Dual-Arm In-Hand Manipulation and Regrasping Using Dexterous Manipulation Graphs
Silvia Cruciani, Kaiyu Hang, Christian Smith and Danica Kragic

Abstract—This work focuses on the problem of in-hand manipulation and regrasping of objects with parallel grippers. We propose Dexterous Manipulation Graph (DMG) as a representation on which we define planning for in-hand manipulation and regrasping. The DMG is a disconnected undirected graph that represents the possible motions of a finger along the object's surface. We formulate the in-hand manipulation and regrasping problem as a graph search problem from the initial to the final configuration. The resulting plan is a sequence of coordinated in-hand pushing and regrasping movements. We propose a dual-arm system for the execution of the sequence where both hands are used interchangeably. We demonstrate our approach on an ABB Yumi robot tasked with different grasp reconfigurations.

Index Terms—dexterous manipulation, grasping, manipulation planning, dual arm manipulation.

I. INTRODUCTION

Motions such as sliding, rotating and regrasping the object are natural to humans and are enabled by the dexterous structure of the human hand. For robots, it is still challenging to execute such manipulations that go beyond simple pick-and-place. One reason is that artificial hands that resemble human dexterity are not yet widespread in robotics due to inherent difficulties in design and control. Still, the most common end-effector of a robot arm is a simple parallel gripper. The problem of correctly grasping an object is of fundamental importance for object and tool use, both in terms of industrial and service robots applications.

In this work, we focus on achieving in-hand manipulation and regrasping of rigid objects using parallel grippers. The main challenge in manipulating objects with a parallel gripper lies in the lack of degrees of freedom (DOF), as these grippers can only open and close. However, parallel grippers are not just a tool for grasping, but they can be used for object pushing, in line with the concept of extrinsic dexterity. With extrinsic dexterity, the lack of DOF can be compensated using external supports, such as contact surfaces, gravity and inertial forces.

In our work, we exploit a dual-arm system, where the inability of the gripper is compensated by the two arms. This is naturally built upon the fact that dual-arm robots are becoming more common (e.g. Rethink Robotics Baxter, ABB Yumi) and we believe that their exploitation will help overcome various challenges that relate to advanced object interaction.

Our particular focus in this paper is on in-hand object manipulation: how to move an object from an initial grasp configuration to a desired one. We formulate the in-hand manipulation and regrasping problem as a planning problem from the initial to the final configuration, using Dexterous Manipulation Graphs (DMG). Our system consists of three main steps:

1) DMG generation, described in section IV
2) Planning given DMG, described in sections V and VI
3) Plan execution, described in section VII
These steps are shown in Fig. 1. The starting point is the DMG generation from the object’s shape. The DMG represents the possible motions of a finger on the object’s surface. Therefore, it depends on the kind of motions considered, which we define as in-hand manipulation primitives. By exploiting the DMG, we plan solutions that move the grasp from an initial configuration to the desired one. This solution can include both in-hand manipulation and regrasping. Finally, the sequence of motions required to execute the planned solution is executed by a dual-arm system.

This work is an extension of [1]. The improvements are:

- A more detailed analysis of the DMG and an in-depth explanation of the DMG search, which was not extensively detailed in our previous work.
- The addition of a regrasp action, which enables many more complex manipulation tasks. This regrasp action connects different components of the Dexterous Manipulation Graph, providing solutions in cases in which solely in-hand manipulation would fail, and it is the main improvement with respect to our previous work.
- New experiments with a different robot platform, which show the capabilities of the DMG for dexterous manipulation planning by exploiting both in-hand manipulation and regrasping.

This paper is organized as follows: in section II, we summarize the related work in the field of dexterous manipulation, with special focus on extrinsic dexterity; we also analyze the use of dual-arm robots for object manipulation. Section III provides an overview of our system and it defines the Dexterous Manipulation Graph and the manipulation primitives that are used for generating it. The process of obtaining the DMG for a certain object’s shape is described in section IV. Section V shows how to exploit the DMG for planning in-hand manipulation and section VI extends this plan to include the possibility of regrasping the object. The motion of the dual-arm robot is described in section VII. Examples of DMG and experiments with the robot are shown in section VII. Finally, we draw our conclusions and discuss future work in section IX.

II. RELATED WORK

This work spans over three important areas: dexterous manipulation, extrinsic dexterity and dual-arm manipulation.

A. Dexterous Manipulation

In many applications, the robot is required to execute a specific grasp on an object, e.g. if the object is to be used as a tool, or when placing an object in a confined space. Despite the abundance of work in grasping [2], achieving a desired grasp on an object is not straightforward: errors in perception and in actuation, as well as limited reachability, lead the execution of a planned grasp to result in a different configuration. To address these, it is possible to place the object back on the supporting surface and apply a new grasp again until the grasp coincides with the desired one [3]. However, this process requires a sequence of many pick-and-place operations for the purpose of grasping an object, resulting in an inefficient solution. Alternatively, fast regrasping can be executed by tossing the object in the air and adjust the pose of the hand to catch it in the desired configuration [4], but this solution is often infeasible due to the requirements for high-speed object tracking and hand actuation.

Dexterous In-Hand Manipulation can be seen as an alternative to regrasping. Instead of releasing the object and grasping it again, the grasp is adjusted by executing different hand motions that lead to the desired configuration. The ability of the hand to translate or rotate the object is dependent on the dexterity of the hand. An excellent example of a dexterous hand is the human hand. It remains extremely challenging to build an artificial hand that can mimic its skills [5], [6], both from mechanical design and control perspectives [7]–[9]. As an alternative to build complex dexterous hands, some works focus on building custom grippers for specific tasks, lowering the dexterity of the system to the movement needed for the target problem [10]–[14].

In addition to the aforementioned challenges, there are also challenges that arise when the end-effectors are used for planning and executing in-hand manipulation. The latter challenges have been addressed in various ways.

The Hierarchical Fingertip Space approach [15] (HFTS) provides a system for grasping the object and successively adjusting the pose of the fingers to increase the stability of the grasp. This work subdivides the object into small areas to find suitable contacts for the fingers, a concept similar to what we use for generating the Dexterous Manipulation Graph. Differently from our work, the main focus of HFTS is to achieve a stable grasp, and not dexterous in-hand manipulation.

The authors in [16] focus on in-grasp manipulation. Their method, called “relaxed rigidity constraint” is designed to adjust the pose of the object inside the hand. However, it can achieve only small changes in the object’s pose, and the contact points are not slid along the surface, but kept stable. Similarly, in [17] the grasp configuration is changed by moving the fingers’ joints and not by changing the contact points on the object. In addition to changing the fingers’ joint configurations, the dexterous manipulation planner in [18] proposes a roadmap to plan regrasping as well: one finger can release the contact and move to a new one, while the other fingers keep the object stable. Thus, this planner also places an important assumption, which is that the fingertips cannot slide or rotate on the object’s surface.

Recent progress in Reinforcement Learning demonstrated the possibility of applying it to the complex task of object manipulation with a multi-fingered hand. However, despite successfully exploiting a system simulator to learn policies that can be used on a real system, these policies cannot yet be well generalized for different objects [19].

B. Extrinsic Dexterity

In contrast to the works presented in the previous section, several studies address the problem of achieving in-hand manipulation under limited dexterity. This includes our work, which focuses on in-hand manipulation with parallel grippers. With end-effectors that are poor in DOF, the in-
hand manipulation task can be achieved by exploiting extrinsic dexterity [20].

Limited dexterity often results in limited in-hand repositioning capabilities, even when the dexterity is enhanced by means of external supports. Therefore, many works focus on simple in-hand manipulations, that still allow for rearrangements of the object pose inside the gripper. An example of this simple manipulation is pivoting. Pivoting is the rotation of the object between the two fingertips, and it admits reorientation along a single axis. This in-hand rotation can be initiated by exploiting gravity or inertial forces [21–23]. An example of gravity based pivoting is the work in [24], which adopts an adaptive control solution to compensate for the modeling uncertainties of the friction at the pivoting point. The work in [25] exploits inertial forces generated by the robot arm for rotating the object, and it uses a Reinforcement Learning approach with noise in the training data to compensate for the friction uncertainty. Pivoting is an in-hand manipulation strategy that only allows for limited repositioning. However, it can be combined with other strategies, such as sliding [26], [27], and provide a more complete range of possibilities. In fact, in our solution, we propose to combine rotational and translational motion, which are treated separately, to achieve the in-hand manipulation goal.

An additional support that enhances the dexterity, other than gravity, friction and inertial forces, is the use of external contacts with other surfaces [28–30]. The presence of external contacts helps in manipulating the object and allows the robot to reorient it and obtain the desired pose. The work in [31], and extended in [32], provides a plan for in-hand manipulation as a sequence of pushes against external fixtures. The plan is based on RRT* and it needs an accurate inverse dynamic solver to predict how the object will slide and rotate within the gripper.

Our approach provides an efficient graph structure, which integrates object representation and manipulation planning. Instead of relying on external fixtures, we propose to exploit the redundancy of a dual-arm system. Therefore, we do not rely on environment models or on properly placed external fixtures, and the dual-arm system provides additional control over how an object can be pushed inside the gripper. Moreover, it allows the robot to overcome limitations of in-hand manipulation by exploiting regrasping, without the need for planning object placements.

C. Dual Arm Manipulation

Dual arm systems can be used to manipulate the object using two grippers at the same time. While there are several studies in coordinating dual arm systems and exploiting them for grasping and regrasping [33], these systems are not commonly used in the context of in-hand manipulation.

