Vision-based robotic dual arm manipulation of soft objects

Célia Saghour

THÈSE POUR OBTENIR LE GRADE DE DOCTEUR DE L'UNIVERSITE DE MONTPELLIER

En Systèmes Automatiques et Microélectroniques (SYAM)

École doctorale : Information, Structures et Systèmes

Unité de recherche LIRMM

Vision-based robotic dual arm manipulation of soft objects

Présentée par Célia SAGHOUR

Le 8 décembre 2023

Sous la direction de Andrea CHERUBINI et Philippe FRAISSE

Devant le jury composé de

M. Alexandre KRUPA, Directeur de Recherche Inria, Université de Rennes, Inria, CNRS, IRISA	Rapporteur
M. Youcef MEZOUAR, Professeur des Universités, Université Clermont Auvergne, Sigma	Rapporteur
M. David NAVARRO-ALARCON, Professeur Associé, The Hong Kong Polytechnic University	Examinateur
M. Vincent CREUZE, Professeur des Universités, Université de Montpellier, LIRMM	Examinateur, *Président*
M. João CAVALCANTI-SANTOS, Professeur des Universités, Université de Montpellier, LIRMM	Invité
M. Andrea CHERUBINI, Professeur des Universités, Université de Montpellier, LIRMM	Directeur de thèse
M. Philippe FRAISSE, Professeur des Universités, Université de Montpellier, LIRMM	Directeur de thèse

Titre : Manipulation robotique bi-bras d'objets souples basee vision

Résumé :

Les objets non-rigides sont amplement présents autour de nous, qu'il s'agisse de notre vie quotidienne ou d'un contexte industriel. Malgré le besoin croissant d'automatisation de la manipulation de tels objets, il n'existe à ce jour pas de méthode générale et facilement implémentable pour contrôler la forme de ceux-ci. En effet, l'important nombre de degrés de liberté des objets souples fait qu'il est difficile de suivre et contrôler la façon dont ils se déforment pendant la manipulation.

L'asservissement visuel non-calibré, une méthode permettant d'estimer en ligne un model caractérisant le contrôle du robot en fonction de données visuelles, a été développé et largement utilisé depuis des années. Pourtant, les premiers travaux implémentant l'asservissement visuel ne considèrent en général que les objets rigides. Plus récemment, des chercheurs ont conçu de multiples contrôleurs de formes, s'appuyant sur l'estimation de modèle, les réseaux de neurones ou encore l'apprentissage par renforcement pour calculer la commande d'un robot. Bien que ces méthodes présentent de bons résultats, il y a néanmoins des inconvénients, comme le besoin de connaissances sur l'objet en amont (forme, paramètres matériaux), un modèle spécifique (câbles, tissus), d'importantes bases de données ou un temps d'apprentissage élevé; tout cela peut limiter l'éventail d'applications possibles ou rendre l'implémentation de ces méthodes laborieuses.

Cette thèse a pour objectif de combiner les concepts d'asservissement visuel, les méthodes de contrôle basées données ainsi que basées modèle, dans le but de contrôler la forme d'objets souples avec un robot bi-bras. Nous présentons d'abord une méthode de contrôle basée données pour déformer des objets en un contour 3D désiré, utilisant des référentiels de tâches coopératives pour coordonner deux bras robotiques. Cette méthode ne requiert aucune connaissance a priori de l'objet manipulé, mais est limitée au contrôle d'un contour 3D.

Dans cette optique, nous proposons ensuite des outils pour construire de simples modèles géométriques et mécaniques en tirant profit du robot bi-bras à disposition. Nous implémentons des simulations physiques permettant de relier la déformation des modèles obtenus avec les déplacements des effecteurs du robot en temps réel.

Enfin, nous utilisons ces simulations en temps réel, combinées à un retour visuel, pour alimenter une boucle de contrôle de forme prenant en compte le volume entier des objets manipulés. Notre méthode peut être utilisée avec peu de connaissance sur l'objet manipulé, permettant une mise en place facilitée et un plus large choix d'applications. De plus, l'utilisation de simulations remédie à la difficulté d'obtenir des données visuelles consistantes pour le calcul de commande. Tout au long du développement, des validations expérimentales sont conduites avec le robot BAZAR composés de deux bras

robotiques KUKA, différents objets, et une seule caméra RBG-D statique.

Mots-clefs : Manipulation d'Objets Déformables, Manipulation Bi-Bras, Asservissement Visuel, Méthode d'Elements Finits, Real-to-Sim.

Title: Vision-based robotic dual arm manipulation of soft objects

Abstract:

Non-rigid objects are present everywhere in our daily life as well as industrial contexts. Despite the increasing need for automating their manipulation, there is to date no generally applicable and easily implementable method to control the shape of such objects. Indeed, the high number of Degrees of Freedom (DOF) of soft objects makes it difficult both to track and control the way they deform during manipulation.

Uncalibrated visual servoing, a method where the model mapping visual data to robot control is estimated online, has been developped and widely used for years. Yet, the early works implementing visual servoing usually focus only on rigid objects. More recently, researchers have designed shape servoing controllers, relying on model estimation, neural networks or reinforcement learning to compute the command. Although these methods present good results, there are drawbacks - the need for prior knowledge of the object (shape, material parameters), specific models (cables, fabrics), huge datasets and high training time - all limiting the range of application or increasing the difficulty of implementation.

This thesis aims at combining visual servoing, data-based and model-based methods with the goal of shaping soft objects with a dual-arm robot. We first present a data-based control framework to shape objects into a desired 3D contour, using cooperative tasks frames to coordinate both arms. This work does not require any knowledge of the manipulated object, but is limited to shaping a contour.

With this in mind, we then propose tools to build geometrical and simple mechanical models while making use of the dual-arm robot setup. We implement physics simulations relating the deformation of the obtained models with the real-time displacements of the end-effectors.

Finally, we use real-time simulations combined with visual feedback to feed the control framework to shape objects in their full volume. Our framework operates with little knowledge of the manipulated object, allowing a large range of applications, and the use of simulations remedies the difficulties of getting visual data for control computation. Throughout the developments, we conduct experimental validations with the robot BAZAR (composed of two KUKA arms) shaping vaious soft objects, thanks to a single fixed RGB-D camera.

Keywords: Soft Objects Manipulation, Dual-Arm Manipulation, Visual-Servoing, Fi-

nite Element Method, Real-to-Sim.

Discipline : Informatique, Structure et Systèmes

Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier
UMR 5506 CNRS/Université de Montpellier
Batiment 5 - 860 rue de St Priest

Acknowledgments

I would like to first and foremost thank Andrea Cherubini and Philippe Fraisse for giving me the opportunity to work with them for a bit more than three years, and for their endless support during our time together.

Of course, I also thank David Navarro-Alarcon for his warm welcome in the ROMI-lab, who's made my stay in Hong Kong a fun and interesting experience. I was also very happy to see have him assist in person to my defense, and hope we'll meet again. With this, I also thank all the friends I made in the ROMI-lab and outside of it, Samantha Lee, Zoë Ma, Bowen Fan and all the others, and most particularly, Luiza Labazanova who I really admire and in whom I found a very dear friend, and who's made every day stuck in the dungeon that is the ROMI-lab fun. I would also like to thank Christina Hoellriegl, with whom I enjoyed sharing a flat and helping me get the motivation to go out of my room.

Back in Montpellier, I of course thank everyone in the IDH team, people who have since left or are newly arrived, for their friendliness. Especially, I thank Julien Roux who was very welcoming when I first arrived, as well as Johan Medrano, Hugo Lefèvre, Carole Fournier, Antonin Dallard, Camille Mutschler with whom I could share these times of both fun and struggles. They all have become dear friends to me in this shared experience. I also thank Guillaume Gourmelen, Benjamin Navarro, and Robin Passama for their help and support. I'd like to also thank Mathieu Célérieu for being a great help during his internship among us.

I also would like to thank my friends from my time in Polytech, Loïna Grondin, Thomas Rouzo, Clement Tosi, Agathe Bignon, Sebastien Trombert, Nathan Collin, Iris Grandjin, Jean Robin Peiteado and Vincent Roussel who I've known for a while now, and that I'm happy to count as my friends. Special thanks to my dear friends from abroad, Elisa Martinez, Maria Paula Castilla and Grace Whited for always supporting me even from far away.

A thank you to my family for always believing in me, and especially for my grand-

mother, I'm very grateful.

Finally, I would like to thank Stephane Sarrade and Gentianne Venture for giving me the opportunity to do an internship in the Tokyo University of Agriculture and Technology back in 2019, which was my first experience in academic research and gave me the desire to pursue a PhD in robotics.

This thesis was supported by European Union Horizon 2020 Research and Innovation Programme as part of the project SOPHIA, whose members I thank for having me as part of this interesting project.

Contents

List of Figures

Bold symbols are for vectors.

Acronyms

DOF	Degrees Of Freedom
FEM	Finite Element Method
ICP	Iterative Closest Point
KNN	K-Nearest Neighbors
PCA	Principal Component Analysis
RKCL	Robot Kinematics Control Library
RMSE	Root-Mean-Square Error
SVD	Singular Value Decomposition

List of symbols

E	Young Modulus
$\dot{\boldsymbol{\theta}}$	Joints command
$\boldsymbol{\epsilon}$	Strain
$\boldsymbol{\sigma}$	Stress
$\mathbf{F}$	Force
$\mathbf{H}$	Holding nodes
$\mathbf{J}$	Jacobian matrix
$\mathbf{L}$	Interaction matrix
$\mathbf{c}$	Contour
$\mathbf{m}^v$	Visual mesh nodes
$\mathbf{m}$	Tetrahedral mesh nodes
$\mathbf{pc}$	Point cloud
$\mathbf{pr}$	Rigid particle
$\mathbf{r}$	Robot pose
$\mathbf{s}$	Shape features
$\mathbf{u}$	Displacement
$\mathcal{M}^v$	Visual (triangle) mesh
$\mathcal{M}$	Tetrahedral mesh
ν	Poisson coefficient

Introduction

Context

Cables, organic matters like tissues or vegetables, polymers foams, clothes, even metal parts - are deformable. Despite the presence of so many non-rigid objects everywhere in our daily life and despite the increasing need for automating their manipulation, to date, there is no generally applicable and easily implementable method to shape such objects. The high number of Degrees of Freedom (DOF) of soft objects, which deform in every direction, makes it difficult to track and control their shape during handling.

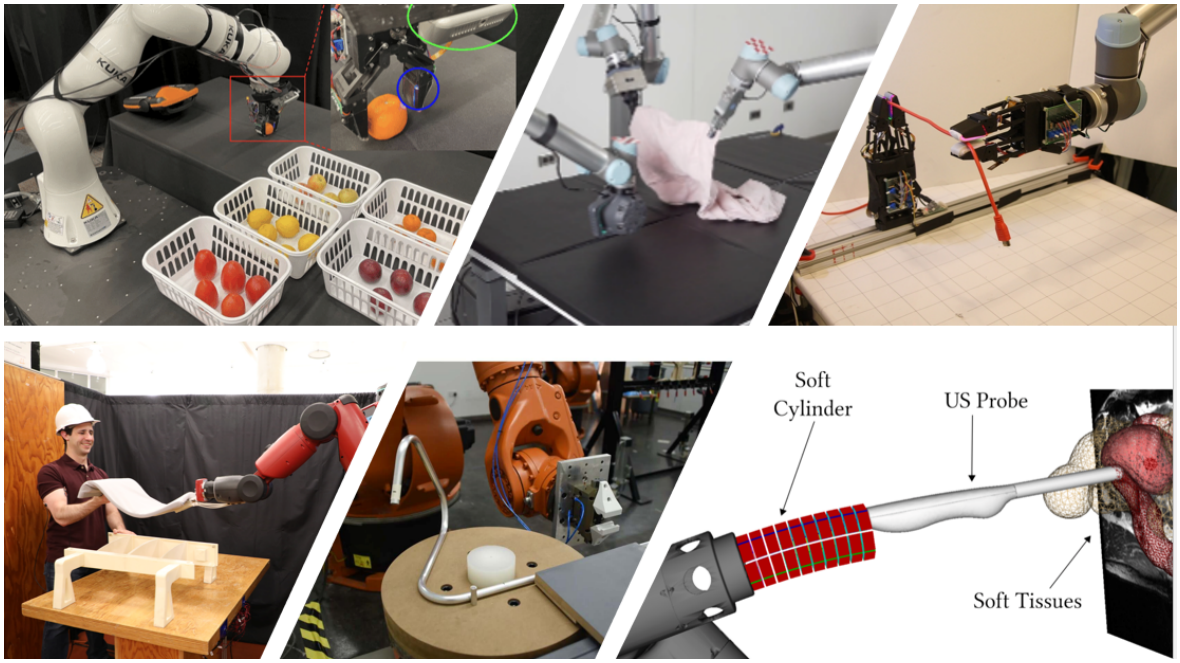


Figure 1: Soft objects handling in robotics. Fruit picking ([HBB+21]), cloth unfolding ([XCB+22]), cable manipulation for plug insertion ([SWD+21]), collaborative lifting ([DR19]), "Multiplanar Robotic Tube Bending" design research project for caadria 2022 post-carbon[2], and robotic prostate probing ([CAL+22]).

Manipulating objects such as clothes [NKBC22], wires [LSGL18], [MS22] or tying suture on biological surfaces [SDO+16], [ZWWL19] constitute important and open challenges in robotics and related areas. Folding clothes [GCLW+20] or picking fruits [UWH+20] and vegetables [SFN+20] are simple tasks for humans, which prove difficult for robots. Such tasks often involve two important aspects: the use of two coordinated arms, and the handling of non-rigid objects. For all these reasons, in this thesis, we focus on the dual-arm manipulation of such objects.

The research conducted in this thesis was funded by the Socio-Physical Interaction Skills for Cooperative Human-Robot Systems in Agile Production (SOPHIA[3]), which is supported by the European Union's Horizon 2020 research and innovation program. The SOPHIA project is a European project pursued in collaboration by several universities (Istituto Italiano di Tecnologia, Univerité de Montpellier, University of Twente, Vrije Universiteit Brussel, Università di Pisa) and industries (Hidria, Volskwagen, Baua, Hankamp, Inail, IMK automotives, DIN) with at its heart, the development of robotic technologies for socially cooperative human-robot systems. SOPHIA's objective is to propose socio-physically adaptive, ergonomic, reconfigurable and intuitive production systems, for the industry. In this context, handling deformable industrial parts like one produced at Hidria and shown in Fig.2 - needs to be studied in order to automatize their processing in the industrial context. Indeed, the robotic manipulation of deformable objects is harder to manage than that of rigid objects. The change in shape must be monitored, to avoid damaging the objects permanently.

Figure 2: Industrial part produced by Hidria.

Motivations

This thesis focuses on the development of methods for automatic dual-arm manipulation of various soft objects. In particular, we investigate the handling - i.e, tracking and control - of deformation through vision, namely *visual shape servoing*. The principle is as follows: the robot end-effectors guide the soft object, from an initial shape and towards a desired shape, as illustrated in Fig. 3. The control scheme is based on visual information, which can be obtained with non-expensive cameras.

[2] https://caadria2022.org/projects/multiplanar-robotic-tube-bending/
[3] https://project-sophia.eu/

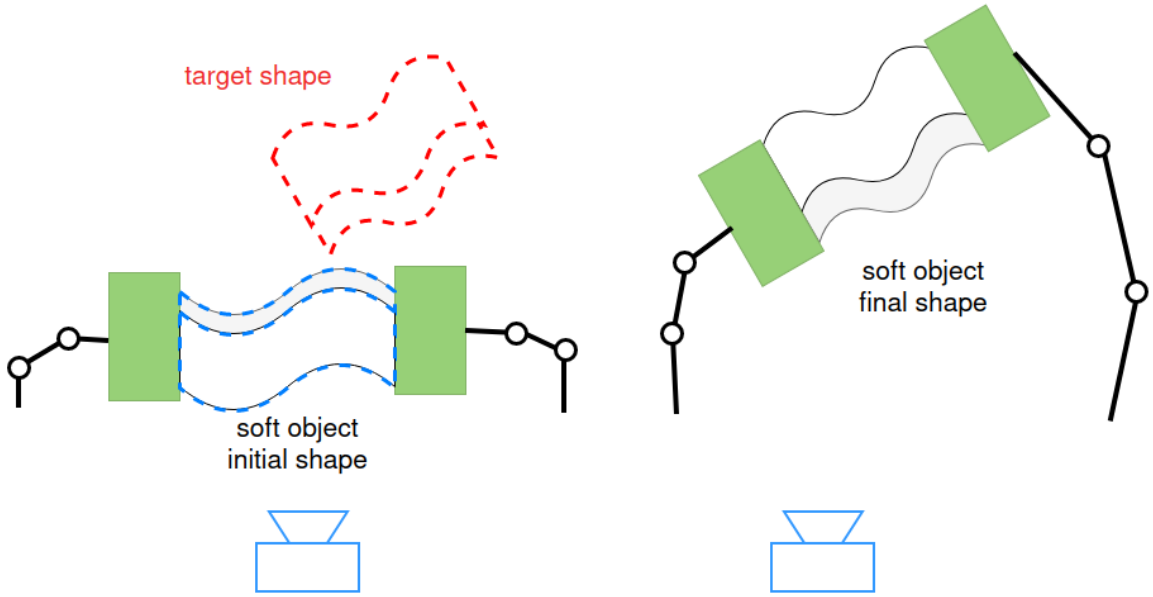


Figure 3: Illustration of visual shape servoing. The robot aims should drive the object from an initial shape (dotted blue) observed by the camera (in blue) to a target shape (dotted red).

The objectives of this thesis are the following:

1. Propose a general framework to control the shape of different soft objects as precisely as possible. The control scheme should apply to the use of two robotic arms in 3D.

2. Include visual information in the control loop in real time.

3. Infer the deformation of the whole object, including its self-occluded parts.

4. Validate the proposed method through experiments with different types of objects.

Setup

The robot performs visual perception with a fixed Intel RealSense D435 RGB-D camera. The algorithm that performs tracking is programmed using Python and the libraries *OpenCV*[4], *pyrealsense*[5] and *Open3d*[6]. The computations of robot poses are also done in a Python script, and sent to the robot via the control library *RKCL*[7], in C++, which converts the pose command into joint commands for the robot. Communication between the Python script and the C++ script is done through a socket using

[4] https://opencv.org/
[5] https://dev.intelrealsense.com/docs/python2
[6] http://www.open3d.org/
[7] https://rkcl.lirmm.net/rkcl-framework/

the library *Nanomsg*[8].

For our experiments, we use the dual-arm robot BAZAR ([CPN+19]), composed of a mobile base and two KUKA LWR4+ arms which are each equipped with ATI Mini 45 force/torque sensor. The setup is shown in Fig. 4.

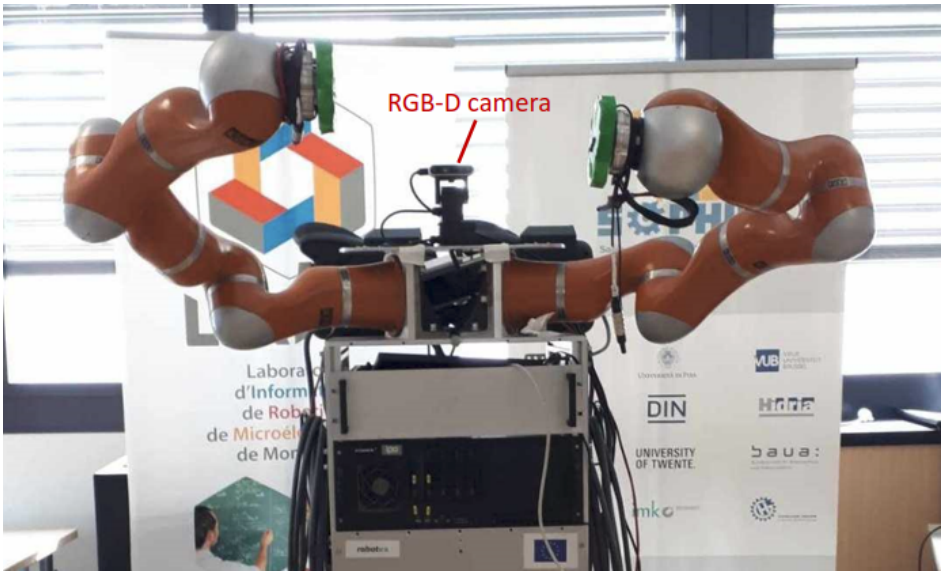


Figure 4: Robotic setup.

Assumptions
Throughout the manuscript, we consider these hypothesis:

- The soft object is already *rigidly* grasped by the manipulator, without any loose contact during the whole task.
- At least one face of the soft object is in the field of vision of the camera throughout manipulation.

Contributions

The main contributions of this work are:

Chapter 2: A visual servoing controller for cooperative dual-arm shaping of soft objects

- A controller in $\mathbb{SE}(3)$ to shape the 3D contour of the visible surface of soft object into a target 3D contour.
- An implementation of the cooperative task space for independent control of the deformation and pose.

[8] https://nanomsg.org/

- The validation of our closed-loop controller on experiments with a dual-arm robot and different objects.

Chapter 3: Tools for soft objects modeling

- A physics-based simulation implementation, to predict the deformation of soft objects according to the robot motions.
- A methodology to estimate the radius and length of Deformable Linear Objects (DLO) from single view RGB-D data.
- A process to create meshes from different types of reconstructed data (either DLO parameters or a reconstructed point cloud).

Chapter 4: A hybrid model-based and visual servoing controller for dual-arm shaping of soft objects

- A controller in $\mathbb{SE}(3)$ to shape the whole volume of soft object, represented by a mesh, into a target mesh.
- An implementation of the physics-based simulation setup introduced in Chapter 3 in the loop of the controller, allowing to estimate the deformation (and breakage) of the entire soft object during robotic manipulation.
- A strategy to evaluate and correct the estimated (via the physics-based simulation) deformation with regards to that observed by the RGB-D camera.
- The validation of of our closed-loop controller controller with a dual-arm robot, for different foam objects.

Organization of the thesis

This thesis is organized as follows, and as summarized in Fig.5:

- Chapter 1 presents a review of the different methods existing in the literature for soft object robot manipulation. In particular, it addresses the *tracking* and *control* challenges.
- Chapter 2 presents a dual-arm visual servoing scheme to shape soft objects in $\mathbb{SE}(3)$. It aims at driving the object's shape, defined by a 3D contour of the visible surface, to a desired shape, while using cooperative tasks to control the deformation and pose separately. We validate the framework with experiments on our robot and with different objects, but highlight the limit of this shape representation.
- Chapter 3 introduces tools for modeling soft objects, in an attempt to answer the challenges that arose in Chapter 2. We present a physics simulation setup to estimate the deformation of an object under external forces, and we propose geometric model construction tools, which can be used to implement these simulations.

- Chapter 4 presents a control scheme using model-based visual servoing for soft objects; it proposes a solution to the vision oriented challenges that were made apparent in Chapter 2, by putting into use the modeling tools introduced in Chapter 3. Experiments are conducted, to validate the approach, both in simulation and on the robot, with several foam objects.
- Chapter 5 summarizes and concludes the work done in this thesis.

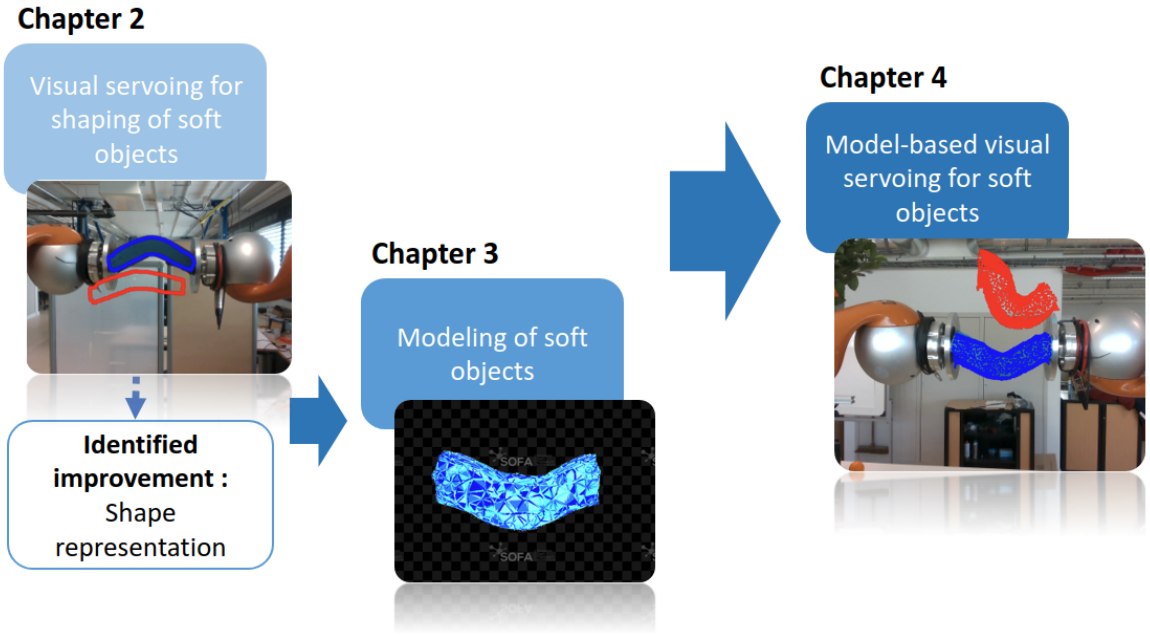


Figure 5: Research process and organization of the manuscript.

CHAPTER 1

State of the art

Robotic manipulation of soft objects is a subject that has been studied in numerous contexts and with various approaches, in recent years (see the survey [YVK21] for examples on multi-robot manipulation of deformable objects).

The literature relevant to our work can be divided into two categories: firstly, **tracking** of non-rigid objects and secondly, robotic **control** of non-rigid objects.

In the first category (**tracking**) are investigated methods to follow and represent the evolution of the deformations the tracked object is subjected to. Tracking methods can be touch-based ([SMC+18]), but we focus on the *visual tracking* of soft object, i.e. the tracking of deformation through visual feedback, as it is most relevant to our work. Visual tracking of deformations can also be divided into two categories of methods:

- *Model-based*, which relies on a known *template* of the object (its topology) and/or a physical model.
- *Model-free*, which does not rely on any model.

In the second category (**control**), which is on the subject of control schemes for soft objects, are presented methods to drive robotic manipulators into accomplishing a given task with a soft object. Here, we mostly focus on shape-servoing, which consists in controlling the deformation that a soft object undergoes in order to reach a desired shape. These control methods can also be divided in different categories:

- *Data-based*, which consists in learning the behavior of a soft object offline, thanks to preliminary collected data.
- *Model-based*, where a chosen physical model of the object is used to predict the behavior of the soft object in response to its manipulation.

- *Online Jacobian estimation*, which aims to iteratively estimate a local and simplified model of the deformation with regards to the robot motions during manipulation. In particular, we focus on the implementation of *visual-servoing* methods applied to soft objects, which consist in mapping the robot inputs with features extracted from visual observations.

These different categories are illustrated in Fig.1.1.

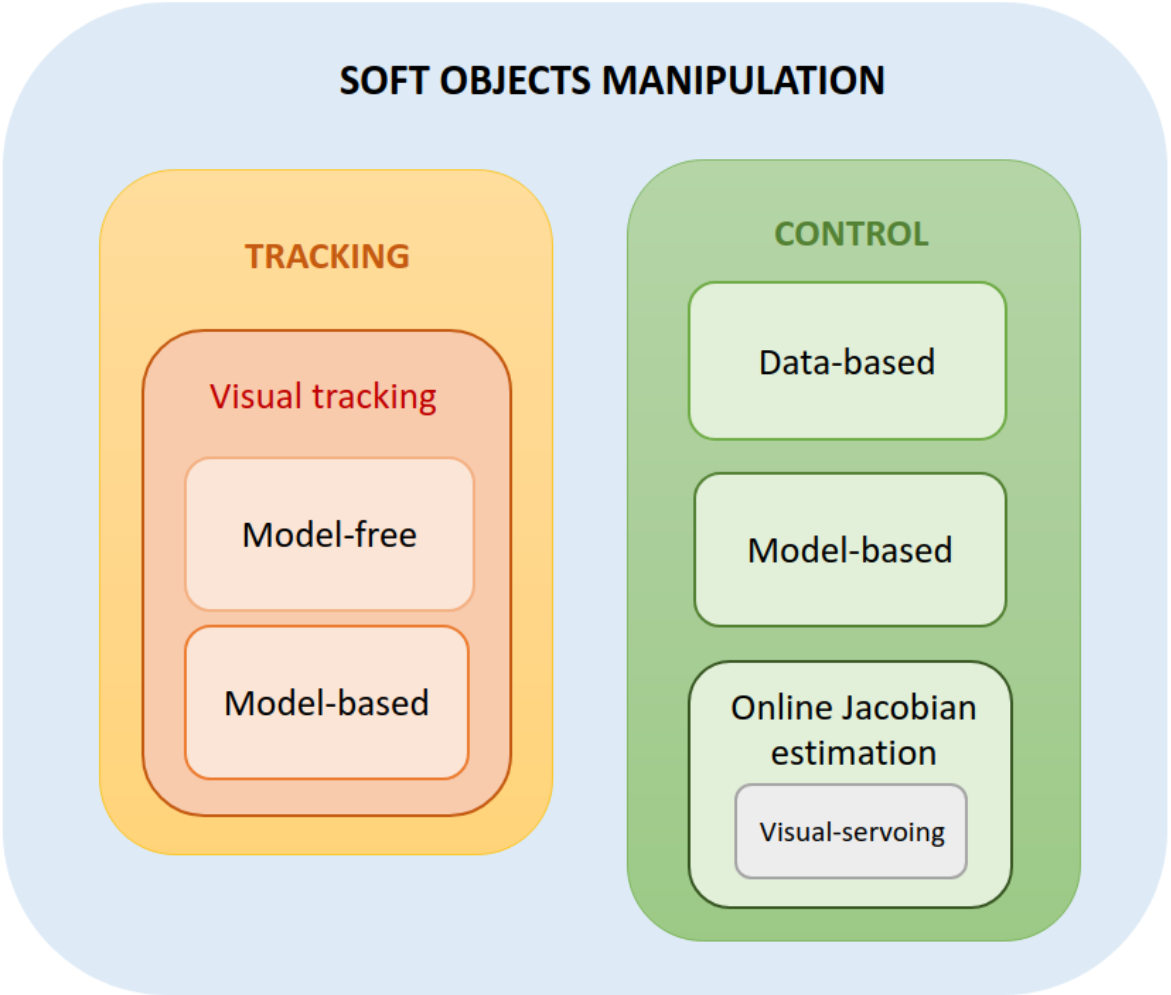


Figure 1.1: Categorization of the presented topics in the state of the art with regards to Robotic manipulation of soft objects.

This thesis aims at studying and developing tools, to both track and control 3D deformation of different soft objects with robotic arms. To that end, we start by presenting a general overview of relatively recent works made in the different topics at stake. We first introduce works related to visual tracking of deformation, then ones related to control of soft objects, with methods which can be either model-based or data-based. We also investigate how visual-servoing can be applied to deformable

objects manipulation in some notable works. Finally, we discuss the position of our work with regard to the literature.

1.1 Visual tracking of deformation

The existing literature for pose (position and orientation) tracking of rigid objects is expansive. The Iterative Closest Point (ICP), first introduced in [BM92] and later developed in [RL01], is an optimization method which aims at minimizing the distance between two point clouds (a source and a target one). The registration consists mainly of two steps: matching the points to the closest geometric entity (curve, surface, plane, point), then estimate the rigid transformation that best aligns the matches. The process is iterated until the resulting alignment is satisfying, and gives the final rigid transformation from the source point cloud to the target one.

David G. Lowe [Low99] first introduces the Scale-Invariant Invariant Transform (SIFT) algorithm in 1999 to answer correspondence problems between different views of an object in images. SIFT selects rotation, scale and illumination invariant features by applying Difference of Gaussians, and matches these features through descriptors generated from a histogram of the gradients. Speeded up robust features (SURF), introduced in [BTVG06], is another method for features detection and matching, based, this time, on square-shaped filters to build a scale-space representation. Once the features are extracted, descriptors are constructed for matching.

In 2004, the authors of [LPF04] estimate key feature points in images to construct a training set, and conduct points matching between two images by using statistical classification. The work [PR12] describes a pose estimation method based on maximizing the energy function, which is established from a probabilistic region-based separation of background and foreground pixels. Both works require 3D models of the tracked objects. Pose estimation using Neural Networks is presented in recent works such as [WZR+18] using semantic labeling, or [XSNF17] using only RGB images.

When it comes to soft objects, however, the issue of tracking objects and their shapes becomes more complex. While rigid objects can be defined by a non-changing geometry - edges, corners, curves - or a pose in space, the geometry of a soft objects varies during manipulation. This makes establishing the correspondences between shape features from one step to the other challenging.

1.1.1 Model-free tracking

Model-free methods for deformable objects tracking are methods that do not require a template or physical model for the object to track, and only relies on it's observed state.

In 2019, the authors of [CB19] present a tracking method from RGB-D data. Their method is based on the Coherent Point Drift algorithm (CPD, first introduced in [MS10]), which is a probabilistic point set registration method. It uses Gaussian Mixture Models, to represent a point set and aims to fit them to a second point set while preserving a structural coherence. The work in [CB19] focuses on handling occlusions, by also considering topological consistency, adding a regularization step based on Locally Linear Embedding [RS00]. Although the presented method is template-less, the topological coherence is ensured with an initial connectivity model of the object composed of ordered vertices and edges (needed as input).

In [CPP12], the authors extract and monitor the shape deformations of soft objects from a video sequence, which they map to force measurements. They use a three-finger robotic hand to sense an object, so the hand fingertips' positions are associated to the contours of a deformed object, tracked in a series of images. The deformations are tracked through a grid, printed on the object surface. The authors then use Neural Networks to predict the deformation of the object when subject to interaction with the robotic hand. [HPC17] tracks the deformation of objects undergoing probing by a robotic hand, by using level set geometric contour representation [SK08].

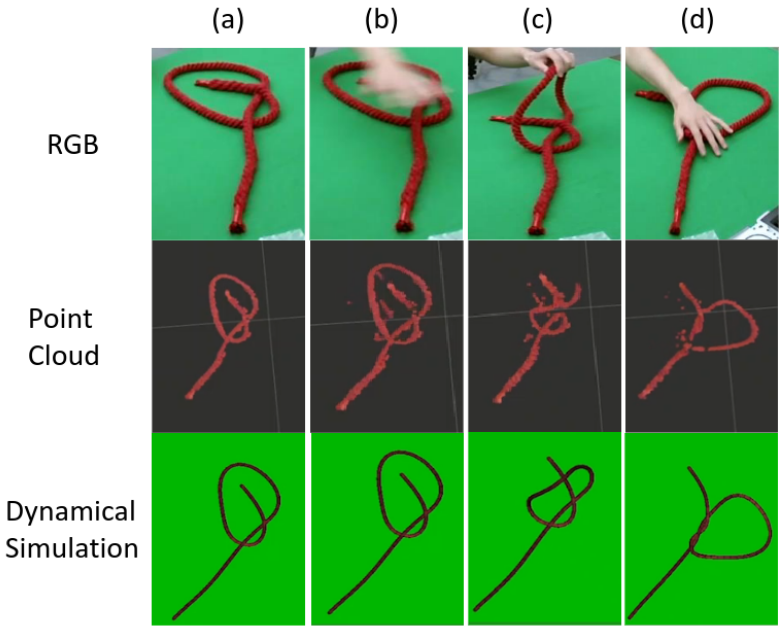


Figure 1.2: Examples of DLO tracking through occlusions - Figure from [JWZ+20].

The work [JWZ+20] presents a closed-loop framework to track DLO, like ropes, especially in case of occlusions. They first complete the point cloud in frame t, using the foreground mask and the recovered point cloud in frame t-1 through image processing. The point cloud is then registered into nodes, using a Structure Preserved Registration algorithm as well as a cubic spline interpolation. Through a physic-based simulator and

a feedback linearization controller, the authors then simulate the nonlinear dynamical behavior of the DLO to stabilize the tracking procedure. Some results of the tracking method are shown in Fig.1.2.

The paper [FNT+11] presents a dual-grid free-form deformation (FFD) framework, which consists in deforming a source shape to the target shape by rigidly aligning characteristic features and subsequently forming a free-form deformation grid. To represent a shape and the space around the surfaces, the authors use the Signed Distance Field (SDF), as well as free-form deformation. The source shape is surrounded by a control point grid, permitting to model the spatial deformation needed to match both surfaces (source and target). A second grid is used for sampling: it is superimposed over the FFD control point grid. This sampling grid is subdivided into sampling regions to enclose each control point, and it is used to find the optimal translation - the one that minimizes the difference between the SDFs in the corresponding sampling region - of the control point within this sampling region. The control points are then translated accordingly, and their new positions represent the deformation.
[YGL+19] proposes a non-rigid registration method for large deformations, using a set of RGB-D scans. The authors perform a coarse-to-fine multi-resolution scheme, to compute the deformation of multiple scans simultaneously, optimizing a global alignment problem with an as-rigid-as-possible constraint.

A local-to-global hierarchical optimization framework is proposed in [WZX17] to non-rigidly register partial RGB-D feedback with dynamic motions. Tracking the deformation from one instant to the other is done through a deformation graph, on which nodes are uniformly sampled throughout the surface, and neighboring nodes are connected with edges. The authors define the distance between a point on a source scan $\mathbf{p}$ and target scan $\mathbf{v}$ as:

$$d(\mathbf{p}, \mathbf{v}) = \max(1 - \frac{||\mathbf{p} - \mathbf{v}||}{d_{max}}, 0) * \max(\mathbf{n}_p * \mathbf{n}_v, 0) \tag{1.1}$$

With $\mathbf{n}_p$ and $\mathbf{n}_v$ the points normal, and d_{max} a limit distance. For each point $\mathbf{p}$, the point $\tilde{\mathbf{v}}$ from source scan that maximizes d - to account for large deformations - is found; a correspondence then yields $d(\mathbf{p}, \tilde{\mathbf{v}}) > 0$.

Digital Volume Correlation (DVC) was first introduced in [BSFS99] as an extension of a known image analysis in 2D (Digital Image Correlation, DIC) to RGB-D data. It is a method used to track deformations using image correlation, and it allows to obtain complete 3D displacements and strain maps. The main attribute of the method is that it requires the material to have a high contrast random pattern, acting somewhat as deformation markers. Some works use Fourier-transform methods [BKTA+15], or spline interpolation [GLH11] to solve a minimization problem and compute the correlation between two sub-images. This method gives great 3D results for local displacements or deformation, but does require highly accurate imaging, usually by using stereo cameras,

which is difficult to attain with usual RGB-D cameras, see for [BJM+18] an extensive review.

Many different algorithms propose solutions for template-less non-rigid registration. One of the main difficulties of deformation tracking is maintaining topological coherence throughout the registration process; while some works deal with this issue by using specific geometries (e.g., DLO or grids) or via constraints added in the optimization problem, this issue can also be solved by the use of an input model (i.e. a mesh of the tracked object).

1.1.2 Model-based tracking

We now present different works using models to track the deformation of soft objects. A model refers to:

- a *geometric* model - that is, a template or topology;
- a *physical* model that contains the equations and different dynamics parameters that rule over the object's behavior. These equations are solved through integration schemes or simulation as the manipulation goes.

In particular, we introduce some tracking schemes using the Finite Elements Method, which will be of interest in our work.

Finite Element Method (FEM) is a modeling approach based on continuum mechanics. It is used on meshes, a set of points (nodes) and connectivity between these points (elements) which describe the topology of the considered object. The method aims to interpolate the displacement of the nodes to approximate the displacement of an element of the mesh.

The authors of [PPP17] combine DIC with FEM in what is called a FE-Stereo-DIC. Considering an initial mesh plate, they use DIC to calibrate it with the 3D points of the plate object extracted from the stereo cameras. The mesh is then corrected to match the points of the stereoscopic image pair. For two pairs of images of the object (an initial and a deformed image), the 3D displacement of each node of the mesh can then be measured.

The authors of [PLS15] also propose a tracking method based on FEM. Knowing the 3D volumetric mesh of the object and the material properties of the object, the authors match the visible nodes of the undeformed mesh, with the point cloud of the currently deformed object (extracted from RGB-D data). They do so first, rigidly through ICP; then, K-Nearest Neighbors (KNN) algorithm is used to find the correspondences, on one hand, of the mesh nodes to the point cloud, and on another hand, of the point cloud to the mesh nodes. Using this pair of correspondences, that are meanwhile weighted, the elastic forces $\mathbf{F}$ are computed, as shown in Fig. 1.3.

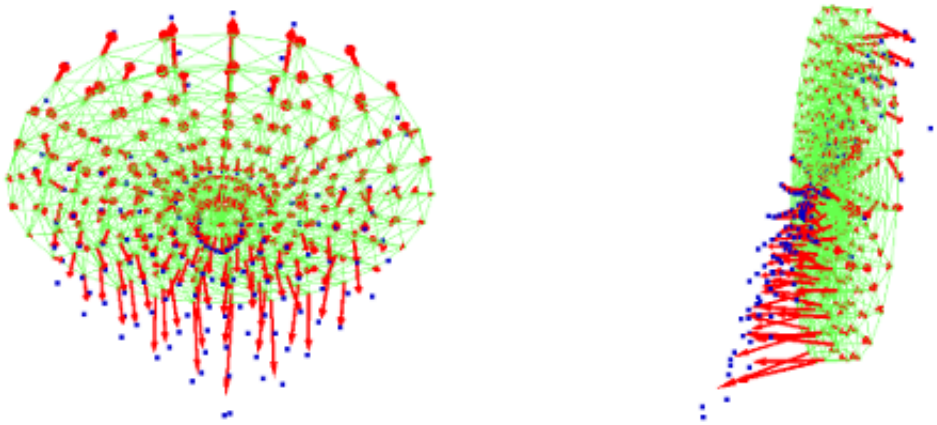


Figure 1.3: Example of the computed external forces applied on the mesh (green) to align it with the point cloud (blue) - Figure from [PLS15].

Using FEM modeling based on co-rotational linear elasticity, at step t_i the displaced mesh nodes $\mathbf{m}(t_i)$ are computed through:

$$\mathbf{F}(t_i) = \mathbf{R}(t_i)\mathbf{K}(t_i)[\mathbf{R}(t_i)^{-1}\mathbf{m}(t_i) - \mathbf{m}(t_{i-1})] \tag{1.2}$$

with $textbf{}R(t_i)$ the rotation matrix output by ICP, $\mathbf{K}(t_i)$ the stiffness matrix that is computed using the material parameters and $\mathbf{m}(t_{i-1})$ the mesh nodes at the previous iteration. The authors expand their work to address collision between non-rigid objects in [PCLS18].

Similarly, the authors of [SKM19] use a similar method for visual tracking, taking into account external forces, including a closed loop minimization technique used to compute the Jacobian matrices relating the displacement of the nodes with the external forces applied on it. This permits to relax the need for precise knowledge of the material parameters.

Instead of using FEM, [ZNI+14] minimize the deformation energy, following a thin shell deformation model using a Conjugate Gradient solver. In this work, the authors present a template-based tracking method, as well as an algorithm for online template acquisition. In their framework, the template is first reconstructed using a stereo matching algorithm, which fuses multiple view RGB-D data with a 3D model, as well as a mesh hierarchy (meshes with different fineness). For the tracking part, each new frame is first rigidly registered to the smooth template, then non-rigidly registered to the meshes form coarser to finer. Non-rigid registration is done by minimizing a fitting energy, composed of dense geometric and lighting constraints in addition to an as-rigid-as-possible (ARAP, [SA07]) regularizer; this fitting energy is minimized with a Gauss-Newton solver. Details are finally fused onto the final mesh using a linear

deformation model (thin-plate spline regularizer).

