j)$ for each pair $(v_i, v_j) \in V \times V$. For this purpose we use the function **Distance**.

The key idea of this algorithm is to find a set of vertices $\{u, v, u', v'\}$ satisfying all the conditions to form a double fork. In the first step of our algorithm, we are searching for vertices that might be located at the centre of some double fork. We enumerate all the vertices in the graph and consider those which potentially could be located at a centre of a double fork. In order to speed up the computations, we consider only those vertices which are located at a distance not greater than $0.2R_G$, where $R_G = \mathrm{dist}(x, M)$ and M is the centre of mass of the set $V \subset \mathbb{R}^3$. This selection is performed by the function **Vert-at-Dist**, and the centre of mass is the output of the corresponding function **Centre-of-Mass**. Then, for each selected vertex a we enumerate all the vertices in the graph at a distance not less than d to a, and consider them as candidates for the first vertex of the double fork – u. At this step, we build a path $p_u : a \to u$ in G. The function **Find-Path** is used for this purpose. After that, for each pair (a, u), we enumerate all the vertices at a distance greater or equal than d to p_u and consider them as promising candidates for u'. We search for a path $p_{u'} : a \to u'$ in G_u^d. If there is no such a path, we consider the next candidate for u'. Once $p_{u'}$ is found, having a triple (a, u, u'), we enumerate all the vertices in G which are located at a distance greater or equal than d to $p_{u'}$, and consider them as candidates for being v. We search for a path $p_v : a \to v$ in $G_{u'}^d$. Finally, having $\{a, u, u', v\}$ we enumerate the vertices separated from p_u and p_v at least at a distance d and choose v' among them. Now u, v, u', v' together with $p = p_u \cup p_v$ and $p' = p_{u'} \cup p_{v'}$ form a double fork.

Let us now discuss in detail how the above used functions are implemented.

Algorithm 2: Compute-Double-Fork(mesh O, graph $G = (V, E)$, float d)

```
input : a mesh O, representing the object;
        a skeleton G = (V, E);
        the lowest acceptable double fork parameter d
output: a fork of parameter d
foreach u ∈ V do
    foreach v ∈ V do
        D_{u,v} ← Distance(u, v)
    end
end
M ← Centre-of-Mass(G)
R ← 0
foreach v ∈ V do
    if D_{v,M} > R then
        R ← D_{v,M}
    end
end
foreach a ∈ V do
    if dist (a, M) ≤ 0.2R then
        UC ← Vert-at-Dist(V, {a}, d)
        foreach u ∈ UC do
            p_u ← Construct-Path(G, a, u)
            if p_u = ∅ then
                continue
            end
            U'C ← Vert-at-Dist(V, p_u, d)
            foreach u' ∈ U'C do
                p_{u'} ← Construct-Path(G, a, u')
                if p_{u'} = ∅ then
                    continue
                end
                VC ← Vert-at-Dist(V, p_{u'}, d)
                foreach v ∈ VC do
                    p_v ← Construct-Path(G, a, v)
                    if p_v = ∅ then
                        continue
                    end
                    V'C ← Vert-at-Dist(V, p_u ∪ p_v, d)
                    foreach v' ∈ V'C do
                        p_{v'} ← Construct-Path(G, a, v')
                        if p_{v'} = ∅ then
                            continue
                        end
                        p = p_u ∪ p_v
                        p' = p_{u'} ∪ p_{v'}
                        return {u, v, u', v'}, p ∪ p'
                    end
                end
            end
        end
    end
end
return ∅
```

- **Centre-of-Mass(Graph G)**
  Given a geometric graph $G = (V, E)$, this function computes a centre of mass of $V \subset \mathbb{R}^3$.
- **Vert-at-Dist(Set S, Set D, Float d)**
  This function returns a subset S' of the set $S \subset \mathbb{R}^3$, such that each point from S' lies at a distance greater than or equal to the parameter $d \in \mathbb{R}$ to any point of the set $D \subset \mathbb{R}^3$.
- **Find-Path(Graph G, Vertex u, Vertex v)**
  The goal of this function is to find a path between the vertices u and v in the undirected graph G. For this purpose, we run Breadth-first search in G starting at u, and then recover the path $p : u \to v$ once v is reached. If we cannot reach v from u, the function returns empty

path.

C. Caging loop synthesis

Let us now describe an approach towards constructing caging loops. We assume that we have already analysed the shape of the object O, and have found either a neck formed by a path p, or a double fork formed by a pair of paths (p, p').

In our algorithm, we consider the edges contained in the object's surface ∂O as a graph G. The output of the algorithm is given by a piecewise linear loop l in this graph. In order to construct a loop, we need a starting point M inside O. Intuitively, M is the point around which we construct a loop, and it is located at the centre of a narrow part of the object. Our algorithm aims to construct a short loop as follows.

Formally, assume we have an object with a double fork (p, p'). Let $M \in p \cap p'$ be a point, such that $M = \arg\min_{x \in p \cap p'}(\text{geod}(x))$. In the case of a neck, let $M = \arg\min_{x \in p}(\text{ds}(x))$. In order to unify our notation for both cases Algorithms 3, 4, 5, 7 accept as input both p and p', however, we set $p' = \emptyset$ in case of a neck features.