Often, dual-arm manipulation is used for large object handling, and it takes advantage of whole body contacts. For instance, the dual-arm manipulation method proposed in [34] is formulated using a transition graph that considers possible contacts with the whole body of a humanoid robot. This transition graph is generated by testing the outcome of basic operations, such as lifting or sliding, and a dynamic model of object and environment is required. The bimanual grasp planner in [35] is also used for handling large objects and not for fine in-hand manipulation. In [36] the authors apply a dual-arm system to the manipulation of articulated objects. This application can be seen as an instance of dexterous manipulation, but the contacts between the gripper and the object are always kept fixed, while we focus on adjusting the grasp configuration, which includes changing the position of the contact points.

An example in which a dual arm system is used to grasp an object multiple times is presented in [37]. However, the goal is to perform tactile exploration to identify the object and later grasp it, and the robot’s exploration is passive: a human moves the robot’s hand on the object’s surface. The IK switch move proposed in [38] is designed to manipulate objects by performing multiple grasps. However, the possibility of sliding or rotating the object while maintaining contact is not taken into account, and the use of regrasping is meant to ease the planning of the two robot manipulators. Similarly, the work in [39], which focuses on humanoid systems with multi-fingered hands, is used to select between different grasping and regrasping poses that have been predefined, and there is no in-hand manipulation possibility between different grasps.

The planning proposed in [40] addresses the problem of two-hand regrasping, in which one hand grasps an object from the other hand. We exploit this concept to enhance the manipulability of the object when the sole in-hand manipulation is not enough. However, in this work the authors use multi-fingered hands and they assume that once the object is grasped or handed over its pose inside the hand cannot change. In contrast, we exploit the DMG to achieve more regrasping possibilities for the dual-arm system.

The preparatory manipulation planner described in [41] exploits a regrasp graph to plan a solution to regrasp an object in the proper configuration. This graph contains grasp configurations and object poses, and it is different from our Dexterous Manipulation Graph, which instead carries the information about fingers moving along the object’s surface. This regrasp graph is used to plan a sequence of pick-and-place and handovers for regrasping to allow the robot to grasp the object as desired. However, the sequence of regrasps on the object does not take into account the possibility of in-hand manipulation, and all the grasp poses must be pre-computed off-line to build the graph.

While it is not always true that using two arms for regrasping is better than a single arm placing the object and picking it up again, dual-arm regrasping is more flexible and it provides a higher success rate; however, single arm regrasp performs better when the grasp and the regrasp pose overlap [42]. In our case, we overcome this complication by exploiting in-hand manipulation.

To achieve in-hand manipulation with dual arm, we assume that the robot is holding the object with one gripper and the other one is used as additional support to enhance the dexterity. We use the Extended Cooperative Task Space (ECTS) [43] to control the motion of the two arms. The ECTS task specification [44] explores different manipulation scenarios, but it does not address the problem of in-hand manipulation.
We obtain the desired relative motion between the grippers given the desired in-hand motion of the object, and the robot’s arms are moved accordingly.

III. DEFINITIONS

In this section, we define the Dexterous Manipulation Graph and the manipulation types that we use to plan and execute grasp reconfigurations.

A. DMG definition

The Dexterous Manipulation Graph (DMG) is a structure proper to each object, and it depends on the object’s shape and partially on the length of the considered gripper’s finger. It is a disconnected undirected graph that represents the possible motions of a finger along the object’s surface. This graph is defined by a set of nodes $N$ and a set of edges $E$ between these nodes.

1) Graph Nodes: A node $n_{ij} \in N$ represents a configuration of the finger on the object’s surface, containing information on both position and orientation. More specifically, each node is a tuple $n_{ij} = (p_i, A_{ij})$; $p_i$ is the position of the contact point between the fingertip and the object, and $A_{ij}$, named angular component, is a continuous set of orientations along which the finger in contact at $p_i$ can rotate without collisions. As shown in the example in Fig. 2, more nodes in the graph can correspond to the same contact point, depending on the shape of the object. In the example, the point $p_i$ corresponds to two nodes $n_{ij}$ and $n_{ih}$, because when the finger is in contact at this point it can rotate continuously in the set of angles $A_{ij}$ or in the set of angles $A_{ih}$.

2) Graph Edges: An edge $e_{n_{ij}n_{gh}} \in E$ connects two nodes $n_{ij}$ and $n_{gh}$ if it is possible to move the finger along the object’s surface between the two configurations. The possible motions taken into account, called in-hand manipulation primitives, are described in section III-B. In the example in Fig. 2 the two nodes $n_{ij}$ and $n_{ih}$ are not connected by an edge because it is not possible to directly move the finger between the two corresponding configurations. In fact, a finger in contact at the point $p_i$ and with an angle $\phi_{ij} \in A_{ij}$ cannot move to an angle $\phi_{ih} \in A_{ih}$ while maintaining the same fingertip contact point, due to collisions with the object.

3) Graph Components: The DMG is a disconnected graph, i.e. it has several connected components that are disconnected from each other. This is due to the nature of the graph, which describes the motions of a gripper’s finger on the object’s surface: it is not possible for the finger to freely move from any point of the object’s surface to another; for instance, a finger cannot move across sharp edges without releasing the contact. Each connected component $C_i \subseteq N$ contains all the nodes between which a finger can move using the in-hand manipulation primitives. Therefore, it is not possible to move the finger between two nodes $n_{ij}$ and $n_{gh}$ if $n_{ij} \in C_i$ and $n_{gh} \in C_g$, with $C_i \neq C_g$, without releasing the contact with the object’s surface. We exploit the different connected components for planning regrasps when an in-hand motion is not possible.

To build the DMG, the object’s shape is analyzed according to how the gripper’s fingers can move along its surface. In the following, we introduce the manipulation primitives that are considered to describe the fingers’ motion. Furthermore, we take into account the possibility of releasing the contact and regrasping in case the desired configuration is not achievable through solely in-hand motions.

B. In-Hand Manipulation Primitives

The in-hand manipulation primitives are motions that describe how a single gripper’s finger moves with respect to the object. These motions lead the finger on the object towards a different configuration.

The configuration of the finger on the object is a tuple $c=(p, \phi)$, in which $p$ is the position of the contact point between the fingertip and the object’s surface, and $\phi$ is the current angle of the finger with respect to the object. It is important to notice that a finger configuration is different from a graph node; in fact, a graph node describes a set of configurations, because it includes the set of possible angles that can be assumed at a certain contact point.

For the DMG generation given the object’s shape, we consider the following motions:

1) Translation: (Fig. 3a). The contact point between the fingertip and the object slides along the object’s surface, while the orientation of the finger with respect to the object does not change. More specifically, a translation moves the finger from a configuration $c=(p, \phi)$ to a new configuration $c’=(p, \phi)$. 

2) Rotation: (Fig. 3b). The contact point between the fingertip and the object is kept fixed, while the finger’s orientation with respect to the object changes. More specifically, a rotation moves the finger from a configuration $c=(p, \phi)$ to a new configuration $c’=(p, \phi’)$.

A combination of these two motions at the same time, a rototranslation (Fig. 3c), is not considered in our current DMG generation because in several cases it requires prehensile
into the component of the DMG that contains the desired grasp. That is, the regrasping action has the role of creating bridges between the disconnected sub-graphs that compose the DMG.

IV. GRAPH GENERATION

This section describes the process of obtaining the DMG starting from the object’s shape. It corresponds to step 1 in Fig. 1.

A. Overview

The main idea for obtaining the DMG is to divide the object’s surface into small areas, corresponding to contact points with the fingertip, and analyze how the gripper’s finger can move between them. The size of the area depends on the chosen approximation of the fingertip contact with respect to the size of the object.

The connection between an area and the neighboring areas is refined based on the possibility for a fingertip to move from one to the other using the in-hand manipulation primitives defined in section III-B. To each area, one or more nodes in the DMG are associated; these nodes contain information about the fingertip contact, identified by the area’s centroid, and the possible range of orientations (angular components) that the finger can assume at that contact point. The detailed process of DMG generation is described in the next session and summarized in Algorithm 1.

B. From Object’s surface to DMG

We represent the object as a point cloud and we subdivide its surface into smaller areas of approximate resolution \( r_{area} \) using the method of supervoxels [45]. Each area is segmented out from the rest of the object given the desired resolution and according to common characteristics, such as geometric information.

An area \( a \), called supervoxel, is associated to its centroid \( p_a \) and its normal \( n_a \). The geometric proximity of an area with its neighbor areas is contained in a graph. This graph is described by a set of nodes \( N_a \) and a set of edges \( E_a \). Each node corresponds to a supervoxel and an edge connects a supervoxel to its neighbors.

The connected graph describing the object’s surface subdivided into areas is the starting point for the generation of the disconnected DMG. An initial set of nodes \( N \) and an initial set of edges \( E \) replicates the structure of the supervoxels graph. However, the new set of nodes differs from \( N_a \) because a node is not representing an area, but it has the same components of a DMG node.