[SLHA13] integrates a probabilistic model to infer the correspondences between physical model of the object (the mesh of n nodes) and its observation (point cloud **pc** of p points), to then estimate the deformed state of the mesh. The authors use the Expectation Maximization algorithm, to find the most probable nodes position, for each point cloud, by maximizing the probability $\mathcal{P}$:

$$\arg\max_{\mathbf{m}_{1:n}} \mathcal{P}(\mathbf{m}_{1:n}|\mathbf{pc}_{1:p}). \tag{1.3}$$

Once the correspondences are estimated, the corresponding forces are deduced, based on a Mass-Spring Model, and applied to the nodes through simulations, to obtain the new positions of the physical model. [LAAB14] also feeds information processed from RGB-D data into a Mass-Spring Model to simulate the physical behavior of the deformable object. In a similar way, the authors of [WWY+15] also apply a probabilistic model, but instead use FEM simulations for a linear-elastic model.

The presented regularization of the external force allows for online estimation of the material parameters.

Other works also use initial templates, but performs registration without a mechanical model. In [CBI10], the first reconstructed mesh is deformed to fit point clouds resulting from the observation of motion, while preserving the local rigidity with respect to the reference pose. Rigid motions around control surface points are locally averaged, to guide the deformation of the mesh. These rigid motions are computed in a manner similar to the ICP algorithm: given a source mesh and target data, point-to-point associations are iteratively re-established, until the error is smalle "enough". In [IZN+16], the authors represent the object with a mesh and a deformation field: a deformed 3D surface mesh is created according to the RGB-D data, representing the current state of the object. Depth correspondences with the non-deformed mesh are computed, as well as SIFT features in the current frame and in all past frames, to further improve the correspondence. The deformation which aligns the correspondences is computed through a non-linear optimizer, which takes the deformation energy $\mathcal{E}_{defo}$ as objective function:

$$\mathcal{E}_{defo}(\mathbf{X}) = w_s\mathcal{E}_{sparse}(\mathbf{X}) + w_d\mathcal{E}_{dense}(\mathbf{X}) + w_r\mathcal{E}_{reg}(\mathbf{X}). \tag{1.4}$$

In this equation: the unknown 3D local deformations are stacked in $\mathbf{X}$, $\mathcal{E}_{sparse}(\mathbf{X})$ is the alignment objective concerning the SIFT features correspondences, $\mathcal{E}_{dense}(\mathbf{X})$ concerns the depth correspondences, $\mathcal{E}_{reg}(\mathbf{X})$ is a regularization term, w_s, w_d, w_r are weights. The overall tracking process is illustrated in Figure 1.4.

The authors of [AFB15] consider general shape tessellations, to track shapes over temporal sequences. The authors consider Centroidal Voronoi tessellation cells, in

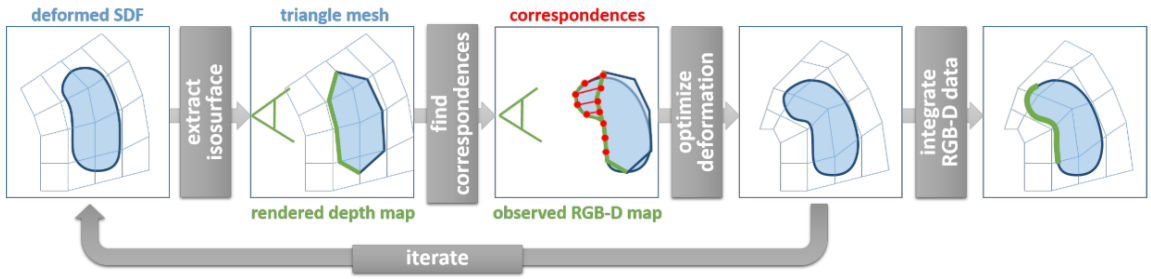


Figure 1.4: Workflow presented in [IZN+16] - Figure from [IZN+16].

which the center of mass of the cell and its centroid coincide. They build both volumetric deformation and observation models, then formulate the tracking problem as the MAP (Maximum A Posteriori probability) estimation of multiple poses of a given geometric template model. [HZSP18] uses a parse deformation graph model [SSP07] to estimate an inter-frame deformation model between a reference shape model and the observations provided by an RGB-D camera. The authors also iteratively reconstruct the model, gradually completing and refining its details, by integrating multiple RGB-D images into the reference shape model.

1.1.3 Conclusions

These works show that many methods using markers, sequences of images, physical models or neural networks can be used in order to track and establish correspondences between different states of one object. Among all methods, model-based ones often rely on a similar workflow: acquisition of a new visual representation, correspondences estimation (ICP, nearest neighbors, SIFT, probabilistic models), resulting deformation computation and finally template correction.

1.2 Control methods for soft object manipulation

While some works favor physics-based modeling for accuracy in the prediction of the behavior of soft objects, other use the advantages of data-driven methods to track and control the deformation of a wide range of objects, without knowing much about their behavior, beforehand.

1.2.1 Data-based control

Among recent works on soft objects manipulation, many choose to use data-based learning approaches such as Deep Neural Networks (DNN) or Reinforcement Learning (RL) to encode relevant information and to learn the behavior of the object when subject to robot manipulation.

In [LLG+15], the authors use the Thin plate spline robust point matching algorithm ([CR03]) to register the initial point cloud. Subsequently, a variable impedance control strategy is learned through demonstration (using either tele-operation or kinesthetic teaching). [TWT18] proposes a full framework for dual arm DLO (Deformable Linear Objects) manipulation using, including state estimation, task planning and trajectory planning. The state estimation is done using the Coherent Point Drift algorithm (similarly to [CB19]) to determine the correspondences between iterations. For trajectory planning, the authors also rely on a learning from demonstration approach. [SFP+19] also proposes two predictive control algorithms (reinforcement learning and learning from demonstration) to manipulate tissues with surgical robots.

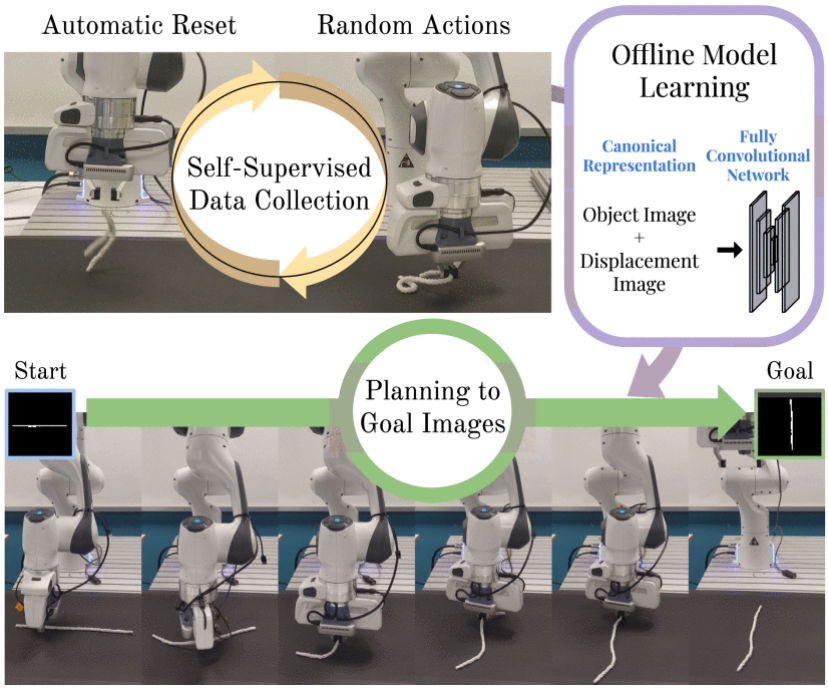


Figure 1.5: Overview of the framework proposed in [LHM+22]: self-supervised data collection, offline training and action selection - Figure from [LHM+22].

In [LHM+22], the robot controller learns to manipulate DLO through self-supervision. The authors propose an image-based prediction system, which utilizes Fully-Convolutional Neural Networks (FCNNs). Random pick-and-place actions are first performed in a self-supervised manner, to collect a data-set (object images and action images). The displacement images are made of a picking point, a placing point, and a vector representing the displacement from one to the other; for each pair of images, both are translated so that the picking point is at the center. The action image is rotated so that the displacement vector aligns with the horizontal axis, and the rotation is reported to the object image. The model is trained offline with the collected data-set and the goals to reach are determined through a predicted image: the action which minimizes the cost function is selected. The selected image-based cost function $\mathbb{C}$ is defined, with $\mathbf{p}^o$ and $\mathbf{p}^g$ pixels in the observation image and goal image respectively, as:

$$\mathbb{C}(o) = \sum_j \sum_i |\mathbf{p}_{i,j}^g - \mathbf{p}_{i,j}^o|^2. \tag{1.5}$$

This method allows to select the action which results in the predicted image state that is closest to the goal state. The framework is illustrated in Fig.1.5.

In an approach that is similar to [LHM+22], [WKL+19] learn to generate a sequence of images for trajectory planning, in order to reach a target shape. In this work, the authors separate the problem into visual planning (using a deep generative model to generate a plan) and control computation (learning inverse models from observations using deep CNN). The system is trained from self-supervised data.

[DZAL+22] also aims to learn to manipulate DLO, this time using Deep Reinforcement Learning (DRL). The DLO is described by a mesh, and the framework aims to generate the robot motion which leads a few selected mesh nodes to a desired position. The reward function is taken as the average Euclidean distance between the current and desired positions of the mesh nodes. The training phase is conducted through simulations, using FEM computation to simulate the deformation of the mesh.

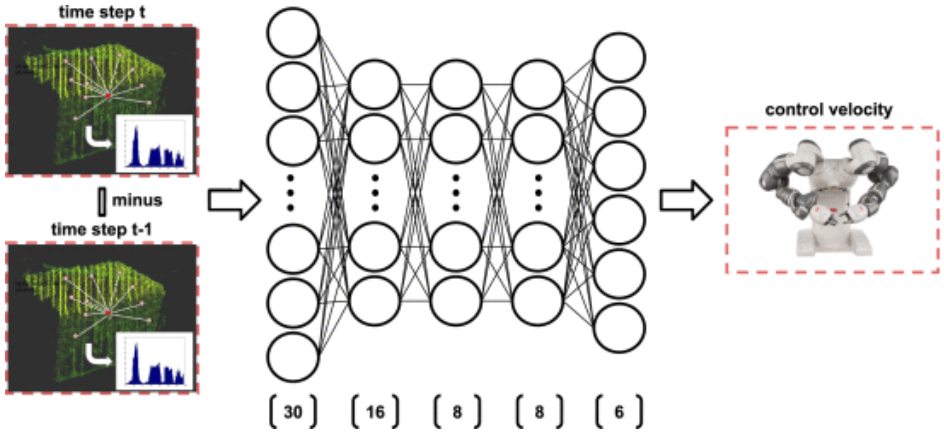


Figure 1.6: Architecture of the designed Neural Network, with the number of neural units for each layer - Figure from [HHS+19].

[TCUM19] presents an approach using RL to manipulate clothes with a humanoid robot. To alleviate the need for heavy data-sets imposed by regular RL methods, the authors proposes a dual DRL algorithm which allows the learning process to be more efficient, thus reducing the number of learning samples needed. The authors apply their method on a humanoid robot, conducting grasp and release actions to manipulate cloth-like objects.

The authors of [HHS+19] encode the state of the deformable object, using a fixed-length feature based on the Fast Point Feature Histogram [RBB09], extended by using Principal Component Analysis (PCA) to extract features from the observed object point cloud. Their controller then uses a DNN to map the end-effectors motion to

the object deformation. Figure 1.6 gives an overview of the framework. The neural network in composed of 5 layers and it is trained with Mean Square Error (MSE) as loss function. The input of the neural network is the velocity of the features, obtained by subtracting the feature resulting from the PCA between two iterations.

1.2.2 Model estimation and model-based control

In model-based methods, a model of the object (both geometric and physic) is used to predict its deformation in response to the external forces applied to it.

To be ale to use a model in a control scheme, it is necessary to have knowledge on the object's mechanical properties. In some papers, robotic manipulation is then used to *estimate* a model, or the mechanical parameters proper to the object being manipulated that are usually needed to be able to implement model-based control.

Iterative methods aim at decreasing an error function step by step. This function usually depends on both a chosen behavior model and the observation of the object, as in [FJP+12], where a physical model and RGB-D data are combined in a joint error function, to estimate the material parameters of objects deforming under gravity. [WMC+20] aims at optimizing a Lagrangian-Eulerian formulation, to solve the inverse elasticity problem. The physical behavior of the object subject to deformation is explained through the stress-train relationship, which the authors describe with the Piola-Kirchoff stress tensor:

$$\boldsymbol{\sigma} = 2\mu\boldsymbol{\epsilon} + \lambda tr(\boldsymbol{\epsilon})\mathbb{I}. \tag{1.6}$$

In this equation, $\boldsymbol{\sigma}$ is the stress, $\boldsymbol{\epsilon}$ is the strain depending on the displacement $\mathbf{u}$, μ and λ are the Lamé coefficients, and can directly be linked to the Poisson ration and Young Modulus through:

$$\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)}, \mu = \frac{E}{2(1+\nu)}. \tag{1.7}$$

[GPIK17] estimates a parameter which determines the degree of deformability of a material, using a position-based dynamics model along with a FEM model. [AMN18] conducts force-based model estimation, in addition to 3D shape and pose recovery from 2D observations, by using a Probabilistic PCA formulation to learn the elastic model.

Once a model is chosen or estimated, it can be used to describe the behavior of an object in relation to the robot motions in a control loop.

The authors of [MB18] choose to design tasks with the multiarmed bandit method. The model of each arm acting on the object is used to compute the corresponding command for the robot, and the reward is given in terms of utility - the reduction of task error - for the considered task.

Instead of relying on physical simulations, the authors of [RMB18] present a framework which constructs a geometry-based model to predict the motions of cloth-like

objects and DLO given the motion of the end-effectors. Their model estimation is based on geometric information like the positions of the grippers, their motion, or obstacles. The authors formulate these constraints as an optimization problem which they solve using gradient descent, with an additional projection step, to estimate the model. [APR+22] and [AALN+22] model DLO behavior in a 2D workspace using an ARAP deformation model.

Some works use Finite Element modeling to describe the behavior of the manipulated objects. The method presented in [FMC+18] uses inverse FEM modeling to compute the motions of three fingers to deform the object to a target shape. Using both visual and force feedback, they first estimate the material parameters E and ν for the FEM model, based on a vision-based offline calibration technique presented in [PLFS17]: the approach consists in minimizing the error between the deformations in simulation *sim* which depends on the mesh of the object $\mathbf{m}$ and the material parameters, and the ones observed by RGB-D camera feedback defined by the point cloud $\mathbf{pc}$. The error is defined as:

$$e(E, \nu) = dist(sim(E, \nu, \mathbf{m}), \mathbf{pc}) \tag{1.8}$$

with E the Young modulus, and ν the Poisson coefficient, used to build the stiffness matrix $\mathbf{K}$ for linear elastic materials. The error minimization problem is solved using a gradient-free Nelder-Mead method. Lagrange multipliers are used to model the forces applied by the finger tips. These are then related to the displacement of the mesh nodes $\mathbf{u}$ through Hooke's law under the assumption of infinitesimal strain:

$$\boldsymbol{F} = \boldsymbol{K}\boldsymbol{u}. \tag{1.9}$$

The inverse FEM problem is then solved for a desired deformation, to obtain the force to be applied by the fingertips in order to reach it.

The authors of [DBPC18] propose a simulation-based control scheme; they model a deformation energy with FEM and sensitivity analysis, as well as a gripping energy, which they aim at minimizing. The deformation of the object is mapped to the joint angle commands, and the model takes into account both collisions and joint limits. The simulations couple the joint representation of the robot with the FE model of the object, as shown in Fig. 1.7, so the total energy of the system $\mathcal{E}$ is minimized through the following formulation:

$$\mathbf{m}(\boldsymbol{\theta}) = \arg\min_{\mathbf{m}} \mathcal{E}(\mathbf{m}, \boldsymbol{\theta}) \tag{1.10}$$

with $\boldsymbol{\theta}$ the robot joint angle.

[ZPC21] proposes a FEM simulation-based trajectory for dual arm manipulation of soft objects. The authors adapt single shooting trajectory optimization strategies to simulations forwarded in time, by using implicit integration schemes, in order to execute task-focused (laying clothes, whipping) trajectories in an open-loop manner. Fig.1.8 presents results of the simulated state of the object in comparison with the real

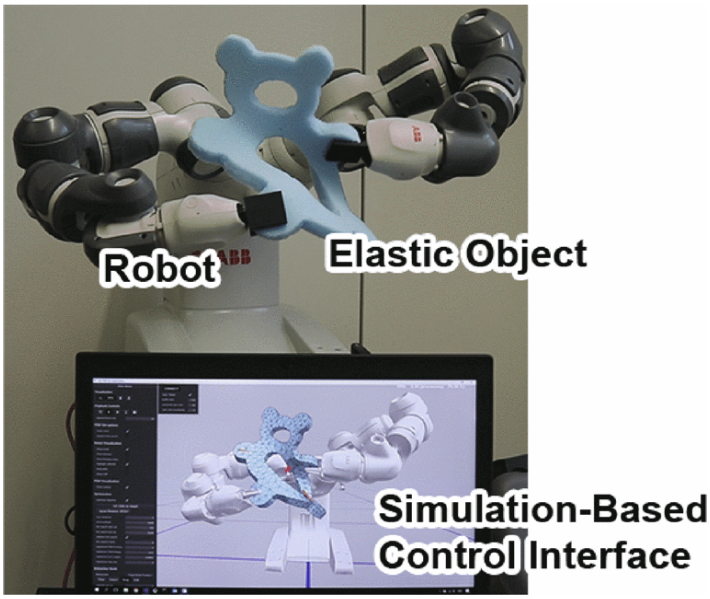


Figure 1.7: Interactive setup to manipulate a soft object via simulations - Figure from [DBPC18].

behavior of the object, observed via visual feedback. A finite element formulation is also used in [KFB+21] for shape control, and to remedy for the lack of real-time simulations.

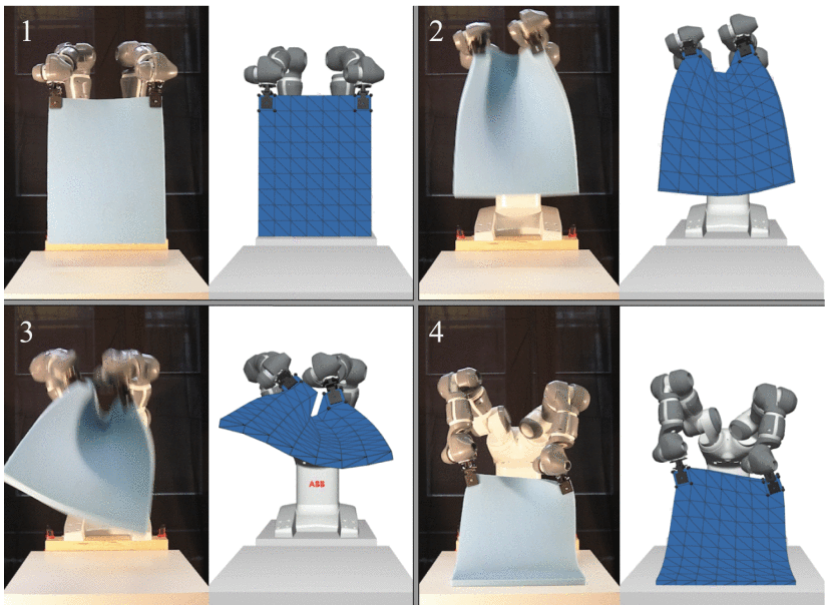


Figure 1.8: Comparison of the simulations and visual feedback from [ZPC21] - Figure from [ZPC21].

1.2.3 Conclusions

Data-based and model-based methods are two different ways of mapping robot motions to the objects' behavior. Both methods have their benefits and drawbacks. Data-based methods require training for each new object. Hence, some model-based methods can be extended more easily than data-based methods, to different objects (in view of approximating some of the material's characteristics, e.g., linear elasticity). Yet, in some cases the model assumptions (again, consider linear elasticity) may be less accurate in describing deformation, or the material parameters can be unknown.

1.3 Visual servoing for soft objects

Visual servoing techniques aim at controlling a dynamic system using visual features extracted from visual data. Unlike in data-driven approaches, the principle is to estimate *online* a mapping between robot inputs and visual features with *locally* collected data, thus estimating an approximate model.

For rigid objects, the basics of visual servoing are exposed in [HHC96],[Cha07]. Visual servoing control schemes aim at minimizing an error $\mathbf{e}(t)$, typically defined by:

$$\mathbf{e}(t) = \mathbf{s} - \mathbf{s}^* \tag{1.11}$$

where $\mathbf{s}$ is a vector of visual features obtained from image measurements, and $\mathbf{s}^*$ represents a constant, motionless target shape. This control scheme is notably used for tasks such as tracking a face by controlling the pan-tilt of a camera [YMH17] or positioning an object to a desired pose with regards to a robot [KGD+03].

Visual servoing control loop The principle is to select k_s visual features $\mathbf{s}$ to control the k_f DOF of the system through a Jacobian matrix, referred to as the "interaction matrix" $\mathbf{L}$, such that:

$$\dot{\mathbf{s}} = \mathbf{L}\mathbf{v}, \tag{1.12}$$

with $\mathbf{v}$ the velocity input to the robot controller. To try to reach a desired shape described by the visual features $\mathbf{s}^*$ and ensure an exponential decoupled decrease of the error $\dot{\mathbf{s}} = -\lambda(\mathbf{s} - \mathbf{s}^*)$, one can then use:

$$\mathbf{v} = -\lambda \mathbf{L}^+(\mathbf{s} - \mathbf{s}^*) \tag{1.13}$$

with $\mathbf{L}^+$ the Moore-Penrose pseudoinverse of $\mathbf{L}$. This yields the closed-loop system shown in Fig.1.9 and summarized by:

$$\dot{\mathbf{s}} = -\lambda \mathbf{L}\mathbf{L}^+(\mathbf{s} - \mathbf{s}^*) \tag{1.14}$$

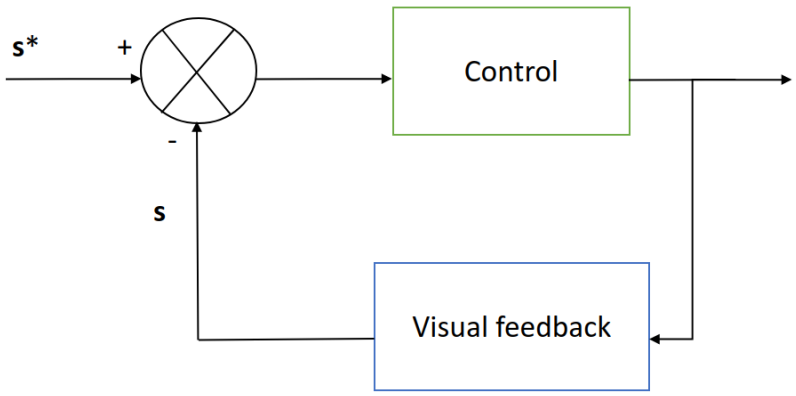


Figure 1.9: Visual servoing closed-loop system.

This control method can be extended to non-rigid objects. The main problem is to choose relevant features to describe deformable objects.

In [NAYW+16], an adaptive deformation model for elastic materials is estimated with regards to feature points of interest. The displacements of the feature points $\delta\mathbf{s}$ and of the robots position $\delta\mathbf{r}$ from the equilibrium is expressed as:

$$\delta\mathbf{s}_i = \mathbf{D}_i\delta\mathbf{r}, \tag{1.15}$$

with $\delta\mathbf{r}$ the stacked displacement of all the end-effectors, and $\mathbf{D}_i$ a deformation matrix. The authors construct a vector of deformation parameters, which iteratively approximates the deformation model, is estimated using a gradient descent, and relates to the deformation features through a Jacobian deformation matrix. The deformation features vector contains both the position of the features $\mathbf{s}$ and shape information; in the end, for a desired deformation features vector, it is then possible to control $\dot{\mathbf{r}}$ to minimize the error between the current and the desired deformation features.

[NAL18] presents a method for Fourier-based shape servoing, by encoding the shape of the object through Fourier coefficients, and iteratively estimating a model, to deform it. If $\mathbf{G}(\alpha)$ is the Fourier series used to approximate the object contour $\mathbf{c}(\alpha)$, according to parameter α, then the vector of shape features $\mathbf{s}$ is computed as:

$$\mathbf{s} = (\mathbf{G}^T\mathbf{G})^{-1}\mathbf{G}^T\mathbf{c} \tag{1.16}$$

with $\mathbf{G}$ a regression-like matrix. Similarly, [ZNF+18] uses Fourier series to both characterize the shape of cables, and control a dual arm robot to shape the cable in the plane. The authors of [LKM20] also consider the manipulation of DLO (shaping of deformable wires) through shape-servoing, but in 3 dimensions. Their approach tracks the deformations and extracts visual features based on a geometric B-spline model. The deformation feature vector $\mathbf{s}$ used in the control law is composed of a desired number of equidistant B-spline points.

The authors of [JHPM18] present a visual servoing method for manipulating cloth-like objects with two end-effectors. They present a novel feature representation for such objects, which is a Histogram of Oriented Wrinkles, resulting from simple image processing and filtering. Their framework uses a dictionary storing pre-computed visual feedback to map the shape variations with the velocities of the end-effectors. [HSP18] presents a Gaussian Process Regression to model and learn the deformation function of a soft object, based on a spring model. This approach allows to adaptively learn a nonlinear deformation function, along with the manipulation process. The authors model a soft object using three classes of points: manipulated points, feedback points, and uninformative points. Model??

The authors of [QMZ+22] use contour moments as a state representation of the manipulated object. For a contour $\mathbf{c}$, the contour moment of order $i+j$ is defined as:

$$h_{ij} = \sum_{k=1}^{p} u_i^k v_j^k \mid\mid \mathbf{c}^k - \mathbf{c}^{k-1} \mid\mid \tag{1.17}$$

where u_i and v_i represent the pixel coordinates of the ith point in the image frame, and $\mid\mid \mathbf{c}_k - \mathbf{c}_{k-1} \mid\mid$ represents the distance between two adjacent pixels on the contour. Their method is applicable for elastic, composite and rigid objects, manipulated in 2D, as shown in 1.10. [Ber13] uses the concept of diminishing rigidity to compute an approximation to the Jacobian of the deformable object, with regards to the gripper motion.

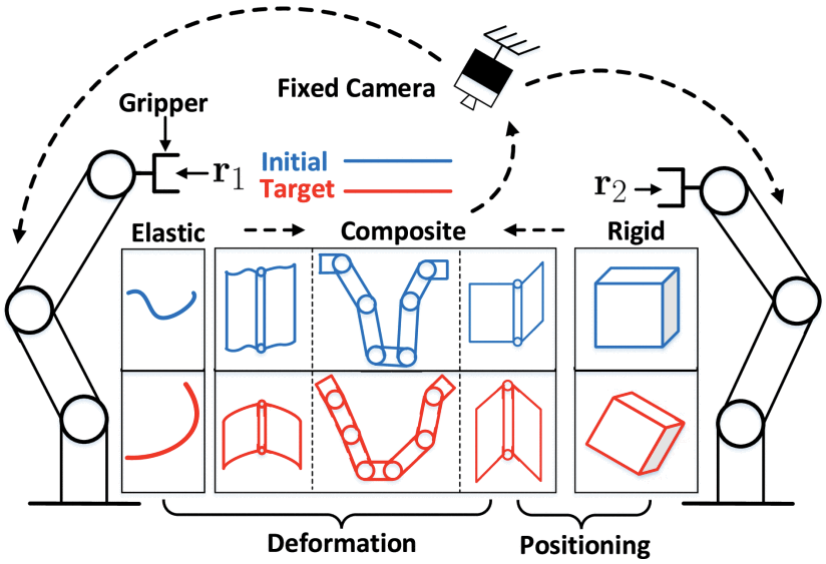


Figure 1.10: Representation of the type of objects defined as elastic, rigid and composite, on which dual-arm manipulation tasks include deforming and positioning until reaching a target configuration - Figure from [QMZ+22].

[AACRM+20] uses monocular 2D images for perception, coupled with a template to shape isometrically deforming objects. The template contains the object's texture map, rest shape and deformation law. It is used both to track the shape thanks to the

Sft algorithm from [BGC+15], and to control the robot motion. Sft proposes a solution for shape reconstruction of a deformable surface from one image and a 3D template, by using the image point locations as well as these point's first-order differential structure. The authors of [AACRM+20] improve the algorithm, to have it track and generate intermediate feasible targets towards the final target, and to control the end-effectors to target poses, as shown in Fig. 1.11. These target poses are extracted from patches on the intermediate target template, to which rigid particles are then aligned - since the patches deform with the template. The aligned rigid particles then define the target positions and orientations of the end-effectors.

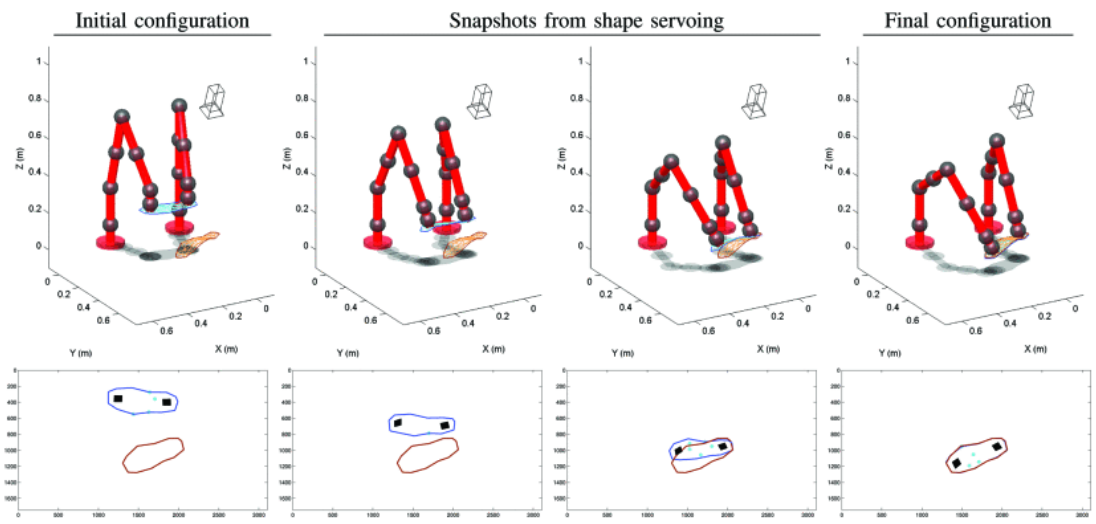


Figure 1.11: Shape servoing experiment from [AACRM+20]. The target shape is represented in brown, the current shape in blue. The black squares represent the target poses of the end-effectors, to which rigid particles are aligned - Figure from [AACRM+20]

The visual-servoing framework presented in [SBAMO22b] uses a 3D lattice representation formed around the manipulated object. Lattice and object are bound together by geometrical constraints. The authors' framework then aims at controlling the deformation of the lattice. The deformation of the lattice is modeled according to an ARAP model (as in [APR+22], [AALN+22]), which is also used to compute the interaction matrix. The authors had introduced this model in [SBAMO22a]. It is computed via numerical differentiation, to control the shape of thin-shell objects. In [SBAMO22b], they present an analytic expression of the ARAP-based interaction matrix.

The authors of [ZNAPC20] encode the object via Principal Component Analysis (PCA) on image contours of deformable/rigid objects. They present experiments for moving and deforming 2D contours with a single arm moving in the plane (3 DOF). Their method applies to both rigid and soft objects, allowing a large range of applications. They start by generating a sequence of small motions around the resting shape

of the object, in order to collect deformed contours and the corresponding end-effector poses. These data are used to estimate the local interaction matrix **L**, which is used to control the end-effector into reaching local targets, until the final target shape. The interaction matrix is re-estimated at each iteration, with the last contour variation and effector pose, so that the estimation stays local. At each iteration, PCA is applied to the matrix containing the past contour points, updated with new data on a sliding window basis - so that it keeps the same dimension as well as locality. PCA allows to select a few principal components which best express the contour variance, to encode the contours with a small number of features.

1.3.1 Conclusions

In robotics, control schemes relying on visual feedback can be used for manipulating rigid and soft objects like-wise. The main challenge with implementing visual-servoing for deforming object is to find a way to encode the shape of the object into features, which can be continually tracked throughout manipulation.

1.4 Conclusion and positioning of our work with respect to the literature

Overall, the presented overview of the state of art shows that shape-servoing for soft objects involves different issues:

- Choosing a relevant shape descriptor (according to the available sensors, the type of object, known template, etc.), which can be *tracked* over time and which is consistent;
- Mapping the robot commands to the shape descriptor (via off-line data-based training, by estimating a Jacobian online with local data, or according to a physical model);
- Closing the control loop with sensor feedback; when a template is used, for instance, the controller is fed with the template's current state, and not necessarily with sensor feedback. This template has to somehow be related to this feedback to ensure a link between model and observation.

All these aspects have to be taken into account depending on the application and resources. A trade-off between the collection of data and the estimation of an accurate model is crucial. Besides, the literature raises the question of the dimensionality of soft objects manipulation: another aspect to ponder about is the number of DOF to consider, be it for the description of the shape (contour, surface, volumetric model), for the number of robotic actuators used (single or multiple arms) or the dimension of the motions (on the plane or in the 3D space). The presented works for robotic manipulation of soft objects are summarized in Tab. 1.1.

	Manipulators		**Workspace**		**Control method**			**Specific objects**
	Single	Multiple	2D	3D	Visual servoing	Data-based	Model-based	
[LLG+15]		✓		✓		Demonstrations		
[TWT18]		✓		✓		Demonstrations		DLO
[SFP+19]		✓	✓			RL, demonstrations		Tissues
[LHM+22]	✓		✓			FCNN		DLO
[WKL+19]	✓		✓			DCNN		DLO
[DZAL+22]	✓			✓		DRL		DLO
[TCUM19]		✓	✓			DRL		Clothes
[HHS+19]		✓		✓		DNN		
[MB18]		✓		✓			MBM	
[RMB18]		✓		✓			Geometry-based	Clothes, DLO
[APR+22]	✓		✓				ARAP	DLO
[AALN+22]	✓		✓				ARAP	DLO
[FMC+18]		✓		✓			FEM	
[DBPC18]		✓		✓			FEM	
[ZPC21]		✓		✓			FEM	
[KFB+21]	✓		✓				FEM	DLO
[NAYW+16]		✓		✓	✓			
[NAL18]	✓			✓	✓			
[ZNF+18]		✓	✓		✓			DLO
[LKM20]		✓		✓	✓			DLO
[JHPM18]		✓		✓	✓			Clothes
[HSP18]		✓		✓	✓			
[QMZ+22]		✓	✓		✓			Clothes
[Ber13]		✓		✓	✓			
[AACRM+20]		✓		✓	✓			
[SBAMO22b]		✓		✓	✓		ARAP	
[ZNAPC20]	✓		✓		✓			

Table 1.1: Summary of the different works presented for soft objects manipulation

With this thesis, we aim at proposing a dual arm soft object shape-servoing framework uniting the benefits of data-driven and Jacobian estimation methods (not needing a precise physical model, learning as we manipulate), with those of model-based methods (no need of huge data-sets and training time, use of a consistent topology and ability to deal with large deformations) while ensuring to incorporate visual feedback into the control loop in order to relate the *estimated* behavior of the object compared to the *observed* one.

While some works focus on specific soft objects (i.e. DLO, clothes), we try to include different types of object for a more general framework. Finally, we aim to develop a control scheme for dual arm manipulation in $\mathbb{SE}(3)$, that takes into account the deformation of the whole object in 3D.

The next chapter presents a visual-servoing method for dual-arm control of the shape of soft objects in 3D. We build on top of the work [ZNAPC20], to expand the number of DOF controlled (3 to 12 DOFs), while implementing the controller in a cooperative task space, allowing the two arms of the robotic system to be controlled in a collaborative way.

CHAPTER 2

Visual-servoing for dual-arm shaping of soft objects using a cooperative task space

While many works in the literature present visual shape servoing controllers, many rely on model estimation, neural networks or reinforcement learning to compute the command. Although these methods present good results, the first one requires a prior knowledge of the object (template, material parameters) which limits its range of application, while the others require huge datasets and training time. Furthermore, some methods focus on specific objects, like DLO or fabrics, or rely only on single arm manipulation.

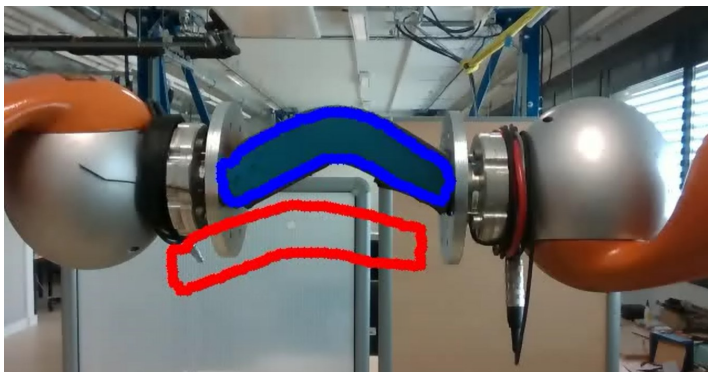


Figure 2.1: The goal of the work presented in this chapter is to control a dual arm robot so that the initial contour (blue) reaches the target contour (red) in three dimensions.

In this chapter, we propose a real-time, dual-arm vision-based shape servoing controller. Our method does not require any prior knowledge of the object, apart from

its color, needed for visual extraction. It applies to soft objects of different shapes and materials, with very short initialization time. In addition, our task representation allows a cooperative control of the two arms, to command 3D deformations and/or 3D motions of the tracked part of the object, represented by a 3D contour.

The authors of [ZNAPC20] obtain a good encoding of the object shape, via Principal Component Analysis (PCA). Their method shows very promising results for moving and deforming 2D contours with a single arm moving in the plane (3 DOF). It can be applied on both rigid and soft objects, allowing a large range of applications.

We build on top of that work, going farther and performing deformations in $\mathbb{SE}(3)$ with two *cooperating* robotic arms. Our contributions are the following:

- While [ZNAPC20] considered 3 DOF planar motion (two translations and one rotation) of a single arm, we increase the robot operational space dimension to 12 DOF (6 per arm), hence extending the range of applications.
- We apply the cooperative task representation [CCS96] to control the deformation and/or the pose of the object separately. We explore how both tasks affect the global deformation and the reaching of the target.
- We validate our controller in 3 dimensions (including orientation) in a series of experiments, with different objects and targets. We also consider 3D deformation of the objects.
- Application-wise, our framework can be useful in industrial contexts, where deforming objects (e.g., for insertion) requires robotic strength and an adaptable and reproducible technique.

2.1 Problem statement

We consider a dual-arm robot with both end-effectors holding an object. The dimension of the robot operational space is 12 (i.e., each end-effector is free to move in $\mathbb{SE}(3)$). A fixed RGB-D camera observes the object, used to extract its visual features in 3D. The goal is to modify the position and/or shape of the object into a target shape (a target 3D contour), shown in Fig. 2.1.

In some cases, the task may consist only in giving the object the desired shape, without caring about its position and orientation. For such a task the robot is *redundant*: it only requires 6 DOF, and can use the other 6 for other purposes. Yet, doing this is not trivial if each end-effector is modeled and controlled separately, as in [LKM20]. A solution is to use the cooperative task representation introduced in [CCS96] and outlined in Fig.2.2, to describe the task as the combination of an absolute task and/or a relative task. The absolute task frame $\mathcal{F}_{ctrl}^{abs}$ is attached to one of the arms chosen arbitrarily (the left one in our case) by a virtual link (dashed orange in

the figure) and it is described in the absolute reference frame $\mathcal{F}_{ref}^{abs}$, which is fixed in the world. The relative task describes the pose of one end-effector $\mathcal{F}_{ctrl}^{rel}$ in the frame of the other one $\mathcal{F}_{ref}^{rel}$. This representation allows to consider as relative task the object deformation, and as absolute task, the object's pose . Then, for operations which solely imply deformation or shaping, only the relative task matters, "freeing" the absolute task's 6 DOF. On the contrary, for rigid object manipulation consisting only in translating or orienting, the relative task's 6 DOF are "freed". More formally, two alternatives are possible ($k = 6$ or $k = 12$ DOF), as one can either control only one of the poses or both the relative and absolute poses:

$$\mathbf{r} = \mathbf{r}^{\mathbf{ref}} \in \mathbb{R}^6 \quad \text{or} \quad \mathbf{r} = \begin{bmatrix} \mathbf{r}^{\mathbf{abs}} \\ \mathbf{r}^{\mathbf{rel}} \end{bmatrix} \in \mathbb{R}^{12}. \tag{2.1}$$

with $\mathbf{ref} = \{\mathbf{abs}, \mathbf{rel}\}$. To represent orientations, we use angle-axis vectors $\mathbf{q}_r = [x_r \; y_r \; z_r]$, so $\mathbf{r}^{\mathbf{ref}}$ is defined as:

$$\mathbf{r}^{\mathbf{ref}} = \left[x^{\mathbf{ref}} \; y^{\mathbf{ref}} \; z^{\mathbf{ref}} \; x_r^{\mathbf{ref}} \; y_r^{\mathbf{ref}} \; z_r^{\mathbf{ref}}\right]^\top \in \mathbb{R}^6. \tag{2.2}$$

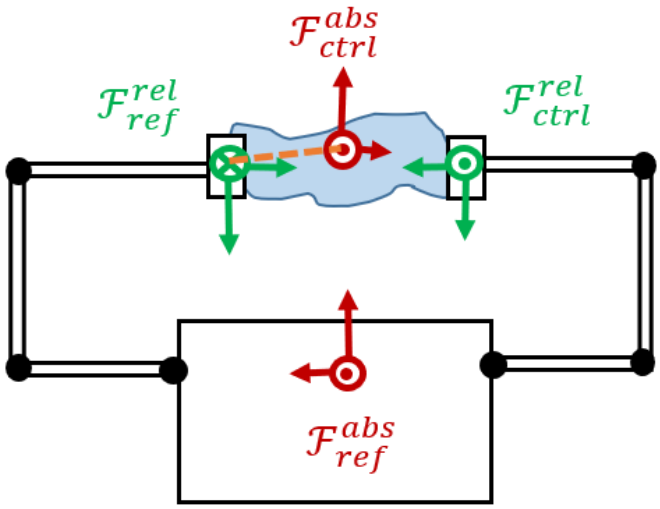


Figure 2.2: Schema of the task representation. The relative task (green) describes the motion of $\mathcal{F}_{ctrl}^{rel}$ in relation to the relative frame $\mathcal{F}_{ref}^{rel}$, i.e. the right end-effector in relation to the left one. The absolute task (red) describes the motion of $\mathcal{F}_{ctrl}^{abs}$, in relation to the absolute frame $\mathcal{F}_{ref}^{abs}$. The dashed line is the virtual link between the end-effector and $\mathcal{F}_{ctrl}^{abs}$.

Another important aspect is the perception of the object. In our work, the robot has no knowledge of the object's material or characteristics, except for its color (needed for visual extraction) and current shape. At each iteration, to represent the object shape, we use the contour seen by the RGB-D camera. We extract the target contour (to be reached) from one of few previously saved images of the same object, held by the robot

in various positions/shapes.