We start by selecting a vertex $v_0 = \arg\min_{v \in \partial O} \text{dist}(v, M)$. We will explore the vertices of ∂O in the increasing order of their geodesic distance to v_0. Here we approximate the geodesic distance by considering the shortest edge paths, determined by means of the Dijkstra's algorithm, as explained below. Let the set $Vertices$ contain the vertices from ∂O. We gradually add explored vertices from $Vertices$ to the set $Marked$. So, at the initial stage $Marked = \emptyset$. We will also need a graph G_{marked} consisting of the vertices from the set $Marked$, and the edges from ∂O such that both their endpoints are in $Marked$. Initially, G_{marked} is empty.

For each vertex v in G, we need their distance to the initial point v_0. By distance here we mean an approximation of the geodesic distance – the geometric length of the minimal edge path from v_0 to v. In addition, for each vertex v we will store its immediate ancestor in the shortest path from v_0 to v. For example, assume that we have a path $path = \{v_0, v_1, ..., v_n, v\}$. The length of this path is the sum of the geometric length of its edges, and the immediate ancestor of v in this path is v_n. This information will be stored in the arrays $Distance$ and $Ancestor$, where $Ancestor[v]$ denotes the immediate ancestor of the vertex v in the shortest path from v_0, and $Distance[v]$ is the length of this path. We run Dijkstra's algorithm in G starting from the point v_0, in order to compute the arrays $Distance$ and $Ancestor$.

For convenience, we sort the vertices in $Vertices$ in an increasing order of their distance from v_0 by running **Sort-Vertices-by-Distance**. We explore the vertices from $Vertices$ in this order, and add them to the set $Marked$. When two adjacent vertices are added to $Marked$, we add the edge between them to G_{marked}. In this process, the graph G_{marked} grows on the surface of the object, and tends to cover the entire surface ∂O (i.e., to become equal to G).

Although in our algorithm we consider only a 1-skeleton of ∂O and therefore do not work with its faces, in Fig. 15 we visualize in purple the faces whose edges and vertices are in G_{marked}. Intuitively, the process of adding vertices and edges to G_{marked} resembles a gradual growth of a surface patch on the mesh, see Fig. 15. Since the object is bounded, the boundary vertices of this 'patch' are becoming closer and closer to each other, and at some moment merge. In Fig. 15, we illustrate this merging for the neck of the bear mesh. When this merging occurs, the resulting closed edge path forms a short loop around the narrow part of the object which we return as a caging loop candidate in Alg.3.

Algorithm 3: Caging-Loop-Generation(mesh O, path p, path p')

input : a mesh O, representing the object ;
two paths p and p' inside O, representing the double fork or neck
output: a caging loop l

if $p' = \emptyset$ **then**
  // If $p' = \emptyset$, then we deal with a neck formed by p
  $M \leftarrow \arg\min_{x \in p}(\text{ds}(x))$
end
else
  // If $p' \neq \emptyset$, then we deal with a double fork formed by (p, p')
  $M \leftarrow \arg\min_{x \in p \cap p'}(\text{geod}(x))$
end
Vertices $\leftarrow \{v : v \in \partial O\}$
Marked $\leftarrow \emptyset$
$v_0 \leftarrow \arg\min_{v \in \partial O}(\text{dist}(v, M))$
$G \leftarrow G(V$ = Vertices, $E = \{e = (v_1, v_2) : v_1, v_2 \in V, e \in \partial O\})$
$G_{marked} \leftarrow G(V = \emptyset, E = \emptyset)$
Distance, Ancestor $\leftarrow$ Dijkstra(G, v_0)
Sort-Vertices-by-Distance(Vertices, Distance)
while *Vertices* $\neq$ *Marked* **do**
  v = Vertices.next()
  foreach $u \in$ *Adjacent*(v) **do**
    if $u \in$ *Marked* **then**
      path $\leftarrow$ Find-Path(G_{marked}, u, Ancestor[v])
      if *Length(path)* $> \deg(v) - 1$ **then**
        $l \leftarrow (v, u) \cup$path $\cup$(Ancestor[v], v)
        if *Verify-Caging-Loop(O, p, p', l)* **then**
          return l
        end
      end
    end
  end
  Marked.push(v)
  Add-Vertex-To-Graph(G_{marked}, (v))
  foreach $u \in$ *Adjacent*(v) **do**
    if $u \in$ *Marked* **then**
      Add-Edge-To-Graph(G_{marked}, (v, u))
    end
  end
end
return $\emptyset$

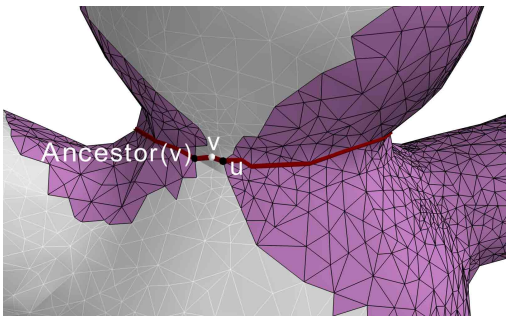

Fig. 15. Illustration of Alg.3. Currently explored vertices and edges are depicted in black; faces whose edges and vertices are black, are depicted in purple and a resulting caging loop is shown in red.