A node \( n_a \in N \), corresponding to the supervoxel \( a \), is a tuple \( n = (p_a, A_{360}) \). \( A_{360} \) is a discrete set that contains all the possible orientations of the finger in the range between \( 0^\circ \) and \( 360^\circ \). We choose to represent the possible orientation with resolution \( r_{angle} \in \mathbb{R} \), so that the set \( A_{360} \) contains all the \( K_{angle} \) multiples of \( r_{angle} \) within \([0, 360)\). Assuming the elements of the set organized in increasing order, the first element is \( \phi_0 = 0 \) and the last one is \( \phi_{K_{angle}-1} = K_{angle} r_{angle} \). Therefore, at this step, each node in the graph corresponds to...
Algorithm 1: DMG generation

Input : \( P, l_f, \delta_n, r_{\text{angle}} \)
// \( P \) is the object represented as point cloud /*
1 \( N, E, \delta_n \leftarrow \) supervoxels of \( P \) and their adjacency
2 \( N \leftarrow \emptyset \) // empty set of nodes
3 \( E \leftarrow \emptyset \) // empty set of edges
4 \( A_{360} \leftarrow \{0 : r_{\text{angle}} : 360\} \) // set of discrete angles
5 foreach \( e_{a_i,a_j} \in E \) do
6 \( p_{a_i} \leftarrow \) centroid of the area \( a_i \)
7 \( p_{a_j} \leftarrow \) centroid of the area \( a_j \)
8 \( n_{a_i} \leftarrow (p_{a_i}, A_{360}) \)
9 \( n_{a_j} \leftarrow (p_{a_j}, A_{360}) \)
10 \( N \cup \{n_{a_i}, n_{a_j}\} \)
11 \( E \cup \{e_{n_{a_i},n_{a_j}}\} \)
12 foreach \( n_{a_i} \in N \) do
13 \( E \leftarrow \) translation_refinement(\( n_{a_i}, N, E, \delta_n \))
14 foreach \( n_{a_j} \in N \) do
15 \( A \leftarrow \) angular_component(\( P, l_f, n_{a_j} \))
16 if \( A \) is empty then
17 foreach \( e_{nm} \in E \) s.t. \( n \) is \( n_{a_0} \) or \( m \) is \( n_{a_0} \) do
18 \( E \leftarrow E \setminus \{e_{nm}\} \)
19 \( N \leftarrow N \setminus \{n_{a_j}\} \) // no contact is possible
20 else
21 \( N, E \leftarrow \) split_node(\( n_{a_i}, A, N, E, r_{\text{angle}} \))
22 foreach \( n_{a_j} \in N \) do
23 \( E \leftarrow \) rotation_refinement(\( n_{a_j}, N, E \))
24 DMG \( \leftarrow \) \( (N, E) \)
25 return DMG

Algorithm 2: translation_refinement

Input : \( n_{a_i}, N, E, \delta_n \)
// \( a_i \) is the supervoxel corresponding to node \( n_{a_i} \) /*
1 \( \hat{n}_{a_i} \leftarrow \) normal to the surface in \( a_i \)
2 foreach \( n_{a_j} \in N \) connected to \( n_{a_i} \) do
3 \( \hat{n}_{a_j} \leftarrow \) normal to the surface in \( a_j \)
4 if \( \|\hat{n}_{a_i} - \hat{n}_{a_j}\|_2 > \delta_n \) then
5 \( E \leftarrow E \setminus \{e_{n_{a_i},n_{a_j}}\} \)
6 return \( E \)

Algorithm 3: angular_component

Input : \( P, l_f, n_a \)
1 \( p_a \leftarrow \) centroid of \( n_a \)
2 \( A_{360} \leftarrow \) set of angles of \( n_a \)
3 \( A \leftarrow \emptyset \) // empty set of nodes
4 \( \phi \leftarrow 0 \)
5 foreach \( \phi \in A_{360} \) do
6 if finger of length \( l_f \) in contact at \( p_a \) with angle \( \phi \) not in collision with object \( P \) then
7 \( A \leftarrow A \cup \{\phi\} \)
8 return \( A \)

Algorithm 4: split_node

Input : \( n_{a_i}, A, N, E, r_{\text{angle}} \)
1 \( K_{\text{angle}} \leftarrow \lfloor r_{\text{angle}} \rfloor \)
2 \( \phi_{\text{max}} \leftarrow K_{\text{angle}} \text{ angle} \)
3 \( k \leftarrow 0 \)
4 \( A_0 \leftarrow \emptyset \) // empty set of angles
5 \( A_A \leftarrow \emptyset \) // empty set of sets
6 while \( k < K_{\text{angle}} - 1 \) do
7 // the set \( A \) is ordered
8 \( \phi_k \leftarrow k\text{-th element of } A \)
9 \( A_k \leftarrow A_k \cup \{\phi_k\} \)
10 \( \phi_{k+1} \leftarrow k+1\text{-th element of } A \)
11 if \( \phi_{k+1} - \phi_k > r_{\text{angle}} \) then
12 \( A_A \leftarrow A_k \)
13 \( A_k \leftarrow \emptyset \)
14 \( k \leftarrow k+1 \)
15 if \( \phi_{\text{max}} \in A_A \) then
16 \( A_0 \leftarrow \text{first set in } A_A \)
17 if \( 0 \in A_0 \) then
18 \( A_A \leftarrow A_A \setminus \{A_0\} \)
19 \( A_0 \leftarrow A_0 \cup A_A \)
20 else
21 \( A_A \leftarrow A_A \cup \{A_0\} \)
22 foreach \( A_i \in A_A \) do
23 \( n_{ai} \leftarrow (p_{a_i}, A_i) \)
24 \( N \leftarrow N \cup \{n_{ai}\} \)
25 foreach \( e_{nm} \in E \) s.t. \( n \) is \( n_{a_i} \) or \( m \) is \( n_{a_i} \) do
26 \( E \leftarrow E \setminus \{e_{nm}\} \)
27 \( e_{nm} \leftarrow E \cup \{e_{nm}\} \)
28 foreach \( e_{nm} \in E \) s.t. \( n \) is \( n_{a_i} \) or \( m \) is \( n_{a_i} \) do
29 \( E \leftarrow E \setminus \{e_{nm}\} \)
30 \( E \leftarrow E \cup \{e_{nm}\} \)
31 \( N \leftarrow N \setminus \{n_{a_i}\} \)
32 return \( N, E \)
yz plane with a common reference between all the connected nodes of the component.

The following step, summarized in Algorithm 5, consists in identifying the admissible rotations for the finger. For each node \( n_a \), the finger in contact at the point \( p_a \) is checked for possible collisions with the object. More specifically, the finger of given length \( l_f \) is rotated at each angle \( \phi \in A_{360} \) and the angle is removed from the set if it is in collision. The angle set associated with \( n_a \) now contains only the collision-free angles that the finger can assume when its fingertip is in contact with the object in \( p_a \). If this set is empty, then this contact cannot be achieved and the node \( n_a \) is removed from the graph along with all its edges.

Given an angle \( \phi_k \) and the subsequent angle in the set \( \phi_{k+1} \), the finger cannot rotate between the two if

\[
|\phi_{k+1} - \phi_k| > r_{\text{angle}}.
\]

The only exception is between the angles \( \phi_0 \) and \( \phi_{\text{K}_{\text{angle}}-1} \), because it is possible to rotate between the two even though (2) is verified.

If there is at least one rotation gap, that is there are two angles in the set between which the finger cannot rotate, this angle set is split into two of more sets \( A_{s1}, A_{s2}, \ldots, A_{sK_{set}} \). Each of these sets contains a subset of the admissible angles for a finger in contact at \( p_a \), between which it is possible to rotate without incurring in collisions.

When \( K_{\text{set}} > 1 \), the area \( a \) of the object must correspond to more than one node in the DMG. Therefore, the node \( n_a \) is removed from the graph and it is split into different nodes \( n_{a1} = (p_a, A_{s1}), n_{a2} = (p_a, A_{s2}), \ldots, n_{aK_{set}} = (p_a, A_{sK_{set}}) \), each one containing a different set of sequential angles. To each of these nodes we introduce new edges that connect it to the same nodes to which \( n_a \) was connected. However, there is no edge that connects these new nodes to each other. This process of splitting one node into multiple nodes according to the angular components is summarized in Algorithm 5.

Finally, the graph with the new nodes refined according to the rotation refinement summarized in Algorithm 5. For each edge \( e_{n_{a1}n_{b1}} \), the two angle sets \( A_{ij} \) and \( A_{gh} \) of the two connected nodes are analyzed. In fact, if \( A_{ij} \cap A_{gh} = \emptyset \), it is not possible for the finger in contact at \( p_i \) to move to the contact point of the other node \( p_j \) and vice-versa, because there is not a common admissible orientation. Therefore, the edge \( e_{n_{a1}n_{b1}} \) is removed from the graph.

After the translation and the rotation refinements, the obtained graph is the DMG. This graph describes how a finger can move from one node to the other according to the motions described in section III-B.

The obtained graph is disconnected. We can identify the subsets of nodes that belong to the same component, for instance using breadth-first search on all the nodes in \( N \) and identifying a new component whenever the search ends without having visited all the nodes. Therefore, we can associate to the DMG a set of components \( C = \{C_1, C_2, \ldots, C_{K_{\text{comp}}} \} \). These components describe the subset of nodes between which it is possible to move; in fact, it is not possible to move between nodes that belong to different components, as there is no edge that connects them.

V. IN-HAND MANIPULATION PLANNING

This section explains how to use the DMG to plan in-hand manipulation and move the object from an initial grasp to the desired one without releasing contact between fingertips and object. This procedure is the first part of step 2 in Fig. 1.

A. Grasp Configuration Description

The goal of object manipulation planning is to find the sequence of manipulation primitives that leads the object from an initial grasp configuration to the desired one. We consider in-hand manipulation with a parallel gripper. The grasp of the gripper on the object can be described by the contact point of the two fingertips on the object \( p_{f1} \) and \( p_{f2} \) and by the orientation of the fingers \( q_f \), described as a quaternion, which is the same for both due to symmetry.

The points \( p_{f1} \) and \( p_{f2} \) lie on the object’s surface; therefore, they belong to one of the supervoxels used to compute the DMG and they can easily be associated with the corresponding nodes that contain the centroid of the supervoxel. By using the reference frame associated to the connected component in the DMG, mentioned in section IV-B, the orientation of the finger in space is translated into an angle in the yz plane of this frame. Using this angle, the fingers are associated to the start configurations \( c_{s1} = (p_{s1}, \phi_{s1}) \) and \( c_{s2} = (p_{s2}, \phi_{s2}) \), where \( p_{s1} \) and \( p_{s2} \) are the centroids of the supervoxels containing the fingertips’ contact points. Similarly, the desired contact points for the first and second finger \( p_{d1} \) and \( p_{d2} \) and the desired orientation \( q'_{d} \) can be described using the configurations \( c_{d1} = (p_{d1}, \phi_{d1}) \) and \( c_{d2} = (p_{d2}, \phi_{d2}) \).