It is worth discussing the limits of the contour representation. Beside deformation instabilities (such as bucking, or privileged curvatures) which will cause problems, it will be impossible to apply any deformation out of the image plane or manage self-occlusions, which do not affect the object contour. Furthermore, the target contour is reachable if it does not differ "too much" from the initial contour, e.g., it should not require changing from convex to concave bending. We show a case of unreachable target in our experiments video, linked in Section 2.3.

We denote the visible contour, composed of the 3D metric coordinates of p points, $\mathbf{c} \in \mathbb{R}^{3p}$. Working with a task of such high dimension (300 if $p = 100$ points) makes it complex to control the largely lower number of DOF ($k = \{6, 12\}$) of our robot. To solve this underactuated problem, we encode the contour $\mathbf{c}$ into a smaller feature vector $\mathbf{s} \in \mathbb{R}^k$, with k the dimension of the required robot pose. Furthermore, we consider that the mapping between feature variation $\delta\mathbf{s}$ and robot pose variation $\delta\mathbf{r}$ is linear, through what we refer to as the interaction matrix $\mathbf{L} \in \mathbb{R}^{k\times k}$:

$$\delta\mathbf{s} = \mathbf{L}\delta\mathbf{r} \tag{2.3}$$

In the rest of the chapter, in addition to the hypothesis stated in the introduction chapter, we assume the following:

- The tracked part of the object stays entirely visible throughout the manipulation, and can be represented by closed contours $\mathbf{c}$.
- The object target shape and (when applicable) pose are reachable, i.e. the tracked part of the object can physically be deformed to (and placed at) the target.

Our goal is to drive (shape and/or place) the 3D contour of the object to a target 3D contour $\mathbf{c}^*$. Our framework (outlined in Fig. 2.3) operates as follows. First, we control the robot end-effectors (initially in open loop), to collect a sequence of $D + 1$ images. For each, the corresponding contour $\mathbf{c_i}$ is extracted and the robot pose $\mathbf{r_i}$ collected, with:

$$\mathbf{c}_i = [x_i^1, \cdots, x_i^p, y_i^1, \cdots, y_i^p, z_i^1, \cdots, z_i^p]^\top \in \mathbb{R}^{3p}, i = 0, ..., D + 1. \tag{2.4}$$

The collected contours are stored in the matrix $\mathbf{C}$:

$$\mathbf{C} = [\mathbf{c}_0, \cdots, \mathbf{c}_D] \in \mathbb{R}^{3p\times(D+1)} \tag{2.5}$$

while we use $\mathbf{r}$ to compute robot pose *variations* $\delta\mathbf{r}$, (see section 2.2.2) and store it in the matrix $\mathbf{\Delta R}$:

$$\mathbf{\Delta R} = \begin{bmatrix} \delta\mathbf{r}_1 & \dots & \delta\mathbf{r}_D \end{bmatrix} \in \mathbb{R}^{k\times D} \tag{2.6}$$

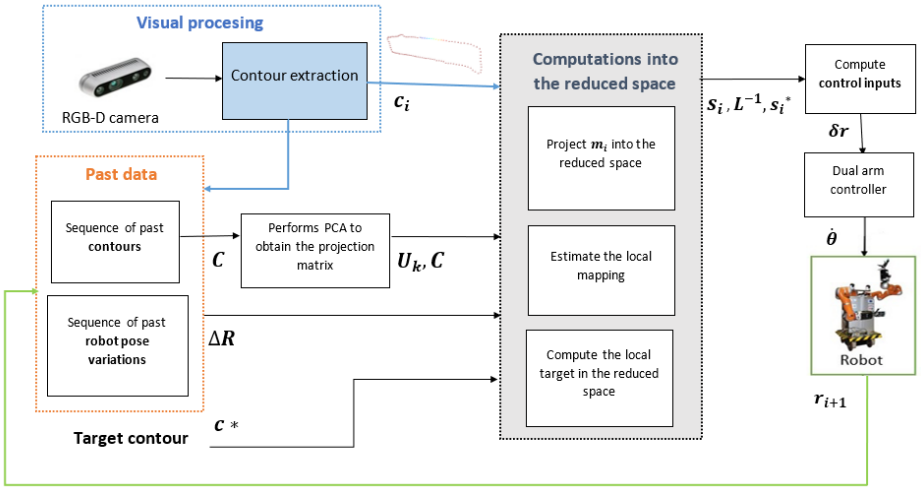


Figure 2.3: Overview of our framework. The robot poses and object contours are stored in matrices $\mathbf{\Delta R}$ *and* $\mathbf{C}$*, respectively. At each iteration, PCA yields the projection matrix* $\mathbf{U}$*, to compute the feature vector* $\mathbf{s}$ *from contour* $\mathbf{c}$*, and the local target* $\mathbf{s_i^*}$*. We also project the contour variation matrix* $\mathbf{C}$ *to estimate the inverse interaction matrix* $\mathbf{L}^{-1}$*. We use* $\mathbf{L}^{-1}$ *to compute the robot desired pose variations* $\delta\mathbf{r}$ *needed to drive* $\mathbf{s}$ *to* $\mathbf{s_i^*}$*. These* $\delta\mathbf{r}$ *are finally sent to the dual-arm controller, which computes the robot joint commands* $\dot{\boldsymbol{\theta}}$*.*

Then, we perform a PCA (Principal Component Analysis) on the sequence of contours $\mathbf{C}$, to encode the current contour $\mathbf{c}_i$ into a smaller k-dimensional feature vector $\mathbf{s}_i \in \mathbb{R}^k$, via projection operator $\mathbf{U}$. This reduces the task shape dimension to the number of required DOF.

We also use these sequences of object contours and corresponding robot poses, to estimate the inverse of the interaction matrix $\mathbf{L}$, which maps feature variations to robot pose variations according to (2.3)). Since linear mapping (2.3) is only valid locally, we must limit the motions to small displacements and cannot drive the manipulators to the final target right away. Therefore, at every iteration i, we compute a local target $\mathbf{c_i^*}$, via a linear interpolation between current and target contours ($\mathbf{c}$ and $\mathbf{c}^*$, respectively). We encode this $\mathbf{c_i^*}$ into a feature vector as well, denoted $\mathbf{s_i^*}$.

We use the inverse of the interaction matrix $\mathbf{L}^{-1}$ to compute the robot pose variations $\delta\mathbf{r}$, required to drive the object to the local target, $\mathbf{s_i^*}$.

We then send $\delta\mathbf{r}$ to our dual-arm controller [Tar19], which relies on hierarchical inverse kinematics, to compute the robot joint commands. The whole process is repeated, by continuously updating data matrices $\mathbf{C}$ and $\mathbf{\Delta R}$ at each iteration i, until the object reaches the target shape and/or position. The entire process is summarized in Algorithm 1. While previous works made use of these different techniques (i.e. cooperative task representations, image Jacobian estimation), our work presents a promising approach to control deformable objects in $\mathbb{SE}(3)$, without considering specific objects (e.g., cables or clothes).

Algorithm 1 Control framework

procedure INITIALIZATION
- $\mathbf{R} = \{\}$
- $\mathbf{C} = \{\}$
- $j = 0$
- **while** $j \leq D$ **do**
  - Control robot in open-loop
  - $\mathbf{C} \xleftarrow{+} \mathbf{c}_j$ ▷ from camera feedback
  - $\mathbf{R} \xleftarrow{+} \mathbf{r}_j$ ▷ from robot
  - $j \leftarrow j + 1$
- Construct $\boldsymbol{\Delta}\mathbf{R}$ ▷ see section 2.2.2
- **return** $\boldsymbol{\Delta}\mathbf{R}$, $\mathbf{C}$

procedure CONTROL LOOP
- Define the target $\mathbf{c}^*$
- $\mathbf{c}_i = \mathbf{c}_D$
- $e_i = RMSE(\mathbf{c}^*, \mathbf{c}_i)$
- **while** $e_i \leq 0.1$ **do**
  - $\mathbf{U}_k = PCA(\mathbf{C})$ ▷ see section 2.2.3
  - $\mathbf{s}_i = project(\mathbf{c}_i, \mathbf{U}_k)$
  - $\mathbf{L}^{-1} = interactionMatrix(\boldsymbol{\Delta}\mathbf{R}, \mathbf{C}, \mathbf{U}_k)$ ▷ see section 2.2.4
  - $\mathbf{s}_i^* = localTarget(\mathbf{c}^*, \mathbf{c}_i, \mathbf{U}_k)$
  - $\boldsymbol{\delta}\mathbf{r}_i = poseCommand(\mathbf{L}^{-1}, \mathbf{s}_i^*, \mathbf{s}_i)$ ▷ see section 2.2.5
  - $\dot{\boldsymbol{\theta}} = robotJointsCommand(\boldsymbol{\delta}\mathbf{r})$ ▷ see section 2.2.6
  - Apply $\dot{\boldsymbol{\theta}}$ to robot
  - Get $\mathbf{c}_i, \mathbf{r}_i$ from camera feedback and robot respectively
  - Update $\boldsymbol{\Delta}\mathbf{R}$, $\mathbf{C}$
  - $e_i = RMSE(\mathbf{c}^*, \mathbf{c}_i)$

2.2 Method

In this Section, we detail each of the modules which compose our framework (again, refer to Fig. 2.3).

2.2.1 Image processing for object contour extraction

We use an Intel Realsense D435 camera, which looks at the robot end-effectors and at the object from a fixed (in the world) position. Both the RGB and depth images are used to obtain a sampled, ordered contour of the object in 3D at each iteration. The different steps of the contour extraction are presented in Fig. 2.4.

To simplify the image processing algorithm – which is not the main scope of our work

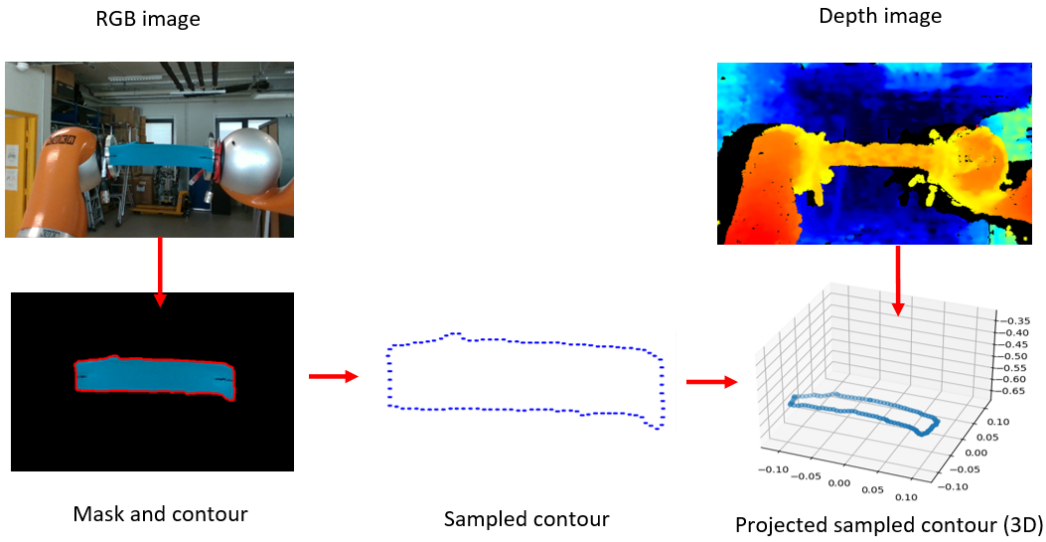


Figure 2.4: Steps of the image processing for object contour extraction (here, a sponge). The 2D RGB image of the camera is used to extract a mask of the object, which in turn lets us obtain its contour. This contour is sampled into a constant number of pixels, which are consistently spaced and ordered. These pixels are then projected as 3D points using their depth and the camera intrinsic parameters.

– we have only considered blue objects or blue parts of the objects. By thresholding in the HSV space, we obtain a mask of the blue part, on which we apply six closings and six openings using a 3x3 kernel. Three erosions are added to obtain a mask well inside the object, to avoid getting outlying depth coordinates later. Then, we use the *findcontour* function of OpenCV to obtain points defining the contour of the previously extracted mask.

The contour is then sampled into a given number of points p; it is essential for the contour points to be uniformly spaced and, more importantly, always ordered the same way from one contour to the next. Namely, the indexes of the points in $\mathbf{c}$ must correspond in $\mathbf{c}_i$ and $\mathbf{c}_{i+1}$, for the PCA to extract the accurate contour variations. We chose to order the contours in a clockwise direction starting from the left end-effector; we recall (second hypothesis in Sec. 2.1) that the contact between object and left end-effector is the same in the current and target contours. Once the contour is sampled and ordered, it is projected in 3D metric coordinates using the *pyrealsense* library. For this, we apply the function *rs2_deproject_pixel_to_point* to each sampled pixel of the contour, using the pixel depth and the camera intrinsic parameters.

The output of this module, at each iteration i, is vector $\mathbf{c}_i$ (2.4).

2.2.2 Generating a sequence of contours and robot poses

The next step consists in generating a sequence of contours and corresponding robot poses. These are respectively the contour variation matrix $\mathbf{C}$ and the robot pose variation matrix $\mathbf{\Delta R}$, which are needed for two purposes: to extract the contour's

principal components, and to estimate the inverse interaction matrix. Matrices $\mathbf{C}$ and $\mathbf{\Delta R}$ are built on a sliding window containing the most recent data: at each iteration, the oldest data in $\mathbf{C}$ and $\mathbf{\Delta R}$ are removed and replaced by the current data.

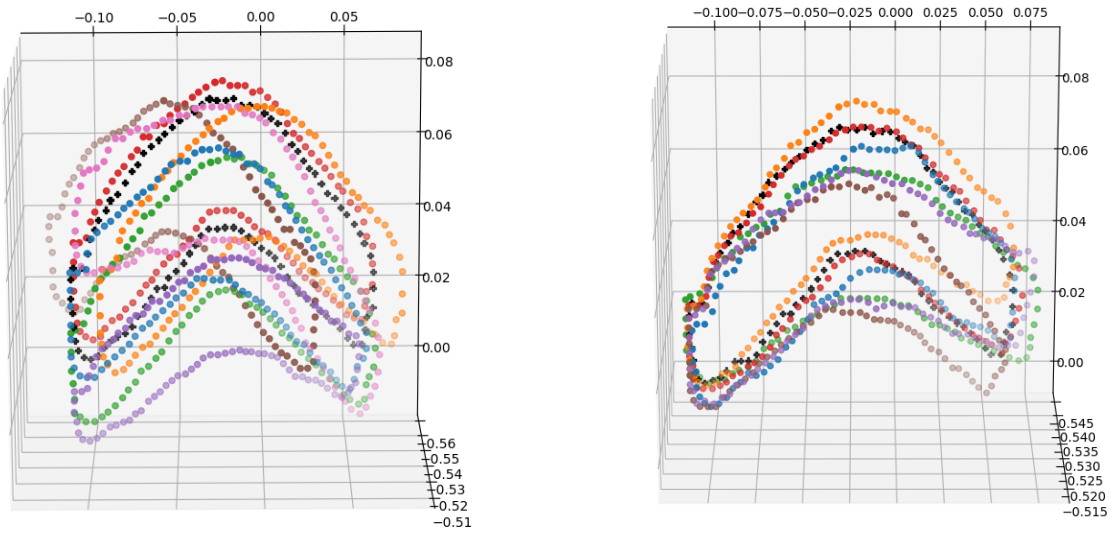


Figure 2.5: Initialization motions of the object (here, a sponge) for the absolute (left) and relative (right) task. Here, each of the $k = 12$ DOF are stimulated: 3 translations and 3 rotations around the initial position (black) for each task.

To build matrices $\mathbf{C}$ and $\mathbf{\Delta R}$, we collect the last $D+1$ samples of contours $\mathbf{c}$ and robot poses $\delta\mathbf{r}$ (so as to get D variations) while the robot moves. While after $D+1$ iterations ($i > D+1$) the robot autonomously moves using the designed control law, for the first $D+1$ iterations we let it move in open-loop. We do this by executing a series of small motions to deform the object, exciting one by one each considered DOF as shown in Fig. 2.5. The magnitude of these motions is chosen arbitrarily so as to sufficiently vary the contour; we set ± 0.02 m for the translations and $\pm\frac{\pi}{6}$ rad for the rotations. The relative motions should be reduced for stiffer or more fragile objects, to avoid breaking them. We decided to move along each robot DOF in both directions ($\pm$) just once yielding $D = 2k$. D should not be too small, to ensure that $\mathbf{L}^{-1}$ is not under-determined, and not too large, so that the model estimation is local and initialized quickly. This window size allows to excite every DOF equally and in both directions, to ensure a well distributed initial model estimation.

To build matrix $\mathbf{C}$, we stack all the $\mathbf{c_i}$ (2.5). Building $\mathbf{\Delta R}$ is a bit more cumbersome. We must obtain the robot pose variation $\delta\mathbf{r}_i$ between consecutive iterations $i-1$ and i. For the translations, we simply subtract the components: $x_i^{\mathbf{ref}} - x_{i-1}^{\mathbf{ref}}$, $y_i^{\mathbf{ref}} - y_{i-1}^{\mathbf{ref}}$, $z_i^{\mathbf{ref}} - z_{i-1}^{\mathbf{ref}}$. For the orientations, we transform rotation vectors $\mathbf{q}_{r_i}^{\mathbf{ref}}$ and $\mathbf{q}_{r_{i-1}}^{\mathbf{ref}}$ to quaternions, compute the distance between the quaternions (the distance between quaternions $\mathbf{q}_i$ and $\mathbf{q}_{i-1}$ is $\mathbf{q}_i \cdot \mathbf{q}_{i-1}^{-1}$) and then transform back to the rotation vector representation. Finally we stack all $\delta\mathbf{r}_i$, to compose robot pose variation matrix, 2.6.

2.2.3 Principal Component Analysis

Even for a small dataset and small number of contour samples, the dimension of matrix $\mathbf{C}$ is too high compared to the robot DOF (for $M = 24$ and $p = 100$ points: $\mathbf{C} \in \mathbb{R}^{300\times 25}$). Hence, we perform a PCA on $\mathbf{C}$, to reduce its dimension. First, we shift each column of $\mathbf{C}$ by the mean of all columns, $\bar{\mathbf{c}} = \frac{\sum \mathbf{c}_j}{M+1}, j = 0, ..., D$, to obtain:

$$\mathbf{C}_m = [\mathbf{c}_0 - \bar{\mathbf{c}}, \cdots, \mathbf{c}_D - \bar{\mathbf{c}}] \in \mathbb{R}^{3p\times D+1}. \tag{2.7}$$

Then, we compute $\mathbf{Q}$, the covariance matrix of $\mathbf{C}_m$. We obtain the eigenvector matrix $\mathbf{U} \in \mathbb{R}^{3p\times 3p}$ by performing a Singular Value Decomposition (SVD) on $\mathbf{Q}$.

We select the first k columns of $\mathbf{U}$ to be able to control k DOF of our system; this choice ensures that no information is lost, as most of the time only fewer singular values are non-zeros. But since we can't predict how many will be "significant", $k = 12$ is the safest maximum choice so as not to risk losing accuracy while not reducing the number of DOF available. They define the projection matrix $\mathbf{U}_k \in \mathbb{R}^{3p\times k}$. These columns correspond to the k principal components (with highest variances) in the dataset, and therefore determine the directions of highest variability in the data.

At each iteration i, the reduced feature vector is then:

$$\mathbf{s}_i = \mathbf{U}_k^\top (\mathbf{c}_i - \bar{\mathbf{c}}) \in \mathbb{R}^k. \tag{2.8}$$

2.2.4 Estimation of the Inverse Interaction Matrix

The next step consists in estimating the inverse interaction matrix $\mathbf{L}^{-1}$ needed for control. Recall that the interaction matrix is the linear mapping between feature variation $\delta\mathbf{s}$ and robot pose variation $\delta\mathbf{r}$ (see (2.3)). This matrix is unknown for a non-rigid object, and it should be inverted to control the robot pose. For both reasons, [ZNAPC20] has shown that for image-based soft object manipulation, it is more efficient to estimate the inverse interaction matrix than the matrix itself.

To this end, we project the contour matrix $\mathbf{C}$ into the reduced space as follows:

$$\mathbf{S} = \mathbf{U}_k^\top \mathbf{C}_m \in \mathbb{R}^{k\times D+1}, \tag{2.9}$$

and then derive the features variation over the D-dimensional window:

$$\mathbf{\Delta S} = \begin{bmatrix} \mathbf{S}_1 - \mathbf{S}_0 & \dots & \mathbf{S}_D - \mathbf{S}_{D-1} \end{bmatrix} \in \mathbb{R}^{k\times D}. \tag{2.10}$$

Finally, the inverse interaction matrix is given by:

$$\mathbf{L}^{-1} = \mathbf{\Delta R \Delta S}^+ \in \mathbb{R}^{k\times k} \tag{2.11}$$

With $\mathbf{\Delta S}^+$ the pseudo-inverse of $\mathbf{\Delta S}$, calculated using the Numpy SVD algorithm.

2.2.5 Controlling the robot pose

Linear approximation (2.3) is only valid locally. Therefore, a control law based on $\mathbf{L}^{-1}$ cannot guarantee convergence if the initial contour and the target contour $\mathbf{c}^*$ are too far. To solve this issue, we design a local target contour $\mathbf{c}_i^*$ at each iteration, via linear interpolation:

$$\mathbf{c}_i^* = \frac{\mathbf{c}^* - \mathbf{c}_i}{n} \tag{2.12}$$

with n big enough to ensure small displacements. Correspondingly, we can project $\mathbf{c}_i^*$ into the feature vector space to obtain the local target feature vector:

$$\mathbf{s}_i^* = \mathbf{U}_k^\top(\mathbf{c}_i^* - \bar{\mathbf{c}}) \in \mathbb{R}^k \tag{2.13}$$

The desired robot pose variations are then computed via:

$$\delta\mathbf{r}_i = \mathbf{\Lambda}\mathbf{L}_i^{-1}(\mathbf{s}_i^* - \mathbf{s}_i) \in \mathbb{R}^k \tag{2.14}$$

with $\mathbf{\Lambda} \in \mathbb{R}^{k\times k}$ a diagonal matrix of control gains. This feedback controller guarantees asymptotic convergence of $\mathbf{s}_i$ to $\mathbf{s}_i^*$, in the ideal case that $\mathbf{L}^{-1}$ is perfectly estimated, as proved using the Lyapunov criterion studied hereby:

Recall the locally linear mapping $\delta\mathbf{s} = \mathbf{L}\delta\mathbf{r}$, which we can write in discrete form as:

$$\mathbf{s}_{i+1} - \mathbf{s}_i = \mathbf{L}_i\delta\mathbf{r}_i \tag{2.15}$$

Here, we aim at proving local convergence. Let us consider a constant local target $\mathbf{s}^*$. We consider that if the system converges to each local target, it will eventually converge to the final one. We define the error

$$\mathbf{e}_i = \mathbf{s}^* - \mathbf{s}_i \tag{2.16}$$

that we regulate via the control law:

$$\delta\mathbf{r}_i = \Lambda\mathbf{L}_i^{-1}(\mathbf{s}^* - \mathbf{s}_i) \tag{2.17}$$

under the assumption that $\mathbf{L}^{-1}$ is perfectly estimated.

Let us define the Lyapunov function:

$$\mathbf{V}(\mathbf{e})_i = \frac{1}{2}\mathbf{e}_i^\top\mathbf{e}_i \tag{2.18}$$

whose derivative is:

$$\delta\mathbf{V}(\mathbf{e})_i = \mathbf{e}_i^\top\delta\mathbf{e}_i \tag{2.19}$$

Since $\mathbf{e}_{i+1} = \mathbf{s}^* - \mathbf{s}_{i+1}$, we can rewrite $\delta\mathbf{e}_i$ as:

$$\begin{aligned}\delta\mathbf{e}_i &= \mathbf{e}_{i+1} - \mathbf{e}_i \\ &= -\Lambda\mathbf{e}_i\end{aligned} \tag{2.20}$$

using (2.15) and (2.17). Then, we get:

$$\delta\mathbf{V}(\mathbf{e})_i = \mathbf{e}_i^\top(-\Lambda)\mathbf{e}_i \tag{2.21}$$

With $\Lambda > 0$. We prove that $\delta\mathbf{V}(\mathbf{e})_i < 0$, thus the system is stable.

2.2.6 Controlling the robot joints

To map the desired robot pose variations $\delta\mathbf{r}$ to the robot joint velocities $\dot{\theta}$, we use the cooperative task representation, outlined in Sect. III, and recalled here (for further details, refer to [Tar19], [CCS96], [JR14]). We consider $\delta\mathbf{r} \in \mathbb{R}^{12}$ as the variation of $\mathbf{r}$ defined in (2.1). From $\mathbf{J_{lef}}$ and $\mathbf{J_{rig}}$ (the Jacobian matrices of the left and right arm in $\mathcal{F}_{ref}^{abs}$), we derive the cooperative task Jacobian matrices:

$$\begin{aligned}\mathbf{J_{abs}} &= \begin{bmatrix}\frac{1}{2}\mathbf{J_{lef}} & \frac{1}{2}\mathbf{J_{rig}}\end{bmatrix} \\ \mathbf{J_{rel}} &= \begin{bmatrix}-\mathbf{\Psi\Omega J_{lef}} & \frac{1}{2}\mathbf{\Omega J_{rig}}\end{bmatrix},\end{aligned} \tag{2.22}$$

where

$$\mathbf{\Psi} = \begin{bmatrix}\mathbf{I} & -\mathbf{\Upsilon_{lef}} \\ 0 & \mathbf{I}\end{bmatrix}, \quad \mathbf{\Omega} = \begin{bmatrix}\mathbf{\Phi_{lef}} & 0 \\ 0 & \mathbf{\Phi_{lef}}\end{bmatrix}, \tag{2.23}$$

with $\mathbf{\Upsilon_{lef}}$ the skew-symmetric matrix of the position of the left arm in the relative task frame, and $\mathbf{\Phi_{lef}}$ the rotation matrix of the left arm in $\mathcal{F}_{ref}^{abs}$.

If both the relative and absolute tasks have to be satisfied at the same time ($k = 12$), a relevant choice is to prioritize the relative task, to avoid undesired internal stress which may damage the object and/or the robot. Taking into account joint (position, velocity and acceleration) limits, the highest priority task is solved through:

$$\begin{aligned}\dot{\boldsymbol{\theta}}_1 \in \min_{\dot{\boldsymbol{\theta}}} &||\mathbf{J_{rel}}\dot{\boldsymbol{\theta}} - \delta\mathbf{r^{rel}}||_2 \\ &\text{subject to: joint limits.}\end{aligned} \tag{2.24}$$

The obtained solution vector $\dot{\boldsymbol{\theta}}_1$ provides a null-space condition to the second task, so the final joint velocities to be sent to the robot are:

$$\begin{aligned}\dot{\boldsymbol{\theta}} \in \min_{\dot{\boldsymbol{\theta}}} &||\mathbf{J_{abs}}\dot{\boldsymbol{\theta}} - \delta\mathbf{r^{abs}}||_2 \\ &\text{subject to: joint limits,} \\ &\mathbf{J_{rel}}\dot{\boldsymbol{\theta}} = \mathbf{J_{rel}}\dot{\boldsymbol{\theta}}_1.\end{aligned} \tag{2.25}$$

We add the relative task solution as a constraint to the absolute task to avoid interference. Using this formulation, the absolute task error is minimized as long as the resulting joint velocity vector provides the best solution for (2.24).

If only the relative task has to be satisfied to deform the object, and we "free" the object pose, $k = 6$ and we can simply apply (2.24) and set $\dot{\boldsymbol{\theta}} = \dot{\boldsymbol{\theta}}_1$.

2.3 Experiments and results

To validate the method, we conducted many experiments, during which we tested different initial and target shapes, as well as different objects (rigid or not, presented in Fig.2.6). A video of experiments is available at https://www.youtube.com/watch?v=cxoqIOt973s and the source codes are accessible at https://gite.lirmm.fr/csaghour/rkcl-bazar-flex-app.git. At the beginning of each experiment, $M = 24$ predefined motions are executed. We set the number of contour points to $p = 100$ and the number of intermediate targets to $n = 20$. The control gain matrix was tuned experimentally to $\mathbf{\Lambda} = 0.07\ \mathbf{I}_k$, to obtain motions neither too fast nor too slow.

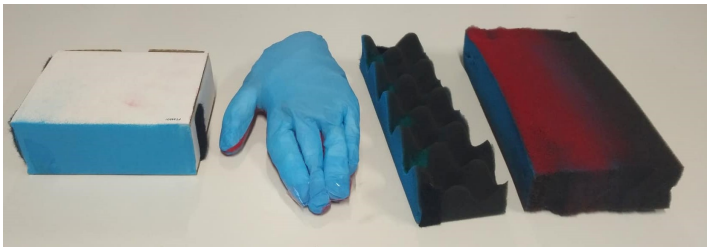


Figure 2.6: Example of objects to be manipulated. Left to right: cardboard box, plastic glove, crown-shaped sponge, sponge. The face in front the camera once grasped (blue) is of a distinctive color to allow the image segmentation. Only the deformation of this face is tracked.

The experiments using the sponge shown in Fig. 2.6 demonstrate successful convergence to the different final targets (in red in the figure), involving the deformation, as well as change in the orientation and position of the object (see Fig. 2.7). We also show that the method gives equally successful results with a different initial position of the object. The results do not depend on the camera-object relative pose since the initialization phase is carried out at the start of each experiment. The evolution of the cooperative task frames is shown in the third column of Fig. 2.7.

We define the task error as the RMSE between the current contour and the final target contour:

$$e_i = \sqrt{\frac{1}{3p}\sum_{i=1}^{3p}(\mathbf{c}^* - \mathbf{c}_i)^2} \tag{2.26}$$

This metric decreases (see the last column of Fig. 2.7) until reaching an acceptable threshold (0.01m in our experiments).

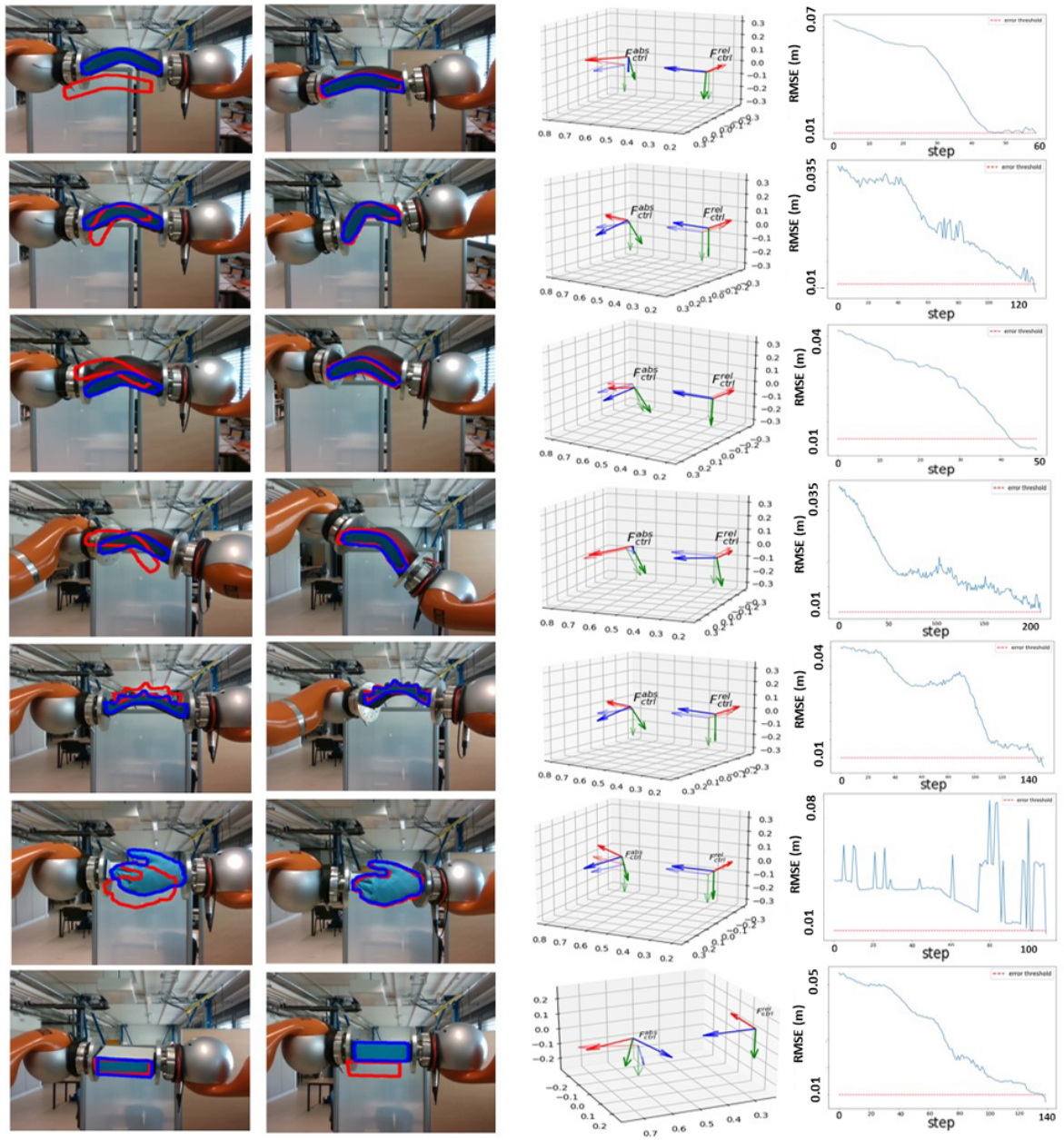


Figure 2.7: The two first columns are the robot camera view of seven experiments. Starting from different initial configurations (in blue on the first column), we reach the final targets (in red). The third column shows the initial (transparent) and final poses of the absolute frame F_{ctrl}^{abs} and relative frame F_{ctrl}^{rel}. The last column shows the evolution of the shape difference defined in (2.26) according to the iteration step. The experiments include, from top to bottom: four different shaping of a sponge, then shaping of different objects: a crown-shaped sponge, a plastic glove, and a cardboard box. For the cardboard box, the relative frame F_{ctrl}^{rel} is freed, so only the absolute task F_{ctrl}^{abs} evolves.

We also performed experiments with the other objects shown in Fig. 2.6, soft (a differently shaped sponge, a plastic glove) and rigid (a cardboard box). The method proves to be effective with those as well, as shown in Fig. 2.7, despite the rough

contours for an object of more complex shape like the glove. For this experiment, noise was detected in the images, explaining the wide oscillations of the error observed in Fig. 2.7. Nevertheless, the controller converged to the final target shape, showing the robustness of the method. Again, tasks like bending, compressing, rotating and translating were successfully carried out, proving the large spectrum of applications of our controller. Tab. 2.1 contains the convergence time of the experiments, as well as the initial error between the initial and goal shape.

Experiment	Initial error	Convergence time (s)
1	0.74	12
2	0.36	26
3	0.42	10
4	0.34	42
5	0.4	30
6	0.41	22
7	0.54	28
8	0.5	21

Table 2.1: Convergence data for all the experiments.

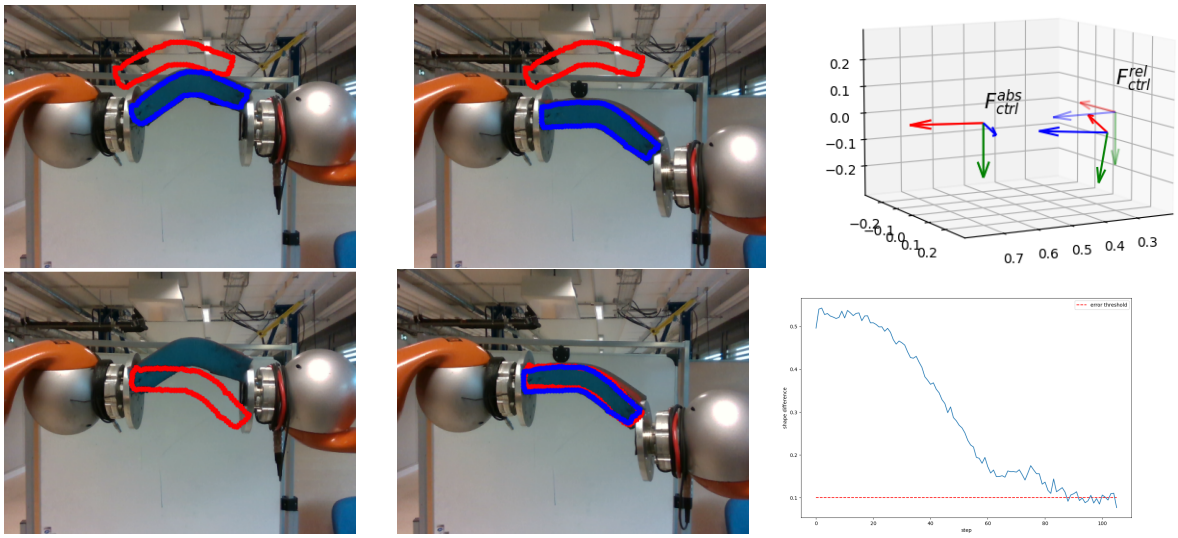


Figure 2.8: Experiment with the sponge, reaching the target configuration from the initial position (in blue, left) while only controlling the relative task. The target (in red, top row) is aligned with the relative reference frame F^{rel}_{ref} (the left end-effector) so that the aligned target (in red, bottom row) can be achieved with only the relative task being controlled. On the right are shown the evolution of the cooperative task frames (top) and of the error (bottom).

Regarding the manipulation of the cardboard box, which we considered as rigid, the experiment consists only in orienting and translating the object. Through this experiment, we show that our framework can be used for manipulating a wider spectrum

of objects, including rigid ones. In this case, only $k = 6$ DOF are needed for the absolute task, and the 6 other DOF are freed. These could be used, for example, for obstacle avoidance. We can, on the contrary, choose to only deform the object and not move its position and orientation in the space. Then, only the relative task is needed, therefore reducing the number of DOF to $k = 6$. Such an experiment is presented in Fig. 2.8. To achieve this, the target is aligned so as to match with the fixed end-effector during the experiment. This allows us to shape the object with no concern on the orientation and position. The error is then computed between the aligned contour and the current contour.

Table 2.2: Range of the displacements of the absolute and relative tasks during the eight experiments (with largest values in bold). x, y, z are the translation in mm, u, v, w the rotations in rad (Euler angles) around axis X, Y, Z, respectively.

Experiment	$x^{\mathbf{abs}}$	$y^{\mathbf{abs}}$	$z^{\mathbf{abs}}$	$u^{\mathbf{abs}}$	$v^{\mathbf{abs}}$	$w^{\mathbf{abs}}$
1	-28	**56**	-7	**-0.31**	**0.73**	0.01
2	7	18	9	**-0.37**	-0.01	0.17
3	-13	-32	-2	**-0.12**	0.1	0.05
4	-27	12	-14	0.06	**-0.14**	0.04
5	20	-20	12	-0.17	**0.9**	0.06
6	**-37**	**37**	32	**-0.25**	**-0.21**	0.02
7	-14	**41**	13	**-0.2**	**0.61**	-0.1
8	-	-	-	-	-	-

(a) Absolute task

Experiment	$x^{\mathbf{rel}}$	$y^{\mathbf{rel}}$	$z^{\mathbf{rel}}$	$u^{\mathbf{rel}}$	$v^{\mathbf{rel}}$	$w^{\mathbf{rel}}$
1	8	-0.6	10	-0.03	0.03	0.03
2	7	**34**	**-39**	-0.02	-0.01	-0.06
3	-1	-7	-8	0.01	0.04	**0.08**
4	8	-10	-14	**0.15**	0.01	0.05
5	8	3	-15	0.05	0.01	-0.01
6	**-43**	8	-7	0.01	0	0.01
7	-	-	-	-	-	-
8	-20	**-80**	14	-0.05	**-0.24**	**-0.13**

(b) Relative task

The range of displacements of the absolute and relative task for each experiment are shown in Table 2.2. The most important absolute displacements during these experiments are rotations, especially on the $Y^{\mathbf{abs}}$-axis. We observe that the relative displacements are small compared to the absolute ones: this is because the objects which we considered can be displaced/oriented on a larger scale than it can be deformed. Unexpectedly, the relative translation on the $Z^{\mathbf{rel}}$-axis is small for experiment 1, which was aimed to be a traction; this serves to show that a desired contour can be attained in more than one absolute/relative displacements combination, and that a contour does not testify of a specific deformation. This is because by representing the

object only with its visible contour, we lose information about its global geometry. In average, the most important relative displacements are translations (experiments 2, 6, 8). In experiments 4 and 8, the rotations are also large enough to be visible on the corresponding frame evolution figures.

We also experimentally investigate how the number of samples, thus the size of the window of past data, impacts the controller. Practical experiments showed that a smaller window ($D = 12$) could lead to cases where $\mathbf{L}^{-1}$ is under-determined. We inferred that $D = 24$ seemed to be a window size complete enough to describe well the span of possible motions in order to start the manipulation unbiased toward a specific direction, since it contains information about every DOF in both directions.

Theoretically, we also aim to keep that window rather small to allow fastest computations, and because our approximation (equation 2.3) is local; a larger window would risk making the model inaccurate. If the contour at $i = 1$ and at $i = D$ are too different, L won't be an accurate linear model of the large deformation that occurs.

We did a few extra experiments to explore this. In those, we aim to reach a same target with either a window of size $D = 24$ or one of size $D = 48$. A few trials for both cases are summarized in Tab.2.3; we define as 'success' an experiment during which the error ϵ (2.26) reaches the desired threshold (0.1). Note that the initialization time is longer for the larger window because it requires the double of initial motions. For $D = 12$, the controller does not converge as a result of $\mathbf{L}^{-1}$ being often under-determined.

D	Initialization time (s)	Mean duration of each iteration	experiment time (s)	Success
24	39.4	0.23	47	yes
24	43.2	0.17	12	yes
24	42.0	0.25	11	yes
24	42.8	0.16	8	yes
48	76.1	0.22	**>60**	yes
48	79.7	0.18	21	yes
48	75.9	0.31	**>60**	**no**
48	77.8	0.21	**>60**	**no**
48	78.5	0.2	**>60**	yes

Table 2.3: Experiments with different window sizes

To summarize, it would seem that the overall convergence (if there is any) time is longer with the larger window, as predicted. We found out that experiments with a larger window do not succeed every time. It is however difficult to conclude on the optimized number of samples taken without more trials, that are difficult implement (open-loop motions to program for initialization, experiments that are unsafe for the

objects/robot...). We do explore this issue in the last chapter of the thesis, 4.8, where we propose to investigate this choice through simulations.

2.4 Conclusion and discussion

This chapter presents a complete method for dual-arm shaping of soft objects in 3D and in real time. We use a few initializing motions to both execute a PCA to reduce the dimensions of the visual data and to compute an interaction matrix, and iteratively obtain robot control inputs to reach a final target shape. The framework is able to handle objects of different geometry and materials, soft and rigid alike without any prior knowledge, although we experimented on a limited variety of objects. Nevertheless, the 3D target shapes could be successfully reached in these experiments.

From a theoretical viewpoint, our work raised questions on the representations, while considering the numerous DOFs needed to shape and move the objects. We believe that using a relative task to describe and control exclusively deformations is a promising approach. Our work addresses different challenges regarding 3D motion representation, while considering the limitations of the objects' geometrical representation. In an industrial context, we aim at proposing solutions for tasks such as "deform and place".

One of the main aspects to improve is vision. Contours are not well suited to describe volumetric objects and limit the deformations to those of the visible surface. Our approach does address 3D deformation, although the object is reduced to a visible surface. The surface is represented in 3D space (thanks to the depth measurement) and not just in a plane. Therefore, the manipulation is indeed in 3D, and we can shape the object out of the image plane (e.g., along the depth axis, or along the two out-of-plane orientations).