At each step, we consider a vertex $v \in Vertices$. We search for its immediate neighbours in G, and consider those of them which are already in $Marked$. Let $u \in Marked$ be one of these neighbours. First of all, we check whether we can obtain a caging loop already at this step. In order to do this, we search for the edge-wise shortest path in G_{marked} between u and $Ancestor[v]$, see Fig. 15 for the intuition. We search for this path by running **Find-Path**, which is the implementation of the breadth-first path search between two points in a given graph. If the length of this path is long enough (longer than $\deg(v) - 1$ edges, where $\deg(v)$ is the degree of the current vertex v), then we suppose that this path together with the edges $(Ancestor[v], v)$ and (v, u) forms a loop. The reason why we want the path to be longer than $\deg(v) - 1$ is that we need to eliminate a priori degenerate loops consisting only of the neighbours of the vertex v. We then run our verification algorithm in order to check whether this loop is a valid cage. If not, we continue the process. Namely, we mark v and append all the edges connecting it to vertices from $Marked$.

- **Compute-Dist-To-Surface(Vertex v, Mesh O)** In this function, we enumerate all the faces in the boundary ∂O of O, and compute the distance between each face and the vertex v. The distance to the surface ∂O is given by the minimum of these distances.
- **Sort-Vertices-by-Distance(Vertices V, Array D)** This function returns an array containing vertices in a decreasing order of their distances to the surface.
- **Vertices-at-Distance(Vertices V, Float d, Array D)** This functions selects vertices from V located at a distance d to the boundary ∂O. The distances are stored in the array D.
- **Update-Filtration(Mesh O, Mesh O_a^+, Array $NewVert$)** This functions adds the vertices from the array $NewVert$ to the filtration O_a^+. It also adds the corresponding edges, faces and tetrahedra from O: each of them is added once all its vertices are in O_a^+.
- **ConComCentre(Vertices V, Array $ConComp$, Integer u)** Given an ID number $ConComp[u]$ of a connected component, this function returns a point belonging to it, such that its distance to the boundary is the largest in this component.
- **Find-Path(Mesh O_a^+, Integer $uCentre$, Integer $vCentre$)** This function computes a path between $uCentre$ and $vCentre$ in O_a^+ using depth-first search.
- **Next-Distance-To-Surface(Mesh O, Mesh O_a^+, Array D, Float a)** This function returns distance to the object's boundary from the closest vertex in $O - O_a^+$. If $O = O_a^+$, it returns zero.

We have implemented the algorithms for double forks and necks detection in Python, see Fig. 16 for the examples. We utilized Python MayaVi module for the visualization of our meshes. The meshes are taken from [1], [5] and [27]. The computation time for the given objects on an Intel i7 CPU laptop is presented in tables under the figure. In the case of the screen and the table, we did not find any appropriate neck features. Instead, for these objects we managed to find double forks. The drawback of the skeletonization technique we currently use [7] is that it requires meshes of a rather high resolution. For this reason, we consider the screen and the table models in a higher resolution when computing double forks. In contrast, for the neck detection algorithm we use simplified meshes with the lower number of vertices, which speeds up the computation. Both necks and double forks detection algorithms require some preprocessing: in the case of necks we tetrahedralize the mesh, while in the case of double forks we extract a skeleton from the object. In both cases, we rely on already existing software: [28] for tetrahedralization and [7] for skeleton extraction. Both of these techniques are rather efficient, but potentially can be replaced by alternatives.

D. Synthesis of caging configurations for PR2

Since the purpose of this work is to introduce a new representation of objects, suitable for caging, we do not focus on motion planning and control here, however we now discuss two simple approaches to compute caging configurations which we evaluate with a simulated PR2 robot. Consider a PR2 manipulator, and let the vectors V_d and V_h indicated in Fig. 17 define its orientation.

Our first approach is described in Alg. 4. Here, a precomputed approximate caging loop is fit to the PR2 manipulator(s) as follows: Let l_a denote the approximate caging loop, consisting of vertices $\{v_1, v_2, .., v_n = v_1\} \subset \mathbb{R}^3$ and the edges between them, such that each vertex has exactly two adjacent vertices. As a preparatory step, we ensure that the loop is constructed in such a way that any triple $\{v_{i-1}, v_i, v_{i+1}\} \subset l_a$ does not consist of three collinear points - this can be achieved for example by means of an infinitesimal perturbation of the vertices.

First, we compute a point C, which is a centre of mass of all the vertices in the loop. Then we enumerate the vertices in l_a, and for each $v \in l_a$, we take its immediate neighbours, v_i and v_j. The triple $\{v_i, v, v_j\}$ defines a plane π_v. Consider a projection $\pi_v(C)$ of C onto π_v, and let $V_1 = \pi_v(C) - v$. Let $V_2 \perp V_1$ be another vector in π_v, see Fig. 18. We place the manipulator at a distance from the loop and in such a way that V_d is collinear to V_1, and V_h is collinear to V_2. We then linearly approach the object by moving the robot hand direction V_1 towards the object.

Once collision occurs, we close the fingers until collision and extract a new caging loop l_h from the resulting hand configuration, and run our verification algorithm. This approach focuses on narrow parts corresponding to our initial caging loop candidate l_a. Note that due to collisions the hand might however not be able to reach a final caging configuration for any of these approach directions along the loop - in future we plan to focus on motion planning algorithms for this purpose.

A second simple caging synthesis algorithm is described in Alg. 5. Here, we do not compute an initial caging loop l_a but instead, we compute a simple graph G_c, suitable for caging, and attempt to close the PR2's fingers around it. In the case of neck, we let $G_c = p$, where p is the path forming the neck while in the case of a fork formed by the paths p and p', we let $G_c = p \cap p'$.