B. Search for In-Hand Path

Given the initial grasp, described by the fingers’ configurations \( c_{s1} \) and \( c_{s2} \), and the desired grasp, \( c_{d1} \) and \( c_{d2} \), the in-hand manipulation planning finds a sequence of in-hand manipulation primitives that moves the object inside the gripper from the initial to the desired configuration.

Each finger configuration is associated to a node in the graph, leading to two initial nodes, \( n_{s1} = (p_{s1}, A_{s1}) \) and \( n_{s2} = (p_{s2}, A_{s2}) \), and to two desired nodes, \( n_{d1} = (p_{d1}, A_{d1}) \) and \( n_{d2} = (p_{d2}, A_{d2}) \). The angular components of these nodes are so that \( \phi_{s1} \in A_{s1}, \phi_{s2} \in A_{s2}, \phi_{d1} \in A_{d1} \) and \( \phi_{d2} \in A_{d2} \). By \( \phi' \) we denote the closest angle to \( \phi \) that belongs to the discrete set of angles \( A_{360} \) with the chosen resolution \( r_{\text{angle}} \).

The in-hand manipulation planning is a search in the DMG for a path that connects \( n_{s2} \) with \( n_{d1} \) and \( n_{s1} \) with \( n_{d2} \).

### Algorithm 5: rotation_refinement

| Input | \( n_a \), \( N \), \( E \) |
|-------|------------------|
| 1. \( A_{a1} \leftarrow \) angular component of \( n_a \) |
| 2. foreach \( n_a \in N \) connected to \( n_a \) do |
| 3. \( A_{a2} \leftarrow \) angular component of \( n_{a2} \) |
| 4. if \( A_{a1} \cap A_{a2} \) is empty then |
| 5. \( E \leftarrow E \setminus \{e_{n_a,n_{a2}}\} \) |
| 6. return \( E \) |
With $C_{s_1}, C_{s_2}, C_d_1,$ and $C_d_2$ denoting the graph’s components containing respectively the two initial nodes and the two desired nodes, there is no in-hand path that can lead from the initial to the desired configuration if

$$C_{s_1} \neq C_d_1 \quad \text{or} \quad C_{s_2} \neq C_d_2.$$  

(3)

In this case, the desired grasp can only be achieved by releasing the contact. The process of releasing and regrasping the object is described in section VI. If the condition in (3) is not verified, an in-hand path is searched for in the DMG.

First of all, we select one finger as principal finger and the other finger as secondary finger. In fact, the DMG is built considering the motion of only one finger on the object, while we consider the problem of in-hand manipulation with a parallel gripper. Therefore, the search in the DMG must be carried out by following the motions on one finger.

Without loss of generality, we assume that 1 denotes the principal finger and 2 denotes the secondary finger. The graph search is carried out in the component of the principal finger $C_1$ (equivalent to $C_{s_1}$ and $C_{d_1}$) using Dijkstra’s algorithm [46]. The inclusion of angular components in the DMG’s nodes can result in different path’s on the object’s surface, even though the initial and final contact points are the same, when the desired orientation differs, as shown in the examples in Fig. 5.

During the search, the secondary finger follows the principal finger in the component $C_2$ (equivalent to $C_{s_2}$ and $C_{d_2}$) and it is used to ensure that the followed path is valid for both of the gripper’s fingers. As shown in Fig. 6, the choice of one finger rather than the other as principal finger does not affect the validity of the in-hand path found, and hence it can be assigned arbitrarily.

At each iteration of the DMG search, a transition between two connected principal nodes is checked, e.g. from $n_{j_1}$ to $n_{i_1}$. To a principal node $n_{i_1} \in C_1$ corresponds a secondary node $n_{i_2}$ that represents the secondary finger. This node is found as the node corresponding to the supervoxel that contains the intersection point between the object and the ray from $p_{i_1}$ in the direction opposite to the normal of the area $i_1$. Depending on the object’s shape, different situations can occur:

1) there is only one intersection point, corresponding to the node $n_{i_2}$.

2) there is only one intersection point, but it corresponds to multiple nodes $n_{i_20}, n_{i_21}, \ldots$ with different angular components. These nodes could also belong to different graph components.

3) there are multiple intersection points, corresponding to different nodes $n_{i_20}, n_{i_21}, \ldots$ that belong to different graph components $C_{i_20}, C_{i_21}, \ldots$.

In the condition of point 1, the node $n_{i_2}$ is a valid node for the secondary finger only if the finger can assume the same orientation of the principal finger in this node. The set of possible angles $A_{i_2}$ can be transformed into a new set $A'_{i_2}$ that correspond to the angles in $A_{i_2}$ in the reference frame of $C_1$. Therefore, the secondary node is valid if

$$A_{i_1} \cap A'_{i_2} \neq \emptyset.$$  

(4)

During the search, the node $n_{i_1}$ will be replaced by the node $n'_{i_1} = (p_{i_1}, A_{i_1} \cap A'_{i_2})$. In the condition of points 2 and 3 a node is only valid if it belongs to $C_2$ and it satisfies the condition in (4).

In addition, the difference between the normals in $n_{i_1}$ and $n_{i_2}$ can be checked to allow or not motions along curved or skewed surfaces, such as the example in Fig. 7.

If there are no valid nodes, a transition of the principal finger that moves from node $n_{j_1}$ to node $n_{i_1}$ is not allowed. Therefore, the cost of edge $e_{n_{j_1}n_{i_1}}$ is set to $+\infty$ to ensure that this path is not chosen. As a consequence, the path solution may not be the shortest path for the principal finger in its corresponding component, because it must consider the motion of the secondary finger as well. The right image in Fig. 6 shows an example: the finger does not proceed straight to the goal because the opposite finger cannot follow it along that path.

Fig. 5: An example of graph search from the same initial configuration (in red) to the same final contact point, but the desired orientation is different (in green) between the left and the right figure. The DGM’s nodes contain angular components, so that the path along the object’s surface can be followed by the finger at the given orientation. The object is the same of Fig. 2.

Fig. 6: An example of graph search from the initial grasp (red fingers) to the desired grasp (green fingers). The top and bottom images show the outcome for the same object with a different finger selected as principal finger. In both cases the solution found, shown as blue path, is valid, but it leads to two slightly different in-hand motions. The DMG is shown on the right, and each color corresponds to a different component.

Fig. 7: The fingers move from the initial grasp (red) to the desired one (green) by sliding on a prismatic object. This translation requires an adjustment of the fingers’ opening because the surfaces are not parallel. This kind of situation can be adjusted depending on the available system and preferences.
If the node is valid, the cost $\xi_{n_j,n_i}$ of an edge connecting two nodes $n_j$ and $n_i$ can be chosen in different ways. A first possible choice is

$$\xi_{n_j,n_i} = \|p_{j} - p_{i}\|_2, \quad (5)$$

that is the simple Euclidean distance between the two centroids of the supervoxels associated with the nodes. This distance minimizes the distance in terms of translations of the fingertips on the object’s surface.

As an alternative, the cost can be adapted and chosen according to how expensive a certain movement is considered to be. For instance, some translation directions could be more difficult to execute, either due to the object’s shape or, as we discuss later in section VII-D, due to the robot’s configuration. Therefore, these directions can be penalized by increasing the cost.

In addition, if moving from one node to another involves a rotation, the requirement of this rotation in addition to the translation can be considered as more expensive and the cost of the edge increased. While the association between translation and edge in the graph is straightforward, the rotation is not explicitly represented in the DMG. In fact, a node contains the Cartesian position of the centroid associated with the fingertip contact point, but it contains more than one angle to associate with the finger’s orientation.

A solution path in the DMG is a sequence of nodes, and not a sequence of finger configurations. For reasons of notation simplifications, we explain how to obtain the sequence of rotations from a sequence of nodes in section V-C. However, this procedure can be executed at the same time of the graph rotations from a sequence of nodes in section V-C. However, the simplifications, we explain how to obtain the sequence of rotations from a sequence of nodes in section V-C. However, this procedure can be executed at the same time of the graph rotations from a sequence of nodes in section V-C. However, the simplifications, we explain how to obtain the sequence of rotations from a sequence of nodes in section V-C. However, this procedure...
E. Sequence Simplification

This sequence is so that \( \gamma \) nation of rotation and translations \( K \) sequence of \( K \) sequence of in hand manipulation primitives is an alteration of \( K \) corresponds to a translation. Therefore, from a sequence \( K \) sequence of \( K \) sequence of \( K \) path \( n_1 \) translations. Given the two consecutive nodes \( n_{path_h} \) and \( n_{path_{k+1}} \), the corresponding translation is defined as the vector

\[ t_k = p_{k+1} - p_k. \]  

From the sequence of \( K+1 \) angles corresponding to the sequence of \( K \) nodes, as shown in section [42] we can obtain a sequence of \( K \) rotations. We assume that a rotation always precedes a translation during the motion execution, and that a final rotation is executed in the end. Given two consecutive angles \( \phi_k \) and \( \phi_{k+1} \), the corresponding rotation is the angle

\[ \gamma_k = \phi_{k+1} - \phi_k. \]  

The sequence of in hand manipulation primitives is an alternation of rotation and translations \( \gamma_0, t_0, \ldots, \gamma_{K-1}, t_{K-1}, \gamma_K \). This sequence is so that

\[ \mathbf{p}_{d_1} = \mathbf{p}_{s_1} + \sum_{k=0}^{K-1} t_k \quad \text{and} \quad \phi_{d_1} = \phi_{s_1} + \sum_{k=0}^{K} \gamma_k. \]  

E. Sequence Simplification

Since the DMG is obtained by subdividing the object’s shape into small areas, the in-hand manipulation sequence is composed of many segments that connect these areas. However, this sequence can often be simplified because several translations lie on the same line and many rotations are zero. By simplifying the in-hand manipulation sequence, the execution of the motions as described in section [7] becomes smoother.