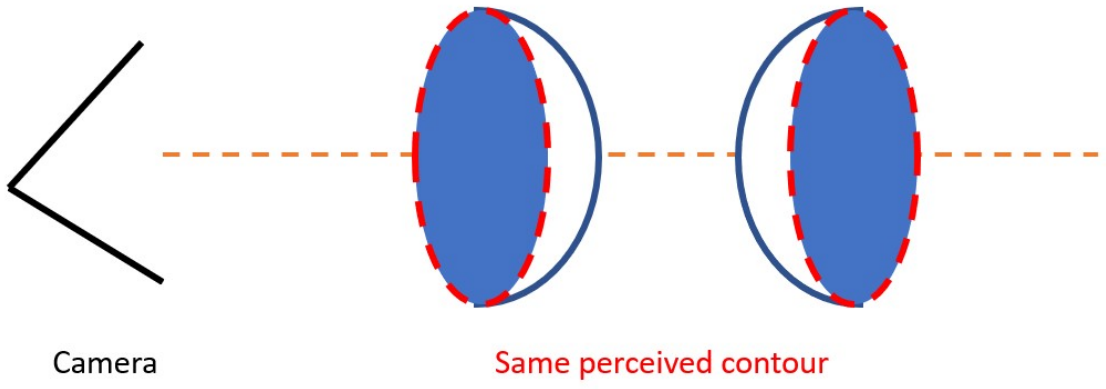


Figure 2.9: Example of an object (e.g., a speaker diaphragm, or a contact lens) with out-of-plane deformations which cannot be controlled using the visible contour.

Yet, we agree that the visual representation that we chose (contours) limits the spectrum of manipulations. For instance:

- We cannot deal with object self-occlusions.
- Out-of-plane deformations, which do not change the contour, are not controllable. An example is shown in Fig.2.9.

Yet, a 3D volumetric sampling might be challenging to achieve, since the success of our method largely depends on the consistency of the points and on their order from one iteration to the other, which is not assured with simple point clouds. Occlusions could also be difficult to handle.

As a remedy to this problem, we focus the rest of our work on the addition of an adaptive deformation model. Such a hybrid controller would be able, for instance, to estimate the deformation of the self-occluded part of the object, and to ensure target reachability.

CHAPTER 3

Modeling and simulations for soft objects manipulation

3.1 Motivations

The preliminary work presented in Chapter 2 led us to focus on modeling, following and controlling the deformation of the manipulated object over its entire volume - including its self-occluded parts. Since our objective is to find a state representation which can be used to track the deformations and used in a control scheme similar to the one described in chapter 2, we look for a geometric representation that verifies the following requirements:

1. have a constant dimension;
2. have a consistent topology between iterations: each element should be matched to its previous position in space, in order to compute local displacement;
3. represent the whole volume of the object.

For these reasons, we choose to use a mesh to describe the object being manipulated. A mesh $\mathcal{M}$ is an entity constituted of:

- $\mathbf{m}$, a set of 3 dimensional points, called nodes: $\mathbf{m} \in \mathbb{R}^{3\times n}$, with n the number of nodes.
- $\mathbf{P}$, called connectivity set (or elements), which links the nodes to each others in arbitrary polygons. For instance, for a triangle mesh, elements will be constituted of sets of three node indexes, each composing a triangle: $\mathbf{P} \in \mathbb{R}^{3\times tr}$ with tr the number of triangles representing the full surface of the object. For a tetrahedral

mesh, each element will be constituted of the indexes of four nodes: $\mathbf{P} \in \mathbb{R}^{4\times te}$ with te the number of tetrahedra representing the full volume of the object.

The number of nodes is constant. Since these nodes can be displaced in space, the connectivity between them remains unchanged, allowing the mesh to be ordered consistently. Throughout the manuscript, we refer to tetrahedral mesh as $\mathcal{M} = [\mathbf{m}, \mathbf{P}]$, while $\mathcal{M}^v = [\mathbf{m}^v, \mathbf{P}^v]$ denotes a triangle mesh, also referred as visual mesh (since it only represents the external surface of the object).

The tetrahedral mesh - for the full volume - is the one used to compute the deformation, via physics simulation. Instead, the triangle mesh is used to represent the external surface of the object, used as comparison with the *observed* state of the object.

Then, using a mechanical model, the displacement of the nodes in relation to external forces can be estimated in simulations. These external forces can be measured from the real robot onto the mesh by means of sensors (force, positions, camera...) to feed a real-to-sim tracking framework. Real-to-sim approaches, sometimes referred as Digital Twins, create simulation models which behave as much as possible like the real system. The goal is to analyze the system behavior, or possible scenarios ([VEBR18], [VSM+21]).

In this chapter, we will first present the basics of FEM as our chosen physical model and its resolution, as well as the simulations setup (section 3.2). Then, we will present various solutions for the model construction (section 3.3.3).

3.2 Finite Element Method (FEM) based simulation

FEM simulations have been used for numerical solving in multiple areas (automobile, aeronautical, chemical, pharmaceutical, infrastructures...) for engineering design and manufacturing. In fact, they allow modeling many different physical processes - structural analysis, fluid flow, heat conduction, and electromagnetic potential, to cite a few ([LLP22]).

The principle of FEM is to obtain a set of algebraic equations to solve for unknown displacement (called *nodal quantity*) given a set of external forces applied to the system. Overall, [Log11] presents the main steps of the solving method, which will be further detailed in the following, as:

1. Sub-divide the solid (the object) into smaller elements;
2. Choose a displacement function within each element, and approximate the nodal unknowns for each element;
3. Define the strain/displacement and stress/strain relationships, and derive the relations among the nodal values of the solution over each element;
4. Assemble the elements and obtain the global solution.

The advantages of numerical methods, and in particular FEM are listed in [Red13]:

- They provide numerical solution which albeit approximate, are easier to find than *analytic* solutions for complex models.
- By relying on simulations, they are more viable than Physical experiments in term of time and resources.
- They enable choosing which relevant features to include and analyze.
- By discretization, they allow for an accurate representation of complex geometry, with simpler subdomains.
- they easily represent the total solution through approximate functions defined within each element, and can therefore capture local effects.

FEM also allow to take into account irregular boundaries, dynamic systems as well as non-linear problems. On the other hand, a drawback resulting from sampling a complex domain of interest is that values are interpolated between the elements; if the samples are big, the nodes are widely spaced, and the approximation in between can draw away from reality. Conversely, numerous very small samples will result in higher costs in term of memory and computation. A good trade-off between accuracy and computation cost should be considered.

3.2.1 Basic principles

The Finite Element Method is a numerical method for solving differential equations, based on the discretization of a continuous domain Ω into a finite number of small elements. To do so, the studied domain is sampled as a mesh $\mathcal{M}$, and modeled according to boundary conditions (constraints on some surfaces or nodes), external forces, and material models (linear elasticity, hyper-elasticity, visco-elasticity, plasticity...). By using a discrete representation $\mathbf{u}$ of the field of unknown displacements, an approximate problem of which $\mathbf{u}$ is the solution is defined.

The aim of the method is primarily to compute a global stiffness matrix $\mathbf{K}$ such as:

$$\mathbf{F} = \mathbf{K}\mathbf{u} \tag{3.1}$$

with $\mathbf{F}$ the external force applied on the surface $\delta\Omega_F$ (Neumann boundary condition), and $\mathbf{u}$ the (unknown) displacement. A kinematically admissible displacement must also satisfy the Dirichlet boundary condition: $\mathbf{u} = U$ an imposed displacement on the surface $\delta\Omega_U$.

Let us hereby detail the FEM *general formulation*.

General formulation, Direct Method, or Displacement method
The displacement $\mathbf{u}^e$ of each element (local) of index e is function of the displacement at the nodes $\mathbf{u}$ as:

$$\mathbf{u}^e = \mathbf{N}\mathbf{u} \tag{3.2}$$

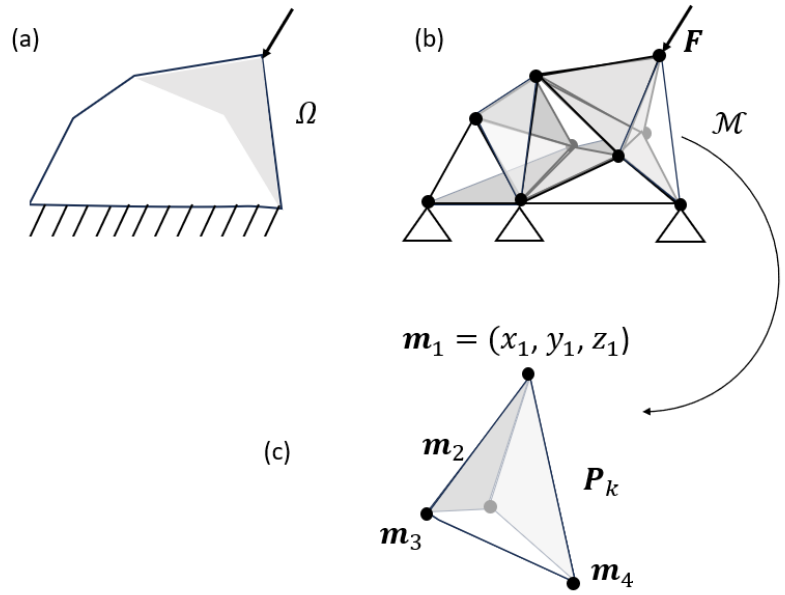


Figure 3.1: Discretization of domain Ω (a) into a mesh $\mathcal{M}$ (b). Each element (tetrahedron $\mathbf{P}_k$) of the mesh is made of four nodes (c). Boundary conditions and external force $\mathbf{F}$ are also modeled and applied on $\mathcal{M}$.

with $\mathbf{N}$ the shape function (also called blending function), which is an interpolation function over non-nodal points. Shape functions are known, and depend on the geometry chosen for the elements, the number of interpolation points, as well as the boundary conditions.

The strain $\boldsymbol{\epsilon}^e$ is then written according to these displacement through a constitutive equation:

$$\boldsymbol{\epsilon}^e = \mathbf{S}\mathbf{u}^e \tag{3.3}$$

. Depending on the hypothesis of the model, $\mathbf{S}$ can be expressed by the Cauchy-Green deformation tensor for large deformations [LRK09], or through infinitesimal strain theory. This entails:

$$\boldsymbol{\epsilon}^e = \mathbf{S}\mathbf{N}\mathbf{u} = \mathbf{B}\mathbf{u} \tag{3.4}$$

The strain-stress relationship can then be expressed through the constitutive equation [Flü65]:

$$\boldsymbol{\sigma}^e = \mathbf{C}\boldsymbol{\epsilon}^e \tag{3.5}$$

Globally, to take into account the external forces applied on the domain, the virtual work of these forces for a virtual displacement $\hat{\mathbf{u}}$ is written as the internal work of the stress on the volume Ω for the same displacement, giving:

$$\hat{\mathbf{u}}^\top \mathbf{F} = \int_\Omega \hat{\boldsymbol{\epsilon}}^\top \boldsymbol{\sigma} dv \tag{3.6}$$

Which is equivalent to:

$$\hat{\mathbf{u}}^\top \mathbf{F} = \hat{\mathbf{u}}^\top \left(\int_\Omega \mathbf{B}^\top \mathbf{C}\mathbf{B} dv \right) \mathbf{u} \tag{3.7}$$

Finally, an expression of the rigidity matrix $\mathbf{K}$ is obtained, so as to write 3.7 as 3.1, with $\mathbf{K}$ as:

$$\mathbf{K} = \int_\Omega \mathbf{B}^\top \mathbf{C}\mathbf{B} dv \tag{3.8}$$

Knowing $\mathbf{F}$ and having built $\mathbf{K}$, it is then possible to use 3.1 to compute the displacement over the mesh.

There exist other formulations of the problem to solve for $\mathbf{u}$; the *variational* (also said *integral*) approach, for instance. The FEM resolution can then be solved as a potential energy minimization problem [BF91].

Variational method

For a virtual displacement $\hat{\mathbf{u}}$, the total external virtual work is done by external force $\mathbf{F}$ and by body forces $\mathbf{f}$. This yields, over the domain, the equilibrium:

$$\int_{\Omega_F} \hat{\mathbf{u}}^\top \mathbf{F} ds + \int_\Omega \hat{\mathbf{u}}^\top \mathbf{f} dv = \int_\Omega \hat{\boldsymbol{\epsilon}}^\top \boldsymbol{\sigma} dv \tag{3.9}$$

Taking into account constitutive equation $\boldsymbol{\sigma} = \mathbf{C}\boldsymbol{\epsilon}$ and knowing that $\boldsymbol{\epsilon}$ depends on $\mathbf{u}$ (3.4), it gives:

$$\int_{\Omega_F} \hat{\mathbf{u}}^\top \mathbf{F} ds + \int_\Omega \hat{\mathbf{u}}^\top \mathbf{f} dv = \int_\Omega \hat{\boldsymbol{\epsilon}}(\hat{\mathbf{u}})^\top \mathbf{C}\boldsymbol{\epsilon}(\mathbf{u}) dv \tag{3.10}$$

$$\Leftrightarrow l(\hat{\mathbf{u}}) = a(\mathbf{u}, \hat{\mathbf{u}}) \tag{3.11}$$

The variational formulation of the problem is then to find the continuous displacement field $\mathbf{u}^h$ such that, for any kinematically admissible displacement $\mathbf{u}_{adm}$:

$$a(\mathbf{u}^h, \mathbf{u}_{adm}) = l(\mathbf{u}_{adm}) \tag{3.12}$$

With a a function that is bi-linear, symmetrical, positive definite.
Defining a mesh entails that $\mathbf{u}^h$ is sampled through *global interpolation functions*, defined as:

$$\mathbf{u}^h = \sum_{i=1}^{n} \psi_i \mathbf{u}_i \tag{3.13}$$

The stiffness matrix $\mathbf{K}$ is then built over the global interpolation functions, by rewriting 3.12 as the approximate variational formulation. For an admissible dis-

placement $\mathbf{u}_{adm} = \psi_j$ and with $\mathbf{u}$ expressed as 3.13, we get:

$$a(\sum_{i=1}^{n} \psi_i \mathbf{u}_i, \psi_j) = l(\psi_j) \tag{3.14}$$

and by linearity of a:

$$\Leftrightarrow \sum_{i=1}^{n} a(\psi_i, \psi_j)\mathbf{u}_i = l(\psi_j) \tag{3.15}$$

$$\Leftrightarrow \mathbf{Ku} = \mathbf{F}, \quad \mathbf{K}_{i,j} = a(\psi_i, \psi_j) \tag{3.16}$$

In our case, we choose to consider linear elasticity as a simple but satisfying approximation of the behavior of vastly used materials. We then take Hooke's equations for isotropic materials as constitutive equations 3.5. It describes the evolution of the stress $\boldsymbol{\sigma}$ in relation to the strain $\boldsymbol{\epsilon}$ considering the Poisson coefficient ν and the Young's modulus E, with the stiffness matrix given as:

$$\mathbf{C} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-2\nu}{2} \end{bmatrix} \tag{3.17}$$

.

3.2.2 Simulations setup

We use Simulation Open Framework Architecture (SOFA)[1] [FDD+12] software to run real-time FEM simulations of the deformation of the object being manipulated. Interactive physical simulations involve multiple components - collisions, mechanical behavior, visual rendering - all of which are computationally expensive. Most importantly, the need to account of real-time input is essential for our framework, since we aim at using the robot feedback to simulate the deformation of the object during manipulation, in a Real-to-sim manner. The end goal is to use these simulations as feedback in the control loop, hence the need for fast and effective computations.

In the simulation scene, the *deformation model* involves the tetrahedral mesh of the object in the robot frame, $\mathcal{M}$. The visual mesh $\mathcal{M}^v$ is imported in the simulator as a *visual model*; it is mapped to the tetrahedral mesh as a child of the parent deformation model. The DOF of the tetrahedral mesh are connected to the visual mesh nodes through SOFA's mapping functions, so that the deformation sustained by the deformation model propagates to the visual model.

[1] https://www.sofa-framework.org/

In the following, we call a *rigid particle*, a rigid mechanical object denoted **pr**. It is an object defined by a position and orientation, and can be moved in the SOFA scene as a single particle.

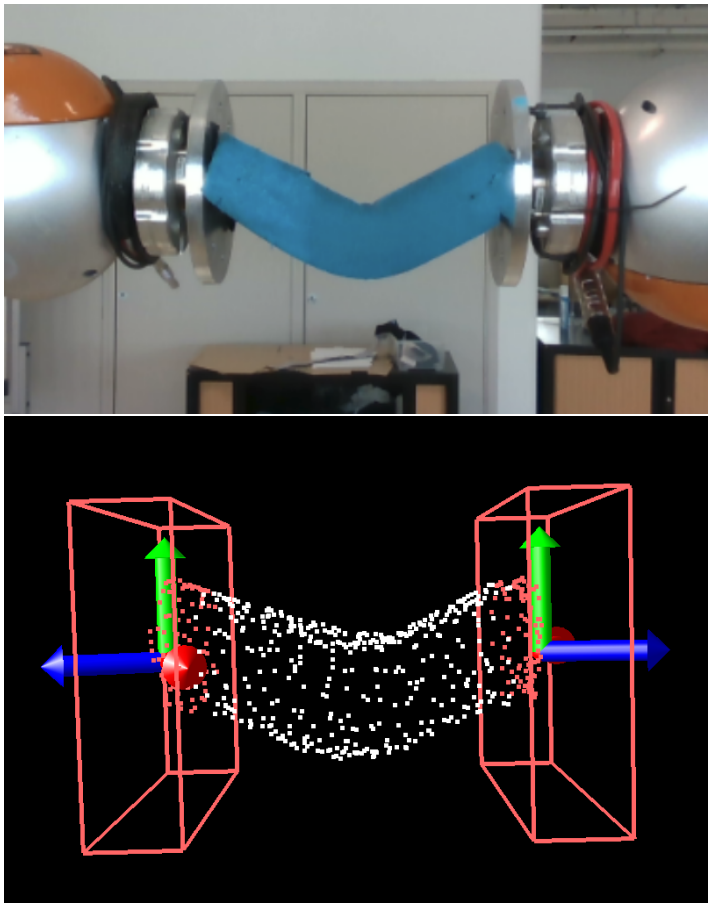


Figure 3.2: Simulation setup. The nodes of the mesh are shown in white. The red lines represent the bounding boxes around the positions of $\mathbf{pr}_l$ *and* $\mathbf{pr}_r$*, represented by the frames. The holding nodes* $\mathbf{H}_l$ *and* $\mathbf{H}_r$ *are the red nodes in left and right boxes respectively.*

The object is already grasped by the robot, and the positions and orientation of both end-effectors, $\mathbf{r}_l$ and $\mathbf{r}_r$ are known. We call $\mathbf{pr}_l$ and $\mathbf{pr}_r$ the rigid particles corresponding to the left and right end-effectors respectively, as illustrated on Fig.3.3. We define the contact between the end-effectors of the robot and the object by a set of mesh nodes, called *holding nodes*. These nodes are contained in each of the bounding boxes built around the position of $\mathbf{pr}_l$ and $\mathbf{pr}_r$ and denoted $\mathbf{H}_l$ and $\mathbf{H}_r$ respectively, see Fig.3.2.

The holding nodes $\mathbf{H}_l$ and $\mathbf{H}_r$ are mapped to the rigid particles $\mathbf{pr}_l$ and $\mathbf{pr}_r$ respectively, and represent the contact between the mesh and the end-effectors. Since the object is considered firmly gripped, the mapping is supposed rigid: when the particles are displaced, the attached holding nodes are transformed in the same way, in a master/slave fashion.

This is equivalent to defining a *displacement constraint* on these nodes. We refer

as "displacement constraint" the rigid motion (translation and rotation) imposed on a rigid particle attached to holding nodes.

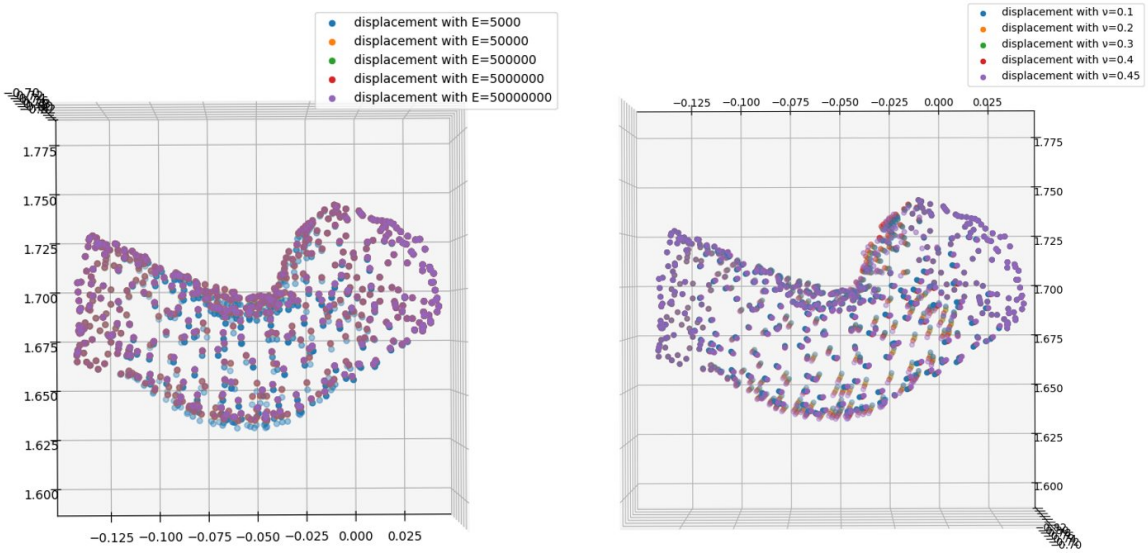
The rest of the mesh is subject to deformation following FEM resolution, since the holding nodes interact with the rest of the object through nodal connections. To simulate the deformation caused by the real end-effectors during manipulation, we relate $\mathbf{pr}_l$ and $\mathbf{pr}_r$ to the position and orientation of the actual robot end-effectors $\mathbf{r}_l$ and $\mathbf{r}_r$. These are retrieved from the robot at each iteration.

Figure 3.3: Simulation setup. The mesh $\mathcal{M}$ is shown in blue. The frames $\mathbf{pr}_l$ and $\mathbf{pr}_r$ represent the the rigid particles related to $\mathbf{r}_l$ and $\mathbf{r}_r$ respectively, acting on the mesh when transformed accordingly to the displacements of the end-effectors.

Note that for small displacements (as are those applied by our controller, see Sec. 2.2.5), the material parameters given as input to the simulator have a small impact on the mesh deformation, unless the object is slacking[2]. Indeed, since the effect of gravity can be neglected (because the object is rigid, or light), the constraints are solely *geometric*. Figure 3.4 shows meshes resulting from simulations, during which a displacement constraint is applied, with different material parameters E and ν.

In this example, the displacement control is applied on the right end-effector, with a span of 0.02 m for the translation and angles of $\frac{\pi}{10}$rad for the rotation on each axis. We try out Young Modulus going from 5000 Pa to 50000000 Pa, resulting in a maximum error (norm L2) between the deformed meshes of 0.05. For the Poisson ratio, we try out values from 0.1 to 0.45, which can correspond to materials such as foams, polymers,

[2]Deformation under the object's own weight



(a) Displacement constraint for different Young Modulus E

(b) Displacement constraint for different Poisson ratio ν

Figure 3.4: Application of the same displacement constraint for different material parameters.

or metals. The maximum RMSE between the deformed meshes after application of a same displacement constraint is 0.06.

These simulations show that for such objects (ones on which gravity can be neglected), the exact knowledge of the material parameters is not necessary to simulate the behavior according to the end-effectors displacement.

The simulation setup presented in this section can be expanded to any objects for which the meshes are known or acquired beforehand. We will now present a method to reconstruct the meshes of specific objects: DLO.

3.3 Tools for mesh construction

To implement simulations for an object, it is necessary to first construct its mesh. Two meshes are necessary for each object:

- a triangle mesh, $\mathcal{M}^v$
- a tetrahedral mesh, $\mathcal{M}$

The triangle mesh is the visual model, representing the *external surface* of the mesh. It is mapped to the tetrahedral mesh, which represent the *volume* of the object and is the one used for the FEM computations. For the simulations to run fast, it is necessary to keep the number of tetrahedra not too high, while also making sure to have enough

tetrahedra so that the mesh isn't too stiff. A high level of detail is not necessary for $\mathcal{M}$, but $\mathcal{M}^v$ should be more detailed, since it represents the visual aspect of the object.

In this section, we will first review visual volume reconstruction. Then, we will present a method for volume reconstruction of *Deformable Linear Objects*, that are objects that can be reduced to a single dimension and referred as DLO in the following, in Section 3.3.2. We will then present tools to automatically construct a mesh given two types of reconstructed volumes: one resulting from our method for reconstruction of DLO, or reconstructed point cloud.

3.3.1 3D volume reconstruction

Volume reconstruction aims at building the three-dimensional model of a scene, object, or person, from observations.

An overview of 3D reconstruction algorithms can be found in recent state of the art [ZSG+18]. In term of static scenes, multi-view reconstruction has interested many works in the field of robot navigation for instance, such as [NZIS13], [CZK15] or SLAM (Simultaneous Localization And Mapping) [NIH+11], [MC13], [LLW+17], [LCL19].

Complex models reconstruction such as that of humans in dynamic settings has been addressed in many works. The authors of [DFF13] propose a non-rigid matching algorithm which aligns 3D observations (partial point clouds) of moving objects, by using both geometry and texture measurements, to construct a model which improves as observations are added. They combine dense point cloud alignment with color consistency, formulate the problem by energy minimization, and solve it with a gradient descent method. The fusion of 3D data is done using a 3D representation that adds two fields: a Signed Distance Field (SDF) and a direction field pointing at the nearest points on the surface. Later, [DKD+16] presents a capture system that uses 24 cameras and multiple GPUs.

The authors of [SBCI17] present an incremental reconstruction method from partial scans, based on level set method ([OFP04]), minimizing an energy which takes into account the geometry of the SDF as well as the deformation field. [NFS15] proposes a SLAM system which reconstructs non-rigidly deforming scenes, by estimating a volumetric warp field that transforms the canonical model space into the current frame.

The mentioned works all consider either complex, detailed scenes, or actively deforming objects. Taking into account our robotic system already grasping the object to be manipulated, simpler methods can be considered. Scanning methods as in [TZL+12] show good result of model reconstruction by using multiple RGB-D cameras to collect different views of the subject. [SZY+18] uses a turntable to scan different views of the object to reconstruct.

3.3.2 DLO reconstruction through parametrization

In order to construct a mesh to represent DLOs, we choose to model a DLO as a cylinder, parameterized by its radius and length.

Many works consider the matter of DLO state estimation, usually representing DLOs as 1D objects. In [CZGP22], deep learning is used to extract features of a DLO through spline modeling. Based on the RGB-D data, [ZFH16] approximates the DLO with Bézier curves as a chain of connected rectangles. The authors of [GPCB19] propose a solution to reconstruct a 3D curve from a 2D image and a 1D template. [WY23] proposed a method to extract the DLO skeleton from depth images. [MX23] conducts rope diameter estimation by the mean of different image processing steps. The authors enclose the rope area in contact with a segmented rod with a minimum area rectangle, whose width is taken as the diameter of the rope.

We propose a few image processing steps in order to extract the desired parameters (radius and length) of the considered DLO from camera feedback. The DLO is already grasped between the end-effectors, and entirely in the frame of the RGB-D camera. First, the pixels representing the DLO are segmented from the RGB image, Fig. 3.5(a). The largest contour $\mathbf{c}^{dlo}$ is extracted (Fig. 3.5(b)), and considered as the main DLO body.

Length estimation: a B-Spline interpolation is conducted on this contour (Fig. 3.5(c)), to obtain a set of 2D points constituting the DLO curve. These points are related to their closest pixel on the RGB image, giving the set of pixels of coordinates (i^{dlo}, j^{dlo}), and then projected into the 3D space using the corresponding depth $d(i^{dlo}, j^{dlo})$ and the camera intrinsic parameters. We obtain the set of 3D points $\mathbf{pc}^{dlo} \in \mathbb{R}^d \times 3$ (see Fig. 3.5(d)). The length of the DLO is then computed as:

$$l = \sum_{k=0}^{n-1} \|\mathbf{pc}_{k+1}^{dlo} - \mathbf{pc}_{k}^{dlo}\|. \tag{3.18}$$

Radius estimation: we find the largest circle fitting in the contour $\mathbf{c}^{dlo}$, see Fig.3.6(a), (b). The pixels contained in the intersection between this circle and $\mathbf{c}^{dlo}$ are projected in 3D (Fig. 3.6), also using the corresponding depth $d(i^{dlo}, j^{dlo})$ and the camera intrinsic parameters and giving the set of 3D points $\mathbf{pc}^{rad}$. The Euclidean distance between each point in $\mathbf{pc}^{rad}$ is computed as $\mathbf{D}$, and the radius is defined as:

$$R = max(\mathbf{D})/2. \tag{3.19}$$

We obtain the parameters of the DLO, (R,l). Table 3.1 presents the result of the DLO parameters estimation from visual feedback for different objects.

This results give a mean error of 7% for the radius and 2% for the length. The error maximum of 15% and minimum of 0% for the radius. The error maximum of 4.5% and minimum of 0.8% for the length.

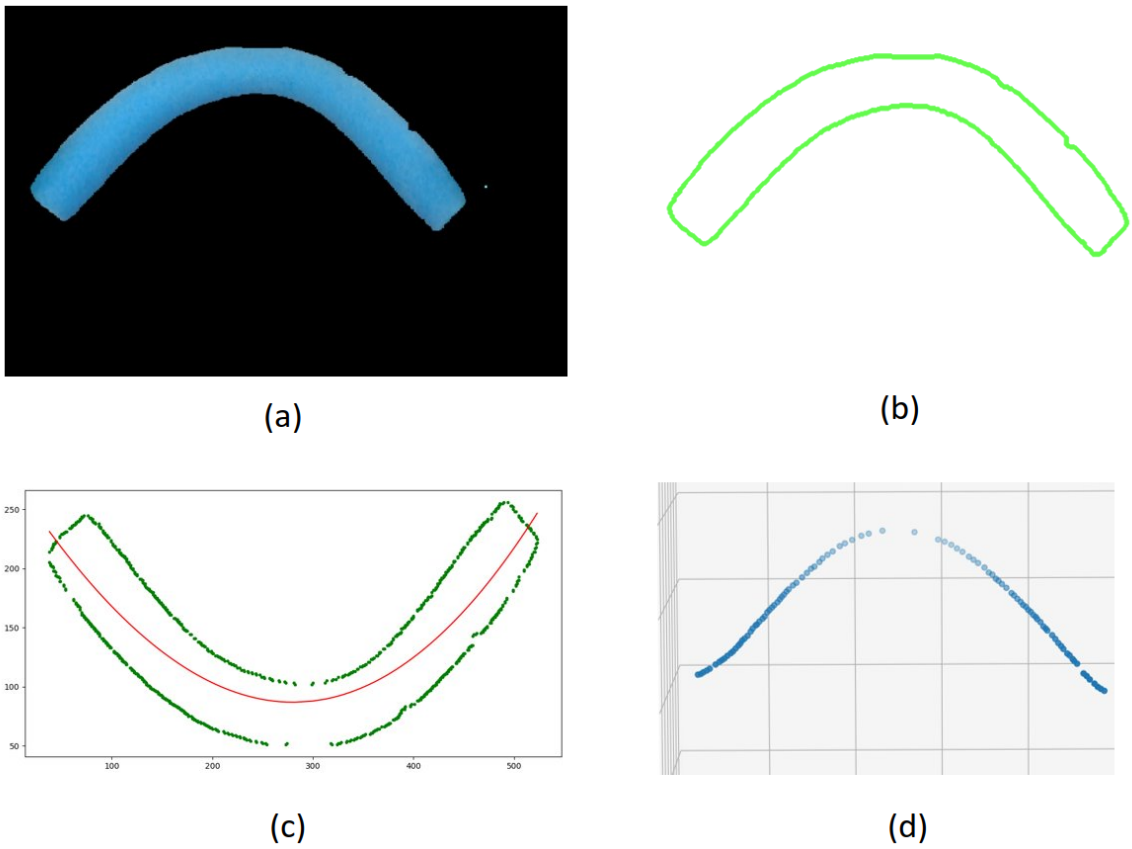


Figure 3.5: Extraction of the 3D points constituting the DLO for length estimation. The body of the DLO is segmented (a), then the largest contour is found (b). B-spline approximation gives the 2D curve (c) and is projected in 3D to give $\mathbf{pc}^{dlo}$ *(d).*

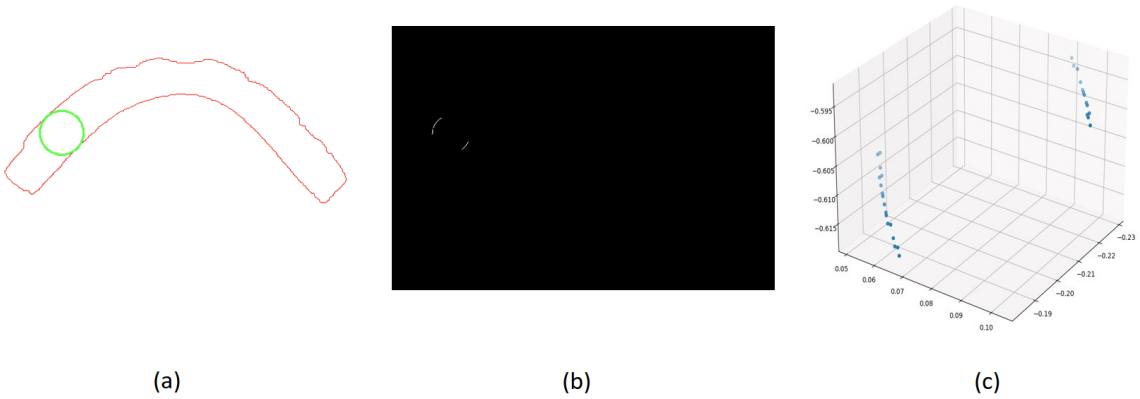


Figure 3.6: Image processing for radius estimation. The largest circle (green) fitting in the contour (red) is found (a), and the mask of their intersection is created (b). These pixels are projected in 3D (c) and used to estimate the radius of the DLO.

3.3.3 Meshing from DLO parameters

In the case of DLO, we use the python library *gmsh*[3] to generate a cylinder from parameters (R, l) and to create the triangle and tetrahedral meshes, as shown in Fig.

[3]https://gmsh.info/

Object	real radius	estimated radius	real length	estimated length
Rope	3	3	525	530
Foam noodle 1	30	29	630	625
Foam noodle 2	30	27	510	487
HIDRIA part	20	17	355	350

Table 3.1: Result of the DLO parameters estimation on different objects, values in mm. The real values were measured by hand. The HIDRIA part is the one presented in Fig.2

.

3.7.

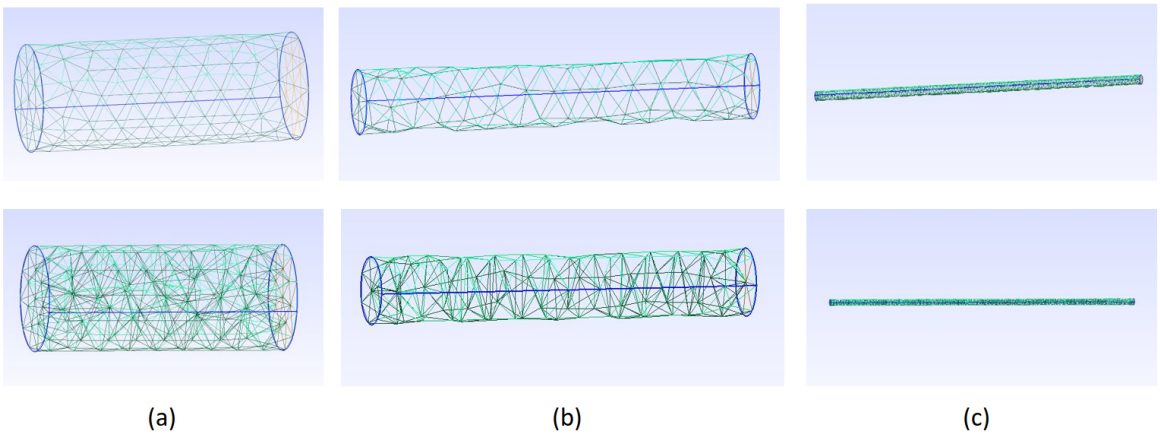


Figure 3.7: DLO meshes generated for different pairs of parameters (R, l): (a)$(R, l) = (0.1, 0.5)$, (b)$(R, l) = (0.05, 0.7)$, (c)$(R, l) = (0.005, 0.5)$. Top row shows the visual meshes $\mathcal{M}_0^v$ and bottom row the tetrahedral meshes $\mathcal{M}_0$.

The obtained meshes are cylinders, whereas the DLO may be deformed and therefore have a non-cylindrical configuration in its observed state, as seen in Fig.3.7. Hence, we need to conduct an extra step to align the meshes with to the current observed DLO.

To this end, we subsample the 3D points constituting the DLO curve $\mathbf{pc}^{dlo}$ (see Fig.3.5 (d)) down to k_s samples. The length between sample $\mathbf{p}_k$ and $\mathbf{p}_{k-1}$ is denoted $dl(k)$, $k \in [0, k_s]$ and it is calculated using all the points in $\mathbf{pc}^{dlo}$ in between. We take $dl(0) = 0$ for the first point in $\mathbf{pc}^{dlo}$. The vector between sample $\mathbf{p}_k$ and the next point in $\mathbf{pc}^{dlo}$ is also computed and expressed as a rotation from the x-axis unit vector, and denoted $q(k)$, see Fig.3.8 (a). The last sample $\mathbf{p}_{k_s}$ is the very last point in $\mathbf{pc}^{dlo}$, and its rotation $\mathbf{q}(k_s)$ is calculated with the closest point in $\mathbf{pc}^{dlo}$.

A FEM simulation scene is created with the cylindrical meshes. The tetrahedral mesh of the DLO, $\mathcal{M}_0$, is also sampled: we create k_s rigid particles. The rigid particle $\mathbf{pr}_k$ is created at length $dl(k)$ from the axis of the cylinder, see Fig.3.8 (b). Each rigid particle is then rigidly attached to the mesh nodes around it, similarly to the end-effectors in section 3.2 which are rigidly mapped to the holding nodes. Finally,

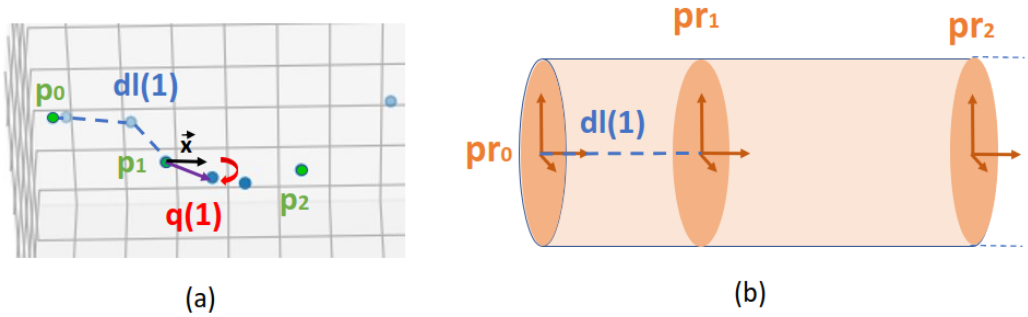


Figure 3.8: Sampling for alignment between the mesh in resting position and the current configuration of the DLO. (a) shows $\mathbf{pc}^{dlo}$ *sampled into* k *points (in green). Each sample* $\mathbf{p}_k$ *is associated to the length separating it from the previous sample,* $dl(k)$*, and to a rotation from the* x*-axis,* $\mathbf{q}(k)$*. (b) shows the corresponding sampling of the mesh* $\mathbf{m}$ *into rigid particles* $\mathbf{pr}_k$*.*

translation to the corresponding sample $\mathbf{p}_k$ and rotation $\mathbf{q}(k)$ is applied to each rigid particle $\mathbf{pr}_k$ to align it with the current configuration of the DLO. The FEM simulation is run, deforming the mesh parts which are not rigidly mapped to any rigid particle accordingly, until quasi-equilibrium is reached. The resulting meshes $\mathcal{M}_{cam}$ and $\mathcal{M}^v_{cam}$ are obtained. Examples of the resulting alignment of DLO meshes are compared to the camera view in Fig.3.9.

The number of samples k_s is tuned depending on the length and shape of the DLO: k_s small for short DLO or smooth curvatures as the foam noodles, and k_s bigger for thin and long objects like the rope. It should also be noted that in case of thin objects, depth camera precision may cause the depth map of the DLO to be incomplete - and possibly miss points on the reconstructed point cloud. This can cause some parts of the DLO not to be aligned, as observed on Fig.3.9(d).

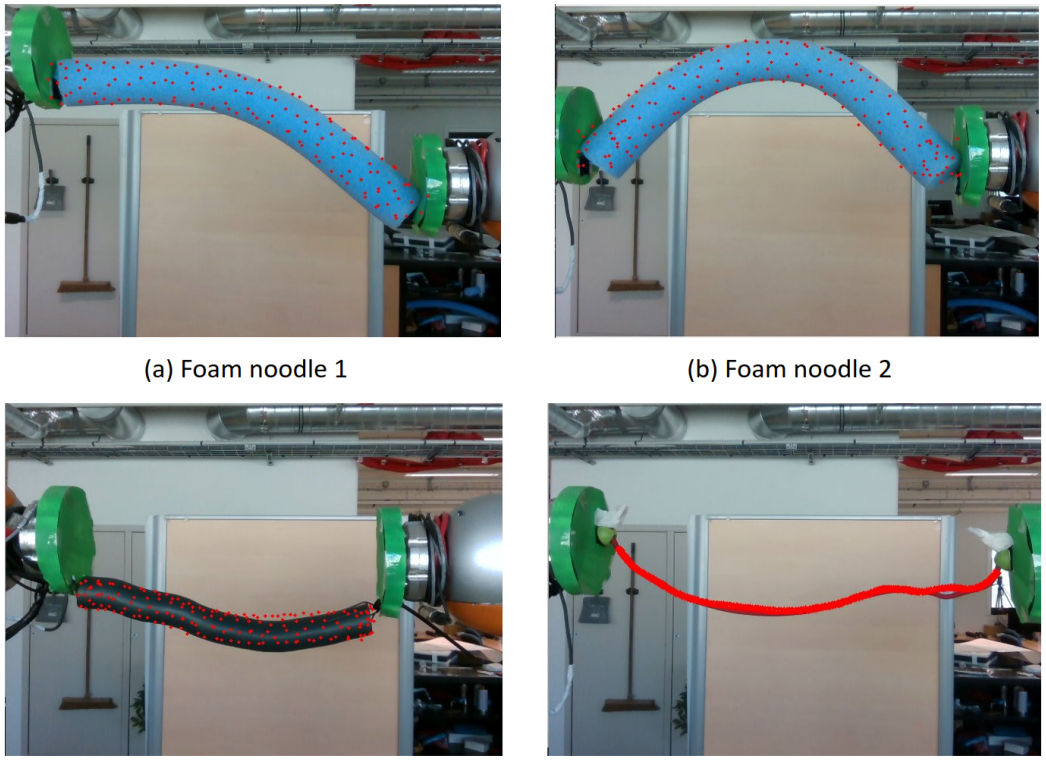
3.3.4 Creating a mesh from reconstructed point cloud

We now present how to use some existing libraries in order to create meshes for objects other than DLO. It consists in building meshes from reconstructed surface in the form of points clouds. This method can be applied to any point cloud of the full volume of the object, which could also be obtained by other methods [SQL22].