For any $e \in G_c$, we perform the procedure described in Alg. 6. Let M be a centre of mass of the segment e. Consider

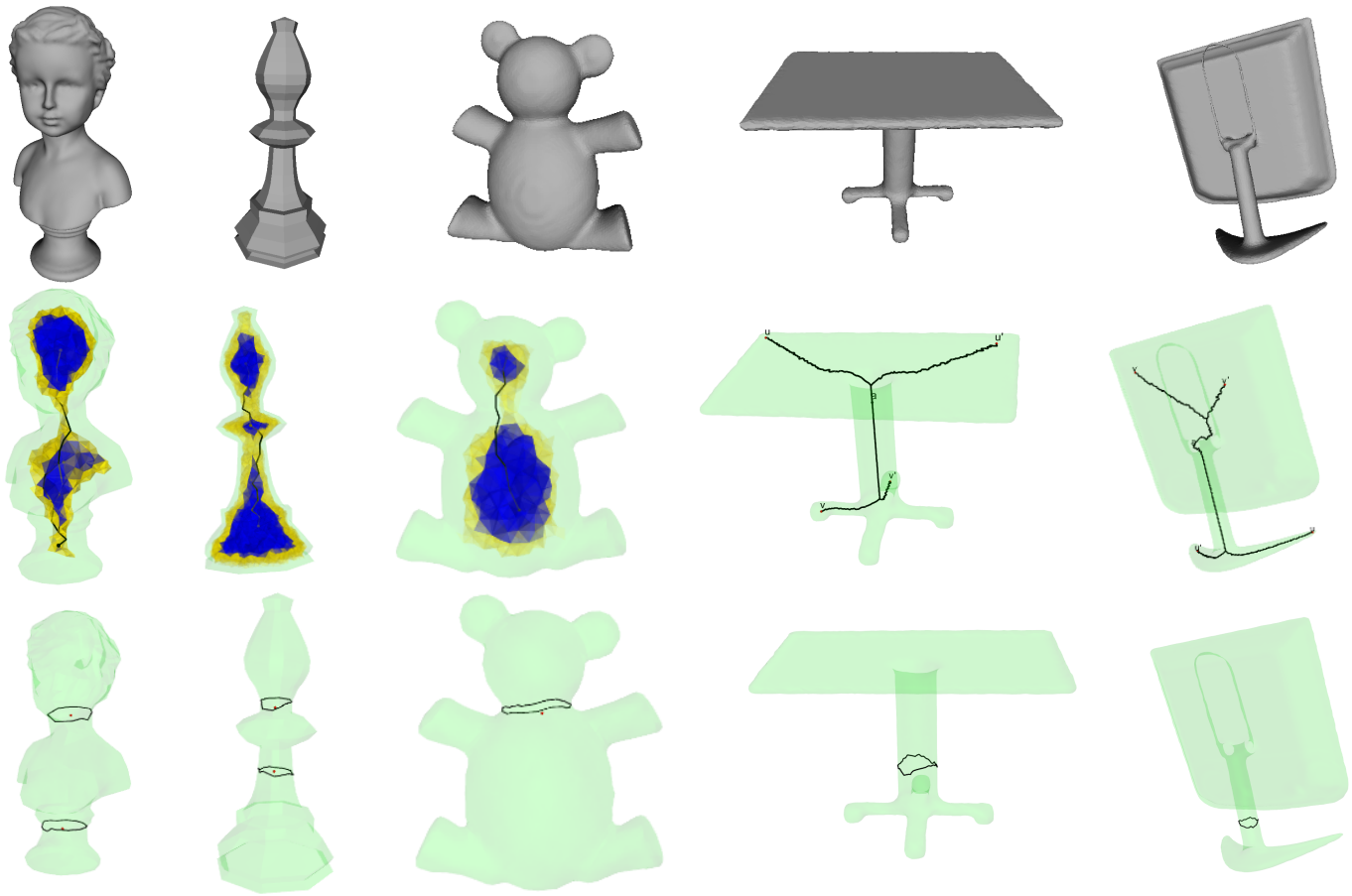

Fig. 16. In the first row, the original meshes are presented. In the second row, shape features are extracted. The first three columns depict objects for which we extracted neck features, while the last two columns show objects with double fork features. We show corresponding filtrations for the first three columns in the second row. The connected components depicted in blue merge to the larger components depicted in yellow. The paths, connecting the blue components, are depicted in black. In the last two columns the objects are depicted together with the double forks in the second column. The third row displays resulting caging loops.

model	number of vert. in the tetr. mesh	number of vert. on the surface	necks detection	loop(s) construction
bust	12862	7047	1524 ms	1203 ms
queen	5990	3589	682 ms	297 ms
bear	12939	7106	2472 ms	1458 ms
table	1661	1464	1901 ms	–
screen	21917	11109	5317 ms	–

model	number of vert. in the skeleton	number of vert. in the mesh	double forks detection	loop(s) construction
table	1148	10650	1776 ms	748 ms
screen	1584	16218	3799 ms	613 ms

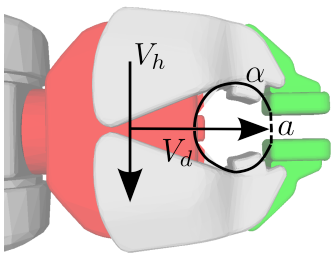

Fig. 17. The vectors V_d and V_h define the direction of the hand. The curve α together with its augmentation a forms an augmented caging loop, representing the manipulator.