The sequence of translations and rotations is simplified by merging translations that move the fingertip in the same direction when the rotation between them is zero.

VI. REGRASPING PLANNING

When there is no in-hand path that moves the grasp from the initial fingers’ configurations \( c_{s_1} \) and \( c_{s_2} \) to the desired fingers’ configurations \( c_{d_1} \) and \( c_{d_2} \), the object must be released and regrasped. Depending on the considered system, the regrasping action could also be initiated if the obtained in-hand path is too complex and would result in a much slower execution. This regrasping action completes step 2 in Fig. [1]

A. Regrasping Sequence Overview

We take advantage of a dual-arm system so that the object does not have to be properly placed an picked up again, because we use the second gripper as a support. We denote a configuration of a finger of the first gripper, the one initially holding the object, as \( c^1 \) and a configuration of a finger of the second gripper as \( c^2 \). Therefore, by first gripper we indicate the gripper for which the desired grasp is defined, and by second gripper we indicate a support gripper. In the following, we use the superscript to indicate the gripper, while the subscript contains information on the principal or secondary finger.

The goal of the regrasping planning is to:

- find release configurations \( c_{s_1}^1 \) and \( c_{s_2}^1 \);
- find grasp configurations \( c_{r_1}^1 \) and \( c_{r_2}^2 \);
- find release configurations \( c_{r_1}^2 \) and \( c_{r_2}^2 \);
- find grasp configurations \( c_{g_1}^1 \) and \( c_{g_2}^1 \).

A release configuration is the configuration of a finger immediately before the gripper opens, and a grasp configuration is the configuration of the finger immediately after the gripper closes on the object.

A release configuration is needed if the gripper cannot fully open at the current grasp pose due to the object’s shape, or if the necessary grasp afterwards is in collision with the current pose of the gripper. For instance, in the example in Fig. [46] the second gripper (blue) is shown after the grasp, but it could slide away from that pose into different release configurations so that the first gripper can directly regrasp in the desired pose. Nonetheless, most of the times for the second gripper the grasp and the release configurations will likely be the same.

The sequence in which the regrasp steps are planned is the following:

1) plan a regrasp for the first gripper towards the desired configuration;
2) find a possible grasp for the second gripper, and plan its release;
3) plan the release of the first gripper.

There are two grasps that need to be planned, one per gripper. The criteria with which the grasp is selected differ according
to which gripper it is. In fact, the first gripper is the one that must reach the desired configuration, while the second gripper has only a support function.

B. Antipodal Points

For grasp planning, we use antipodal point grasps to find a configuration suitable for the two fingers. Antipodal points are two points on the object’s surface whose normals are collinear and in opposite directions. Under the assumption of soft finger contact, these points guarantee force closure [47], [48]. Instead of looking for antipodal points on the whole object’s surface, and since common solutions pose assumptions on the object’s shape that the objects we consider do not fulfill [49], our search constrains the points to be on determined components of the DMG, and in particular we focus only on the centroids of the supervoxels.

Given a point on the object $p_{i_1}$, corresponding to the supervoxel $i_1$, as a candidate contact point for the first fingertip after the grasp, in case there are points intersecting the object along the ray from $p_{i_1}$ directed as the normal $\hat{n}_{i_1}$, this point is not considered suitable for a direct regrasp as the gripper cannot directly close on it. If it is a suitable candidate, we proceed into evaluating the contact of the second fingertip.

The object intersects the line directed opposite to the normal $\hat{n}_{i_1}$ in one or more points. Since the grasp is achieved by approaching the object with a fully open gripper and then closing the fingers, the selected intersection point is the most external one, i.e. the furthest away from $p_{i_1}$, denoted by $p_{i_2}$.

Assuming a maximum opening of the gripper equal to $d_{\text{max}}$, these two points are not valid for grasping if

$$\|p_{i_1} - p_{i_2}\|_2 > d_{\text{max}}.$$  \hfill (9)

Otherwise, we consider the two points good antipodal points if the direction of the normal $\hat{n}_{i_2}$ of the supervoxel that contains the intersection point $p_{i_2}$ is along the same line of $\hat{n}_{i_1}$ and with opposite direction. That is

$$\begin{cases} \|\hat{n}_{i_1} \times \hat{n}_{i_2}\| < \delta_c \\
\hat{n}_{i_1} \cdot \hat{n}_{i_2} < 0 \end{cases},$$  \hfill (10)

where $\delta_c$ is a tolerance threshold to allow for grasps also in surfaces that are not perfectly parallel. While (10) ensures only that the two normals are parallel and opposite, the collinearity is already ensured by the procedure to find $p_{i_2}$.

The choice of fingertip contact point $p_{i_1}$ depends on which grasp is being planned, as the grasp defined by the two antipodal contact is subject to different constraints for the first and for the second gripper. Similarly to the in-hand path solution, a regrasp can differ depending on which finger is chosen as principal finger.

C. First Gripper Regrasp

The regrasp planned for the first gripper takes into account the desired fingers configurations $c_{d_1}$ and $c_{d_2}$. The goal is to plan the two grasps configurations $c_1$ and $c_2$ so that it is easy for the gripper to reach the target pose. If it is possible to directly regrasp at the desired pose, then $c_{d_1} = c_1$ and $c_{d_2} = c_2$. Otherwise, the grasp configurations are found by looking for feasible antipodal contact points. These points are searched starting from all the points $p_{i_1}$ that correspond to the centroid of nodes in the DMG component $C_{d_1}$, $n_{d_1}$ is used as a starting point so that the selected nodes are as close to it as possible. By keeping the followed connections in memory, the subsequent search for in-hand solution can be optimized.

The antipodal point $p_{i_2}$ must belong to a supervoxel that corresponds to a node in the DMG component $C_{d_2}$. The angles $\phi_{i_1}$ and $\phi_{i_2}$ are chosen among the angles that keep the fingers in the desired DMG components so that the gripper’s orientation points towards the first gripper’s side of the robot.