In our case, we only reconstruct a partial surface point cloud of the our different objects by acquiring different views of the objects grasped by the robot, through rotation. The reconstruction method described bellow is only possible because of these conditions:

1. The object is in *stable* configuration; it is either light or rigid enough for gravity to be neglected compared to other external forces, i.e. the shape of the object does not change, as long as the force applied by the robot is constant.

2. The object presents faces, i.e. is *non-spherical*.



(a) Foam noodle 1 (b) Foam noodle 2

(c) HIDRIA mechanical part (d) Rope

Figure 3.9: Result of the DLO mesh alignment for different objects. The triangle mesh nodes after alignment are projected on the corresponding RGB image, in red.

3. The back side of the object is considered as approximately planar.

We consider the object already grasped by the robot end-effectors, that are placed such that the effectors' normals are parallel. We can reconstruct the geometry of the object, while avoiding the use of extra tools or cameras, is to "scan" the object, by rotating it around the axis of the effectors normal, $\mathbf{x}_e$. A single, static RGB-D camera placed in front of the object observes the different views at each rotation increment during the open-loop manipulation.

In open-loop, the end-effectors rotate by an angle $\theta < \frac{\pi}{2}$, see Fig. 3.10(a). They then rotate back, so the object returns to its initial position, Fig. 3.10(b). The reversed rotations of $-\theta$ are then performed, Fig. 3.10(c), before returning to the initial position once more. This sort of scan is possible because of *condition* 1, ensuring that the object does not deform significantly in between views. Existing non-rigid registration algorithms would remedy this point.

The point cloud of the object is extracted from the collected views. Iterative Closest Point (ICP) algorithm is used to rigidly transform the different rotated point clouds, to align them to the final point cloud of the object in initial position $\mathbf{pc}^0$, thus reconstructing different faces of the object. Without *condition* 2, ICP would not be able

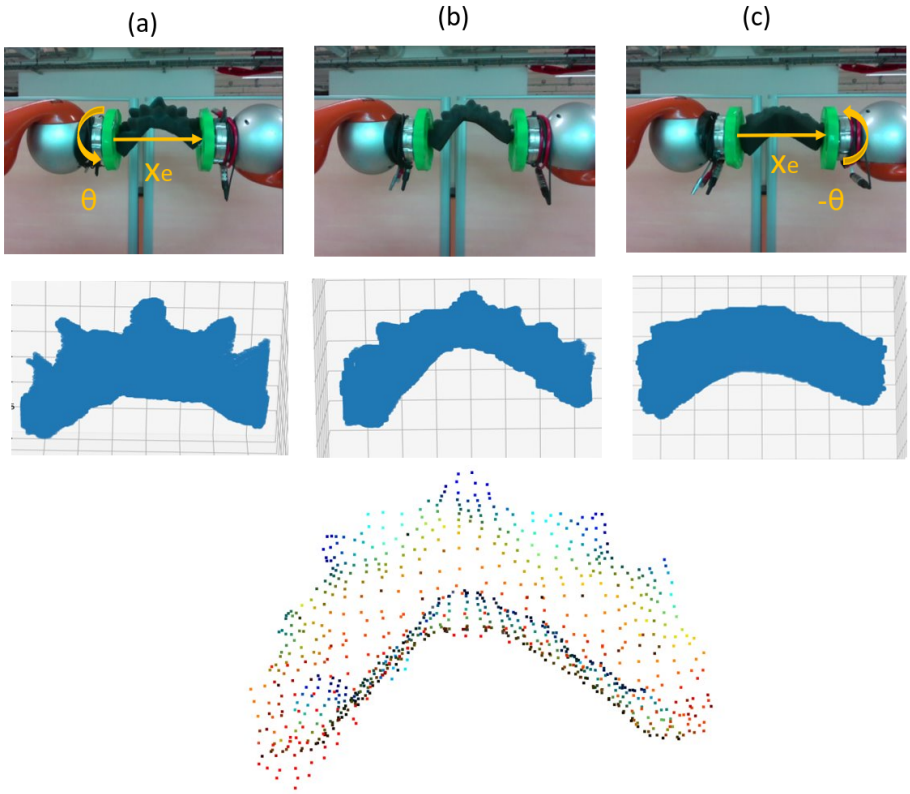


Figure 3.10: Multi-view point cloud reconstruction. The first row shows the RGB camera feedback, the second row shows the extracted point cloud of the object, and the last row shows the reconstructed point cloud. First, the robot rotates the object by an angle θ around the axis of the effectors $\mathbf{x}_e$ (a), rotates it back to the original position (b), then rotates it about an angle $-\theta$ (c).

to construct the surface properly and stack the different views together. *Condition* 3 allows for a partial point cloud of the external surface to be sufficient to construct the meshes, as the surface reconstruction algorithm (presented in the following) give good results even with holes in the point cloud.

This method is very specific to the objects we considered in this chapter but proved to be satisfying. For objects not fulfilling these conditions, any other existing point cloud reconstruction method can also be used.

Once the point cloud is obtained, we first run ICP again to align the reconstructed point cloud with the current observed point cloud of the object $\mathbf{pc}^0$: the reconstructed is then in the camera frame, and denoted $\mathbf{pc}_{cam}$. Then is performed surface reconstruction using Alpha shapes ([EKS83]). We perform it twice, with different trade-off parameters:

- one to obtain a reconstruction as detailed as possible, which will give the triangle mesh $\mathcal{M}^v_{cam}$;
- another with a lower level of detail, $\mathcal{M}_{cam}$, to obtain a smaller number of nodes

allowing computations to run fast enough.

The least detailed reconstruction is tetrahedralized using the *CGAL*[4] engine, giving $\mathcal{M}_0$.

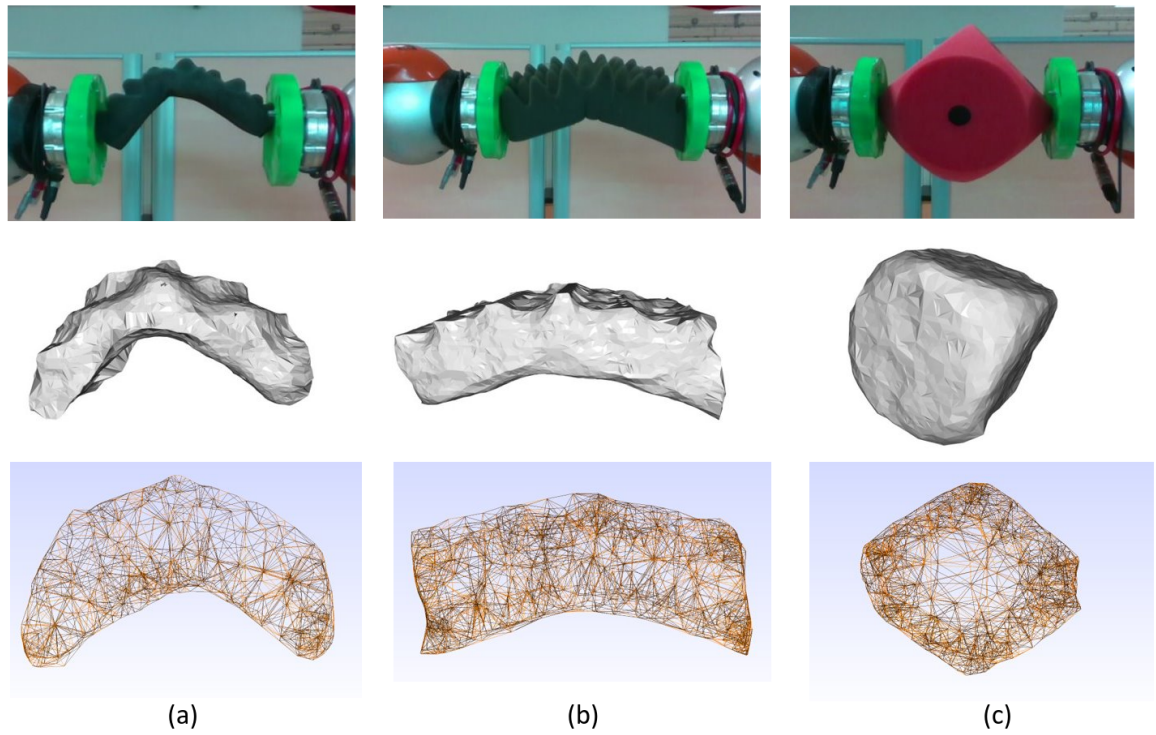


Figure 3.11: Triangle (middle row) and tetrahedral (bottom row) meshes reconstructed from point clouds for different objects. (a) dented foam 1, (b) dented foam 2, (c) foam cube.

3.4 Conclusion

In this chapter, we proposed a simulation setup for estimating in real-time the deformation of an object according to the robot end-effectors displacement, through the use of a model, both geometric (a template) and mechanical (FEM). Knowing the initial tetrahedral and triangle meshes of the object, we are able to apply the motions of the end-effectors to the object in simulation and estimate the resulting deformations of the object.

We also presented a simple volume reconstruction method for DLOs, which take advantage of the dual-arm robot setup. We also proposed tools to build the meshes of the reconstructed DLO volume, as well as for reconstructed point cloud. To summarize, once the object is grasped by the end-effectors, the model construction consists in:

[4] https://www.cgal.org/

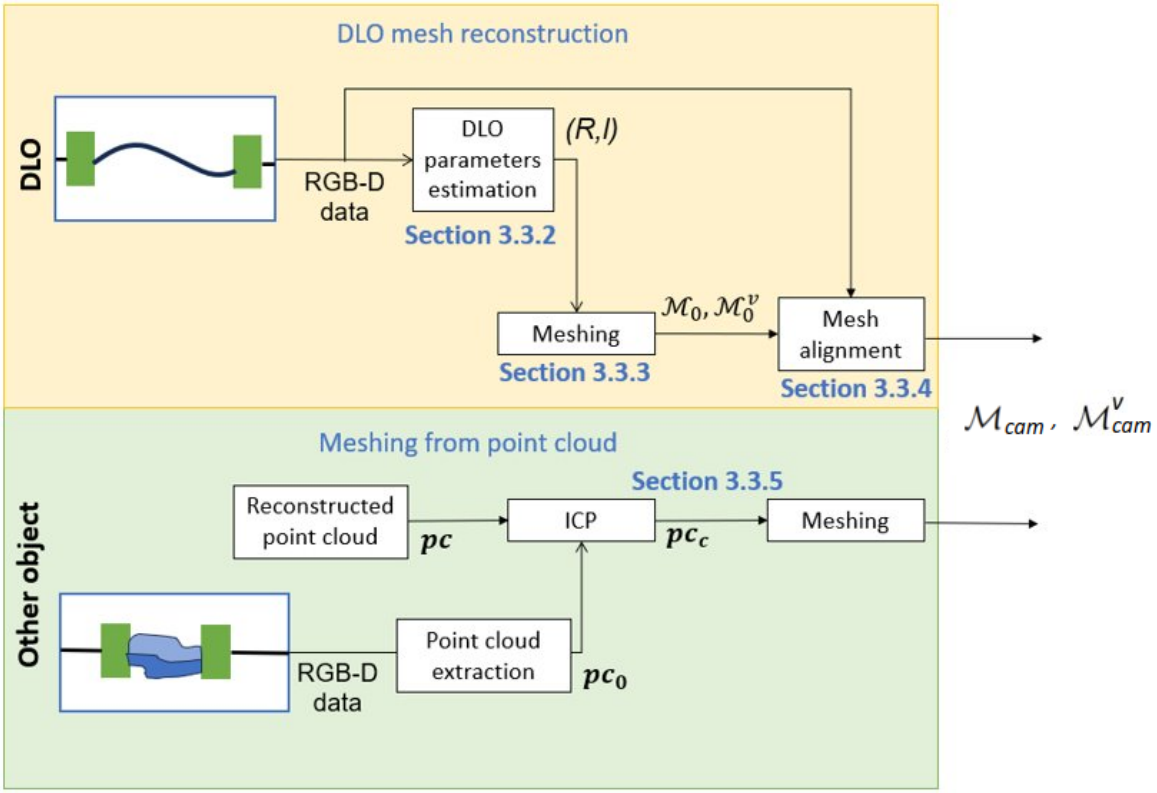


Figure 3.12: Model estimation process: mesh construction.

1. Volume reconstruction:
   - for DLOs: parameters estimation
   - for other objects: point cloud acquisition
2. Meshing: creation of tetrahedral and triangle meshes.
3. For DLOs: alignment of the meshes with the visual feedback: sections of the mesh are aligned to sample points extracted from the camera feedback beforehand.

The process is illustrated in Fig. 3.12.

In the next chapter, we will present how to use the presented simulations in a robot control scheme for deformable objects shape servoing.

CHAPTER 4

Dual-arm shaping of soft objects in 3D based on visual-servoing and FEM simulations

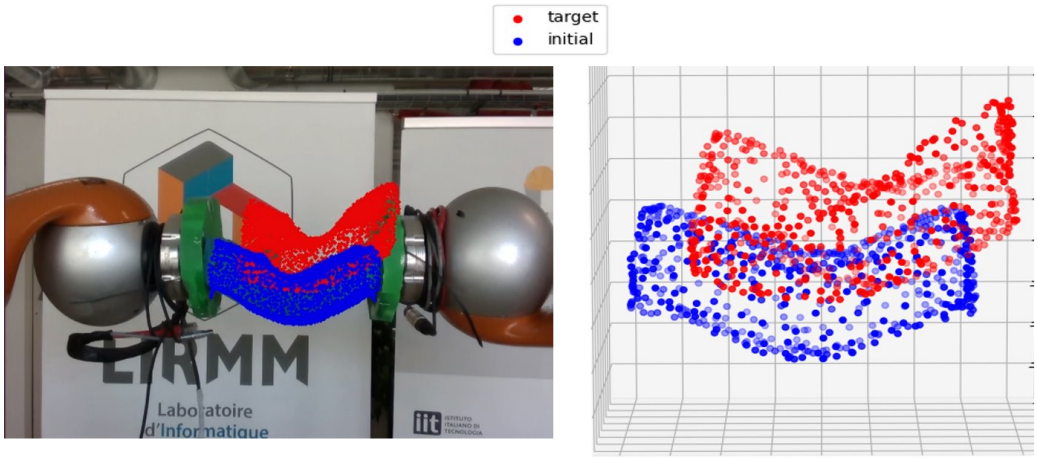


Figure 4.1: The goal of the work presented in this chapter is to control a dual arm robot so that the initial mesh (blue) reaches the target mesh (red) in three dimensions.

We propose a servoing scheme that is vision and model-based, using the object's template as well as a linear elastic mechanical model. The template is then used as the object geometric representation, and by running physics-based simulations in the control loop, it is then possible to estimate the deformation of the object resulting from the robot motions. The visual feedback is also included in the loop in order to ensure that the estimation matches with the observed behavior of the object.

In the previous work, the robot motions were not restricted to plane motions. However, the proposed method could only control the projections in 3D of the observed surface. In this chapter, we build on top of the control scheme developed in 2, by now taking into account the whole volume of the manipulated object - and not only its visible surface - thanks to the principles and tools introduced in 3. The proposed framework is then model *and* vision-based.

We now represent the manipulated soft object with a mesh, and track its deformation through the robot and RGB-D camera feedback. The shaping task then involves deforming the mesh from one initial configuration to a target one, as shown in the example of Fig.4.1.

4.1 Motivations

We aim to track and control the deformation of an entire object, in $\mathbb{SE}(3)$. In our previous work (chapter 2), we solely use projected contours to describe the geometry of the manipulated object. The major drawback of this is that only the visible part of the object is tracked, and therefore the hidden part of the object is not controlled. Instead, in this chapter, we implement real-time FEM simulations during manipulation, to track the deformation of the full volume of the object, and to relate it to visual feedback.

The benefits are:

- We obtain a consistent topology of the whole object, as introduced in chapter 3. Such a representation allows us to implement a Jacobian estimation as in Chapter 2. Yet, instead of considering only the contours of the visible surface, as in Chapter 2, we now also consider the self-occluded parts of the object;

- We can use simulations to collect the data-set initially needed to associate robot pose variation and corresponding shape variation. This avoids having to control the robot and object in open loop, with the risk of breaking either or the other;

- We can use the simulations to determine the target shape: it is not required to reach the target shape manually (by moving the robot or the real object) beforehand. We introduce an intuitive and user-friendly target shape selection process.

However, accurate mechanical models are complex to define. They depend on the material properties and are computationally expensive. Instead of trying to estimate a precise mechanical model, we use linear elasticity as a simple approximation of the mechanical behavior of the manipulated objects, and use the information provided by the robot end-effectors to run simple and efficient simulations. The simulations allow to estimate the deformation of the whole object, including of its occluded parts, according to the end-effectors position and orientation.

4.2 Overview of the framework

In contrast with the previous work, we use simulations to track in real time the shape of the object. A mesh model of the manipulated object is either known, or reconstructed from visual feedback with the method presented in 3.3, when applicable (DLO, or reconstructed point cloud). The material parameters of the object are not necessarily needed for manipulation, as the constraints are considered as *geometric* (see section 3.2.2) but they can be used to monitor the deformation.

At each iteration, we perform a simulation, to deform the mesh of the object according to the displacements of the robot end-effectors, as presented in section 4.6.4. We use the tetrahedral mesh nodes to represent the current object shape. The point cloud of the object is extracted from the RGB-D camera, and used to correct the simulated mesh, as detailed in section 4.7.2. The target shape, denoted $\mathbf{m}^*$, is also composed of mesh nodes and it is chosen in the simulation, via keyboard commands. Our goal is to generate a sequence of commands for the dual-arm robot to deform and drive the current mesh $\mathbf{m}$ to the target mesh $\mathbf{m}^*$.

Since the representation as tetrahedral mesh has a high dimension wrt the number of robot DOF, we encode the mesh in a smaller feature vector $\mathbf{s} \in \mathbb{R}^k$ with $k << n$ (n indicating the number of mesh nodes). As presented in chapter 2, we consider that the mapping between feature variation $\delta\mathbf{s}$ and robot pose variation $\delta\mathbf{r}$ is linear, through the interaction matrix $\mathbf{L} \in \mathbb{R}^{k\times k}$:

$$\delta\mathbf{s} = \mathbf{L}\delta\mathbf{r} \in \mathbb{R}^k \tag{4.1}$$

In the rest of the chapter, we assume the following:

- The mesh of the object is known, or in case of a DLO, we have constructed it, following our method prior to the manipulation.
- The color of the object is different enough of that of the end-effectors to be segmented separably.

The control framework is similar to the one presented in chapter 2, and presented in Fig.4.2. However, the tetrahedral mesh nodes, $\mathbf{m}$ are used instead of 3D contours to represent the shape of the manipulated object:

$$\mathbf{m} = \begin{bmatrix} m_{1,x} & m_{1,y} & m_{1,z} \\ \vdots & \vdots & \vdots \\ m_{n,x} & m_{n,y} & m_{n,z} \end{bmatrix} \in \mathbb{R}^{n\times 3} \tag{4.2}$$

In addition, we use the triangle mesh nodes $\mathbf{m}^v$ to compare the simulations with the visual feedback. First, the simulation is set up according to the current state of

the robot and object, in the setup which we will detail in section 4.3.

Instead of implementing an open-loop control phase, to collect an initial sequence of shapes with corresponding robot poses, we collect these through simulations, as will be detailed in section 4.4. We then obtain the matrices containing the resulting mesh nodes and robot pose variations, $\mathbf{M}$ and $\Delta\mathbf{R}$ respectively.

In order to be stored in $\mathbf{M}$, the mesh nodes $\mathbf{m}$ are arranged in a single dimension vector $\check{\mathbf{m}}$:

$$\check{\mathbf{m}} = [m_{1,x}, \cdots, m_{n,x}, m_{1,y}, \cdots, m_{n,y}, m_{1,z}, \cdots, m_{n,z}]^\top \in \mathbb{R}^{3n}. \tag{4.3}$$

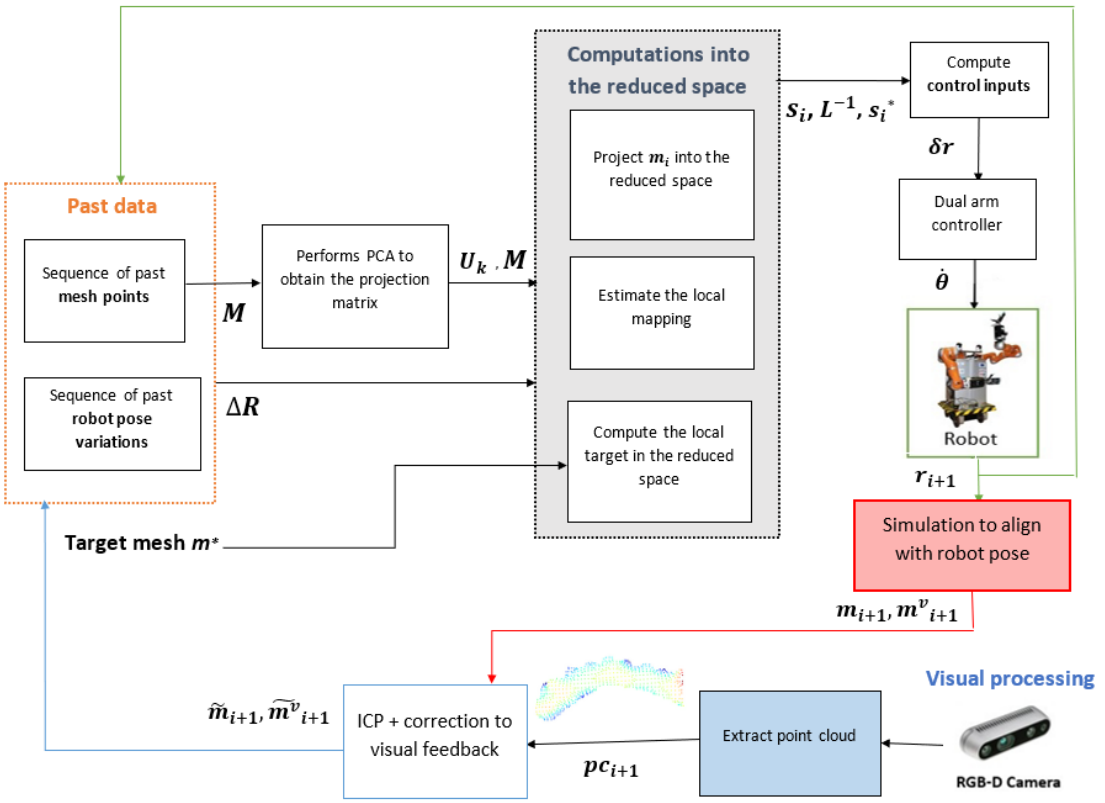


Figure 4.2: Overview of our framework at each iteration i. Given a sequence of past mesh nodes $\mathbf{m}$ and corresponding robot poses $\mathbf{r}$, composing matrices $\mathbf{M}$ and $\Delta\mathbf{R}$ respectively, we apply PCA on $\mathbf{M}$, to obtain projection matrix $\mathbf{U}_k$. This is then used to reduce the dimension of $\mathbf{M}$, to estimate $\mathbf{L}^{-1}$, linearly mapping the features variation to the robot pose variation. With $\mathbf{L}^{-1}$, we compute the robot input $\delta\mathbf{r}$ driving current shape $\mathbf{s}_i$ to intermediary target $\mathbf{s}_i^*$ in the reduced space. The robot input is sent to the dual-arm robot controller to obtain the joint command $\dot{\theta}$. Once the command is achieved, the new robot pose $\mathbf{r}_{i+1}$ is sent to the simulator, and the simulation runs until quasi equilibrium is reached. The resulting triangle mesh nodes $\mathbf{m}^v_{i+1}$ are compared to the point cloud of the object $\mathbf{pc}_{i+1}$ obtained via visual processing, through ICP. If the resulting error is higher than the acceptable threshold, a step of correction to visual feedback is conducted. The past data matrices are updated and everything is repeated at iteration $i+1$.

We extract features and compute control inputs as in chapter 2, by conducting PCA on the mesh nodes matrix, to obtain the projection matrix $\mathbf{U}_k$, encoding the features in the reduced space of k highest variance eigen values. Both the current and intermediary target mesh nodes $\check{\mathbf{m}}_i$ and $\check{\mathbf{m}}_i^*$ are projected to encode the shape into a smaller number of k features, giving $\mathbf{s}_i$ and $\mathbf{s}_i^*$ respectively. We estimate the inverse interaction matrix $\mathbf{L}^{-1}$, mapping the robot pose variation to the shape variation, and use it to compute the control input $\delta\mathbf{r}$ driving the current shape to the intermediary target. The dual-arm controller computes the corresponding robot joint commands, $\dot{\theta}$ to achieve the motion.

Once the command is achieved, the new robot poses $\mathbf{r}_{i+1}$ are retrieved and sent to the simulator. The rigid particles $\mathbf{pr}_r$ and $\mathbf{pr}_l$ are displaced to correspond to the new robot poses. The simulation runs a few steps until a state of quasi-equilibrium of the mesh is reached. This equilibrium configuration yields the new mesh nodes $\mathbf{m}_{i+1}$ and $\mathbf{m}^v_{i+1}$. Meanwhile, the point cloud of the new state of the object $\mathbf{pc}_{i+1}$ is retrieved through visual processing, as we will explain in section 4.7.1.

The meshes resulting from the simulation are compared and corrected to $\mathbf{pc}_{i+1}$ if necessary. This step will be detailed in section 4.7.2. The data matrices are updated using the corrected mesh points $\tilde{\mathbf{m}}_{i+1}$ and robot poses at every iteration i. The process is repeated until the target shape is considered reached. The whole process is summarized in algorithm 2.

4.3 Robot setup, simulations and reference frames

Before starting the manipulation, the robot firmly grasps the object. A RGB-D camera, in our case the Intel Realsense D435, is fixed on the robot, and looks at the object.

The meshes of the object may be already known, or else be reconstructed as explained in section 3.3 in the case of DLOs or for objects reconstructed from point cloud.

In any case, the meshes must represent the current state of the object. The meshes denoted $\mathcal{M}_{cam}$ and $\mathcal{M}^v_{cam}$ are expressed in the camera frame $\mathcal{F}_{cam}$. Since the camera position and orientation in the robot frame $\mathcal{F}_{rob}$ is known, the transformation from the camera frame to the robot frame $\mathcal{T}^{rob}_{cam}$ is obtained, see Fig. 4.3.

Both tetrahedral and triangle meshes are transformed into the robot frame, and will be denoted simply $\mathcal{M}$ and $\mathcal{M}^v$ for clarity. Changing the frame of the mesh consists in transforming all their nodes:

$$\mathbf{m} = \mathcal{T}^{rob}_{cam}\mathbf{m}_{cam} \tag{4.4}$$

and similarly,

$$\mathbf{m}^v = \mathcal{T}^{rob}_{cam}\mathbf{m}^v_{cam}. \tag{4.5}$$

Algorithm 2 Control framework

procedure INITIALIZATION
- $\mathbf{R} = \{\}$
- $\mathbf{M} = \{\}$
- $j = 0$
- **while** $j \leq D$ **do**
  - Generate pose $\mathbf{r}_j$ and apply in simulation ▷ section 4.4
  - $\mathbf{M} \xleftarrow{+} \check{\mathbf{m}}_j$
  - $\mathbf{R} \xleftarrow{+} \mathbf{r}_j$
  - $j \leftarrow j + 1$
- Construct $\boldsymbol{\Delta}\mathbf{R}$ ▷ section 4.4
- **return** $\boldsymbol{\Delta}\mathbf{R}$, $\mathbf{M}$

procedure CONTROL LOOP
- Define the target $\mathbf{m}^*$ ▷ section 4.5
- $\check{\mathbf{m}}_i = \check{\mathbf{m}}_D$
- $e_i = ||\check{\mathbf{m}}^* - \check{\mathbf{m}}_i||$
- **while** $e_i \leq 0.1$ **do**
  - $\mathbf{U}_k = PCA(\mathbf{M})$ ▷ section 4.6.1
  - $\mathbf{s}_i = project(\check{\mathbf{m}}_i, \mathbf{U}_k)$
  - $\mathbf{L}^{-1} = interactionMatrix(\boldsymbol{\Delta}\mathbf{R}, \mathbf{M}, \mathbf{U}_k)$ ▷ section 4.6.2
  - $\mathbf{s}_i^* = localTarget(\check{\mathbf{m}}^*, \check{\mathbf{m}}_i, \mathbf{U}_k)$
  - $\boldsymbol{\delta}\mathbf{r}_i = poseCommand(\mathbf{L}^{-1}, \mathbf{s}_i^*, \mathbf{s}_i)$ ▷ section 4.6.3
  - $\dot{\boldsymbol{\theta}} = robotJointsCommand(\boldsymbol{\delta}\mathbf{r})$ ▷ section 4.6.3
  - Apply $\dot{\boldsymbol{\theta}}$ to robot
  - Get $\mathbf{pc}_{i+1}, \mathbf{r}_{i+1}$ from camera feedback and robot respectively ▷ section 4.7.1
  - $\mathbf{m}_{i+1} = displacementConstraint(\mathbf{m}_i, \mathbf{r}_{i+1})$ ▷ in simulation, section 4.6.4
  - $fit_i = ICP(\mathbf{pc}_{i+1}, \mathbf{m}^v_{i+1})$ ▷ section 4.7.2
  - **if** $fit_i < threshold$ **then**
    - $\tilde{\mathbf{m}}_{i+1}, \tilde{\mathbf{m}}^v_{i+1} = correction(\mathbf{m}_{i+1}, \mathbf{m}^v_{i+1}, \mathbf{pc}_{i+1})$ ▷ section 4.7.2
  - **else**
    - $\tilde{\mathbf{m}}_{i+1}, \tilde{\mathbf{m}}^v_{i+1} = \mathbf{m}_{i+1}, \mathbf{m}^v_{i+1}$
  - Update $\boldsymbol{\Delta}\mathbf{R}$, $\mathbf{M}$
  - $e_i = ||\check{\mathbf{m}}^* - \check{\mathbf{m}}_i||$

The object meshes are then imported into the simulator, which operates in the robot frame. Since the object is gripped by the robot arms, the rigid particles $\mathbf{pr}_l$ and $\mathbf{pr}_r$ representing each end-effectors are created according to the current end-effector poses $\mathbf{r}_l$ and $\mathbf{r}_r$. In contrast with the work presented in chapter 2, here the dual arm robot is *not* controlled following the cooperative tasks space: instead, each end-effector (left and right) is controlled independently and denoted by the letter l and r respectively.

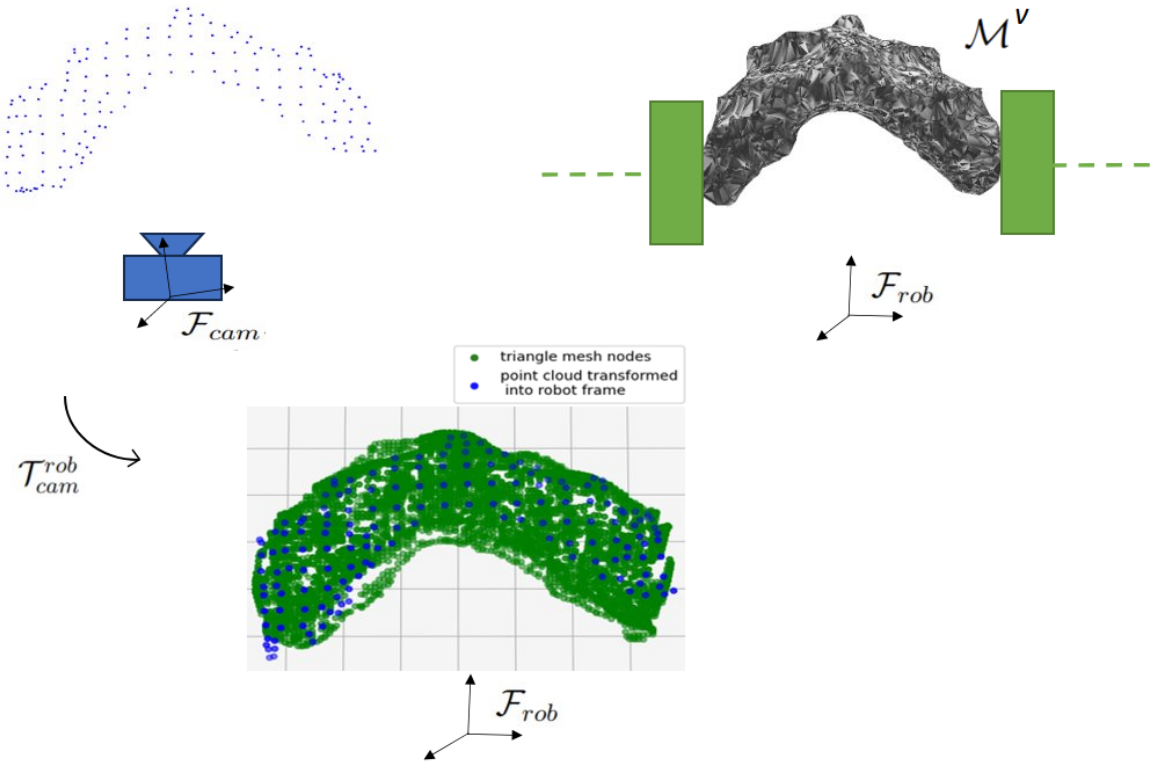


Figure 4.3: Camera frame $\mathcal{F}_{cam}$ and Robot frame $\mathcal{F}_{rob}$

We made this choice since it is more direct to use the same reference frame ($\mathcal{F}_{rob}$) for both effectors when applying the robot motions in the simulation. It also seems more intuitive in the graphic interface for keyboard control (see section 4.5).

These particles are attached to the holding nodes $\mathbf{H}_l$ and $\mathbf{H}_r$ of the tetrahedral mesh, as described in section 3.2.2. A rigid particle is defined by a position and an orientation:

$$\mathbf{pr}_{ref} = [x_{ref}, y_{ref}, z_{ref}, \mathbf{q}_{ref}] \tag{4.6}$$

With $\mathbf{q}$ a quaternion, and corresponds to a robot pose:

$$\mathbf{r}_{ref} = [x_{ref}, y_{ref}, z_{ref}, \boldsymbol{\rho}_{ref}] \tag{4.7}$$

with $\boldsymbol{\rho}$ the angle-axis vector representation of the end-effector orientation, $ref = \{l, r\}$ for the left and right end-effector respectively. The dual-arm robot pose $\mathbf{r}$ is then defined as:

$$\mathbf{r} = [\mathbf{r}_l, \mathbf{r}_r] \tag{4.8}$$

As detailed in section 3.2.2, if the manipulated object is light or rigid enough for its shape not to be affected by gravity (i.e. it does not slack), the material parameters (mass, Young modulus and Poisson coefficient) will not impact the simulation. This is

due to the fact that the shape is constrained by the displacement of the end-effectors, and not by the applied force magnitude. In such cases, we can still use the knowledge of the parameters to monitor the internal stress, and to assure that the object is being manipulated in the elastic domain and does not undergo irreversible deformations. Then, the mass of the object can be obtained through the force sensors on the end-effectors and the Young modulus and Poisson coefficient included in the simulator FEM model. Else, these parameters are set to default values.

4.4 Generation a sequence of mesh nodes and robot poses

In order to construct the initial sequence of mesh nodes $\mathbf{M}$, and the corresponding robot poses variation $\Delta\mathbf{R}$ for a first estimation of the mapping $\mathbf{L}^{-1}$, as well as for features extraction, we now use simulations. There is then no need to control the robot in open-loop to collect the needed shape samples, as was the case in Chapter 2.

In this section, we explain how we generate the data matrices $\mathbf{M}$ and $\Delta\mathbf{R}$. $\mathbf{M}$ is constituted of $D+1$ different samples of shapes (deformed meshes), so as to have D variations. To generate them, we consider the simulation already set up according to the current state of the object, and that the robot poses are known. Let us denote l_0 the distance between the two end-effectors $\mathbf{r}_{l0}$ and $\mathbf{r}_{r0}$ in this current state. We choose a maximum radius R_0 to describe a sphere inside which the particles can be displaced.

We then consider, for a sample i, the position for the left rigid particle as:

$$\begin{bmatrix} x_l \\ y_l \\ z_l \end{bmatrix}_i = \begin{bmatrix} x_l \\ y_l \\ z_l \end{bmatrix}_0 + R_0 \begin{bmatrix} a_l \\ b_l \\ c_l \end{bmatrix}_i \tag{4.9}$$

With $a_l, b_l, c_l \in [-1, 1]$. We then define the position of the right rigid particle in spherical coordinates from the new position of pr_{li}. This ensures that the new position of pr_{ri} is at an acceptable distance and avoids stretching the object too much, while also included in a sphere of radius R_0 around its original position:

$$\begin{cases} R_i = a_r R_0 + l_0 \\ \theta_i = \frac{\pi}{2} - \alpha_{xy} + b_r(\arctan \frac{y_{r0} - y_{li} + R_0}{x_{r0} - x_{li}} - \alpha_{xy}) \\ \phi_i = \frac{\pi}{2} - \alpha_{xz} + c_r(\arctan \frac{z_{r0} - z_{li} + R_0}{x_{r0} - x_{li}} - \alpha_{xz}) \end{cases} \tag{4.10}$$

With $\alpha_{xy} = \arctan \frac{y_{r0} - y_{li}}{x_{r0} - x_{li}}$ and $\alpha_{xz} = \arctan \frac{z_{r0} - z_{li}}{x_{r0} - x_{li}}$. Figure 4.4 illustrates how the spherical coordinates are used to compute pr_{ri}.

The random coefficient a_r is chosen depending on the desired stretching of the object: $a_r \in [-0.75, 0.25]$ will allow stretching up to $0.25R_0$, while $a_r \in [-1, 0]$ will only allow the object to be bent. Finally, we have $b_r, c_r \in [-1, 1]$.

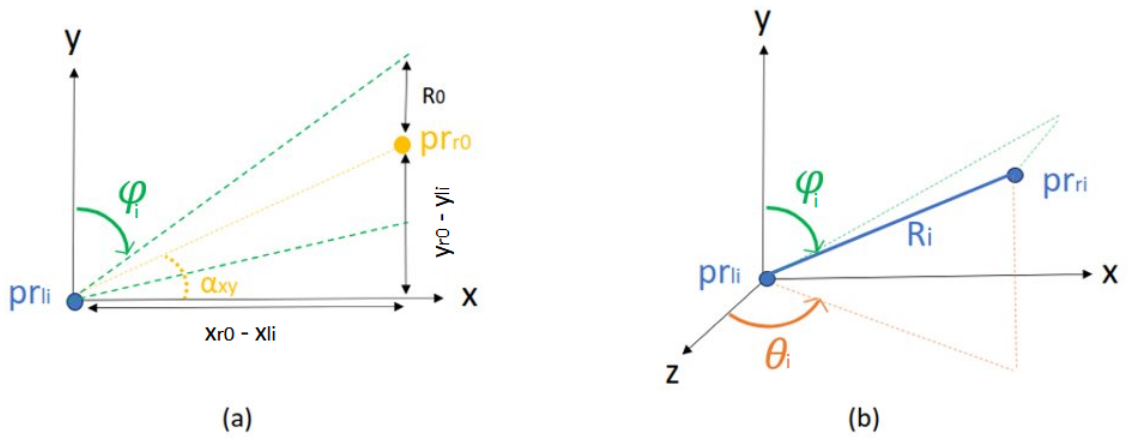


Figure 4.4: Spherical coordinates for random pose generation.

This yields, in Cartesian coordinates:

$$\begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix}_i = \begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix}_0 + \begin{bmatrix} R_i \sin\phi_i \sin\theta_i \\ R_i \cos\phi_i \\ R_i \sin\theta_i \cos\phi_i \end{bmatrix} \tag{4.11}$$

In terms of orientation, random Yaw, Pitch, Roll Euler angles are generated between $-\frac{\pi}{4}$ and $\frac{\pi}{4}$ and converted into the quaternion $\mathbf{q}_i$ (following Euler convention ZYX), to correspond to the rigid particle object (4.6). The resulting rigid particle poses are $\mathbf{pr}_{li} = [x_l, y_l, z_l, \mathbf{q}_l]_i$ and $\mathbf{pr}_{ri} = [x_r, y_r, z_r, \mathbf{q}_r]_i$.

If the material parameters are known, another condition for shape collection can be added on the Von Mises stress during the simulation. By monitoring the value of the Von Mises stress, we can discard the shapes which generate too high of a stress on a part of the object. These shapes would have higher chances to break or deform irreversibly. This condition and the Von Mises stress are detailed in the next section 4.5.

Some examples of generated shapes are shown on Fig. 4.5. The $D+1$ resulting meshes nodes $\mathbf{m}_i \in \mathbb{R}^{3\times n}$ are stacked in a column vector $\check{\mathbf{m}}_i \in \mathbb{R}^{3n}$, stored in matrix $\mathbf{M}$:

$$\mathbf{M} = [\check{\mathbf{m}}_0, \cdots, \check{\mathbf{m}}_D] \in \mathbb{R}^{3n\times D+1} \tag{4.12}$$

In this initialization phase, the corresponding rigid particles pose $[\mathbf{pr}_l, \mathbf{pr}_r]$ are used to compute the robot pose variation between iterations $i-1$ and i. During manipulation, these variations will be computed from the robot end-effectors poses feedback. The position components are simply subtracted, while the rotation is computed as the

distance between quaternions:

$$\begin{cases} \delta x_i = xr_i - xr_{i-1} \\ \delta y_i = yr_i - yr_{i-1} \\ \delta z_i = zr_i - zr_{i-1} \\ \delta \mathbf{q}_i = \mathbf{q}_i \cdot \mathbf{q}_{i-1}^{-1} \end{cases} \tag{4.13}$$

Finally, the variation quaternion $\delta\mathbf{q}_i$ is converted to an angle-axis vector $\delta\boldsymbol{\rho}_i$ to give the robot pose variation $\delta\mathbf{r}_i = [\delta x_i, \delta y_i, \delta z_i, \delta\boldsymbol{\rho}_i]$. The D robot pose variations obtained are stacked into matrix $\boldsymbol{\Delta}\mathbf{R}$:

$$\boldsymbol{\Delta}\mathbf{R} = \begin{bmatrix} \delta\mathbf{r}_1 & \dots & \delta\mathbf{r}_D \end{bmatrix} \in \mathbb{R}^{k\times D}. \tag{4.14}$$

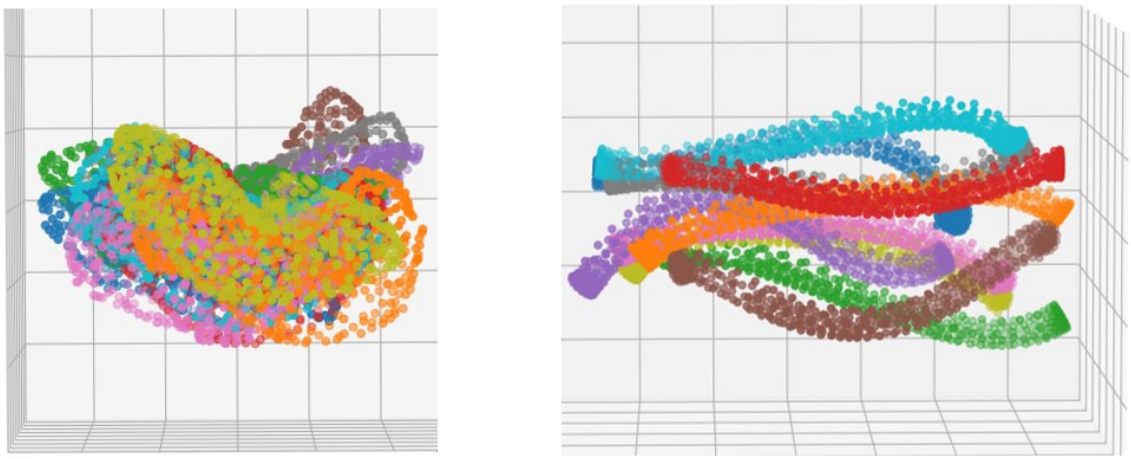


Figure 4.5: Examples of different shapes obtained with simulations. Mesh nodes resulting from $D = 10$ randomly computed rigid particle poses, (a) for a sponge, (b) for a DLO.