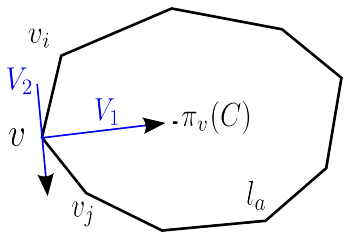

Fig. 18. The approximate caging loop l_a and the vectors V_1 and V_2. The former is pointing towards the centre of the loop, while the latter is perpendicular to it.

the plane μ orthogonal to e and passing through M. We sample a random vector V_1 in μ with respect to uniform distribution,

Algorithm 4: Loop-Based-Caging-Generation(mesh O, path p, path p', loop l_a)

input : a triangle mesh O, representing the object;
two paths p and p' in O, representing the double fork or neck;
an approximate caging loop $l_a \subset \partial O$
output: a caging configuration c_f

$C \leftarrow$ Centre-Of-Mass(l_a)
foreach $v \in l_a$ **do**
  $v_i, v_j \leftarrow$ Adjacent-Vertices(v, l_a)
  $\pi_v \leftarrow$ Plane(v, v_i, v_j)
  $\pi_v(C) \leftarrow$ Project-Onto-Plane(π_v, C)
  $V_1 \leftarrow (\pi_v(C) - v)$
  $V_2 \leftarrow$ Compute-Orthogonal-Vector(π_v, V_1, v)
  EEF-position$\leftarrow$ Collision-Free-EEF-Position (V_1, V_2, O)
  $c_0 \leftarrow$ Solve-IK(EEF-position)
  $c_f \leftarrow$ Move-Until-Collision(c_0, V_1, O)
  $c_f \leftarrow$ Close-Gripper-Until-Collision()
  $l \leftarrow$ Extract-Caging-Loop(c_f)
  if *Verify-Caging-Loop(O, p, p', l)* **then**
    return c_f
  end
end
return $\emptyset$

Algorithm 5: Region-Based-Caging-Generation(mesh O, path p, path p')

input : a triangle mesh O, representing the object ;
two paths p and p' in O, representing the double fork or neck
output: a caging configuration c_f

$G_c \leftarrow \emptyset$ // G_c is a 1D cageable subset of O
if $p' = \emptyset$ **then**
  // If $p' = \emptyset$, then we deal with a neck formed by p
  $G_c \leftarrow p$
end
else
  // If $p' \neq \emptyset$, then we deal with a double fork formed by (p, p')
  $G_c \leftarrow p \cap p'$
end
foreach $e \in p \cup p'$ **do**
  $w_1, w_2 \leftarrow$ Incident(e)
  if *($w_1 \in p \cap p'$* **and** *$w_2 \notin p \cap p'$)* **or** *($w_1 \notin p \cap p'$* **and** *$w_2 \in p \cap p'$)* **then**
    $G_c \leftarrow G_c \cup \{e\}$
  end
end
foreach $e \in G_c$ **do**
  $c_f \leftarrow$ Cage-Around-Edge(O, p, p', e)
  if $c_f \neq \emptyset$ **then**
    return c_f
  end
end
return $\emptyset$

Algorithm 6: Cage-Around-Edge(mesh O, path p, path p', edge e)

input : a triangle mesh O, representing the object ;
two paths p and p' in O, representing the double fork or neck ;
an edge e, around which the object should be caged
output: a caging configuration c_f

$M \leftarrow$ Middle-of-Segment(e)
$i \leftarrow 0$
while $i < 50$ **do**
  $i \leftarrow i + 1$
  $V_1 \leftarrow$ Generate-Random-Orthogonal-Vector($\vec{e}$, M)
  $V_2 \leftarrow \vec{e} \times V_1$
  EEF-position$\leftarrow$ Collision-Free-EEF-Position (V_1, V_2, O)
  $c_0 \leftarrow$ Solve-IK(EEF-position)
  $c_f \leftarrow$ Move-Until-Collision(c_0, V_1, O)
  $c_f \leftarrow$ Close-Gripper-Until-Collision()
  $l \leftarrow$ Extract-Caging-Loop(c_f)
  if *Verify-Caging-Loop(O, p, p', l)* **then**
    return c_f
  end
end
return $\emptyset$

and compute another vector V_2 in μ, orthogonal to V_1. Then, we place the robot's hand into a collision free configuration such that V_d is collinear to V_1, V_h is collinear to V_2, and the end effector is located at a small distance from M. We then move the hand towards M until collision and close the fingers. Finally, we extract the caging loop representation from the manipulator's configuration, and run our verification algorithm in order to check whether our configuration is a cage. If the verification fails, we sample another random vector V_1 and repeat the procedure. If we are not able to construct a caging configuration for the edge e after repeating this procedure 50 times, we consider the next edge from G_c.

If the object is big enough, we can also use two hands of the PR2 as caging tools. In this case, each of the hands approaches the object, as described above. The only restriction we impose is that the angle between the vectors, defining the direction of each hand, must be between $\frac{3\pi}{4}$ and π. This is to ensure that the hands are placed roughly in opposition to each other. Since now we are dealing with two independent manipulators, a single augmented caging loop runs through both hands, as described in the previous section. We extract the caging loop representation and run the verification algorithm.