Algorithm 7: DMG_search

```
Input: DMG, $P$, $c_{d_1}$, $c_{d_2}$, $c_{d_1}$, $c_{d_2}$
if $S^1 \neq \text{Null}$ then
  /* Only one in-hand path is needed, no regrasp */
  return Null, Null, $S^1$, Null, Null, Null
/* first gripper regrasp */
if can regrasp in $c_{d_1}$, $c_{d_2}$ then
  $c_1$, $c_2$ ← $c_{d_1}$, $c_{d_2}$
  $S^1$ ← Null
else
  /* This iteration starts from $n_{d_1}$ and proceeds backwards from the goal */
  foreach $p_{i_1}$ centroid of node $n_{i_1} \in C_{d_1}$ do
    $p_{i_2}$ ← centroid opposite to $p_{i_1}$
    if $p_{i_2}$ and $p_{i_1}$ are antipodal and can be grasped then
      $\phi_{i_1}$, $\phi_{i_2}$ ← closest angles to $\phi_{d_1}$
      $c_1$, $c_2$ ← $\langle p_{i_1}, \phi_{i_1} \rangle$, $\langle p_{i_2}, \phi_{i_2} \rangle$
      $S^1$ ← in_hand_search(DMG, $P$, $c_{d_1}$, $c_{d_2}$, $c_{d_1}$, $c_{d_2}$)
    break
  /* second gripper grasp and release */
  $c_1$, $c_2$ ← regrasp configuration maximizing $d_{\text{max}}$ in $S^1$
if $d_{\text{max}}$ is 0 then
  $c_1^2$, $c_2^2$ ← configurations far away from $c_{d_1}$, $c_{d_2}$ so that $c_1^2 \in C_{d_1}$ and $c_2^2 \in C_{d_2}$
  $S^2$ ← in_hand_search(DMG, $P$, $c_{d_1}$, $c_{d_2}$, $c_{d_1}$, $c_{d_2}$)
else
  $c_1^2$, $c_2^2$ ← configurations far away from $c_{d_1}$, $c_{d_2}$ so that $c_1^2 \in C_{d_1}$ and $c_2^2 \in C_{d_2}$
  $S^2$ ← in_hand_search(DMG, $P$, $c_{d_1}$, $c_{d_2}$, $c_{d_1}$, $c_{d_2}$)
/* first gripper release */
if $d_{\text{max}}$ is too small, or direct release not possible then
  $c_{r_1}$, $c_{r_2}$ ← regrasp configuration maximizing $d_{\text{max}}$ in $S^1$
if $c_{r_1} \in C_{d_1}$ and $c_{r_2} \in C_{d_2}$ then
  $S^0$ ← in_hand_search(DMG, $P$, $c_{d_1}$, $c_{d_2}$, $c_{r_1}$, $c_{r_2}$)
else
  $c_{r_1}$, $c_{r_2}$ ← $c_{d_1}$, $c_{d_2}$
  $S^0$ ← Null
return $S^0$, $S^2$, $S^1$, $[c_1^1]$, $[c_2^1]$, $[c_1^2]$, $[c_2^2]$, $[c_1^1]$, $[c_2^1]$
and then to minimize the difference between \( \phi^1_g \) and \( \phi^2_g \). The chosen grasp configurations are then given by \( c^1_g = (p^1_i, \phi^1_i) \) and \( c^2_g = (p^2_i, \phi^2_i) \).

It is expected to be executed that the expectation of the grasp will not lead the fingertips in contact exactly at the two chosen points; however, the contacts will belong to components in the DMG from which it is easy to manipulate the object as desired.

D. Second Gripper Grasp and Release

The second gripper is chosen to maximize the distance from both the current first gripper’s grasp and the planned first gripper’s regrasp.

Since the second gripper has only the function of support, there is no constraint on which DMG components the two antipodal points \( p^1_i \) and \( p^2_i \) should belong to. Their search is biased towards points that are as far away as possible from the first gripper’s grasp and regrasp points.

First, each node in the DMG is weighted according to the distance between its centroid and the grasp line (i.e. the line between the two fingertips) of the first gripper. More specifically, given the distance from two regrasp lines of the first gripper, one for the current grasp \( d_{g\text{grasp}} \) and one for the planned regrasp \( d_{r\text{grasp}} \), we associate two values \( d_{\text{sum}}=d_{g\text{grasp}}+d_{r\text{grasp}} \) and \( d_{\text{diff}}=|d_{g\text{grasp}} - d_{r\text{grasp}}| \). To each node, the distance value is chosen as

\[
  d_n = \max(0, d_{\text{sum}} - \xi d_{\text{diff}}),
\]

in which \( \xi \) is a weighting coefficient. Candidate nodes for the second gripper grasp are those with highest \( d_n \). The angles \( \phi^1_i \) and \( \phi^2_i \) are chosen, given the nodes, as angles that allow the fingers to point towards the second gripper’s side of the robot and maximize the distance between the first gripper’s release and regrasp configurations. The release configurations of the second gripper, in this case, coincide with the grasp configurations, i.e. \( c^1_r = c^2_g = (p^1_i, \phi^1_i) \) and \( c^2_r = c^2_g = (p^2_i, \phi^2_i) \).

If there are no suitable candidate nodes, i.e. \( d_n=0 \) for all the nodes, a new search is performed preferring distance only from the current initial grasp. The two configurations \( c^1_g \) and \( c^1_r \) previously computed as in section [VI-C] should be modified to increase the distance from the newly found \( c^2_g \) and \( c^2_r \). An example in which the first gripper regrasp configuration is changed is shown in Fig. [9]. If the distance cannot be increased sufficiently, then the second gripper must proceed into the release configurations \( c^2_r \) and \( c^2_g \) that are found to be far away from the planned regrasp of the first gripper. The search for the grasp and release configurations is subject to keep them in the same DMG components \( C^g \) and \( C^r \), so that an in-hand path between them can be found. If still is not possible to find configurations that allow the distance between the grippers to keep a tolerable threshold, the next step is to move away the first gripper and plan according to the new pose, as explained in the next session. However, we do not expect this situation to be frequent.

E. First Gripper Release

Most of the times \( c^1_r \) and \( c^1_g \) coincide with \( c_{s1} \) and \( c_{s2} \). However, in case this gripper is required to be moved, it is done along the DMG components in which the fingers currently lie. The new fingers’ configurations are chosen among the nodes that lie in \( C_{s1} \) and \( C_{s2} \) so that they increase the distance from the second gripper grasp configurations \( c^2_g \) and \( c^2_r \).

F. Summary

Algorithm [7] summarizes how to use the DMG with both the in-hand manipulation search and the regrasp planning. The obtained solution consists of the regrasp and release configurations for the two grippers and three in-hand sequences. The sequence \( S^0 \) moves the first gripper from the initial configuration \( [c_{s1}, c_{s2}] \) to the release configuration \( [c^2_r, c^2_g] \); the sequence \( S^1 \) moves the second gripper from the grasp configuration \( [c^1_g, c^1_r] \) to the release configuration \( [c^2_r, c^2_g] \); finally, the sequence \( S^2 \) moves the first gripper from the grasp configuration \( [c^1_g, c^1_r] \) to the desired configuration \( [c_{d1}, c_{d2}] \).

We defined a procedure for regrasp that focuses on moving the gripper that is initially grasping the object towards the desired grasp pose. However, if the desired grasp does not pose constraints on which gripper has to be used, our procedure can be simplified because there is only one regrasp involved. More specifically, there is no need for the first gripper regrasp (section [VI-C]), and the search for the second gripper grasp should be constrained to \( C_{d1}, C_{d2} \) and the distance \( d_n \) should try to minimize the distance from the goal while keeping the grasp far away from the first gripper.

VII. Execution

Once an object repositioning has been planned, we use a dual-arm robot to perform the execution. In this section we briefly summarize the ECTS formulation that we use to control the robot, and then we define the the motion procedure for executing the in-hand manipulation primitives and regrasping. This section corresponds to step 3 in Fig. [1].

A. Dual Arm Coordination

The ECTS [43] is a formulation that enables the control of two robot’s arms to achieve a common task. This task is specified by an absolute and a relative motion between the end-effectors. These motions are described by the absolute velocity vector \( v_a = (v_a, \omega_a)^T \) and the relative velocity vector.
The ECTS allows a dual-arm robot to share the task between the two arms with a desired degree of coordination. Given the desired relative and absolute motions, the two arms’ velocities $\mathbf{x}_1$ and $\mathbf{x}_2$ define how each end-effector will move so that the given coordinated task is achieved. With $\mathbf{x}= (\mathbf{x}_1 \mathbf{x}_2)^T$ as the vector containing the two end-effector’s velocities, and $\mathbf{x}_E= (\mathbf{x}_0 \mathbf{x}_E)^T$ as the vector containing the desired motions, the relation between the two is given by

$$\mathbf{x} = \left( \begin{array}{c} \alpha \mathbf{I}_6 \\
-\mathbf{I}_6 \\
\end{array} \right) \mathbf{x}_E, \quad (12)$$

where $\mathbf{I}_6$ is the 6-dimensional identity matrix. The coefficient $\alpha \in [0, 1]$ manages the share of a coordinated task between the two arms; for instance, if $\alpha$ is 0.5 the task is shared evenly and the end-effector’s motion is symmetric.

The in-hand manipulation primitives define the motion of the object inside one of the robot’s grippers. We describe this motion as relative motion between the two grippers, as one gripper is holding the object and the other one is pushing it to help the in-hand motion. Therefore, we can specify an in-hand manipulation sequence by using $\mathbf{x}_a$.

The velocity $\mathbf{x}_a$ specifies an absolute motion of the two grippers. In our experiments we keep it to $0$ to minimize noise in sensors and errors in actuations, but it can be exploited to move the arms while the in-hand manipulation task is executed. We see this possibility useful to accelerate the task execution, for instance, in assembly lines, in which the robot picks up an object, then properly repositions it inside the gripper and then it is supposed to place or insert the object in a certain way in a different part of the workspace.

### B. Comments on Single Arm Motions

While we specifically target dual-arm systems as ideal platforms for our method, certain solutions can be implemented also on a single arm system. However, this removes the ability of regrasping and it lowers the object repositioning possibilities. A single arm can execute the in-hand manipulation primitives described in section [II-B] by exploiting contacts with the environment, similarly to the works in [31], [32]. The formulation that we propose, based on the ECTS framework, can be kept for a single arm by assuming $\alpha=0$, so that only one arm moves to achieve the desired relative motion, and the environment is seen as an immobile second arm.

### C. Common Procedure for In-Hand Primitives

The two in-hand primitives, translation and rotation, both require an external push for the object to move within the gripper. In our case, we use the second robot’s gripper to push the object. The first robot’s gripper is holding the object at a certain configuration $c$; the in-hand manipulation primitive has the purpose of changing the grasp on the object to a new configuration $c'$, as explained in section [II-B]. The execution procedure is the following:

1. **Find push point**: A push point is a point on the object from which the second gripper can push so that the motion corresponds to the desired translation or rotation. The procedure to find this contact point depends on the type of motion primitive, and its location depends on the current configuration $c$. Examples of how to find the push point are shown in Fig. 10. Push point locations are a factor that can affect the feasibility of certain in-hand motions, and they can be taken into account when searching the DMG by altering the cost of edges, as explained in section VII-B.

2. **Approach push point**: The two robot’s arms move so that the second gripper is in contact with the object at the push point. Normally only the second gripper’s arm is required to move, but both arms can move and adjust the relative pose if necessary, depending on the chosen $\alpha$ value, to improve maneuverability.