4.5 Simulation for target selection

In contrast with chapter 2, the target shape does not have to be selected from already reached shapes (i.e, by moving the robot or object manually to obtain a desired shape, to then go back to an initial shape and deform it again in closed-loop). Instead, we profit from the SOFA graphic interface, to control the object mesh. In practice, the user can select the target shape by displacing the rigid particles $\mathbf{pr}_l$ and $\mathbf{pr}_l$ via keyboard commands, as shown in Fig.4.6, until s/he obtains the desired shape. This Human-Machine Interface (HMI) for target shape selection is user-friendly, and allows an easy use of the framework without the necessity of knowledge on how to control the

robot manually.

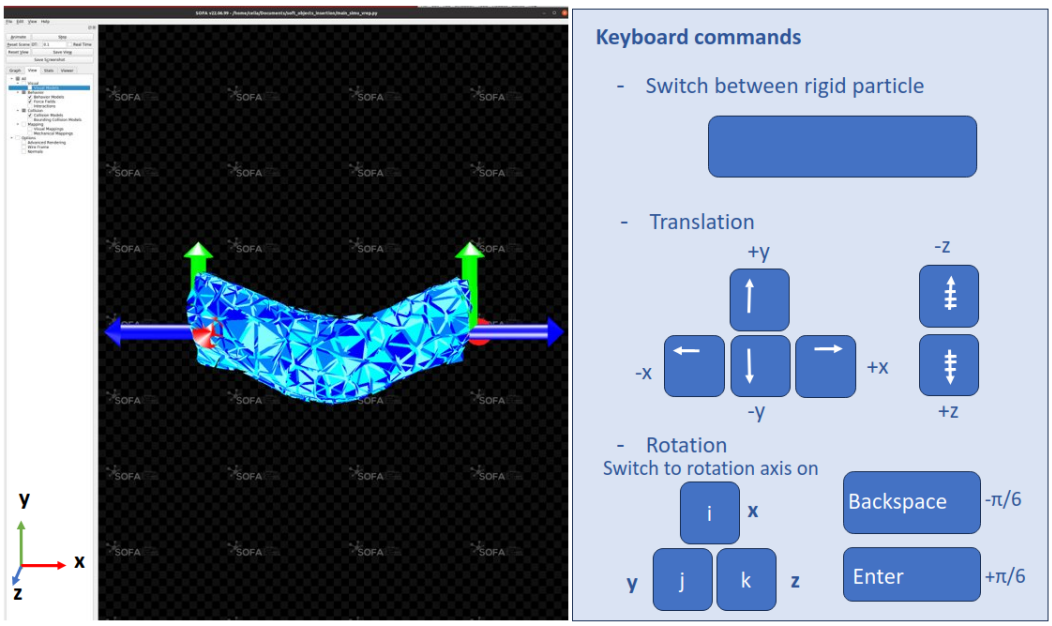


Figure 4.6: Keyboard commands for target selection via simulation.

If the material parameters are known or estimated, one can choose to compute the Von Mises stress during simulations.

The general form of Von Mises stress is:

$$\sigma_v = \sqrt{\frac{1}{2}[(\sigma_{11} - \sigma_{22})^2 + (\sigma_{22} - \sigma_{33})^2 + (\sigma_{33} - \sigma_{11})^2] + 3\sigma_{12}^2 + \sigma_{23}^2 + \sigma_{13}^2} \quad (4.15)$$

The Von Mises criterion states that the body will yield if the applied stress is greater than a critical value σ_y, called the yield strength ([Arm16]) (see Fig.4.7). The criterion is defined as:

$$\sigma_v \geq \sigma_y \quad (4.16)$$

This criterion is used to predict yielding for ductile materials.

The Von Mises stress is represented visually in the SOFA graphic interface through a color scale (where red indicates higher internal stress, hence higher damaging risk). Figure 4.8 shows the visual representation of the Von Mises stress on a mesh. Zones of high stress are circled in red. With this tool, the user can easily monitor if a target shape is hazardous or unreachable.

Similarly, it is then possible to impose a stress limit on the mesh, so that:

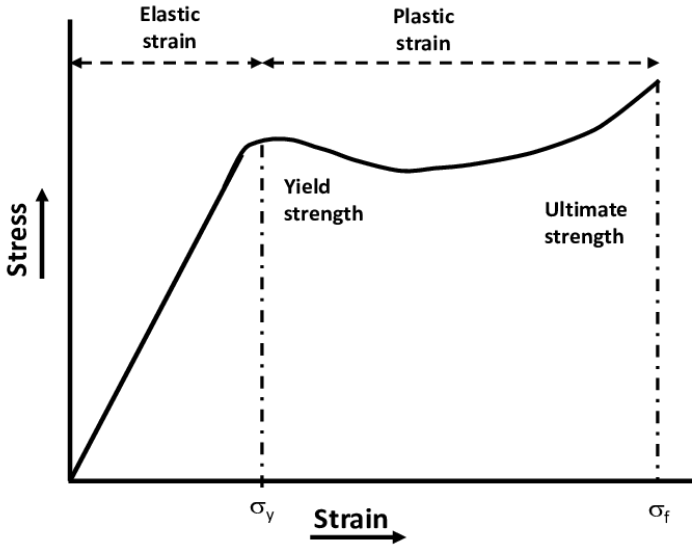


Figure 4.7: Typical strain-stress behavior for polymers. In the elastic domain, the body recovers its original shape when the stress is released (reversible deformation). In the plastic domain, instead, the deformation is irreversible. The yield strength defines the limit between elastic and plastic strain. Figure from [CPS+17].

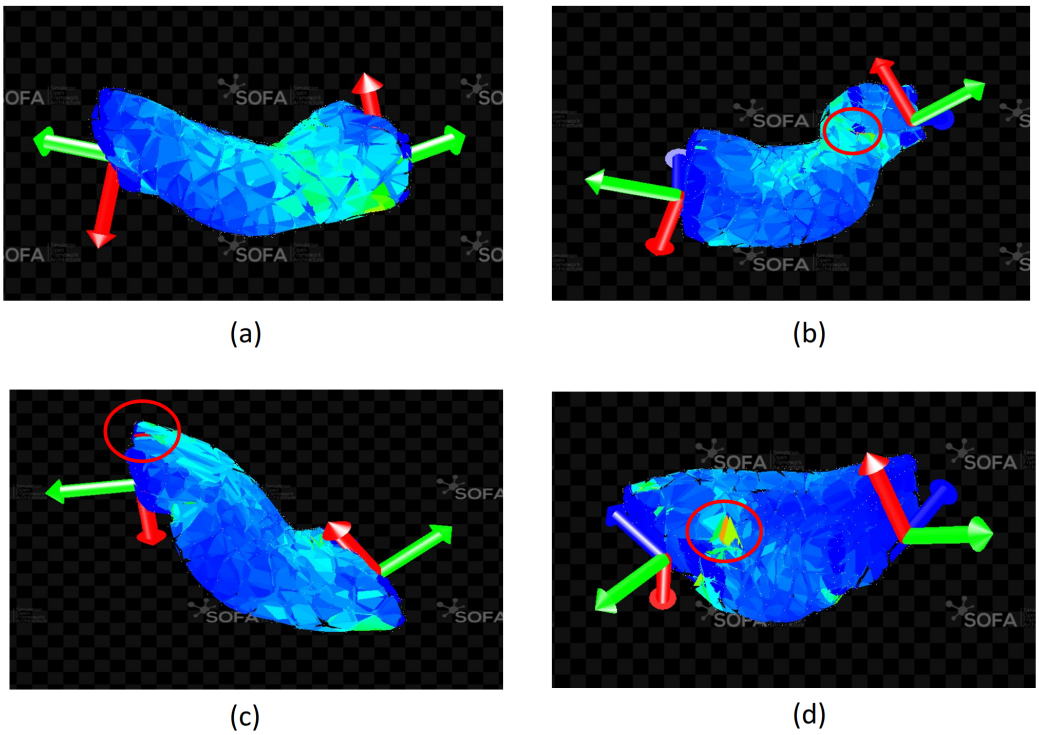


Figure 4.8: Examples of visual representations of the Von Mises stress on a mesh in SOFA during target selection. On subfigures (b), (c) and (d), we circled in red the mesh areas presenting high Von Mises stress.

- During the initial data collection, the generated meshes with internal stress higher than the limit are discarded;

- During manipulation, the control loop can automatically stop if the internal stress reaches high values, and therefore avoid damaging the object.

4.6 Run time control

In this Section, we detail each of the modules which compose our control loop (again, refer to Fig. 4.2). The framework is very similar to the one presented in chapter 2, although it uses tetrahedral mesh nodes instead of visible contours to represent the object.

4.6.1 Principal Component Analysis

We start by decreasing the size of the past data matrix $\mathbf{M}$ containing the mesh node samples, via PCA. First, we shift each column of $\mathbf{M}$ by the mean of all columns, $\bar{\mathbf{m}} = \frac{\sum \check{\mathbf{m}}_j}{D+1}, j = 0, ..., D$, to obtain:

$$\mathbf{M}_m = [\check{\mathbf{m}}_0 - \bar{\mathbf{m}}, \cdots, \check{\mathbf{m}}_D - \bar{\mathbf{m}}] \in \mathbb{R}^{3n \times D+1}. \tag{4.17}$$

We then compute $\mathbf{Q}$, the covariance matrix of $\mathbf{M}_m$, and (via Singular Value Decomposition - SVD - on $\mathbf{Q}$) the eigenvector matrix $\mathbf{U} \in \mathbb{R}^{3np \times 3n}$.

We select the first k columns of $\mathbf{U}$ to be able to control k DOF of our system; this choice is investigated in section 4.8.2. These columns form the projection matrix $\mathbf{U}_k \in \mathbb{R}^{3n \times k}$, whose columns correspond to the k principal components (with highest variances) in the dataset, and therefore determine the directions of highest variability in the data.

At each iteration i, the reduced feature vector is then:

$$\mathbf{s}_i = \mathbf{U}_k^\top (\check{\mathbf{m}}_i - \bar{\mathbf{m}}) \in \mathbb{R}^k. \tag{4.18}$$

4.6.2 Estimation of the Inverse Interaction Matrix

The next step consists in estimating the inverse interaction matrix $\mathbf{L}^{-1}$ needed for control. Recall that the interaction matrix is the linear mapping between feature variation $\delta\mathbf{s}$ and robot pose variation $\delta\mathbf{r}$ (see (4.1)). This matrix is unknown for a non-rigid object, and it should be inverted to control the robot pose.

To this end, we first project the mesh nodes matrix $\mathbf{M}$ into the reduced space as follows:

$$\mathbf{S} = \mathbf{U}_k^\top \mathbf{M}_m \in \mathbb{R}^{k \times D+1}, \tag{4.19}$$

and then derive the features variation over the D-dimensional window:

$$\mathbf{\Delta S} = \begin{bmatrix} \mathbf{S}_1 - \mathbf{S}_0 & \dots & \mathbf{S}_D - \mathbf{S}_{D-1} \end{bmatrix} \in \mathbb{R}^{k \times D}. \tag{4.20}$$

Finally, the inverse interaction matrix is given by:

$$\mathbf{L}^{-1} = \mathbf{\Delta R \Delta S}^{+} \in \mathbb{R}^{k \times k}, \tag{4.21}$$

with $\mathbf{\Delta S}^{+}$ the pseudo-inverse of $\mathbf{\Delta S}$.

4.6.3 Controlling the robot joints

Linear approximation (4.1) is only valid locally. Therefore, a control law based on $\mathbf{L}^{-1}$ cannot guarantee convergence if the initial and final shapes are too far. To solve this issue, we design an intermediary target mesh with nodes $\check{\mathbf{m}}_i^*$ at each iteration, via linear interpolation:

$$\check{\mathbf{m}}_i^* = \frac{\check{\mathbf{m}}^* - \check{\mathbf{m}}_i}{n} \tag{4.22}$$

with n big enough to ensure small displacements. Correspondingly, we can project $\check{\mathbf{m}}_i^*$ into the feature vector space to obtain the intermediary target feature vector:

$$\mathbf{s}_i^* = \mathbf{U}_k^{\top}(\check{\mathbf{m}}_i^* - \bar{\mathbf{m}}) \in \mathbb{R}^k. \tag{4.23}$$

The desired robot pose variations are then computed via:

$$\delta \mathbf{r}_i = \mathbf{\Lambda L}_i^{-1}(\mathbf{s}_i^* - \mathbf{s}_i) \in \mathbb{R}^k \tag{4.24}$$

with $\mathbf{\Lambda} \in \mathbb{R}^{k \times k}$ a diagonal matrix of control gains. As in chapter 2, this feedback controller guarantees asymptotic convergence of $\mathbf{s}_i$ to $\mathbf{s}_i^*$, in the ideal case that $\mathbf{L}^{-1}$ is perfectly estimated.

The robot pose variation command $\delta \mathbf{r} \in \mathbb{R}^{12}$ is mapped to the joint velocities $\dot{\boldsymbol{\theta}}$ through the standard QP (Quadratic Programming) optimization problem:

$$\begin{gathered} \min_{\dot{\boldsymbol{\theta}}} ||\mathbf{J}\dot{\boldsymbol{\theta}} - \delta \mathbf{r}||_2 \\ \text{subject to: joint limits.} \end{gathered} \tag{4.25}$$

We take into account joint limits in terms of position, velocity and acceleration, and use

$$\mathbf{J} = \begin{bmatrix} \mathbf{J_{lef}^T} & \mathbf{J_{rig}^T} \end{bmatrix}^T \tag{4.26}$$

with $\mathbf{J_{lef}}$ and $\mathbf{J_{rig}}$ the Jacobian matrices of the left and right arm respectively.

4.6.4 Obtaining the new shape

Once the command has been achieved, the new robot pose $\mathbf{r}_{i+1}$ is retrieved, and transferred to the simulator: the rigid particles $\mathbf{pr}_l$ and $\mathbf{pr}_r$ representing the end-effectors are constrained in displacement to reach the new position and orientation, $\mathbf{r}_{i+1}$.

For a given robot pose $\mathbf{r}_{ref} = [x_{ref}, y_{ref}, z_{ref}, \boldsymbol{\rho}_{ref}]$, $ref = \{l, r\}$, a displacement constraint of the corresponding rigid particle amounts to simply imposing:

$$\mathbf{pr}_{ref} = [x_{ref}, y_{ref}, z_{ref}, \boldsymbol{q}_{ref}] \tag{4.27}$$

with $\mathbf{q}_{ref}$ the quaternion equivalent to the angle-axis vector $\boldsymbol{\rho}_{ref}$. This constraint is, by proxy, imposed to the corresponding holding nodes $\mathbf{H}_{ref}$, which are rigidly attached to the rigid particle.

After the constraint is imposed, the FEM simulation runs to deform the rest of the mesh until quasi-equilibrium is reached, i.e. for simulation step j, we have:

$$\mathbf{m}_j \approxeq \mathbf{m}_{j-1} \tag{4.28}$$

The resulting meshes are denoted $\mathbf{m}_{i+1}$, $\mathbf{m}^v_{i+1}$. The new point cloud of the object $\mathbf{pc}_{i+1}$ is transformed into the robot frame through the transformation matrix $\mathcal{T}^{rob}_{cam}$. ICP[1] is then conducted between $\mathbf{pc}_{i+1,rob}$ and the triangle mesh $\mathbf{m}^v_{i+1}$. Running ICP between the point cloud in the robot frame $\mathbf{pc}_{i+1}$ and the visual mesh nodes $\mathbf{m}^v_{i+1}$ gives the transformation $\mathcal{T}_{i+1}$, from the point cloud to the mesh. A fitness score between the triangle mesh and transformed point cloud is computed. The process is illustrated in Fig. 4.9.

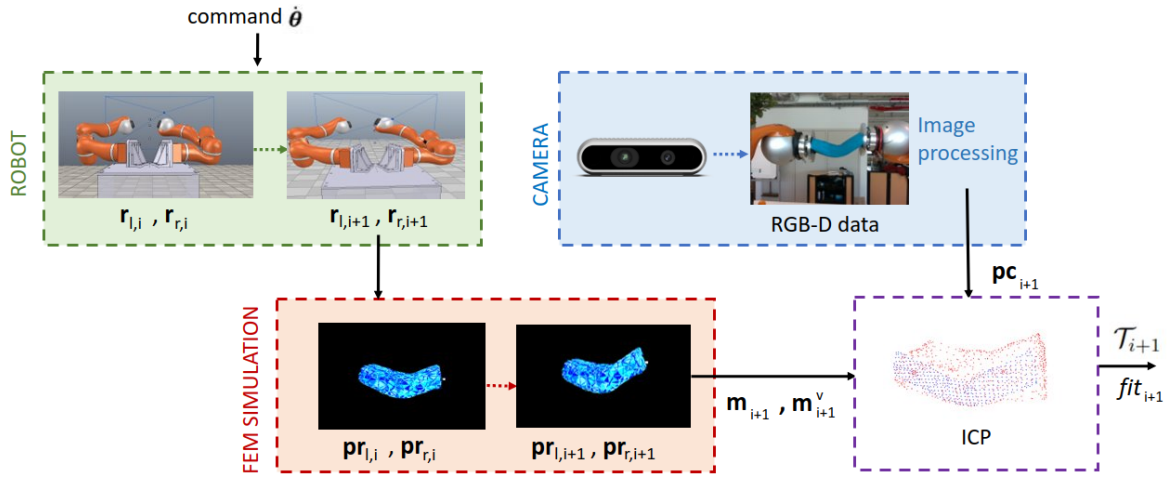


Figure 4.9: Process to obtain the new shape from both simulation and visual feedback.

If the fitness result of the ICP is lower than an imposed threshold, a correction step is conducted to improve the similarity between meshes and visual feedback. The visual processing and correction steps are detailed in the following section. The corrected

[1] http://www.open3d.org/docs/release/tutorial/pipelines/icp_registration.html, section *Point-to-plane ICP*

mesh nodes are denoted $\tilde{\mathbf{m}}_{i+1}$ and $\tilde{\mathbf{m}}^v_{i+1}$.

If the ICP gives satisfying results (this evaluation will be detailed in section 4.7.2), the correction step is skipped and we simply take:

$$\begin{cases} \tilde{\mathbf{m}}_{i+1} = \mathbf{m}_{i+1} \\ \tilde{\mathbf{m}}^v_{i+1} = \mathbf{m}^v_{i+1} \end{cases} \tag{4.29}$$

The past data matrices $\Delta\mathbf{R}$ and $\mathbf{M}$ are updated with the new pose variation computed via 4.13 and the nodes $\tilde{\mathbf{m}}_{i+1}$.

The task error at iteration i is defined as the norm error between between the *transformed* corrected mesh nodes and the target mesh nodes, in meters:

$$e_i = \|\mathbf{m}^* - \mathcal{T}_i^{-1}\tilde{\mathbf{m}}_i\| \tag{4.30}$$

NOTE:
The rigid transformation $\mathcal{T}_{i+1}$ resulting from the ICP is not applied to the mesh nodes when added to the data matrix $\mathbf{M}$, since the estimated *deformation* (the mesh) is considered close enough to the observed deformation (the point cloud). Applying the rigid transformation also transforms the holding nodes, resulting in their displacement not corresponding to the displacement of the end-effectors anymore. To avoid this issue, which may cause the algorithm not to converge for lack of coherence, the rigid transformation $\mathcal{T}_{i+1}$ is only accounted for in the computation of the task error, and used as well in the correction step that will be detailed in the next section. This is also the reason why the correction step is not performed at every iteration, ensuring coherence in the data matrices.

4.7 Including visual feedback

4.7.1 Image processing

Instead of relying on the *known* color of the object, as in chapter 2, here we decides to use the color of the robot end-effectors to segment the object in the RGB image. We assume that:

- the object is at all times contained in-between the two end-effectors, i.e. it is at the right of the left end-effector and at the left of the right end-effector;
- the color of the object is not too close to that of the end-effectors.

Since the end-effectors' color remains unchanged regardless of the object being manipulated, the image acquisition needs no tuning from one manipulation to the next. Our setup uses the green color of the end-effectors, but it is straightforward to replace color segmentation by markers on the end-effectors, if needed.

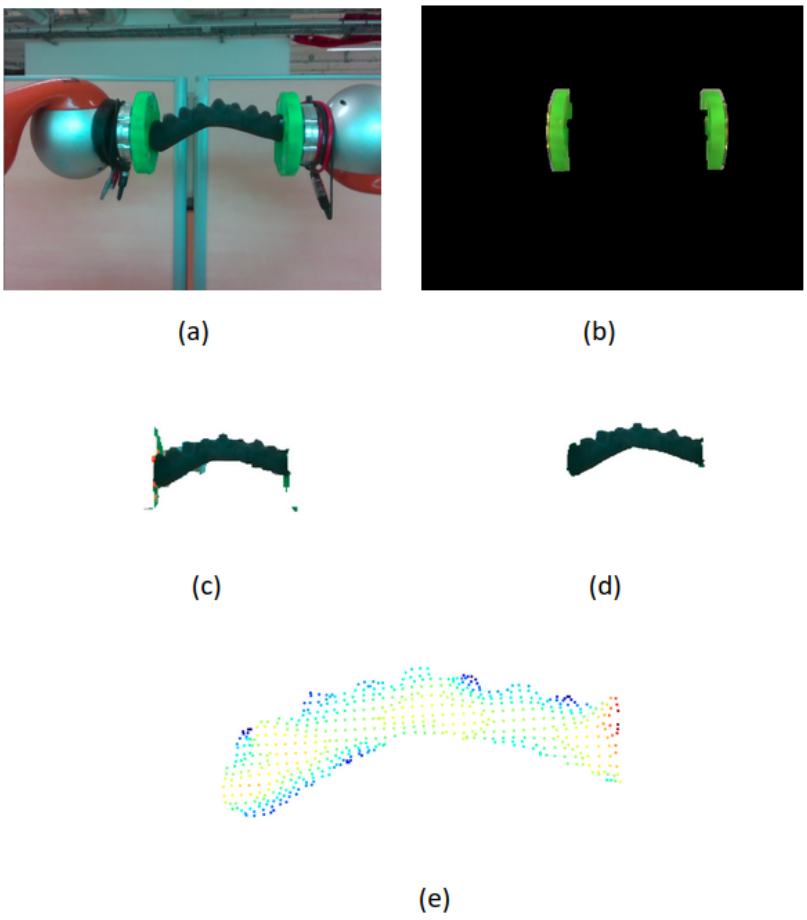


Figure 4.10: Different steps of point cloud acquisition. From RGB image (a), the robot end-effectors are segmented by color (b). An inverted mask is applied to the depth map, to nullify the depth value of the effectors. The depth of pixels right of the right effector and left of the left one are also set to 0, as are the background pixels. This constitutes a mask which is applied to (a), resulting in (c). From there, the dominant color of (c) is segmented as well, giving (d). (d) is then projected in 3D, resulting in the point cloud (e) expressed in the camera frame.

The process start by extracting the pixels of both end-effectors, from their known color (see Fig.4.10(b)). The depths of the end-effectors pixels (i^e, j^e) are set to zero, and so are the depths of pixels left of the left end-effector and right of the right end-effector. The depth values which are farther than a limit distance $d_{bg} = max(d(i^e, j^e)) - 0.5$ are considered as background, and nullified as well:

$$d(i,j) = 0 \text{ if } \begin{cases} (i,j) \in (i^e, j^e) \\ i < \min(i^e) \\ i > \max(i^e) \\ d(i,g) > d_{bg} \end{cases}$$

for a pixel of coordinate (i, j), with depth $d(i, j)$. The resulting depth map is trans-

formed into a binary mask and applied to the RGB image to obtain a new image where all pixels such as $d(i,j) = 0$ are colored white, Fig.4.10(c).

When getting the very first camera feedback during the manipulation, we conduct Hue Clustering [Mac67] to extract the dominant color of the remaining pixels of the resulting RGB image. This step can be longer, so to avoid having to repeat it at every iteration, we save the dominant color extracted from the first camera feedback and pass it as an input for the rest of the experiment.

Another color segmentation is then applied to the new RGB image to extract the pixels corresponding to the dominant color. These will characterize the manipulated object, Fig.4.10(d). Finally, these pixels are projected into 3D coordinates using the camera intrinsic parameters and depth value, as was done in 2.2.1. The object point cloud pc_{cam} is obtained, Fig.4.10(e). The point cloud is finally transformed to the robot frame using the transformation matrix $\mathcal{T}_{cam}^{rob}$ to give:

$$pc = \mathcal{T}_{cam}^{rob} pc_{cam} \tag{4.31}$$

4.7.2 Evaluation of the mesh with the visual feedback and correction

For many different reasons (the material not following a perfectly linear-elastic behavior, small errors on the gripping model, bulking effects, privileged direction of bending, and so on), the simulation may mismatch the actual object deformation. Hence, after each control step, we evaluate the mesh resulting from the simulation, via visual feedback.

Given a source set of 3D points $\mathbf{P}$ and the target set of points $\mathbf{Q}$, the ICP algorithm finds, via Nearest Neighbors Search, the correspondence set $\mathcal{K}$, defined as $\mathcal{K} = \{(\mathbf{p}, \mathbf{q})\}$ with $\mathbf{p}$ from $\mathbf{P}$ and $\mathbf{q}$ from $\mathbf{Q}$.

For κ correspondences, and a target set of points of size n_q, we define the fitness fit of the ICP algorithm as:

$$fit = \frac{\kappa}{n_q} \in [0, 1] \tag{4.32}$$

We consider the alignment of the simulation with the visual feedback satisfactory, if the ICP outputs a fitness value $fit_{i+1} > 0.8$, i.e. if the point cloud and the nodes overlap by more than 80%.

When the fitness is not satisfactory, we proceed with a correction step. Our approach consists in considering the mesh as an articulated object: the goal is to find the maximum displacement between corresponding points and nodes, and create a particle acting as a joint, to apply this displacement to the corresponding slice of the mesh, and deform the rest of the mesh accordingly.

For easier reading, let us denote a current point cloud aligned to the mesh with the transformation (resulting from ICP) $\mathcal{T}$:

$$\mathbf{pc} = \mathcal{T}_{cam}^{rob}\mathcal{T}\mathbf{pc}_{cam}. \tag{4.33}$$

We also denote the current (updated with the displacement of the end-effectors) tetrahedral mesh nodes $\mathbf{m}$ and the corresponding triangular mesh nodes $\mathbf{m}^v$. Using KNN (K-Nearest Neighbors, [MM99]) algorithm, we find the nearest triangle mesh node to each point of the point cloud. The result is a list of nodes $\mathbf{KNN}$, such that node $\mathbf{KNN}_i \in \mathbf{m}^v$ is the triangle mesh node closest to point $\mathbf{pc}_i$. We call $(\mathbf{p}*, \mathbf{n}*)$ the pair of point cloud point and corresponding triangle mesh node respectively, see Fig.4.11(a), such as:

$$\begin{cases} \mathbf{p}* = \mathbf{pc}_\kappa \\ \mathbf{n}* = \mathbf{KNN}_\kappa, \end{cases} \tag{4.34}$$

with κ such that :

$$\text{Find } \kappa \in \max_{\kappa \in [0,p]} (||\mathbf{pc}_\kappa - \mathbf{KNN}_\kappa||_2). \tag{4.35}$$

Once $\mathbf{n}*$ is known, we find the holding nodes $\mathbf{H}_n$: The tetrahedral mesh $\mathbf{m}$ is "sliced" in the direction of the plane normal to the end-effectors axis $\mathbf{p}_{pr}$, with a small tolerance on the distance to the plane, to get more nodes (see Fig.4.11(b)). The nodes in $\mathbf{H}_n$ are such that:

$$\left|\mathbf{p}_{pr} \cdot \mathbf{n} - \mathbf{p}_{pr} \cdot \mathbf{n}*\right| \leq tol, \mathbf{n} \in \mathbf{m}. \tag{4.36}$$

with $tol = 0.01$.

Next, the holding nodes $\mathbf{H}_n$ are rigidly attached to a rigid particle $\mathbf{pr}_n$ created after the position $\mathbf{n}*$, see Fig.4.11(c). Since the correction is applied with translation alone, the orientation of the rigid particle is set to $\mathbf{q}_n = [0, 0, 0, 1]$.

We finally apply a displacement constraint on the rigid particle $\mathbf{pr}_n$ to reach the position $\mathbf{p}*$. The displacement is, due to the rigid link, also applied to the nodes in $\mathbf{H}_n$. The simulation runs and deforms the rest of the geometry accordingly, resulting in the corrected meshes $\tilde{\mathbf{m}}_{i+1}$ and $\tilde{\mathbf{m}}^v_{i+1}$, as shown in Fig. 4.11(d).

The matrix containing the past mesh nodes $\mathbf{M}$ is updated using the corrected mesh points $\tilde{\mathbf{m}}_{i+1}$ for the next iteration, see Fig.4.2.

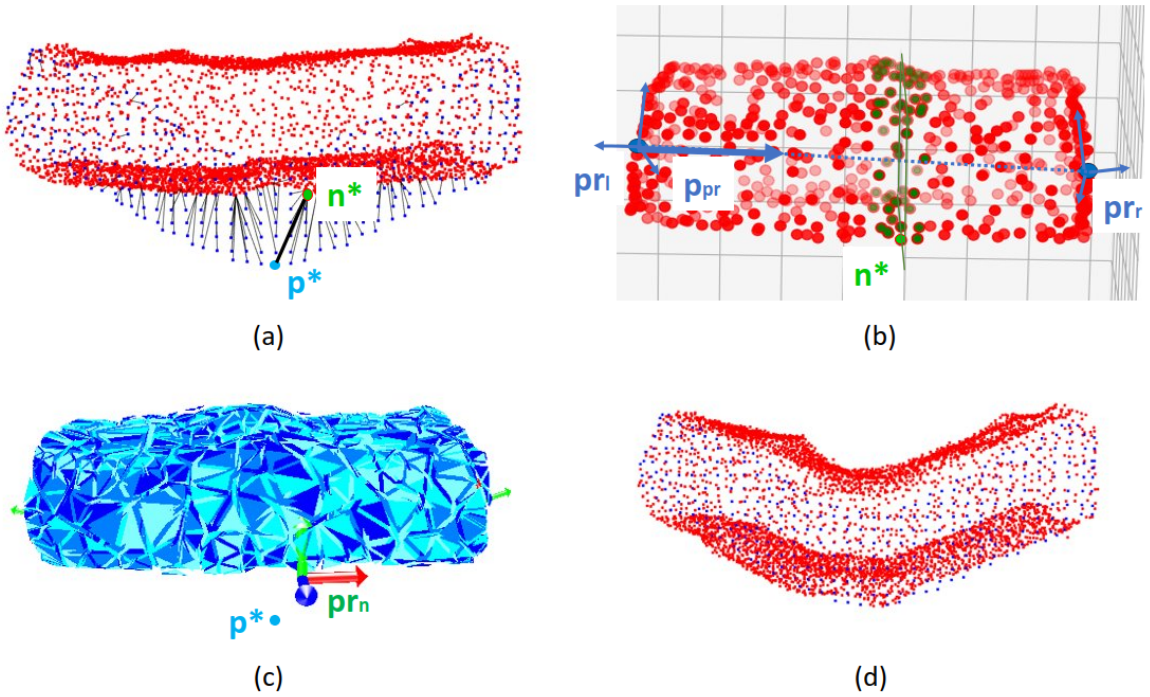


*Figure 4.11: Steps of the correction to visual feedback. (a) We find the node of the triangle mesh (**n**$*$ in red) which is the farthest from its corresponding point in the point cloud (**p**$*$ in blue). (b) In the tetrahedral mesh, we select the holding nodes ($\mathbf{H}_n$, in darker green), which have a distance to the plane orthogonal to the axis of end-effectors $\mathbf{p}_{pr}$ lesser than a threshold. (c) A rigid particle $\mathbf{pr}_n$ is added; it is rigidly attached to all the points in $\mathbf{H}_n$. We displace $\mathbf{pr}_n$ to have it reach **p**$*$. (d) The rest of the mesh is deformed, resulting in $\tilde{\mathbf{m}}^v_{i+1}$ shown in red.*

4.8 Experiments with robot simulator

Simulations are critical in robotics to analyze performances and design control systems (see [Žl08]), and many simulators (Coppelia-Sim[2], MuJoCo[3], Gazebo[4], to cite a few) allow users to model different robots and environments. If rigid objects can easily be included in robot simulators, it is seldom the case for soft objects.

MuJoCo includes a collection of basic elements designed to simulate ropes, cloth, and soft bodies represented by standard geometries like cubes or balls, but not specific geometries. The authors of [DBPC18] design their own simulator, which couples the shape representation (FEM energy) to the robot joint representation. Another approach is to use available rigid models to approximate the behavior of soft objects. For instance, the authors of [CP20] model cables using rigid links connected by revolute joints. The authors of [ZPC21] use FEM simulations for their trajectory optimization strategy, using a compressible neo-Hookean material model.

[2] http://www.coppeliarobotics.com/
[3] https://mujoco.org/
[4] https://gazebosim.org/home

When it comes to visual servoing for soft object manipulation, in addition to the robot model, an accurate visual result must be generated as part of the control loop. Hence, for methods based on camera feedback, the deformed shape corresponding to the robot end-effectors' action must be generated in parallel. To this end, the authors of [ZNAPC20] include, in parallel to their control framework, simulations of DLO (cables) based on differential geometry.

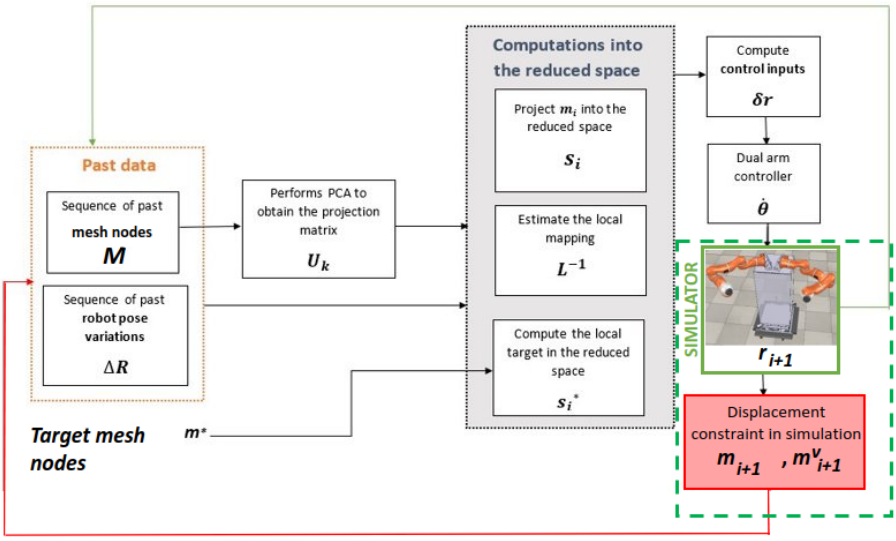


Figure 4.12: Overview of the framework in simulation. The difference with 4.2 are framed in doted green lines. The joint command is sent to the robot simulator instead of the robot, and the visual processing blocks are skipped.

In our framework, the use of mesh nodes for shape feedback allows us to test out the control framework entirely in simulation, by sending the joint command to the simulator instead of sending it directly to the robot. This allows feeding the result of the FEM simulations directly into the data matrix $\mathbf{\Delta M}$, as shown in Fig. 4.12. Figure 4.13 shows, on one side, the model of the robot in the simulator (in our case, CoppeliaSim), and on the other, the mesh of the object corresponding to the pose of the end-effectors, acting like a visual representation of the object.

Figure 4.14 presents the results of the shaping simulations. These results shows the dual-arm robot reaching successfully three different target shapes involving different DOFs in $\mathbb{SE}(3)$, and task with translating, bending, twisting, and stretching. Simulations are also very useful for investigating more in depth the dimension of the data matrices required in the framework, which would be cumbersome to do with real robot experiments. We discuss this in the following section.

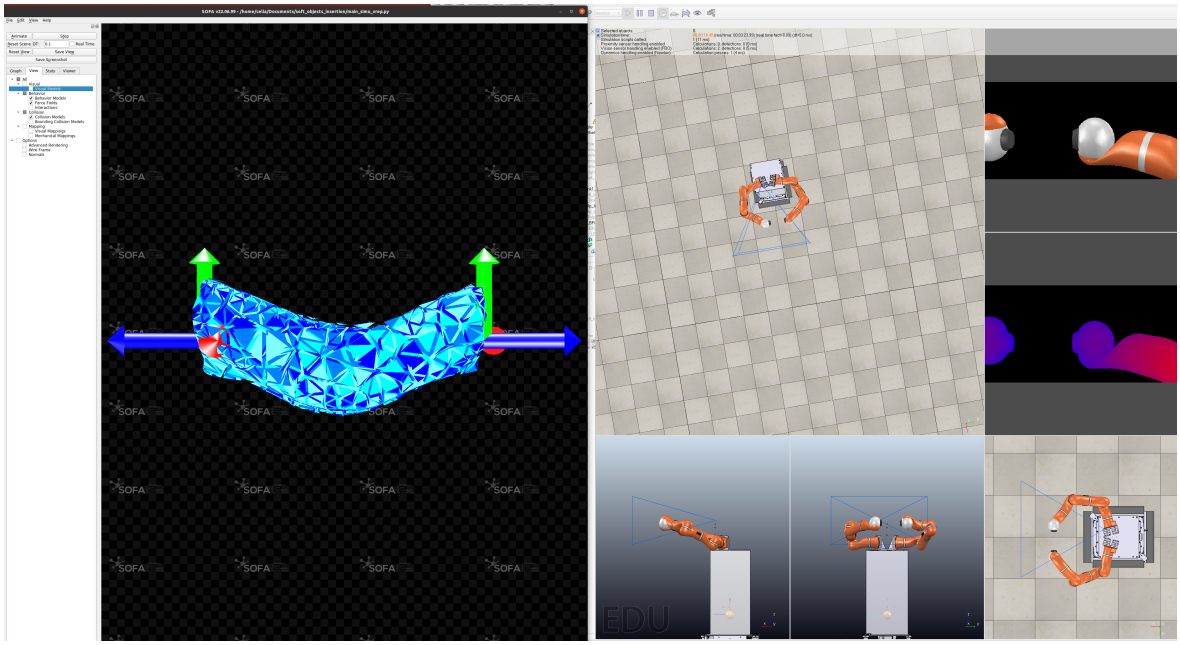
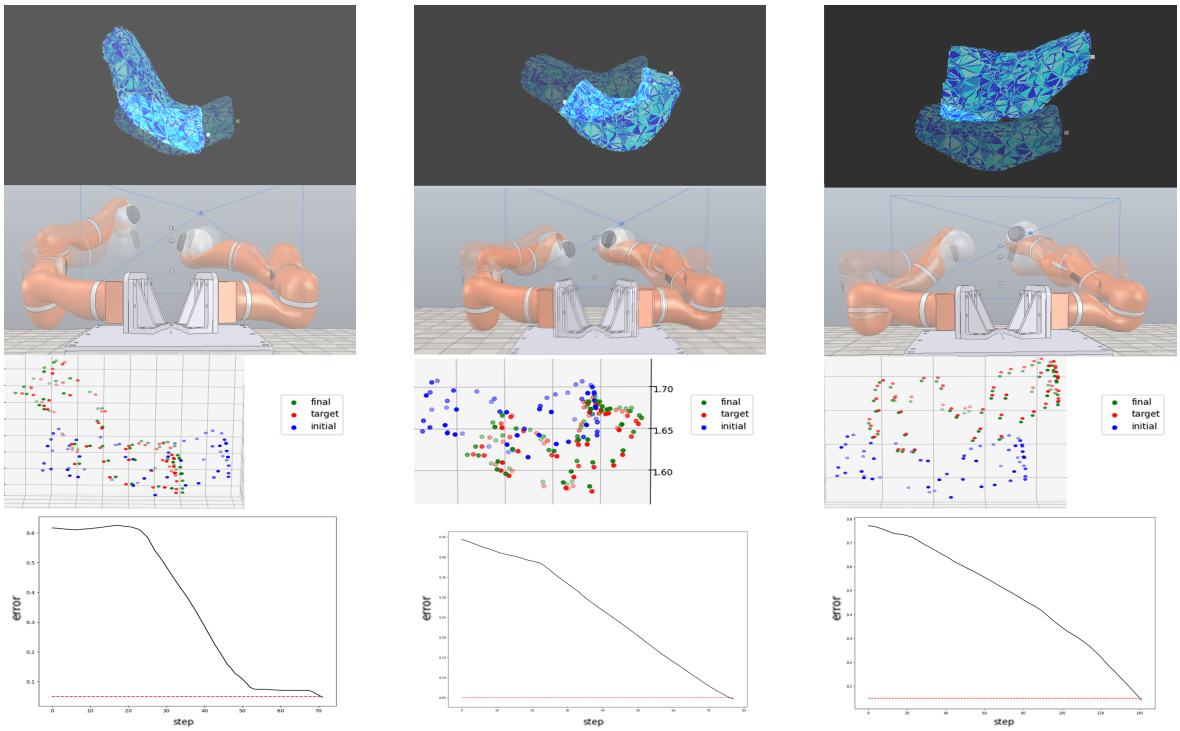


Figure 4.13: Simulated experiment visuals. On the right is robot simulator, moving the end-effectors according to the control inputs. On the left is the FEM simulation of the object, which deforms the mesh according to the new end-effectors poses, updated by the robot simulator.



(a) Target shape 1 *(b) Target shape 2* *(c) Target shape 3*

Figure 4.14: Example of 3 different simulated experiments. The first two rows show the initial and final configurations of mesh (top) and robot (bottom). The third row shows the initial, final and target mesh (all sampled for visibility) of the experiment. The very last row shows the evolution of the shape error (4.30) during the experiment, with the doted red line representing the acceptable threshold $e = 0.05$. For these experiments, we use $D = 24$ and $k = 12$.

4.8.1 Choice of the number of samples

Let us first look at the number of samples which constitute data matrices $\mathbf{\Delta M}$ and $\mathbf{\Delta R}$, which respectively contain past shapes, and corresponding end-effector poses. The authors of [ZNAPC20] do not discuss the number of samples chosen. They only mention that it should be $D \geq k$ for $\mathbf{\Delta R} \in \mathbb{R}^{k \times D}$ to have full row rank.

Since these data can be collected through simulations, we run three, with the goal of reaching the target shapes shown in 4.14, referred to as *target shape 1*, *target shape 2* and *target shape 3* for columns 1 to 3 respectively. We attempt to reach each target shape, with a variable number of samples, to explore the impact of D on the controller's success. The experiments are repeated thrice, with different initialization data for each trial.