We have implemented our cage generation algorithms using OpenRave. They are simple, as in this paper we only want to illustrate how one can cage an object given its shape features and precomputed examples of caging loops. Here we provide some illustrations of the successful output, Fig. 19. A study on how to choose a caging generation algorithm (the loop-based or the region-based) and how many grippers to use for each particular object is beyond the scope of this paper. We plan to get back to this problem and incorporate it with motion planning in the near future. When the position of the robot base and the object are chosen properly, and objects features are computed, it takes 10-15 seconds at the average to compute and verify a caging configuration. In the case of a bear mesh we have generated the cage based on a previously computed approximate caging loop (see Fig. 16). For the shown monitor mesh, this approach however failed due to collisions of the robot hand as the upper part of the monitor prevents the robot hand from approaching it. Our second approach however returns a cage utilizing both of the PR2's hands. In the case of a wine glass, we computed a neck and both of our caging methods described above succeeded. The figure displays the caging configuration of the PR2 hand computed with the latter algorithm based on G_c.

E. Verification of a given caging configuration

Let us now discuss how to verify whether a configuration constructed by our algorithm is a valid cage. Theorems 1, 2 provide us with a constructive way to perform such a verification. Indeed, assume that we are given an object $O \subset \mathbb{R}^3$,

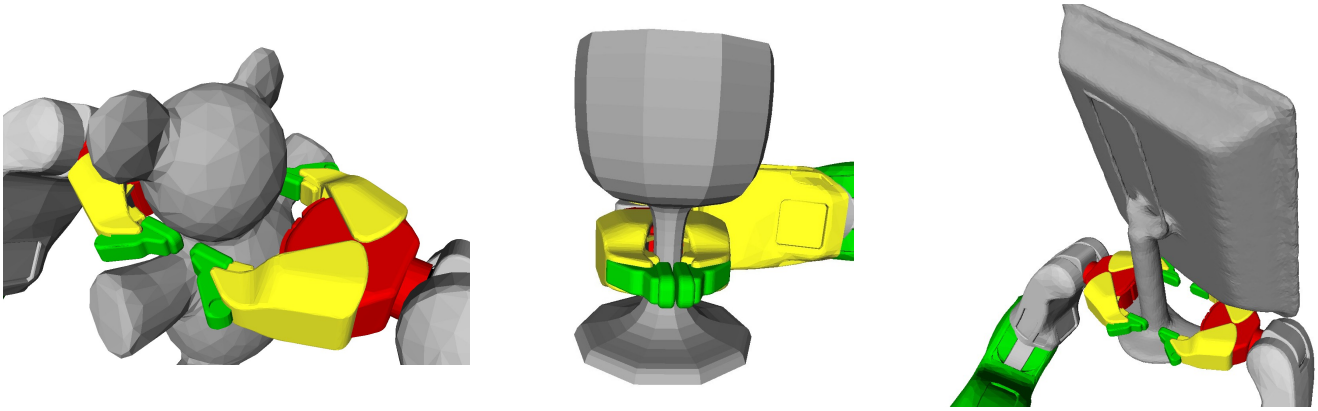

Fig. 19. In the first case, the caging configuration was generated based on the loop around the neck of the bear. At the second figure, the cage was generated based on a neck representation of the object (second approach). In the third case, we did not manage to fit the precomputed approximate caging loop due to the obstacle presence. The caging configuration was generated based on the double fork representation (second approach).

Algorithm 7: Verify-Caging-Loop(mesh O, path p, path p', loop l)

input : a tetrahedral mesh O, representing the object;
paths $p, p' \in \mathbb{R}^3$;
a loop l, representing a rigid manipulator
output: 'l is a valid cage' – True or Undecided

$\varepsilon \leftarrow$ Get-Loop-Augmentation(l)
$Conv \leftarrow$ Convex-Hull(l)
$p_{aug} \leftarrow$ Construct-Augmentation(p, $Conv$)
if *no augmentation found* **then**
  return *Undecided*
end
if $p' = \emptyset$ **then**
  // If $p' = \emptyset$, then we deal with a neck formed by p
  if $\varepsilon \leq \min_{x \in p}(\mathrm{ds}(x))$ **then**
    if $lk(p, p_{aug}) \neq 0$ **then**
      return *True*
    end
    else
      return *Undecided*
    end
  end
  else
    return *Undecided*
  end
end
else
  // If $p' \neq \emptyset$, then we deal with a double fork formed by (p, p')
  $p'_{aug} \leftarrow$ Construct-Augmentation(p', $Conv$)
  if *no augmentation found* **then**
    return *Undecided*
  end
  if $\varepsilon \leq \min_{x \in (p \cup p')}(\mathrm{ds}(x))$ **then**
    if $lk(p, p_{aug}) \neq 0$ **and** $lk(p', p'_{aug}) \neq 0$ **then**
      return *True*
    end
    else
      return *Undecided*
    end
  end
  else
    return *Undecided*
  end
end

and a piecewise-linear closed curve l, representing the caging tool. Our goal is to check whether l bounds the mobility of O.