3. **Push**: The object is pushed inside the gripper to follow the desired in-hand manipulation primitive. This push is the result of a relative motion between the two arms, so only one or both of them can move according to the desired degree of coordination $\alpha$, as mentioned in section VII-A. While the object is pushed, the first gripper enlarges or tightens the grasp to ease the translational or rotational motion.

4. **Leave push point**: The second gripper leaves the contact with the object. As for the approach motion, the two arms can move or only one of them, depending on the need. At the end of this motion, the first gripper is still grasping the object, but in a new configuration $c'$.

### D. Translation

When the in-hand manipulation primitive is translation, the relative motion between the two grippers is a linear motion along the translation direction. This motion is specified by the translation vector $t_k$ in the gripper’s reference frame. By denoting with $\mathbf{R}_g$ the rotation matrix describing the orientation of the gripper with respect to the base frame, this vector is expressed in global coordinates as $t_k' = \mathbf{R}_g t_k$. For the fingertip to slide as desired, the object must be translated in the opposite direction, denoted by $t_k$, that is $t_k = -\frac{t_k'}{\|t_k'\|}$. This direction corresponds to the direction along which the second gripper should push the object.

The motion of the object inside the gripper corresponds to the relative motion between the two grippers. Our strategy
consists in pushing the object with a given velocity until the first gripper’s fingertip contact reaches the desired point, i.e. the desired translation has been executed. This velocity is \( v = m \dot{t}_k \). We adjust the velocity’s magnitude \( m \) using a P controller. The dual-arm robot’s motion is described by

\[
\dot{x}_r = \begin{pmatrix} v \\ 0 \end{pmatrix},
\]

in which the zero vector indicates that there is no angular velocity, as the object’s orientation should not change. When visual feedback is available, we adjust \( \dot{x}_r \) to correct errors in the object’s motion arising during the execution.

It is assumed that the friction between the second gripper and the object is sufficient to push the latter in the desired direction, even when this direction does not match the surface’s normal. However, in case of high mismatches, it is possible to steer the in-hand path search by adjusting the cost of the DMG edges as mentioned in section V-B. Similarly, translations that require a pull instead of a push, i.e. motions that are in the opposite direction with respect to the first gripper’s finger, can be penalized as well, to prefer a sequence of rotations and translations that can be executed more easily. These penalizations are dependent from the initial grasp configuration; therefore they do not modify the DMG, solely object dependent, but they modify only the DMG search strategy.

**E. Rotation**

When the in-hand manipulation primitive is rotation, the second gripper should push the object along an arc of a circle centered in the current contact point between the first gripper’s fingertip and the object. This contact is fixed on the object, i.e. it does not slide. The relative motion between the two gripper should follow this circle. In addition, we introduce angular velocity so that the contact between the object’s surface and the second gripper does not change during this rotational motion.

\( \gamma_k \) is the desired rotation specified for the first gripper’s finger. The push point \( \mathbf{p}_p \) on the object’s surface should rotate around the first gripper’s fingertip contact, in the opposite direction of the desired rotation. Assuming that the desired rotation \( \gamma_k \) is an angle in the yz plane of the first gripper’s reference frame, the planar circular motion during time is expressed by the vector \( (\cos(\gamma_0 + \gamma(t)) \sin(\phi_0 + \gamma(t)))^T \). \( \phi_0 \) is the initial angle between the push point \( \mathbf{p}_p \) and the first gripper’s finger. The function \( \gamma(t) \) describes the variation of this angle. Assuming the rotation execution between an initial time \( t = t_0 \) and a final time \( t = t_f \), this function is so that \( \gamma(t_0) = 0 \) and \( \gamma(t_f) = -\gamma_k \).

We express the relative motion in the first gripper’s reference frame as

\[
\dot{g} \mathbf{x}_r = \begin{pmatrix} 0 \\ -\sin(\phi_0 + \gamma(t))\gamma(t) \\ \cos(\phi_0 + \gamma(t))\gamma(t) \\ \gamma(t) \\ 0 \\ 0 \end{pmatrix}.
\]

This velocity is then transformed into a relative velocity \( \dot{x}_r \) expressed in the base frame. Similarly to the translation velocity, we adjust the magnitude of the Cartesian velocity during the execution, until the object reaches the desired angle, and the velocity vector is adapted to compensate for errors when visual feedback on the object’s motion is available.

Since large rotations could lead to collisions between the two grippers, they should be penalized similarly to the penalization of different translation directions during the DMG search.

**F. Regrasping**

The regrasping execution follows this sequence:

1) **Reach First Release Configuration**: At the beginning, the first gripper is holding the object at the initial fingers’ configurations \( c_{1g_1}^{1} \) and \( c_{1g_2}^{1} \). If the release configurations \( c_{2g_1}^{1} \) and \( c_{2g_2}^{1} \) do not coincide with them, an initial in-hand motion is performed until they are reached.

2) **Second Gripper Grasp**: The second gripper should grasp the object so that its fingers are in the configurations \( c_{1g_1}^{2} \) and \( c_{1g_2}^{2} \). If these configurations are difficult to reach given the current relative position of the two robot’s arms, the first arm moves so that the line starting from the first gripper to the desired second gripper’s grasp points is directed towards the second arm. However, since this side of the robot is preferred during the search for regrasp, the need for this reconfiguration is minimal. Once the desired configurations are within reach, the second gripper closes on the object. At the end of this phase, both grippers are grasping the object.

3) **First Gripper Release**: The first gripper opens the fingers to release the object and it is moved away from it. The object is now held only by the second gripper.

4) **Reach Second Release Configuration**: If the second gripper’s release configurations \( c_{2g_1}^{1} \) and \( c_{2g_2}^{1} \) differ from the grasp configurations, in-hand motion is required to readjust the pose of the object. In this case, the in-hand motion sequence is executed as described previously, with the only difference of assuming the first gripper as pusher and the second gripper as holding the object.

5) **First Gripper Grasp**: The first gripper should grasp the object to reach the two fingers’ configurations \( c_{1g_1}^{1} \) and \( c_{1g_2}^{1} \). Similarly to the second gripper grasp, the relative pose between the two arms should be properly adjusted when needed. At the end of this phase both grippers are grasping the object.

6) **Second Gripper Release**: The second gripper opens the fingers to release the object and it moves away from it. The object is now held only by the first gripper. If the current grasp configurations are not the desired ones \( c_{1g_1}^{1} \) and \( c_{1g_2}^{1} \), a final in-hand manipulation execution is necessary to drive the object into the desired pose.

**VIII. Experiments**

In this section we present the results obtained using the DMG for grasp reconfiguration.
TABLE I: Times of DMG generation. The total time is a sum between the time spent in the supervoxel segmentation and the one spent in refining the graph.

| object | bounding box | $t_{\text{area}}$ [mm] | $t_{\text{angle}}$ [deg] | Time [s] | $\delta_{c}$ | Neigh | Comp |
|--------|--------------|------------------------|---------------------------|---------|-------------|--------|------|
| 50×50×40 | 10 | 1.861 (0.299+1.562) | 1.847 (0.299+1.548) | 20.453 | 0.15 | 15 | 20 |
| 100×80×50 | 15 | 0.945 (0.353+0.592) | 0.943 (0.353+0.590) | 20.453 | 0.15 | 15 | 20 |
| 87.8×107.2×104.8 | 20 | 1.050 (0.465+0.585) | 1.049 (0.465+0.584) | 20.453 | 0.15 | 15 | 20 |

A. General Setup

To obtain the subdivision of an object’s shape into small areas, we used the supervoxel method available in the Point Cloud Library (PCL) [50]. For the robot experiments, we used an ABB Yumi with a Kinect v2 camera mounted on top. The camera was used to detect the object’s pose with respect to the gripper’s fingertips, using Apriltags [51]. This pose detected at the beginning of the execution was used to infer the initial fingers configurations, and during the in-hand manipulation and after regrasp it was used as visual feedback to adjust Yumi’s motion. Yumi’s finger implied $\delta_{c}$ and after regrasp it was used as visual feedback to adjust the desired configuration.

TABLE I: Times of DMG generation. The total time is a sum between the time spent in the supervoxel segmentation and the one spent in refining the graph.

B. Graphs Generation

A new DMG should be generated for each object we want the robot to manipulate. The complexity of the generation procedure (Algorithm [1]) can easily be obtained: with $|N|$ denoting the number of nodes derived from the supervoxels, directly dependent on the chosen resolution $t_{\text{area}}$, $K_{\text{neigh}}$ as the maximum number of edges per node and $K_{\text{comp}}$ as the maximum number of angular component per supervoxel centroid, we can write the complexity as

$$O((\max((3 + K_{\text{comp}})K_{\text{neigh}}, 2K_{\text{angle}}))|N|). \quad (15)$$

Since these numbers are subject to geometric constraints of the object’s shape, $K_{\text{neigh}}$ is usually at most 5 and $K_{\text{comp}}$ is 1 for most of the supervoxel centroids, and it can be expected to remain a low number (2 or 3). $K_{\text{angle}}$ depends on the chosen resolution for the angles $t_{\text{angle}}$.

We tested the graph generation on different objects, varying the resolution of the supervoxels $t_{\text{area}}$ and the angle resolution $t_{\text{angle}}$. Table [4] shows the time spent to generate the DMG, which is a sum of the time (in average) required by the supervoxel method to provide a subdivision of the object into small areas and the time to refine the connectivity between the nodes associated to the centroids of these areas. The time has been obtained from running the DMG generation on an Intel Core i7-6700 3.40 GHz processor.

The main difference between the first and the second object is due to the increase in size. The third object, a mug from the YCB object dataset [52], requires less time than the second object to generate the DMG, despite being the biggest object. This is due to the particular object’s shape, that presents a curved surface that causes the mismatch between the normals to exceed the threshold for translation much more frequently. In addition, the object model is noisy. The result is a graph with many small isolated components, for which fewer iterations are required. We increased the value of $\delta_{c}$ up to 0.3 and verified that this increase does not produce significant changes both in the graph structure and in the generation time, due to the curvature and the noisy reconstruction of this shape.

These results show that the main change on the graph generation time is given by the different values of $t_{\text{area}}$, which has a direct influence on the term $|N|$. The value $t_{\text{angle}}$, which instead influences $K_{\text{comp}}$ and $K_{\text{angle}}$, has a much smaller effect in the variation of the required time.