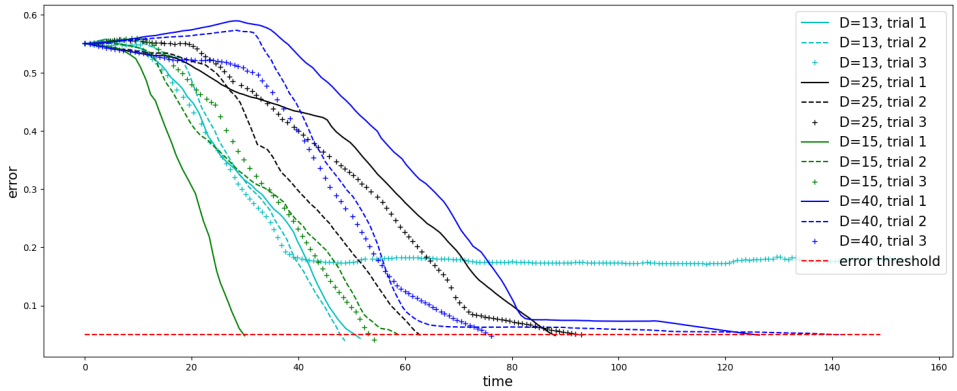
We consider as failed an experiment which has not reached the acceptable error threshold after 200 iterations of the control loop. The results are summarized in Table 4.1, while Figure4.15 shows the evolution of the shape error 4.30 during the 36 simulations.

Target shape	D	Average total time (s)	Success
1	13	50	yes, yes, **no**
2	13	64	yes, yes, yes
3	13	105	yes, **no**, yes
1	15	48	yes, yes, yes
2	15	64	yes, yes, yes
3	15	104	yes, yes, yes
1	25	81	yes, yes, yes
2	25	71	yes, yes, yes
3	25	119	yes, yes, yes
1	40	114	yes, yes, yes
2	40	89	yes, yes, yes
3	40	-	**no**, **no**, **no**

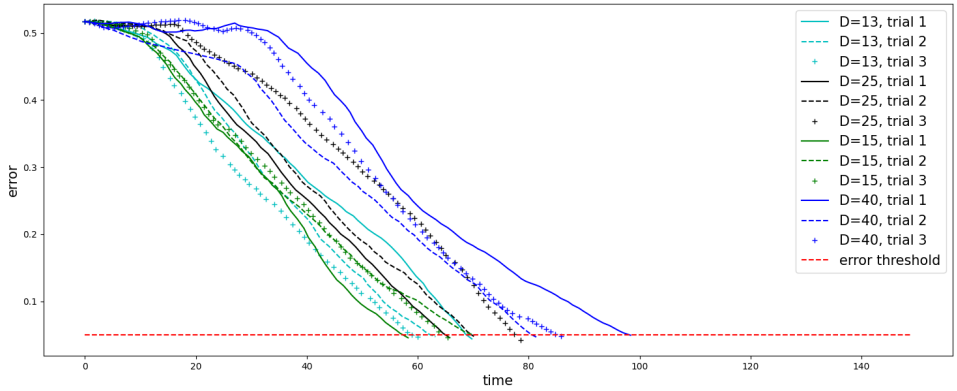
Table 4.1: Simulated experiments with different number of samples. $D = 15$ and $D = 25$ both give a 100% success rate.

The simulated experiments show that a number of samples too small (13) or too high (40) leads to some experiment not converging. Indeed, the number of samples too small will lead to little variety, thus a bad representation of the available motions; on the contrary, too many samples imply losing local information about motion.

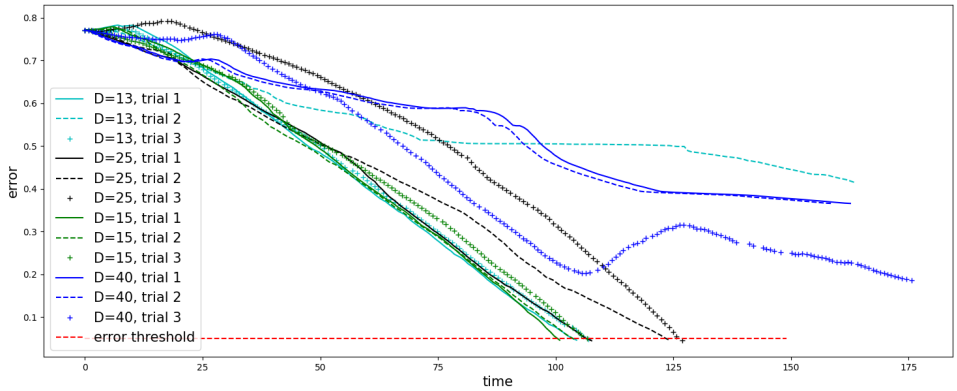
In terms of performances, by comparing the cases of $D = 15$ and $D = 25$, we notice that the lesser the samples, the fastest the computations. Keeping all of this in mind,



(a) Target shape 1



(b) Target shape 2



(c) Target shape 3

Figure 4.15: Evolution of shape error (4.30) during simulated experiments with different number of samples. Lines in cyan represent the trials for $D = 13$, green for $D = 15$, black for $D = 25$ and blue for $D = 40$.

it appears that choosing a number of samples between 15 and 25 would be a good trade-off between performance, locality while reducing the risk of under-representing the available motions. To ensure not an under-determined (similarly to experiments with $D = 13$), we take $D = 17$ for real robot experiments.

4.8.2 Choice of the number of features

It is not required for the interaction matrix $\mathbf{L}$ to be square. It is then worth questioning how many features are needed to represent the shape of an object. Therefore, in this section, we investigate the significance of the number of principal component and its impact on the command computations.

Since it is custom in visual servoing to consider a number of features at least as great as the number of DOF controlled [Cha07] (to avoid redundant visual data), here, $[\mathbf{r}_l, \mathbf{r}_r]$, our first choice is to consider $k = 12$ features.

To verify the validity of this choice, we conduct a series of simulations. As metric, we use the explained variance, which indicates how much each of the principal components encodes the dataset variation. The variance of the i-th principal component is defined, for a column vector of size $3p$, as:

$$var(\lambda_i) = \frac{\lambda_i}{\lambda_1 + \lambda_2 + \ldots + \lambda_{3p}} \tag{4.37}$$

with λ_i the eigenvalue associated with the i-th principal component resulting from the SVD (refer to Sec. 4.6.1). This metric allows to compare their significance, thus their encoding of useful information about the shape. We then observe the explained variance of the principal components, resulting from the PCA during 4 different trials, and summarize it in Table 4.2. For each trial, the PCA is performed on a new set of $D = 17$ randomly generated samples. For principal component i, Cumulative proportion of explained variance is given by:

$$CP(i) = \sum_{n=1}^{i} var(\lambda_n) \tag{4.38}$$

Nb principal components (i)	Trial 1	Trial 2	Trial 3	Trial 4	Average
6	0.899	0.893	0.906	0.901	0.899
8	0.971	0.958	0.978	0.968	0.968
10	0.993	0.986	0.978	0.968	0.981
12	0.999	0.998	0.998	0.999	0.998

Table 4.2: Cumulative proportion of explained variance for different number of principal components, for different trials.

As Table 4.2 shows, more than 90 % of the shape is encoded by 6 to 8 principal components, but 10 to 12 principal components represent a variance close to 1, giving

the best representation of the data. These simulations comfort us in the choice of taking $k = 12$ features for our control strategy.

All in all, the control framework could, with the exception of the modules involving visual feedback, be validated through simulations only, allowing us to tune most parameters offline, and to avoid cumbersome tests on the real robot.

4.9 Experimental results

We conduct experiments with different foam objects, of varying shapes, densities and rigidities. They are presented in Fig.4.16.

Figure 4.16: Different foam objects used during the experiments. From left to right: sponge, thin convoluted foam, convoluted foam, foam noodle.

The material parameters are considered unknown, and set on the FEM simulations to $E = 2.5e^6$ Pa and $\nu = 0.3$. Once again, the objects are gripped at the beginning of each experiment. The experiments consist in shaping the object held by the robot arms into a target shape. The deformation of the object is observed through a RGB-D camera, Intel Realsense D435.

At each control iteration i, the displacement of the end-effectors of the robot from the previous iteration is applied to the FEM simulations, by displacement constraint (see Sec. 4.6.4). The resulting mesh points (deformed according to the end-effectors displacement) $\mathbf{m}_i$ are obtained. Meanwhile, the point cloud of the deformed object $\mathbf{pc}_i$ is extracted, and ICP is conducted between the obtained triangle mesh nodes and the point cloud, giving $\mathcal{T}_i$, the transformation from $\mathbf{pc}_i$ to best fit $\mathbf{m}_i^v$. The ICP fitness score is obtained, and if it lesser than a chosen threshold (0.8 as default), we perform the correction step explained in section 4.7.2. The resulting mesh is $\tilde{\mathbf{m}}_i$. We compute the shape error 4.30, between the target mesh nodes $\mathbf{m}*$ and the corrected mesh nodes

$\tilde{\mathbf{m}}_i$, and the manipulation continues until the error decreases bellow a threshold set to $e = 0.05$ m.

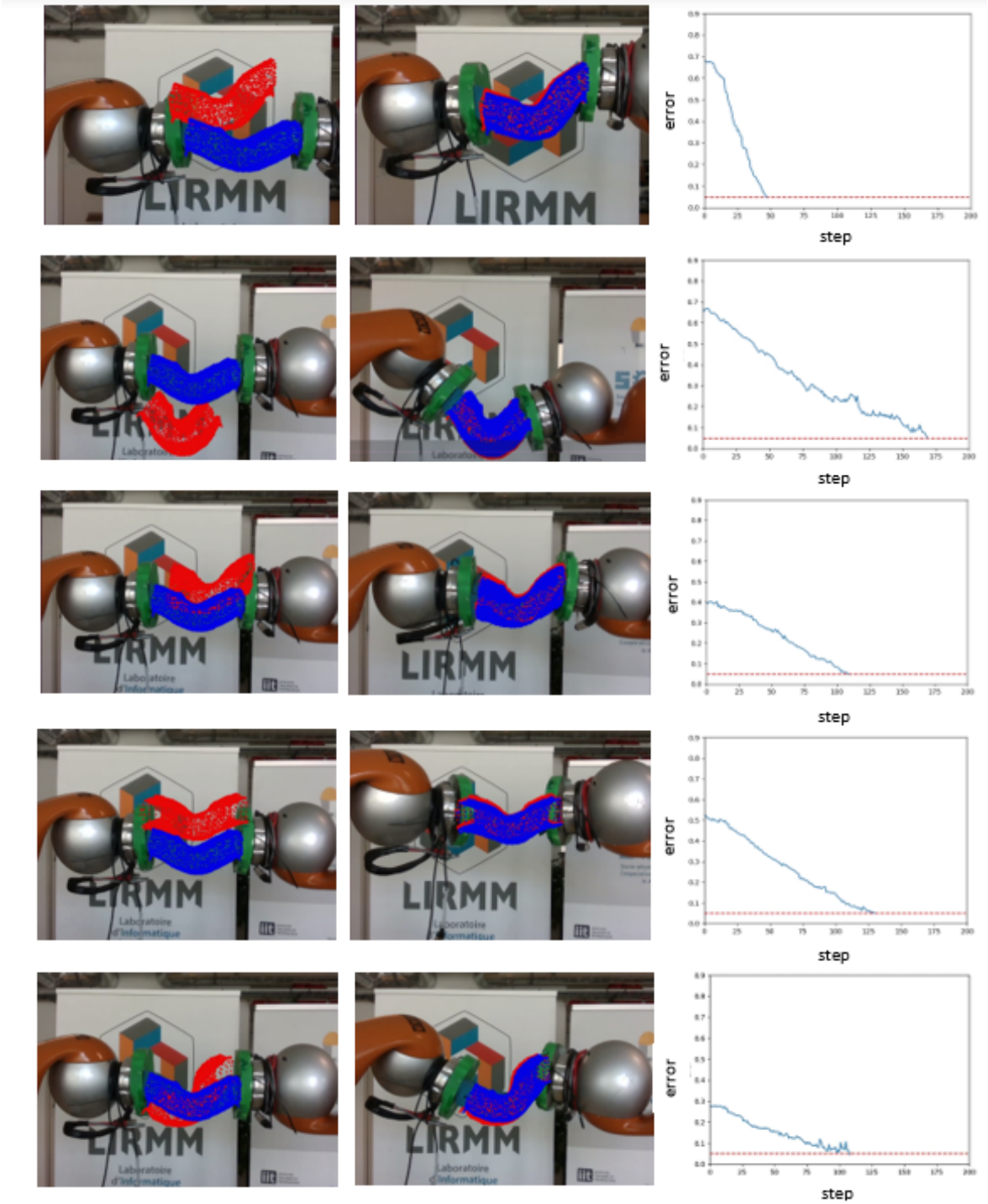


Figure 4.17: Different experiments with the sponge. First column: initial shape of the mesh (blue) compared to target shape (red), both projected on the RGB image. Second column: obtained final shape. Third column: evolution of the error e during the experiments, until the threshold $e < 0.05m$ is reached.

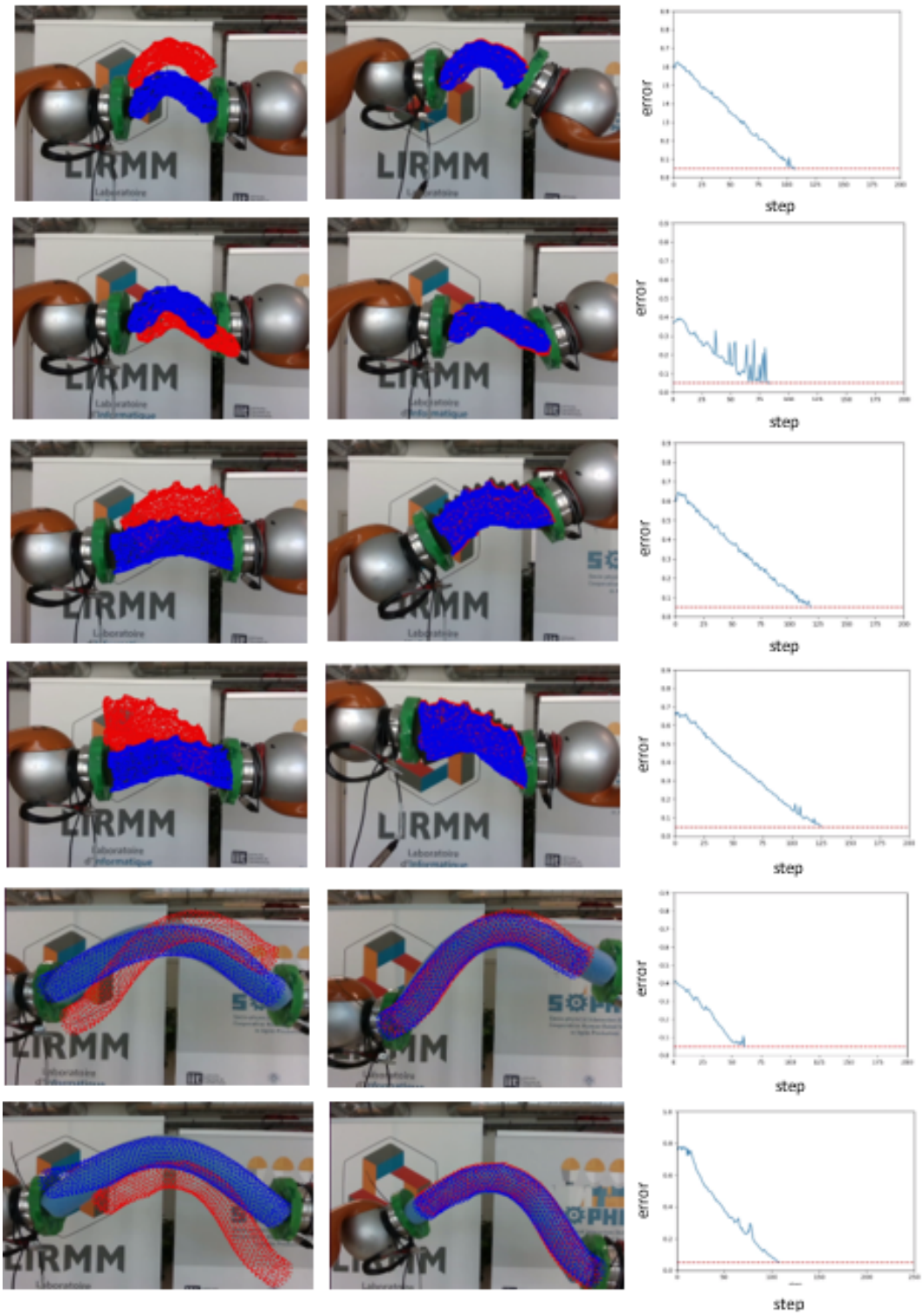


Figure 4.18: Different experiments with other objects - two dented foams and a foam noodle. First column: initial shape (blue) and target shape (red). Second column: final shape compared to the target. Third column: evolution of the error e until the threshold $e < 0.05$ is reached.

Figures 4.17 and 4.18 show the experiments. In the figures, the blue points represent the corrected mesh nodes transformed in the camera frame $\mathcal{T}_{rob}^{cam}\mathcal{T}_i^{-1}\tilde{\mathbf{m}}_i$, projected on the RGB image. The red points represent the target mesh nodes transformed in the camera frame $\mathcal{T}_{rob}^{cam}\mathbf{m}*$, also projected on the image.

Figure 4.17 presents different experiments with the sponge object, involving the 3D workspace. The experiments include bending in different axis and torsion (third row). All succeed in reaching the target shape. The fourth row experiment also presents convergence to the target shape, despite visual occlusion of the object by the robot end-effectors. Figure 4.18 shows different experiments with other foam objects. On the error plot of the second experiment with the thin dented foam, presented on the second row of the figure, we can see the task error e oscillating due to the ICP converging alternatively to different transformations. Since the data matrix $\mathbf{M}$ is not directly impacted by the result of ICP, the control algorithm manages nonetheless to decrease the task error, until convergence. Additionally, one can observe that on the last experiment with the foam noodle (fifth row), the initial mesh is not exactly aligned with the point cloud of the object. Yet, the shift between the two is corrected during manipulation.

The final RMSE for all the objects presented in these experiments are presented in Tab. 4.3.

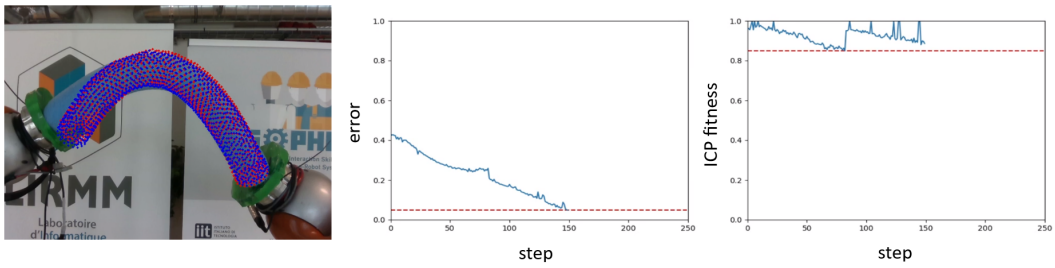
Experiment	sponge	thin convoluted foam	convoluted foam	foam noodle
Mesh nodes	579	564	800	102
RMSE (m)	0.001	0.001	0.001	0.003

Table 4.3: RMSE between the target mesh and the transformed final mesh reached at the end of the experiments with the objects presented in Fig.4.17 and 4.18.

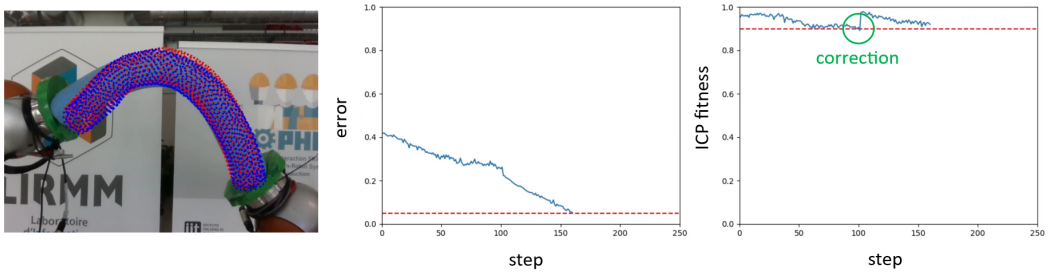
Concerning the foam noodle experiments, it is noticeable that the mesh is displaced compared to the tips of the noodle (i.e, the tips of the mesh and the object are not aligned). However, the ICP gives a high fitness score nonetheless, since it is sensitive to the curvatures. The task error e, which is computed between the mesh nodes (blue points on the figures) and the target mesh nodes (red points on the figures), is not impacted directly by the visual feedback as long as the ICP fitness stays above threshold; this is why it converges nonetheless.

In the case of the foam noodle, the curvature of the point cloud and the mesh nodes "match", even if the tips of object and mesh are not aligned. Since the ICP fitness score stays high, visual feedback correction does not occur. It is then necessary to tune the fitness score threshold, to control the frequency of correction steps.

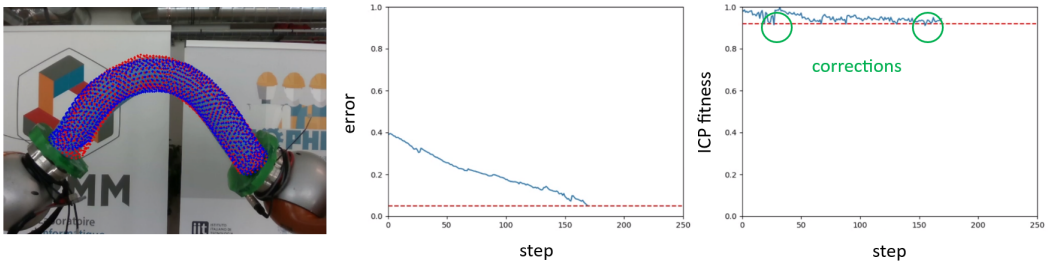
Figure 4.19 presents the results of experiments with the foam noodle reaching the same target shape with different thresholds of the ICP fitness, resulting in different occurrences of the correction step. We can observe that with a higher threshold - and therefore more correction steps during the experiment - the final shape of the mesh



(a) *Experiment with a threshold for ICP fitness at* 0.85.



(b) *Experiment with a threshold for ICP fitness at* 0.9.



(c) *Experiment with a threshold for ICP fitness at* 0.92.

Figure 4.19: Experiments showing how tuning the ICP fitness threshold may ensure adequate correction to visual feedback. The first column shows the final shape in blue, compared to the target shape in red, projected on the RGB image. The second column shows the evolution of the error during the experiments, reaching the threshold $e < 0.05$*. The third column shows the evolution of the ICP fitness score during the experiments, with the correction steps circled in green.*

resembles more the observed shape of the noodle. It is then necessary to tune the threshold of the ICP fitness accordingly to obtain a adequate correction to the visual feedback.

Finally, on Fig.4.20 is shown an example of two consecutive correction steps at the initial ICP alignment of the mesh with the point cloud of the foam noodle.

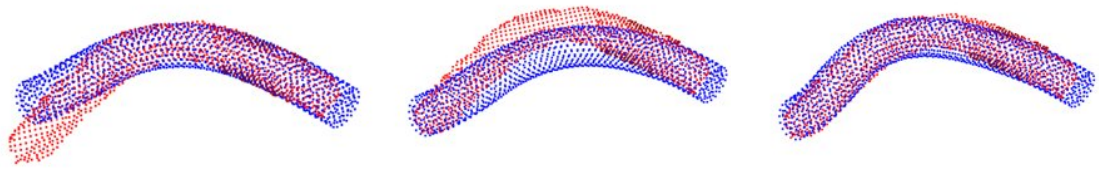


Figure 4.20: Example of correction of the mesh nodes (blue) with the point cloud of the object (red), after ICP alignment.

4.9.1 Stress monitoring experiments

In this section, we present a bending experiment while implementing stress monitoring. The object used in this experiment is a planar test tube 3D-printed in with PLA. The material parameters are set to $E = 3120\ MPa$, $\nu = 0.36$ and a density $\rho = 1240\ kg.m^{-3}$.

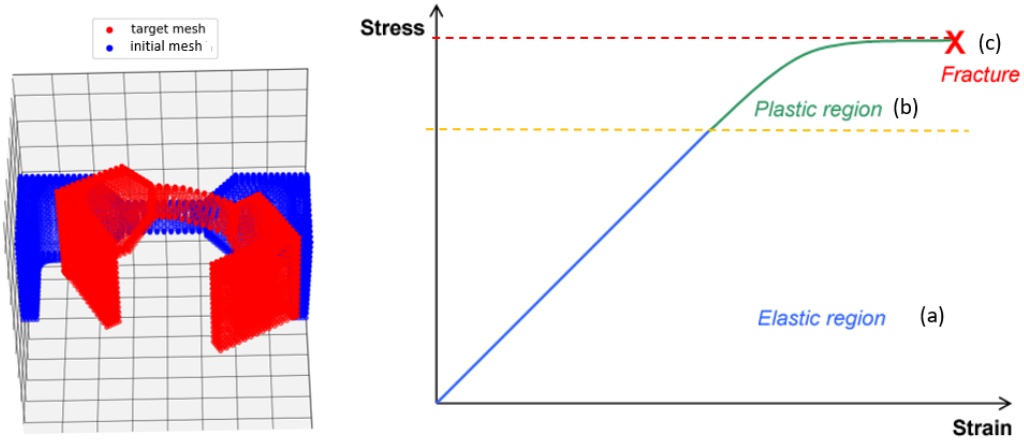


Figure 4.21: The objective of the experiment is to drive the initial shape (blue) to the target shape (red). We monitor the internal stress during the experiment to observe the different type of deformation: elastic (blue), plastic (green) and finally, rupture (red).

As we consider linear elasticity, we aim to apply a stress threshold during the manipulation in order to control the type of deformation applied to the object. As previously detailed in sec.4.5, the deformation applied to the object evolves with the value of the stress applied to it. At first, the deformation is *elastic*, and the object comes back to its original shape when the applied stress returns to zero. When the applied stress exceeds a first threshold (orange dotted line in Fig.4.21), the deformation becomes *plastic*. From there, even when the stress diminishes again, the object will remain deformed: it is permanently damaged. Finally, when the stress reaches the plastic limit (red dotted line in Fig.4.21), the object will reach its rupture point and

break.

In Fig.4.22 are presented three experiments in order to reach the same shape. During the experiment, the Von Mises stress (4.15) at each node is computed and plotted. In the first experiment, a stress limit is applied in order to keep the deformation of the object in the elastic domain. In the second one, we allow the object to be damaged (i.e. the deformation is in the plastic domain) but apply a stress limit to avoid breaking. Finally, the last experiment shows the manipulation without stress limit. For each experiment, the evolution of the Von Mises stress is plotted (first row), and we show the resulting shape of object after experiment (last row).

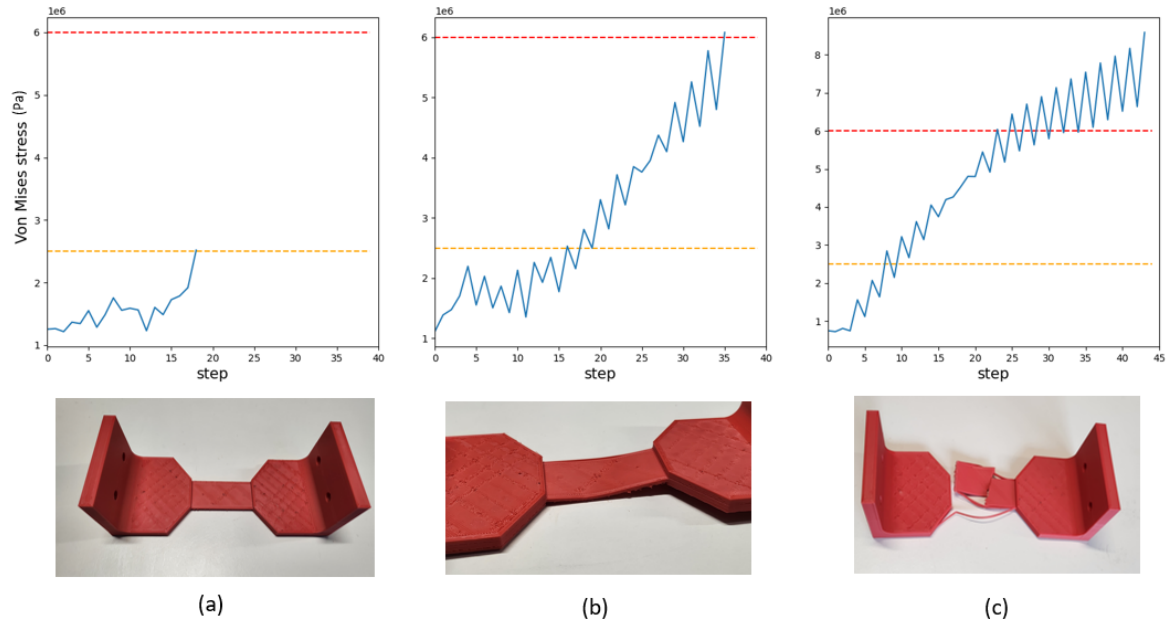


Figure 4.22: Different stress monitoring experiments. In the first experiment (first column), the deformation of the object stays elastic during manipulation, and the shape of object after experiment goes back to its original shape. In the second experiment (middle column), the deformation is plastic: the object is permanently damaged even after relaxation of the applied stress. The last experiment (last column) is conducted without stress limit, showing the deformation going beyond plastic domain and resulting in the object breaking.

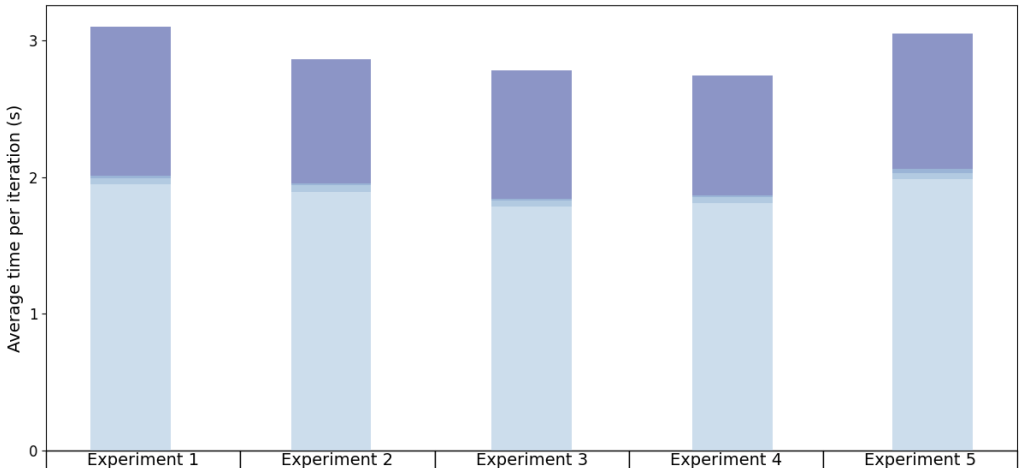
These experiments show that applying a stress limit during manipulation can avoid the damaging and/or breaking of the object manipulated, in the case where the material parameters are known.

4.9.2 Time performances

Let us now study the time performances of the control loop on the sponge experiments presented in Fig. 4.17, numbered from one to five. The total time of the experiments differs widely in function of the complexity of the target and of the initial error, but we can compare the average duration of one control iteration, for these different experiments.

A single control iteration can be broken down to four different modules:

- *simulation*: application in the FEM simulation, of displacement constraint updated by the robot poses, and running the simulation until quasi-equilibrium is reached (Section 4.6.4).
- *image processing*: acquisition of the current point cloud of the object (section 4.7.1).
- *correction*: running ICP, evaluation, and if necessary, correction step, Section 4.7.2.
- *control*: update of data matrices and PCA (Sec. 4.6.1), estimation of the inverse interaction matrix (Sec. 4.6.2), computation of the command (Sec. 4.6.3), sending the command to the robot, performing the command and acquiring the new robot poses.



	Experiment 1	Experiment 2	Experiment 3	Experiment 4	Experiment 5
simulation	1.950	1.893	1.782	1.808	1.985
image processing	0.043	0.045	0.045	0.044	0.046
correction	0.018	0.016	0.016	0.017	0.031
control	1.091	0.905	0.935	0.873	0.984

Figure 4.23: Average time per iteration of the modules for the five sponge experiments shown in Fig. 4.17.

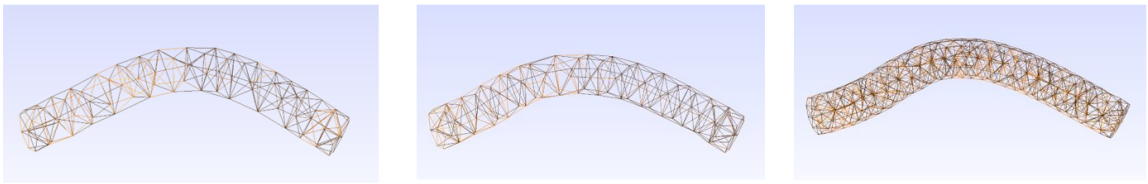
Module	simulation	image processing	correction	control
Average time	1.884	0.045	0.019	0.958

Table 4.4: Average time of the modules for the experiments presented in Fig. 4.17.

Figure 4.23 shows the average duration of each of these modules per iteration, for the five sponge experiments, and Tab.4.4 summarizes the average time of each modules for all these experiments. We see that for the sponge object, the average duration of a single iteration is close to $3s$, with the biggest part of it dedicated to the *simulation* module (almost $2s$). The second most time-consuming module is *control*, which

takes about a second per iteration. This module depends on the gains of the robot (which are tuned so that the motions don't look too jerky), since it waits for the motion of the end-effectors to be executed to obtain the new poses. In this module are also included the PCA (4.6.1), Jacobian computation (4.6.2), control and robot joints commands computation (4.6.3), communication by socket[5] between python and the RKCL C++ robot control codes (both ways: sending the command and receiving the new pose). Point cloud acquisition takes around $0.045s$ in average. ICP algorithm runs quite quickly compared to the rest, with an average time bellow $0.02s$ when no correction step is needed. However, this time increases quickly with the correction step, since additional simulations are needed. Looking at the last experiment shown in Fig.4.19, where several correction steps are performed, we obtain an average time for the *correction* module during the experiment of $0.0168s$ per iteration, versus $0.751s$ in average for a correction step.

In summary, additionally to the code not being optimized, the control loop is mostly slowed down by the FEM simulations. One way of reducing their duration, is to study the effect of the mesh size. We conduct additional experiments with meshes of different sizes, presented in Fig. 4.24. The timing of the *simulation* and *correction* modules, as well as the average timing of one displacement constraint generated for the data initialization (4.4), referred to as *initialization step*, are plotted on Fig. 4.25.



(a) 89 nodes, 381 elements

(b) 102 nodes, 452 elements

(c) 374 nodes, 1766 elements

Figure 4.24: Meshes of different sizes for the foam noodle

From these experiments, it is clear that the simulation time largely depends on the size of the meshes, for the initialization data generation as well as for the simulation during the control loop. Both modules take an average time close to $0.5s$ for a mesh made of around 100 nodes, while they take between 2 to $3s$ for a mesh containing 374 nodes. The *correction* module time is, in total, lower, as it is not performed often. In the sponge experiments, the tetrahedral mesh used for the computations contains 579 nodes and 1628 elements, which is quite significant, and which explains the slow pace of the simulations and as a result, of the iterations.

It is however important not to reduce the size of the mesh too much, so as to not

[5]https://nanomsg.org/

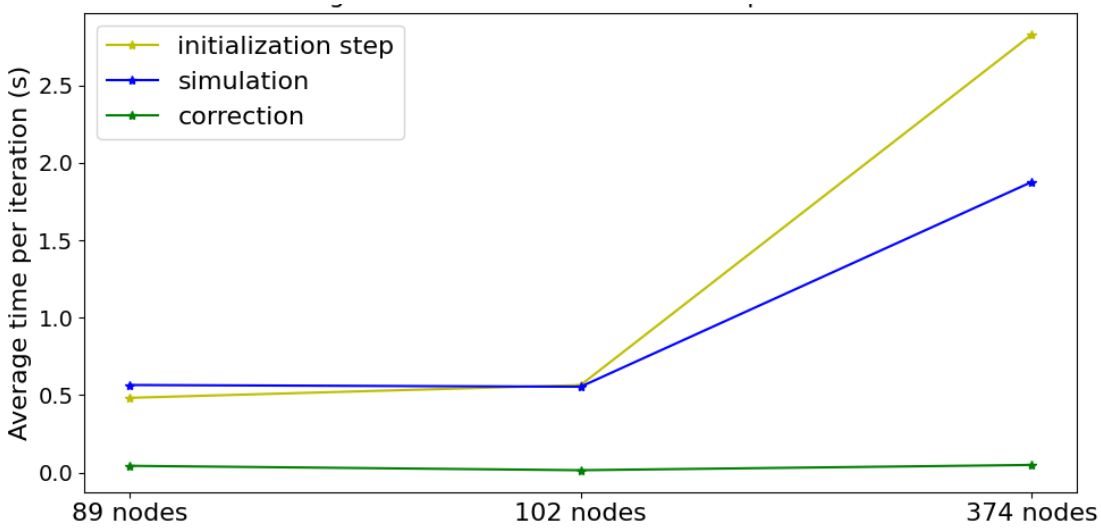


Figure 4.25: Average time per iteration for different mesh sizes

lose shape data, as one can observe a curvature difference between the meshes in Fig. 4.24, with mesh (c) closest to the observed one. It is also important to keep in mind that the least the number of elements, the more rigid the mesh. Choosing the size of the tetrahedral mesh consists in finding a satisfying trade-off between shape details, computation time and rigidity. The whole code could also be optimized to be faster, using multi threading for instance. Our work was however more focused on the method, but it is to keep in mind for future work.

4.10 Conclusion and discussion

In this chapter, we present a framework for shaping soft objects, based on both visual-servoing and model-based methods. Similarly to the method presented in chapter 2, we use PCA to reduce the dimension of the object, and we estimate a linear local mapping between past features and the corresponding robot poses. However, in this chapter, we also represent the object by a triangular and a tetrahedral mesh, in order to estimate the deformation of the object with regards to the motions of the end-effectors at every step, through FEM simulation. We present experiments with different object and tasks in $\mathbb{SE}(3)$, summarized in Tab. 4.5.

Meshes of the objects are required to use this framework. However, this is the case in this industry where CAO models of the manipulated objects are usually known. There also exist many methods to reconstruct meshes from either scans, point clouds or DLO parameterization, as discussed in chapter 3. Knowledge on the material - material parameters, weight - are also to be known if gravity affects the shape significantly. The assumption of linear elasticity also cannot be applied to all objects, and this could cause the deformation estimation from the FEM simulation not to fit

Object	Experiment	Tasks
Sponge	1	Bending, translations only
	2	Compression with rotations
	3	Torsion
	4	Bending (around y-axis)
	5	Bending with rotations
Thin dented foam	1	Bending with rotation
	2	Stretching
Dented foam	1	Bending, spinning
	2	Torsion
Foam noodle (DLO)	1	Out of plane deformation, translations only
	2	Out of plane deformation, with rotation

Table 4.5: Summary of the experiments preformed in this chapter.

with the observed behavior of the object during manipulation. In contrast, the framework presented in chapter 2, if there was no prior knowledge of the manipulated object.

The control loop running time also proves to be quite slow. Future work should involve code optimization, to allow faster running, and keeping in mind the size of the meshes while constructing them. The investigation of other types of objects - like clothes - and models (non-linear) with this framework would also be interesting.

Conclusion

Short summary of the thesis

In this thesis, we presented two frameworks for controlling the shape of deformable object, using visual feedback (chapters 2 and 4), as well as various tools for modeling non-rigid objects deformation (3).

Chapter 2 presents a dual-arm controller to shape the 3D contours of an object's visible surface, observed with a RGB-D camera, into desired 3D contours.

Chapter 3 introduces a setup for real-time FEM simulation to reproduce the motions of the robot end-effectors in simulation, in a Real-to-Sim manner, and consequently estimate the resulting deformation of the object. This chapter also presents tools for meshes reconstruction, for DLOs as well as for reconstructed, partial point clouds.

Finally, chapter 4 puts into practice the setup presented in Chapter 3, in a control framework which relies on both a model and on visual feedback. The presented dual-arm controller allows to shape the 3D mesh of a considered object into a desired one, selected through an intuitive graphic interface.

Comparison of the proposed frameworks and discussion

The presented control frameworks (2 and 4) both allow to:

- Shape objects in $\mathbb{SE}(3)$ with a dual-arm robot;
- Shape objects of different topology and materials;
- Achieve out-of-plane deformations such as torsion, bending or compression.
- Consider large deformations.

However, they rely on different hypotheses and prior knowledge, and they present different advantages and drawbacks, as detailed in Tab.4.6.

Framework	Chapter 2	Chapter 4
Hypotheses	- Object can be segmented from the scene by its color - The visible part of the object does not change too much from the start to final shape	- Object can be segmented from the end-effectors - The behavior of the object can be approximated with linear elasticity
Prior knowledge	- Color of the object	- Color of the end-effectors - Meshes of the object - m, E, ν if the gravity cannot be neglected w.r.t. the forces applied by the robot
Advantages	- Fast computations - Little prior knowledge is needed	- Whole volume of the object, including the self-occluded parts, are controlled - Possibility of monitoring the applied stress - Target selection through user-friendly HMI
Drawbacks	- Only the 3D contour of the visible surface is controlled - No monitoring of the applied stress (risk of damage) - The target must be manually reached before manipulation to obtain the target 3D contour	- Slow computations - More prior knowledge of the object is needed

Table 4.6: Comparison of the two control frameworks presented in this thesis.

The major difference between the two methods is the shape representation; in chapter 2, it consisted of visible contours projected in 3D. The use of meshes in chapter 4 is

a step-up from this, allowing to put into greater use the workspace, and to control the deformation of the whole volume of the object in 3D, and not only that of its visible part. This representation also allows to deal with the self-occlusions of the objects during manipulation, and the use of FEM simulations allows to estimate the deformation of the object even in case of occlusions on the image.

Another notable difference is in the target shape selection. While the framework of chapter 2 requires a "reachable" target, which must be manually reached beforehand, either by moving the robot or by manually deforming the object, the framework presented in chapter 4 does not require this. A simple graphic interface (HMI) and keyboard controller allow a target selection that is intuitive, visual and does not require the robot. This target selection can also be used to monitor the stress, hence ensuring that the chosen target will not damage the object.

All in all, chapter 4 presents a framework to shape soft objects more precisely (in their full volume) than the one presented in chapter 2. Yet, this framework requires more information about the manipulated object than that of Chapter 2.

Perspectives

Let us now discuss improvement leads and future work.

Optimization of the performances
As it was discussed in 4.9.2, a short-term improvement would be to optimize the control loop so that the manipulation of the object is faster. Other than optimizing the code, generating smaller meshes (i.e. with less nodes) could be achieved by parametrising the method presented in chapter 3.3.

Cooperative tasks space
Another interesting point would be the implementation of the cooperative tasks space presented in chapter 2 in the model-based control framework presented in chapter 4. This would allow to free DOFs in the case where a rigid object is manipulated, or in the case where only the *deformation* of the object (and not its position/orientation in space) is of interest.

Material parameters estimation
A longer-term work would be the implementation of material estimation through manipulation, for instance by relying on force sensors on the robot end-effectors. Yet, the precision of the force sensors would need to be high. We could envisage making an estimation through an initial manipulation, by relating the observed displacement (from visual feedback) to the applied force (measured by the force sensors), for example through Hooke's law.

Human interaction
Finally, the implementation of human interaction with the object for cooperative manipulation, would be interesting. Application like handling of large objects (like sheets), folding, or ”deform and place” especially, would make compelling use-cases.