As input to our verification algorithm, we consider l, together with some paths $p, p' \subset O$ (allowing one of them to be empty in case of a neck feature). Alg.7 then determines if p is linkable to l via an $S(l)-$avoiding augmentation, where we consider $S(l) = \mathrm{Conv}(l)$ To do so, we compute $S(l)$ and an augmentation p_{aug} (and similarly p'_{aug}, in the case of a double fork), connecting the endpoints of p (and p') through the complement of $S(l)$ by means of **Construct-Augmentation(Path f, Set** $\mathrm{Conv}(l)$**)**. We then compute the linking number $lk(p, p_{aug})$ and test if $\varepsilon \leq \min_{x \in p}(\mathrm{ds}(x))$ (neck case) or $\varepsilon \leq \min_{x \in (p \cup p')}(\mathrm{ds}(x))$ (double fork case) to apply Theorems 2 and 1 respectively.

In detail, **Construct-Augmentation(Path f, Set** $\mathrm{Conv}(l)$**)** constructs a path f', connecting the endpoints u, v of f via the complement of $\mathrm{Conv}(l)$ and returns failure if u or v lies in $\mathrm{Conv}(l)$. We linearly rescale $\mathrm{Conv}(l)$ by a factor of 2 and call the resulting set M, so that $\mathrm{Conv}(l)$ is strictly contained in M. We then determine closest vertices $u' = \arg\min_{x \in \partial M}(\mathrm{dist}(x, u))$ and $v' = \arg\min_{x \in \partial M}(\mathrm{dist}(x, v))$ and return an edge path p from u' to v' through the edges of ∂M using breadth-first search on the graph of edges of ∂M. The function then returns the concatenated path $f' = (u, u') \cup p \cup (v', v)$.

VII. Conclusions and Future Work

We proposed a methodology for the synthesis and provably correct verification of caging grasps on a class of 3D objects that exhibit geometric features which we call necks and double forks. We utilized the classical notion of the linking number from algebraic topology as the basis for proving the sufficient conditions for caging grasps and have proposed algorithms to compute necks, double forks as well as resulting caging loops and caging configurations for a PR2 manipulator. In future work, we are planning to integrate motion planning and to provide an extensive experimental evaluation of the technique on natural objects and different caging tools. Apart from this, we plan to try different skeletonization techniques to extract double forks. In particular, we want to try approaches allowing to extract a skeleton from an imperfect point cloud. We are also planning to investigate further classes of features suitable for caging 3D objects.

REFERENCES

[1] INRIA gamma research group repository. https://www.rocq.inria.fr/gamma/gamma/download/download.php. Accessed: 2015-11-09.

[2] A. S. Besicovitch. A net to hold a sphere. *The Mathematical Gazette*, pages 106–107, 1957.

[3] A. Bicchi and V. Kumar. Robotic grasping and contact: A review. In *ICRA*, pages 348–353. Citeseer, 2000.

[4] Jeannette Bohg, Aythami Morales, Tamim Asfour, and Danica Kragic. Data-driven grasp synthesisa survey. *Robotics, IEEE Transactions on*, 30(2):289–309, 2014.

[5] X. Chen, A. Golovinskiy, and T. Funkhouser. A benchmark for 3D mesh segmentation. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 28(3), August 2009.

[6] R. Deimel and O. Brock. A novel type of compliant, underactuated robotic hand for dexterous grasping. In *Proceedings of Robotics: Science and Systems*, Berkeley, USA, July 2014.

[7] T. K. Dey and J. Sun. Defining and computing curve-skeletons with medial geodesic function. In *Symposium on Geometry Processing*, pages 143–152, 2006.

[8] R. Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010.

[9] R. Diankov, S. S. Srinivasa, D. Ferguson, and J. Kuffner. Manipulation planning with caging grasps. In *Humanoid Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on*, pages 285–292. IEEE, 2008.

[10] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *Discrete and Computational Geometry*, 28(4):511–533, 2002.

[11] X. Feng and Y. Tong. Choking loops on surfaces. *IEEE Trans. Vis. Comput. Graph.*, 19(8):1298–1306, 2013.

[12] K. Gopalakrishnan and K. Goldberg. D-space and deform closure grasps of deformable parts. *The International Journal of Robotics Research*, 24(11):899–910, 2005.

[13] A. Hatcher. *Algebraic topology*. Cambridge University Press, Cambridge, 2002.

[14] K. Klenin and J. Langowski. Computation of writhe in modeling of supercoiled DNA. *Biopolymers*, 54:307–317, 2000.

[15] W. Kuperberg. Problems on polytopes and convex sets. In *DIMACS Workshop on polytopes*, pages 584–589, 1990.

[16] Jeffrey Mahler, Florian T Pokorny, Brian Hou, Melrose Roderick, Michael Laskey, Mathieu Aubry, Kai Kohlhoff, Torsten Kroeger, James Kuffner, and Ken Goldberg. Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards. 2016.

[17] S. Makita and Y. Maeda. 3d multifingered caging: Basic formulation and planning. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 2697–2702. IEEE, 2008.

[18] S. Makita, K. Okita, and Y. Maeda. 3d two-fingered caging for two types of objects: sufficient conditions and planning. *International Journal of Mechatronics and Automation*, 3(4):263–277, 2013.

[19] R. M. Murray, Z. Li, and S. Sastry. *A mathematical introduction to robotic manipulation*. CRC press, 1994.

[20] P. Pipattanasomporn and A. Sudsang. Two-finger caging of concave polygon. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 2137–2142. IEEE, 2006.