We think of the DMG as a tool that can be exploited multiple times once it has been generated for a given object, for instance in industrial applications where the same object’s shape must be manipulated several times in different ways. However, the times reported in Table [4] show that it is also suitable for single use applications in which a few seconds can be sacrificed before initiating the in-hand manipulation task.

C. Regrasping Execution

We tested the regrasping strategy combined with in-hand manipulation. We gave the object to one of the robot’s grippers and set the goal pose so that regrasping was required to achieve the desired configuration.

In the example shown in Fig. 11, the goal pose of the fingers is inside a concavity of the object; directly regrasping the object inside the concavity is not trivial, as the gripper cannot easily approach the desired contact point with open fingers. Therefore, the DMG solution provides a regrasp on the same graph component of the desired configuration, from which the fingertip can slide into the concavity. This solution was obtained from a DMG generated with $t_{\text{area}}=10$ mm.

The execution of this grasp reconfiguration is shown in Fig. 12. Yumi is initially grasping the object as shown in Fig. 12a. From this initial grasp, the initial configuration of...
Fig. 11: Two views of the planned grasp reconfiguration. The gripper’s fingers should go from the initial configuration, shown in red, to the desired configuration inside the object’s concavity, shown in green. The Second gripper should help regrasping by holding the object in the configuration shown by the blue fingers. The first gripper should regrasp the object as shown by the dark green fingers so that the fingertips are in contact as shown by the dark green sphere and then slide inside the concavity to reach the desired configuration.

the fingers on the object is obtained, given the object’s pose visible from the Kinect. To achieve the desired grasp, Yumi has to regrasp the object, as shown in Figs. 12b and 12c and then push the object to execute the in-hand manipulation task, as in Fig. 12d. After the push, the fingers have reached the desired contact, shown in Fig. 12e. As it is possible to see from a closer look of the final configuration (Fig. 12f), while the contact point is inside the object’s concavity as desired, the final pose of the object inside the gripper is slightly rotated with respect to the desired one. This is due to the lack of feedback during the push, as the arm was covering the camera; therefore, the pushing action could not be adjusted during the execution. However, a new in-hand plan providing a simple rotation action is enough to adjust the final orientation of the object in cases like this.

D. In-Hand Manipulation Execution

In our previous work [1] we tested different in-hand manipulation solutions using a Baxter robot. We executed more experiments with Yumi and we also show an example of in-hand manipulation exploiting the environment as an immobile second arm.

Fig. 13a depicts an initial grasp on an object and Fig. 13b shows the same fingers’ configuration on the object, in red, and the desired pose of the fingers, in green. Since the two configurations lie in the same DMG components, i.e. the condition in (3) is not verified, it is possible to move the fingers from one to the others using in-hand manipulation. The path in blue in Fig. 13b shows the desired motion of the fingertip along the object’s surface. Despite being planned along the flat surface, this solution avoids the obstacle on the opposite side of the object, as explained in section V-B. The sequence of in-hand motions is composed of two translation segments and a rotation of $90^\circ$. Fig. 14 shows Yumi executing the rotational motion of this in-hand motions sequence.

Using the same object, we tested a simple in-hand manipulation task with an external support to help pushing, instead of using the second Yumi’s arm. This situation is the one described in section VII-B. We set as contact point for
Fig. 13: An example of in-hand manipulation task that we executed with the Yumi robot. The goal is to move the fingers on the object from the initial configuration to the desired one without releasing the grasp.

(a) Yumi’s initial grasp.
(b) In-hand solution.

Fig. 14: Yumi executing an in-hand rotation by pushing the object with the second gripper along an arc of a circle.

Fig. 15: The initial grasp of the left image is shown in red in the right images, and the desired grasp is in green. The image on the bottom right shows the object’s DMG and the solution (in blue) to move the fingers to the desired configuration.

(a) Yumi’s initial grasp on the object.
(b) Initial and desired grasp.
(c) In-hand solution from DMG.

Fig. 16: The execution of a translational push using a contact surface instead of the second robot’s arm.

Pushing a point on the table in front of Yumi. Therefore, the grasping arm should move with respect to this point to achieve a successful push. Fig. 15a shows the initial grasp of Yumi’s gripper on the object. This time, Yumi was grasping the object on the cylindrical part. The desired grasp is shown in Fig. 15b. Fig. 15c shows the in-hand solution obtained from the DMG. The solution is composed of an in-hand translation with no rotation along the cylindrical part of the object. Fig. 16 show the execution of this translation exploiting the table to push the object inside the gripper. It is important to notice that the pushing strategy that we designed assumes contacts between points, and not surfaces. Therefore, for more complex in-hand tasks, simply replacing the second gripper with a contact surface might lead to undesired behaviors. However, the DMG solution can still be used with a single arm robot, as long as the contacts for pushes are properly designed and, if necessary, taken into account in the graph search phase. The main drawback lies in the requirement to properly model the surrounding environment or to place ad-hoc fixtures to execute the desired pushes.

IX. Conclusions

We have presented and analyzed the Dexterous Manipulation Graph as a tool for object manipulation. We have described how this graph representation is obtained given an object’s shape and how to exploit it to plan grasp reconfigurations. More specifically, the DMG can be used both for in-hand manipulation and for more general reconfigurations that involve regrasping. We have formulated the in-hand manipulation execution for a dual-arm robot as a relative motion between the two grippers by using ECTS. Experiments with Yumi have shown how the application of the DMG helps in planning in-hand manipulation and other manipulation tasks.

As future work, we plan to extend our in-hand manipulation planning to take into account unknown object. This involves a generation of the DMG from the available sensory data, which will only provide an incomplete shape. The DMG should be modified to take into account the lack of full information and have the capability to adapt itself to new available sensory information. In addition, we will explore modifications of the DMG search that are more connected to the dual-arm robot’s kinematic structure, so that the robot’s arms motion and the object’s motion are more dependent on each other and are chosen according to joint optimization criteria. Finally, we plan to adapt the DMG to take into account multi-fingered hands and allow for planning of other dexterous manipulation tasks.

REFERENCES

[1] S. Cruciani, C. Smith, D. Kragic, and K. Hang, “Dexterous manipulation graphs,” in 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Oct 2018, pp. 2040–2047.
[2] J. Bohg, A. Morales, T. Asfour, and D. Kragic, “Data-driven grasp synthesis—a survey,” IEEE Transactions on Robotics, vol. 30, no. 2, pp. 289–309, April 2014.
[3] P. Tourmassoud, T. Lozano-Perez, and E. Mazer, “Regrasping,” in Robotics and Automation. Proceedings. 1987 IEEE International Conference on, vol. 4, Mar 1987, pp. 1924–1928.
[4] N. Furukawa, A. Namiki, S. Taku, and M. Ishikawa, “Dynamic regrasping using a high-speed multifingered hand and a high-speed vision system,” in Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006., May 2006, pp. 181–187.
[5] C. D. Santina, C. Piazza, G. Grioli, M. G. Catalano, and A. Bicchi, “Toward dexterous manipulation with adapted adaptive synergies: The pisa/lutf softhand 2,” IEEE Transactions on Robotics, pp. 1–16, 2018.
[6] K. Oz, M. Tomura, A. Schmitz, S. Funabashi, and S. Sugano, “Interpolation control posture design for in-hand manipulation,” in 2015 IEEE/SICE International Symposium on System Integration (SII), Dec 2015, pp. 187–192.
[51] E. Olson, “AprilTag: A robust and flexible visual fiducial system,” in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA). IEEE, May 2011, pp. 3400–3407.
[52] B. Calli, A. Singh, J. Bruce, A. Walsman, K. Konolige, S. Srinivasa, P. Abbeel, and A. M. Dollar, “Yale-cmu-berkeley dataset for robotic manipulation research,” The International Journal of Robotics Research, vol. 36, no. 3, pp. 261–268, 2017.

Silvia Cruciani received the B.S. degree in Computer Science and Control Engineering in 2013 and the M.S. degree in Artificial Intelligence and Robotics in 2015 from Sapienza University of Rome, Italy. She is currently a Ph.D. student at the Robotics, Perception and Learning Lab at KTH Royal Institute of Technology, Stockholm, Sweden. Her research is focused on vision-based dexterous manipulation and extrinsic dexterity.

Kaiyu Hang is a postdoctoral associate with the Department of Mechanical Engineering and Material Science, Yale University, New Haven, USA. He received his Ph.D. in Computer Science, specialized in Robotics and Computer Vision in 2016 from KTH Royal Institute of Technology, Stockholm, Sweden. Before that, he received the B.S. degree in information and communication engineering from Xian Jiaotong University, Xian, China in 2010 and the M.Sc. degree in communication systems from KTH Royal Institute of Technology, Stockholm, Sweden in 2012. His research interests include dexterous grasping and manipulation, mobile manipulation, and robot end-effector design and optimization.

Christian Smith is an Associate Professor in Computer Science at the Division of Robotics, Perception, and Learning at The Royal Institute of Technology in Stockholm (KTH). He received his M.Sc. in Engineering Physics in 2005, and Ph.D. in Computer Science in 2009, both at KTH. Between 2010-2011, he was a post-doctoral researcher at Advanced Telecommunications Research International (ATR) in Kyoto, Japan. He is the secretary of the IEEE Robotics and Automation Society Swedish Chapter. Research interests include control and modelling for manipulation and grasping in human-centric environments and human-robot interaction.

Danica Kragic is a Professor at the School of Electrical Engineering and Computer Science at the Royal Institute of Technology, KTH. She received MSc in Mechanical Engineering from the Technical University of Rijeka, Croatia in 1995 and PhD in Computer Science from KTH in 2001. She has been a visiting researcher at Columbia University, Johns Hopkins University and INRIA Rennes. She is the Director of the Centre for Autonomous Systems. Danica received the 2007 IEEE Robotics and Automation Society Early Academic Career Award. She is a member of the Royal Swedish Academy of Sciences, Royal Swedish Academy of Engineering Sciences and Young Academy of Sweden. She holds a Honorary Doctorate from the Lappeenranta University of Technology. She chaired IEEE RAS Technical Committee on Computer and Robot Vision and served as an IEEE RAS AdCom member. Her research is in the area of robotics, computer vision and machine learning. In 2012, she received an ERC Starting Grant. Her research is supported by the EU, Knut and Alice Wallenberg Foundation, Swedish Foundation for Strategic Research and Swedish Research Council. She is an IEEE Fellow.