[21] P. Pipattanasomporn and A. Sudsang. Two-finger caging of nonconvex polytopes. *Robotics, IEEE Transactions on*, 27(2):324–333, 2011.

[22] F. T. Pokorny, J. A. Stork, and D. Kragic. Grasping objects with holes: A topological approach. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 1100–1107. IEEE, 2013.

[23] E. Rimon and A. Blake. Caging planar bodies by one-parameter two-fingered gripping systems. *The International Journal of Robotics Research*, 18(3):299–318, 1999.

[24] A. Rodriguez, M. T. Mason, and S. Ferry. From caging to grasping. *The International Journal of Robotics Research*, pages 886–900, 2012.

[25] D. Rolfsen. *Knots and links*, volume 346. American Mathematical Soc., 1976.

[26] Ashutosh Saxena, Justin Driemeyer, and Andrew Y Ng. Robotic grasping of novel objects using vision. *The International Journal of Robotics Research*, 27(2):157–173, 2008.

[27] P. Shilane, P. Min, M. Kazhdan, and T. Funkhouser. The princeton shape benchmark. In *Shape modeling applications, 2004. Proceedings*, pages 167–178. IEEE, 2004.

[28] Hang Si. Constrained delaunay tetrahedral mesh generation and refinement. *Finite elements in Analysis and Design*, 46(1):33–46, 2010.

[29] J. A. Stork, F. T. Pokorny, and D. Kragic. Integrated motion and clasp planning with virtual linking. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Tokyo, Japan, 2013.

[30] J. A. Stork, F. T. Pokorny, and D. Kragic. A topology-based object representation for clasping, latching and hooking. In *IEEE-RAS International Conference on Humanoid Robots (HUMANOIDS13), Atlanta, USA*, 2013.

[31] M. Vahedi and A. F. van der Stappen. Caging polygons with two and three fingers. *The International Journal of Robotics Research*, 27(11-12):1308–1324, 2008.

APPENDIX

Here we present the proof of the supporting claim used in Section V.

Lemma 1. *Caging under rigid transformations (Def. 10) is an example of caging under D-deformations (Def. 11), where D-deformations is a class of isotopies representing rigid transformations.*

Proof. Any path $g : I \to SE(3)$ such that $g(t).int(O) \cap M = \emptyset$ for all $t \in I$ starting at the identity element $e = g(0)$ corresponds exactly to an isotopy $H : O \times I \to \mathbb{R}^3$ defined by $H(x,t) = g(t).x$ and preserving distances between points: $\|x - x'\| = \|H(x,t) - H(x',t)\|$ for all $x, x' \in O$ and $t \in I$. Furthermore, $H(x,0) = e.x = x = \iota(x)$ for all $x \in O$. Let us denote the class of isotopies generated by such paths $g : I \to SE(3)$ by D. Since $SE(3) = \mathbb{R}^3 \times SO(3)$, and $SO(3)$ is compact, the identity element $e \in C_O = \{g \in SE(3) : M \cap g(int(O)) = \emptyset\}$ lies in a bounded path-component of C_O if and only if the distance between the initial and any other reachable position of the object is bounded: $\mathrm{dist}(O, g(t).O) \leq C$ for all $t \in I$, $g : I \to SE(3)$ and for some fixed $C \geq 0$. Since M does not change its position in the space, the latter implies that for any reachable position of the object its distance to M is bounded as well: $\mathrm{dist}(M, g(t).O) \leq C'$ for all $t \in I$ and for some real constant $C' \geq 0$.

The latter implies that for any D-isotopy H and for each $t \in I$ we have

$$\inf_{x_1 \in M, x_2 \in H(O,t)} \|x_1 - x_2\| \leq C'$$

for some real constant $C' \geq 0$. □

Anastasiia Varava received a B.Sc. degree in Computer Science from Karazin Kharkiv National University, Ukraine, in 2013 and a M.Sc. degree in Computer Science from the University of Nice Sophia Antipolis, France, in 2014. She is a Ph.D. student at the Centre for Autonomous Systems and the Computer Vision and Active Perception Lab, KTH Royal Institute of Technology, Stockholm, Sweden. She is interested in mathematical representations for robotic manipulation.

Danica Kragic (SM'12) received the M.Sc. degree in Mechanical Engineering from TU Rijeka, Rijeka, Croatia, in 1995 and the Ph.D. in Computer Science from the KTH Royal Institute of Technology, Stockholm, Sweden, in 2001. She is a Professor with the School of Computer Science and Communication, KTH Royal Institute of Technology. Her research interests include computer vision and robotics. Prof. Kragic is a Member of the Swedish Royal Academy of Sciences and Swedish Young Academy. She received the 2007 IEEE Robotics and Automation Society Early Academic Career Award.

Florian T. Pokorny received his B.Sc. degree in Mathematics from the University of Edinburgh and a Master of Advanced Study (Part III) from the University of Cambridge. After that, he completed his Ph.D. in Mathematics at the University of Edinburgh. He is a Postdoctoral Researcher at the AMPLab and the Berkeley Automation Science Lab at the University of California, Berkeley. Before that, he was a Postdoctoral Researcher at the Centre for Autonomous Systems and the Computer Vision and Active Perception Lab, KTH Royal Institute of Technology. He is interested in robotic manipulation, machine learning and topological data analysis.