APPENDIX A

Résumé des travaux

A.1 Introduction et contexte

Dans ce manuscrit, nous présentons des travaux de recherche portant sur la manipulation robotique bi-bras d'objets déformables.

De nombreux travaux existants traite de la manipulation d'objets souples; nous pouvons distinguer deux catégories de travaux:

- Ceux portant sur le *traçage* de la déformation;
- Ceux portant sur le *contrôle* de la déformation.

Cette première catégorie englobe les travaux visant à suivre et représenter l'évolution de la forme d'un objet sujet à des déformations. Si nous nous concentrons sur les travaux portant sur un traçage *visuel* des déformations, c'est-à-dire un traçage basé sur un retour visuel. Ces méthodes peuvent être *basée modèle*, s'appuyant sur un modèle géométrique (une topologie) ou un modèle mécanique. A l'inverse, il existe aussi des méthodes dites *sans modèle*, ne s'appuyant que sur les retours visuels.

En ce qui concerne la seconde catégorie (*contrôle*), il s'agit de travaux visant a proposer des méthodes pour contrôler des objets non-rigides pour accomplir certaines tâches, comme plier un vêtement, insérer un câble, ou encore cueillir un fruit. Nous étudions en particulier *le contrôle de forme*, dont le but est de donner une forme désirée à un objet. De telles tâches peuvent être accomplie avec des méthodes *basée données*, consistant à apprendre le comportement de certains objets grâce à des données collectées au préalables, à travers de l'apprentissage par renforcement ou par démonstration, par exemple. D'autres méthodes sont elles dites *basées modèle*, et s'appuient sur un modèle physique choisit pour prédire le comportement de l'objet, et ainsi les

changements de forme. Enfin, d'autres méthodes passent plutôt par une *estimation de Jacobienne en ligne*, utilisant quelques données passées pour estimer itérativement un modèle simplifié entre les commandes du robot et la déformation de l'objet.

Dans l'ensemble, les travaux existants montrent que le contrôle de forme d'objets non-rigides implique différentes problématiques à considerer:

- Choisir un descripteur de forme pertinent (en fonction des capteurs disponibles, le type d'objet, le modèle connu, etc.) et traçable;
- Décrire les commandes du robot par rapport au descripteur de forme (via un apprentissage basé données, part estimation de matrice Jacobienne à partir de données passées, ou selon un modèle physique);
- Fermer la boucle de contrôle grâce à un retour capteur.

Les méthodes basées données sont adéquates pour estimer les déformations d'ojets souples dont peu d'informations sont connues: similairement à la façon dont un être humain procéderai pour déformer un objet souple, apprenant en manipulant et touchant. Cependant, l'utilisation de réseaux de neuronnes profond ou autres méthodes d'apprentissage (par démonstration ou renforcement) nécessite des bases de données conséquentes pour l'entrainement (qui peuvent être laborieuses à récolter), et la phase d'apprentissage est couteuse en temps. Quant aux méthodes basées modèles, elles se montrent précises pour prédire les déformations d'objets déformables tant que les propriétés physiques (matériau, loi de comportement) et la géométrie (modèle 3D) de l'objet sont connues, limitant les possibilités d'application. Un compromis entre la collecte de données et l'estimation d'un modèle précis est donc crucial. Nous commençons par présenter une méthode de contrôle bi-bras pour déformer des objets souples ou rigides en un contour désiré, dans $\mathbb{SE}(3)$.

A.2 Asservissement visuel en tâches cooperatives pour contrôle de la forme d'objets souples à deux bras

Les auteurs de [ZNAPC20] présentent un contrôleur de forme et/ou position d'objets souples ou rigides. Leur methode implémente une Analyse en Composantes Principales (ACP) pour encoder le contour dans l'image de l'objet manipulé en un plus petit vecteur d'informations visuelles. Ils présentent des expériences de déplacement et déformation de contour 2D avec un seul bras se déplaçant dans le plan (3 Degrés De Liberté, DDL).

Notre but dans cette section est de proposer une méthode construite à partir du travail de ces auteurs, étendue à la manipulation à deux bras et en 3D.

Nos contributions sont les suivantes:

- Nous augmentons le nombre de DDL de [ZNAPC20] de 3 (mouvements planaires à un bras) à 12 (mouvements dans $\mathbb{SE}(3)$ - trois rotations et trois translations, pour deux bras), permettant de plus nombreuses applications.

- Nous implémentons une représentation en tâches coopératives ([CCS96]) afin de contrôler d'une part la *déformation*, d'autre part la *position/orientation* de l'objet, avec 6 DDL pour chaque.
- Nous validons le contrôleur dans des experiences sur différents objets, en considérant leur déformation en 3D.

La boucle de contrôle se déroule comme suit: d'abord, une séquence de petits mouvements autour de la forme au repos de l'objet est générée, afin de collecter les contours déformés $\mathbf{c}$ (de l'objet observé par une caméra RBG-D, et obtenus par traitement visuel, en 3D) et les positions des Organes Terminaux (OT) $\mathbf{r}$ correspondantes. Les contours collectés permettent, à travers une APC, d'obtenir une matrice de projection dans l'espace réduit. Celle-ci est utilisée pour réduire les contours en vecteur d'information visuelles de plus petite dimension, notés $\mathbf{s}$. Ces données collectées permettent aussi d'estimer la matrice d'interaction locale inverse $\mathbf{L}^{-1}$, telle que:

$$\delta\mathbf{s} = \mathbf{L}\delta\mathbf{r} \tag{A.1}$$

Avec $\delta\mathbf{s}$ la variation du vecteur d'informations visuelles et $\delta\mathbf{r}$ la variation des positions des OTs. Cette matrice est utilisée pour calculer les commandes en pose des OTs de sorte que l'objet atteigne des formes intermédiaires, jusqu'à finalement obtenir la forme désirée, exprimée as $\mathbf{s}*$. La boucle de contrôle est illustrée dans la Fig.A.1.

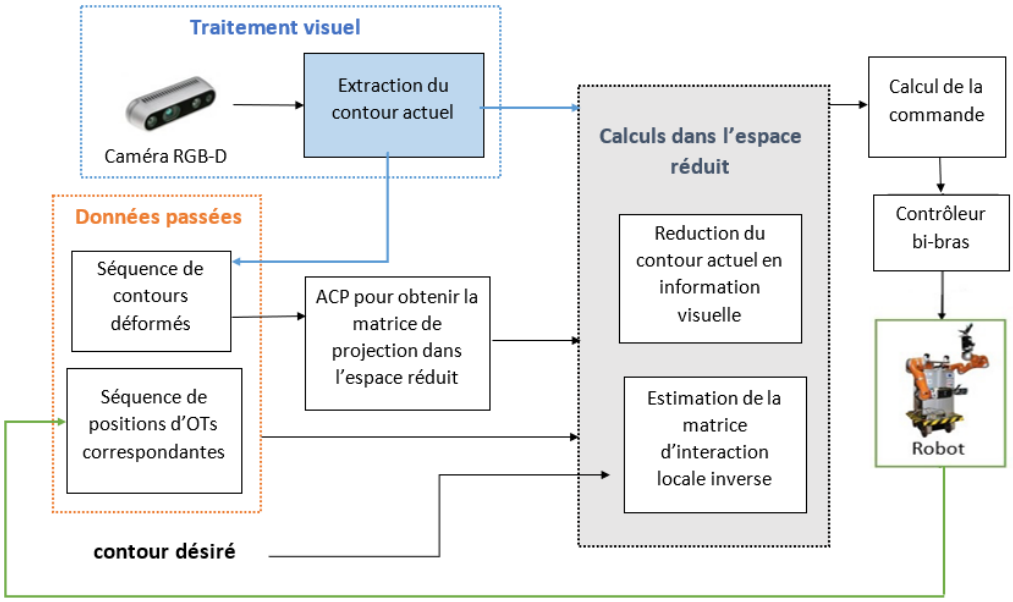


Figure A.1: Résumé de la boucle de contrôle pour déformation de contours.

Nous montrons les performances du contrôleur en manipulant diverse objets (mousses, gant en plastique, boite en carton) jusqu'à ce qu'ils atteignent une forme désirée, incluant des déformations telles que torsion, compression, rotation larges ou simplement changement de position/orientation dans l'espace 3D.

A.3 Estimation de modèle et simulations

De la section précedente ressort une limite dans la représentation d'un objet utilisant des contours visibles: il est impossible de suivre, et donc contrôler, les parties de l'objet non visibles par la caméra. Pour étendre le contrôle à l'ensemble de l'objet, nous présentons dans cette section une solution basée sur la mise en place de simulations physiques en temps réels.

La Méthode des Eléments Finis (MEF) est une méthode numérique de résolution d'équations différentielles, basée sur la discrétisation d'un domaine continu (par example, un objet) en un nombre fini de petits éléments. Pour ce faire, le domaine étudié est échantillonné en un maillage $\mathcal{M}$ contenants des noeuds $\mathbf{m}$ et des élements (tétrahèdres), et ensuite modélisé en fonction de conditions aux limites (contraintes sur certaines surfaces ou nœuds), de forces extérieures et de modèles de matériaux (élasticité linéaire, hyper-élasticité, visco-élasticité...). Nous utilisions le logiciel Simulation Open Framework Architecture (SOFA)[1] [FDD+12] pour effectuer des simulations MEF en temps réel, et obtenir une estimation de la déformation de l'objet manipulé en prenons en compte les entrées du robot (les positions des OTs) en temps réel. L'objectif final est d'utiliser ces simulations comme retour sur l'état de la forme de l'objet dans la boucle de contrôle, d'où la nécessité de calculs rapides et efficaces.

Nous représentons les OTs par ce que l'on appelle des particules rigides, notées $\mathbf{pr}_l$ et $\mathbf{pr}_r$ pour l'OT gauche et droit respectivement. Les noeuds contenus dans des boites d'encombrement autour de ces particules rigides sont dénotés $\mathbf{H}_l$ et $\mathbf{H}_r$ respectivement, et sont attachés de façon rigide à la particule rigide correspondante. Ainsi, lorsque le les OTs ce déplacent (en position et rotation), il est possible de reporter ses nouvelles positions dans la simulation et estimer comment celles-ci influent la forme de l'objet grâce aux calculs élements finits. Dans la Fig. A.2 est illustrée la mise en place d'une simulation dans SOFA.

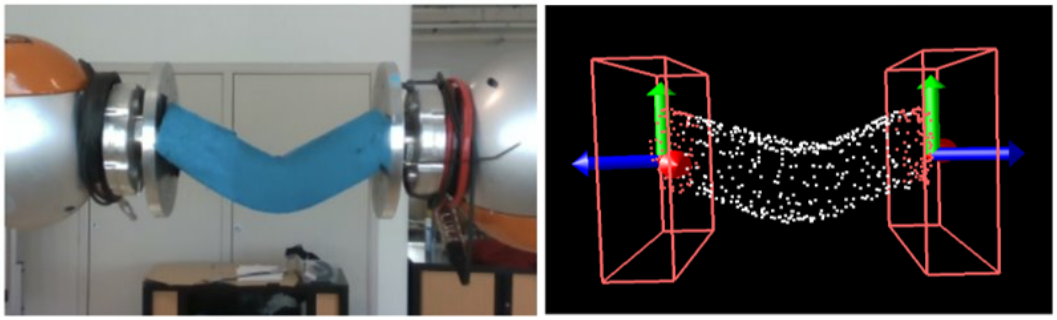


Figure A.2: Simulation de l'objet observé par la camera (à guache) dans SOFA (à droite). Les particules rigides, repésentée par les repères, sont attaché de façon rigide aux noeuds contenus dans les boites d'encombrement autour de celles-ci, en rouge.

[1] https://www.sofa-framework.org/

Nous présentons aussi des méthodes de maillage afin d'obtenir les modèles géomètriques des objets considerés, dans deux cas:

- L'objet est un Objet Linéaire Déformable (OLD); il s'agit d'objets dont la longueur est assez grande par rapport aux autres dimensions que ceux-ci peuvent être réduits à une seule dimension, comme un cable, une corde, etc.
- Un objet dont la surface extérieure a été reconstruite sous forme d'un nuage de points.

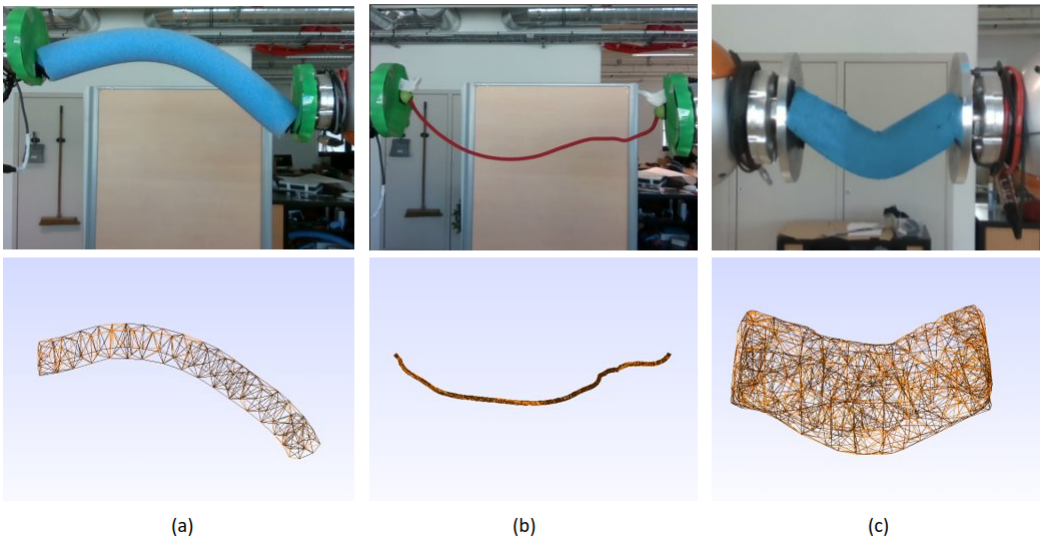


Figure A.3: Examples de maillages obtenus pour une frite en mousse (a), une corde (b), et une éponge (c).

La Fig. A.3 présente des résultats de maillages pour différents objets obtenus avec ces méthodes.

A.4 Déformation d'objets souples à deux bras en 3D, basée données et modèle

Dans cette section, nous intégrons la simulation présentée dans la section précédente dans une boucle de contrôle de forme pour obtenir un contrôleur basé à la fois vision et modèle. Nous utilisons maintenant les noeuds du maillage comme descripteur de forme, permettant de suivre et contrôler l'objet dans son intégralité, y compris les parties non-observées par la caméra. Pour se faire, nous proposons de reporter les mouvements du robot dans la simulation MEF décrite précédemment, en temps réel. Ainsi, une déformation estimée de l'objet résultante est appliquée au maillage. Le maillage déformé est ensuite comparé au nuage de points de l'objet observé par la camera. Cette déformation *estimée* (le maillage) est donc évalué par rapport à la déformation

observée (le nuage de point); dans le cas où l'erreur entre les deux est supérieure à un seuil déterminé, une étape de correction au point cloud est alors réalisée. Le calcul de la commande se déroule similairement à celui du contrôleur présenté section A.2: une matrice contenant les maillages déformés précédants est utilisée pour extraire les composantes principales, et par conséquent réduire la dimension des maillages en vecteurs d'information de forme $\mathbf{s}$. Utilisant aussi la matrice contenant les données de position des OTs précédantes, la matrice d'interaction inverse $\mathbf{L}^{-1}$ est estimée et utilisée pour calculée les positions des OTs qui permettrons d'atteindre un maillage désiré $\mathbf{m}*$. La boucle de contrôle est illustrée dans Fig.A.4.

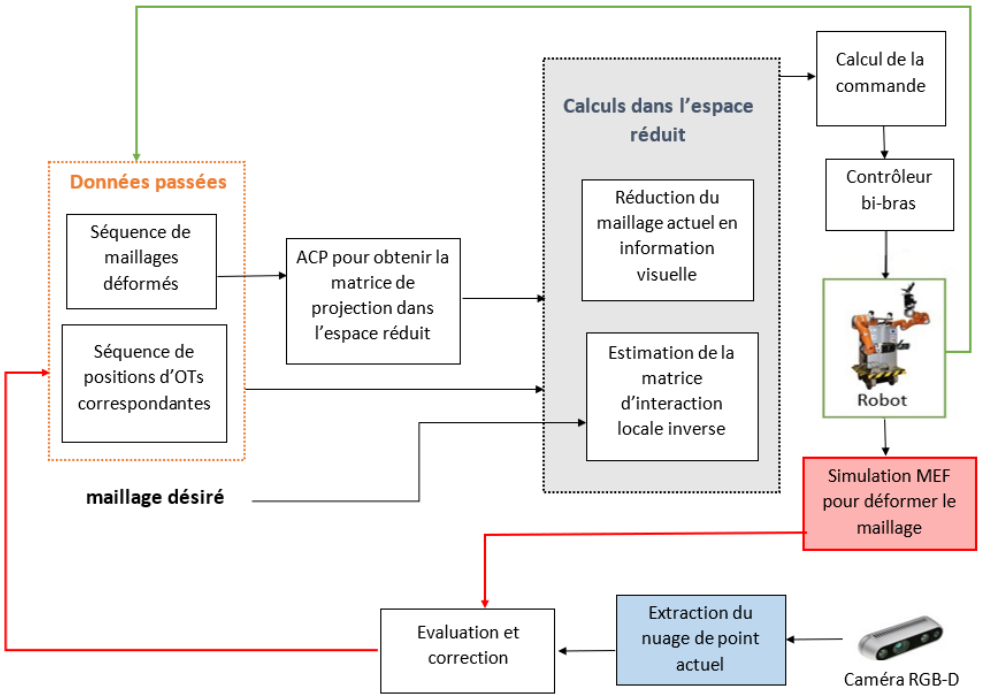


Figure A.4: Résumé de la boucle de contrôle pour deformation de maillages.

Les avantages de cette méthode de contrôle sont diverses; l'utilisation de simulation physiques permet notamment de choisir une forme désirée de façon interactive, à travers une interface graphique facile d'utilisation. De plus, si les paramètres matériaux de l'objet sont connus, il est possible de surveiller les contraintes internes et de s'assurer que l'objet n'est pas abîmé (par exemple, déformé dans le domaine plastique) durant la manipulation. Enfin, il est aussi possible de tester la boucle de contrôle en simulation uniquement, en faisant abstraction du retour visuel, et ainsi s'assurer du fonctionnement de la méthode sans risque et sans avoir à utiliser un robot.

Nous présentons des résultats expérimentaux en simulation et avec un robot bi-bras, et des objets de différentes formes et rigidités.

A.5 Conclusion

Dans cette thèse, nous présentons deux méthodes pour contrôler la forme d'objets déformables se basant sur un retour visuel, ainsi que de divers outils de modélisation de déformation pour objets non-rigides. Le chapitre 2 présente un contrôleur bi-bras pour déformer le contour 3D de la surface visible d'objets déformables, observé avec une caméra RGB-D, en un contour 3D désiré.

Le chapitre 3 présente la mise en place de simulation MEF en temps réel pour reproduire les mouvements des OTs du robot en simulation, et par conséquent estimer la déformation résultante de l'objet. Ce chapitre présente également des outils pour reconstruire des maillages, pour les OLD ainsi que pour les nuages de points reconstruits.

Enfin, le chapitre 4 met en pratique le dispositif de simulation présenté au chapitre précédent, dans une boucle de contrôle qui s'appuie à la fois sur un modèle et sur un retour visuel. Le contrôleur bi-bras présenté permet de déformer le maillage d'un objet considéré en celui souhaité, sélectionnés via une interface graphique intuitive.

Les deux méthodes de contôle présentent des differences en terme d'hypothèses et de connaissances sur l'objet préalables (par example, un maillage). Dans l'ensemble, le chapitre 4 présente une méthode pour déformer plus précisément les objets non-rigides (en considérant leur volume complet) que celle présentée au chapitre 2. Cependant, cette méthode nécessite plus d'informations sur l'objet manipulé que celle du chapitre 2.

Les méthodes présentées permettent cependant toutes deux de :

- Déformer des objets dans $\mathbb{SE}(3)$ avec un robot à deux bras ;
- Déformer des objets de topologies et de matériaux différents ;
- Réaliser des déformations hors plan telles que la torsion, la flexion ou la compression.

Bibliography

[AACRM+20] Miguel Aranda, Juan Antonio Corrales Ramon, Youcef Mezouar, Adrien Bartoli, and Erol Özgür. Monocular visual shape tracking and servoing for isometrically deforming objects. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7542–7549, 2020. 14, 43, 44, 46

[AALN+22] Omid Aghajanzadeh, Miguel Aranda, Gonzalo López-Nicolás, Roland Lenain, and Youcef Mezouar. An offline geometric model for controlling the shape of elastic linear objects. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2175–2181, 2022. 39, 44, 46

[AFB15] Benjamin Allain, Jean-Sébastien Franco, and Edmond Boyer. An efficient volumetric framework for shape tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 268–276, 2015. 34

[AMN18] Antonio Agudo and Francesc Moreno-Noguer. Force-based representation for non-rigid shape and elastic model estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(9):2137–2150, 2018. 38

[APR+22] Omid Aghajanzadeh, Guillaume Picard, Juan Antonio Corrales Ramon, Christophe Cariou, Roland Lenain, and Youcef Mezouar. Optimal deformation control framework for elastic linear objects. In *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*, pages 722–728, 2022. 39, 44, 46

[Arm16] Anthony E Armenàkas. *Advanced mechanics of materials and applied elasticity*. CRC Press, 2016. 95

[Ber13] Dmitry Berenson. Manipulation of deformable objects without modeling and simulating deformation. In *2013 IEEE/RSJ International Confer-*

ence on Intelligent Robots and Systems, pages 4525–4532. IEEE, 2013. 43, 46

[BF91] Franco Brezzi and Michel Fortin. Variational formulations and finite element methods. In *Mixed and hybrid finite element methods*, pages 1–35. Springer, 1991. 71

[BGC+15] Adrien Bartoli, Yan Gérard, François Chadebecq, Toby Collins, and Daniel Pizarro. Shape-from-template. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(10):2099–2118, 2015. 44

[BJM+18] Ante Buljac, Clément Jailin, Arturo Mendoza, Jan Neggers, Thibault Taillandier-Thomas, Amine Bouterf, Benjamin Smaniotto, François Hild, and Stéphane Roux. Digital volume correlation: review of progress and challenges. *Experimental Mechanics*, 58:661–708, 2018. 32

[BKTA+15] E Bar-Kochba, J Toyjanova, E Andrews, K-S Kim, and Christian Franck. A fast iterative digital volume correlation algorithm for large deformations. *Experimental Mechanics*, 55(1):261–274, 2015. 31

[BM92] P.J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992. 29

[BSFS99] Brian K Bay, Tait S Smith, David P Fyhrie, and Malik Saad. Digital volume correlation: three-dimensional strain mapping using x-ray tomography. *Experimental mechanics*, 39:217–226, 1999. 31

[BTVG06] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *Computer Vision–ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I 9*, pages 404–417. Springer, 2006. 29

[CAL+22] Eulalie Coevoet, Yinoussa Adagolodjo, Meichun Lin, Christian Duriez, and Fanny Ficuciello. Planning of soft-rigid hybrid arms in contact with compliant environment: Application to the transrectal biopsy of the prostate. *IEEE Robotics and Automation Letters*, 7(2):4853–4860, 2022. 13, 21

[CB19] Cheng Chi and Dmitry Berenson. Occlusion-robust deformable object tracking without physics simulation. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6443–6450, 2019. 30, 36

[CBI10] Cedric Cagniart, Edmond Boyer, and Slobodan Ilic. Iterative deformable surface tracking in multi-view setups. In *3DPVT 2010-5th*

International Symposium on 3D Data Processing, Visualization and Transmission, 2010. 34

[CCS96] Pasquale Chiacchio, S. Chiaverini, and Bruno Siciliano. Direct and inverse kinematics for coordinated motion tasks of a two-manipulator system. *Journal of Dynamic Systems Measurement and Control-transactions of The Asme - J DYN SYST MEAS CONTR*, 118, 12 1996. 50, 59, 127

[Cha07] François Chaumette. *Visual Servoing*, chapter 6, pages 279–336. John Wiley and Sons, Ltd, 2007. 41, 109

[CP20] Peng Chang and Taşkin Padif. Sim2real2sim: Bridging the gap between simulation and real-world in flexible object manipulation. In *2020 Fourth IEEE International Conference on Robotic Computing (IRC)*, pages 56–62. IEEE, 2020. 104

[CPN+19] Andrea Cherubini, Robin Passama, Benjamin Navarro, Mohamed Sorour, Abdellah Khelloufi, Osama Mazhar, Sonny Tarbouriech, Jihong Zhu, Olivier Tempier, André Crosnier, Philippe Fraisse, and Sofiane Ramdani. A collaborative robot for the factory of the future: Bazar. *The International Journal of Advanced Manufacturing Technology*, 105:1–17, 12 2019. 24

[CPP12] Ana-Maria Cretu, Pierre Payeur, and Emil M. Petriu. Soft object deformation monitoring and learning for model-based robotic hand manipulation. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(3):740–753, 2012. 30

[CPS+17] John Campbell, Nicole Petta, Ori Stein, Ying Liu, Y. Lu, and L. J. Jiang. Three-dimensional printing and deformation behavior of low-density target structures by two-photon polymerization. page 66, 08 2017. 17, 96

[CR03] Haili Chui and Anand Rangarajan. A new point matching algorithm for non-rigid registration. *Computer Vision and Image Understanding*, 89(2):114–141, 2003. Nonrigid Image Registration. 36

[CZGP22] Alessio Caporali, Riccardo Zanella, Daniele De Greogrio, and Gianluca Palli. Ariadne+: Deep learning–based augmented framework for the instance segmentation of wires. *IEEE Transactions on Industrial Informatics*, 18(12):8607–8617, 2022. 77

[CZK15] Sungjoon Choi, Qian-Yi Zhou, and Vladlen Koltun. Robust reconstruction of indoor scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5556–5565, 2015. 76

[DBPC18] Simon Duenser, James M. Bern, Roi Poranne, and Stelian Coros. Interactive robotic manipulation of elastic objects. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3476–3481, 2018. 13, 39, 40, 46, 104

[DFF13] Mingsong Dou, Henry Fuchs, and Jan-Michael Frahm. Scanning and tracking dynamic objects with commodity depth cameras. In *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 99–106, 2013. 76

[DKD+16] Mingsong Dou, Sameh Khamis, Yury Degtyarev, Philip Davidson, Sean Ryan Fanello, Adarsh Kowdle, Sergio Orts Escolano, Christoph Rhemann, David Kim, Jonathan Taylor, et al. Fusion4d: Real-time performance capture of challenging scenes. *ACM Transactions on Graphics (ToG)*, 35(4):1–13, 2016. 76

[DR19] Joseph DelPreto and Daniela Rus. Sharing the load: Human-robot team lifting using muscle activity. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7906–7912, 2019. 13, 21

[DZAL+22] Mélodie Hani Daniel Zakaria, Miguel Aranda, Laurent Lequièvre, Sébastien Lengagne, Juan Antonio Corrales Ramón, and Youcef Mezouar. Robotic control of the deformation of soft linear objects using deep reinforcement learning. In *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*, pages 1516–1522, 2022. 37, 46

[EKS83] H. Edelsbrunner, D. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, 29(4):551–559, 1983. 82

[FDD+12] François Faure, Christian Duriez, Hervé Delingette, Jérémie Allard, Benjamin Gilles, Stéphanie Marchesseau, Hugo Talbot, Hadrien Courtecuisse, Guillaume Bousquet, Igor Peterlik, et al. Sofa: A multi-model framework for interactive physical simulation. *Soft tissue biomechanical modeling for computer assisted surgery*, pages 283–321, 2012. 72, 128

[FJP+12] Andreas Rune Fugl, Andreas Jordt, Henrik Gordon Petersen, Morten Willatzen, and Reinhard Koch. Simultaneous estimation of material properties and pose for deformable objects from depth and color images. In *Pattern Recognition: Joint 34th DAGM and 36th OAGM Symposium, Graz, Austria, August 28-31, 2012. Proceedings 34*, pages 165–174. Springer, 2012. 38

[Flü65] S. Flügge. *Handbuch Der Physik*. Number vol. 3,ptie. 3 in Handbuch Der Physik. Springer, 1965. 70

[FMC+18] Fanny Ficuciello, Alessandro Migliozzi, Eulalie Coevoet, Antoine Petit, and Christian Duriez. Fem-based deformation control for dexterous manipulation of 3d soft objects. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4007–4013. IEEE, 2018. 39, 46

[FNT+11] Kent Fujiwara, Ko Nishino, Jun Takamatsu, Bo Zheng, and Katsushi Ikeuchi. Locally rigid globally non-rigid surface registration. In *2011 International Conference on Computer Vision*, pages 1527–1534, 2011. 31

[GCLW+20] Irene Garcia-Camacho, Martina Lippi, Michael C. Welle, Hang Yin, Rika Antonova, Anastasiia Varava, Julia Borras, Carme Torras, Alessandro Marino, Guillem Alenyà, and Danica Kragic. Benchmarking bimanual cloth manipulation. *IEEE Robotics and Automation Letters*, 5(2):1111–1118, 2020. 22

[GLH11] M Gates, J Lambros, and MT Heath. Towards high performance digital volume correlation. *Experimental Mechanics*, 51(4):491–507, 2011. 31

[GPCB19] Mathias Gallardo, Daniel Pizarro, Toby Collins, and Adrien Bartoli. Shape-from-template with curves. *International Journal of Computer Vision*, 128:121 – 165, 2019. 77

[GPIK17] Puren Guler, Alessandro Pieropan, Masatoshi Ishikawa, and Danica Kragic. Estimating deformability of objects using meshless shape matching. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5941–5948, 2017. 38

[HBB+21] Yunhai Han, Rahul Batra, Nathan Boyd, Tuo Zhao, Yu She, Seth Hutchinson, and Ye Zhao. Learning generalizable vision-tactile robotic grasping strategy for deformable objects via transformer. 12 2021. 13, 21

[HHC96] Seth Hutchinson, Gregory D Hager, and Peter I Corke. A tutorial on visual servo control. *IEEE transactions on robotics and automation*, 12(5):651–670, 1996. 41

[HHS+19] Zhe Hu, Tao Han, Peigen Sun, Jia Pan, and Dinesh Manocha. 3-d deformable object manipulation using deep neural networks. *IEEE Robotics and Automation Letters*, 4(4):4255–4261, 2019. 13, 37, 46

[HPC17] Fei Hui, Pierre Payeur, and Ana-Maria Cretu. Visual tracking of deformation and classification of non-rigid objects with robot hand probing. *Robotics*, 6(1):5, 2017. 30

[HSP18] Zhe Hu, Peigen Sun, and Jia Pan. Three-dimensional deformable object manipulation using fast online gaussian process regression. *IEEE Robotics and Automation Letters*, 3(2):979–986, 2018. 43, 46

[HZSP18] Tao Han, Xuan Zhao, Peigen Sun, and Jia Pan. Robust shape estimation for 3d deformable object manipulation. *arXiv preprint arXiv:1809.09802*, 2018. 35

[IZN+16] Matthias Innmann, Michael Zollhöfer, Matthias Nießner, Christian Theobalt, and Marc Stamminger. Volumedeform: Real-time volumetric non-rigid reconstruction. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VIII 14*, pages 362–379. Springer, 2016. 13, 34, 35

[JHPM18] Biao Jia, Zhe Hu, Jia Pan, and Dinesh Manocha. Manipulating highly deformable materials using a visual feedback dictionary. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 239–246, 2018. 43, 46

[JR14] Rodrigo Jamisola and Rodney Roberts. A more compact expression of relative jacobian based on individual manipulator jacobians. *Robotics and Autonomous Systems*, 63:158–164, 08 2014. 59

[JWZ+20] Shiyu Jin, Changhao Wang, Xinghao Zhu, Te Tang, and Masayoshi Tomizuka. Real-time state estimation of deformable objects with dynamical simulation. 2020. 13, 30

[KFB+21] A. Koessler, N. Roca Filella, B.C. Bouzgarrou, L. Lequièvre, and J.-A. Corrales Ramon. An efficient approach to closed-loop shape control of deformable objects using finite element models. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1637–1643, 2021. 40, 46

[KGD+03] Alexandre Krupa, Jacques Gangloff, Christophe Doignon, Michel F De Mathelin, Guillaume Morel, Joël Leroy, Luc Soler, and Jacques Marescaux. Autonomous 3-d positioning of surgical instruments in robotized laparoscopic surgery using visual servoing. *IEEE transactions on robotics and automation*, 19(5):842–853, 2003. 41

[LAAB14] Ibai Leizea, Hugo Alvarez, Iker Aguinaga, and Diego Borro. Real-time deformation, registration and tracking of solids based on physical simulation. In *2014 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 165–170, 2014. 34

[LCL19] Tristan Laidlow, Jan Czarnowski, and Stefan Leutenegger. Deepfusion: Real-time dense 3d reconstruction for monocular slam using single-view

depth and gradient predictions. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 4068–4074. IEEE, 2019. 76

[LHM+22] Robert Lee, Masashi Hamaya, Takayuki Murooka, Yoshihisa Ijiri, and Peter Corke. Sample-efficient learning of deformable linear object manipulation in the real world through self-supervision. *IEEE Robotics and Automation Letters*, 7(1):573–580, 2022. 13, 36, 37, 46

[LKM20] Romain Lagneau, Alexandre Krupa, and Maud Marchal. Automatic shape control of deformable wires based on model-free visual servoing. *IEEE Robotics and Automation Letters*, 5(4):5252–5259, 2020. 42, 46, 50

[LLG+15] Alex X Lee, Henry Lu, Abhishek Gupta, Sergey Levine, and Pieter Abbeel. Learning force-based manipulation of deformable objects from multiple demonstrations. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 177–184. IEEE, 2015. 36, 46

[LLP22] Wing Kam Liu, Shaofan Li, and Harold S Park. Eighty years of the finite element method: Birth, evolution, and future. *Archives of Computational Methods in Engineering*, 29(6):4431–4453, 2022. 68

[LLW+17] Qiang Lv, Huican Lin, Guosheng Wang, Heng Wei, and Yang Wang. Orb-slam-based tracing and 3d reconstruction for robot using kinect 2.0. In *2017 29th Chinese control and decision conference (CCDC)*, pages 3319–3324. IEEE, 2017. 76

[Log11] D.L. Logan. *A First Course in the Finite Element Method, SI Version*. Cengage Learning, 2011. 68

[Low99] David G Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999. 29

[LPF04] V. Lepetit, J. Pilet, and P. Fua. Point matching as a classification problem for fast and robust object pose estimation. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 2, pages II–II, 2004. 29

[LRK09] W Michael Lai, David Rubin, and Erhard Krempl. *Introduction to continuum mechanics*. Butterworth-Heinemann, 2009. 70

[LSGL18] Xiang Li, Xing Su, Yuan Gao, and Yun-Hui Liu. Vision-based robotic grasping and manipulation of usb wires. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3482–3487. IEEE, 2018. 22

[Mac67] J MacQueen. Classification and analysis of multivariate observations. In *5th Berkeley Symp. Math. Statist. Probability*, pages 281–297. University of California Los Angeles LA USA, 1967. 102

[MB18] D. Mcconachie and D. Berenson. Estimating model utility for deformable object manipulation using multiarmed bandit methods. *IEEE Transactions on Automation Science and Engineering*, 15(3):967–979, 2018. 38, 46

[MC13] Maxime Meilland and Andrew I. Comport. On unifying key-frame and voxel-based dense visual slam at large scales. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3677–3683, 2013. 76

[MM99] Songrit Maneewongvatana and David M Mount. Analysis of approximate nearest neighbor searching with clustered point sets. *arXiv preprint cs/9901013*, 1999. 103

[MS10] Andriy Myronenko and Xubo Song. Point set registration: Coherent point drift. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(12):2262–2275, 2010. 30

[MS22] Itamar Mishani and Avishai Sintov. Real-time non-visual shape estimation and robotic dual-arm manipulation control of an elastic wire. *IEEE Robotics and Automation Letters*, 7(1):422–429, 2022. 22

[MX23] Zhaoyuan Ma and Jing Xiao. Robotic perception-motion synergy for novel rope wrapping tasks. *IEEE Robotics and Automation Letters*, 2023. 77

[NAL18] David Navarro-Alarcon and Yun-Hui Liu. Fourier-based shape servoing: A new feedback method to actively deform soft objects into desired 2-d image contours. *IEEE Transactions on Robotics*, 34(1):272–279, 2018. 42, 46

[NAYW+16] David Navarro-Alarcon, Hiu Man Yip, Zerui Wang, Yun-Hui Liu, Fangxun Zhong, Tianxue Zhang, and Peng Li. Automatic 3-d manipulation of soft objects by robotic arms with an adaptive deformation model. *IEEE Transactions on Robotics*, 32(2):429–441, 2016. 42, 46

[NFS15] Richard A. Newcombe, Dieter Fox, and Steven M. Seitz. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015. 76

[NIH+11] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136, 2011. 76

[NKBC22] Olivia Nocentini, Jaeseok Kim, Zain Muhammad Bashir, and Filippo Cavallo. Learning-based control approaches for service robots on cloth manipulation and dressing assistance: a comprehensive review. *Journal of NeuroEngineering and Rehabilitation*, 19(1):1–25, 2022. 22

[NZIS13] Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Marc Stamminger. Real-time 3d reconstruction at scale using voxel hashing. *ACM Transactions on Graphics (ToG)*, 32(6):1–11, 2013. 76

[OFP04] Stanley Osher, Ronald Fedkiw, and K Piechor. Level set methods and dynamic implicit surfaces. *Appl. Mech. Rev.*, 57(3):B15–B15, 2004. 76

[PCLS18] Antoine Petit, Stéphane Cotin, Vincenzo Lippiello, and Bruno Siciliano. Capturing deformations of interacting non-rigid objects using rgb-d data. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 491–497, 2018. 33

[PLFS17] Antoine Petit, Vincenzo Lippiello, Giuseppe Andrea Fontanelli, and Bruno Siciliano. Tracking elastic deformable objects with an rgb-d sensor for a pizza chef robot. *Robotics and Autonomous Systems*, 88:187–201, 2017. 39

[PLS15] Antoine Petit, Vincenzo Lippiello, and Bruno Siciliano. Real-time tracking of 3d elastic objects with an rgb-d sensor. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3914–3921, 2015. 13, 32, 33

[PPP17] J-E Pierré, J-C Passieux, and J-N Périé. Finite element stereo digital image correlation: framework and mechanical regularization. *Experimental Mechanics*, 57(3):443–456, 2017. 32

[PR12] Victor A Prisacariu and Ian D Reid. Pwp3d: Real-time segmentation and tracking of 3d objects. *International journal of computer vision*, 98:335–354, 2012. 29

[QMZ+22] Jiaming Qi, Guangfu Ma, Jihong Zhu, Peng Zhou, Yueyong Lyu, Haibo Zhang, and David Navarro-Alarcon. Contour moments based manipulation of composite rigid-deformable objects with finite time model estimation and shape/position control. *IEEE/ASME Transactions on Mechatronics*, 27(5):2985–2996, 2022. 13, 43, 46

[RBB09] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (fpfh) for 3d registration. In *2009 IEEE International Conference on Robotics and Automation*, pages 3212–3217, 2009. 37

[Red13] Junuthula Narasimha Reddy. *An introduction to the finite element method*, volume 3. McGraw-Hill New York, 2013. 68

[RL01] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the icp algorithm. pages 145–152, 02 2001. 29

[RMB18] Mengyao Ruan, Dale McConachie, and Dmitry Berenson. Accounting for directional rigidity and constraints in control for manipulation of deformable objects without physical simulation. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 512–519, 2018. 38, 46

[RS00] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000. 30

[SA07] Olga Sorkine and Marc Alexa. As-rigid-as-possible surface modeling. In *Symposium on Geometry processing*, volume 4, pages 109–116. Citeseer, 2007. 33

[SBAMO22a] Mohammadreza Shetab-Bushehri, Miguel Aranda, Youcef Mezouar, and Erol Ozgur. As-rigid-as-possible shape servoing. *IEEE Robotics and Automation Letters*, 7(2):3898–3905, 2022. 44

[SBAMO22b] Mohammadreza Shetab-Bushehri, Miguel Aranda, Youcef Mezouar, and Erol Ozgur. Lattice-based shape tracking and servoing of elastic objects. *arXiv preprint arXiv:2209.01832*, 2022. 44, 46

[SBCI17] Miroslava Slavcheva, Maximilian Baust, Daniel Cremers, and Slobodan Ilic. Killingfusion: Non-rigid 3d reconstruction without correspondences. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1386–1395, 2017. 76

[SDO+16] Azad Shademan, Ryan S Decker, Justin D Opfermann, Simon Leonard, Axel Krieger, and Peter CW Kim. Supervised autonomous robotic soft tissue surgery. *Science translational medicine*, 8(337):337ra64–337ra64, 2016. 22

[SFN+20] Delia SepúLveda, Roemi Fernández, Eduardo Navas, Manuel Armada, and Pablo González-De-Santos. Robotic aubergine harvesting using dual-arm manipulation. *IEEE Access*, 8:121889–121904, 2020. 22

[SFP+19] Changyeob Shin, Peter Walker Ferguson, Sahba Aghajani Pedram, Ji Ma, Erik P Dutson, and Jacob Rosen. Autonomous tissue manipulation via surgical robot using learning based model predictive control. In *2019 International conference on robotics and automation (ICRA)*, pages 3875–3881. IEEE, 2019. 36, 46

[SK08] Yonggang Shi and William Clem Karl. A real-time algorithm for the approximation of level-set-based curve evolution. *IEEE transactions on image processing*, 17(5):645–656, 2008. 30

[SKM19] Agniva Sengupta, Alexandre Krupa, and Eric Marchand. Tracking of non-rigid objects using rgb-d camera. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 3310–3317, 2019. 33

[SLHA13] John Schulman, Alex Lee, Jonathan Ho, and Pieter Abbeel. Tracking deformable objects with point clouds. In *2013 IEEE International Conference on Robotics and Automation*, pages 1130–1137, 2013. 34

[SMC+18] Jose Sanchez, Carlos M Mateo, Juan Antonio Corrales, Belhassen-Chedli Bouzgarrou, and Youcef Mezouar. Online shape estimation based on tactile sensing and deformation modeling for robot manipulation. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 504–511. IEEE, 2018. 27

[SQL22] Haiqing Si, Jingxuan Qiu, and Yao Li. A review of point cloud registration algorithms for laser scanners: applications in large-scale aircraft measurement. *Applied Sciences*, 12(20):10247, 2022. 80

[SSP07] Robert W Sumner, Johannes Schmid, and Mark Pauly. Embedded deformation for shape manipulation. In *ACM siggraph 2007 papers*, pages 80–es. 2007. 35

[SWD+21] Yu She, Shaoxiong Wang, Siyuan Dong, Neha Sunil, Alberto Rodriguez, and Edward Adelson. Cable manipulation with a tactile-reactive gripper. *The International Journal of Robotics Research*, 40(12-14):1385–1401, 2021. 13, 21

[SZY+18] Buyun Sheng, Feiyu Zhao, Xiyan Yin, Chenglei Zhang, Hui Wang, and Peide Huang. A lightweight surface reconstruction method for online 3d scanning point cloud data oriented toward 3d printing. *Mathematical Problems in Engineering*, 2018, 2018. 76

[Tar19] Sonny Tarbouriech. *Dual-Arm control strategy in industrial environments*. PhD thesis, Université Montpellier, 2019. 53, 59

[TCUM19] Yoshihisa Tsurumine, Yunduan Cui, Eiji Uchibe, and Takamitsu Matsubara. Deep reinforcement learning with smooth policy update: Application to robotic cloth manipulation. *Robotics and Autonomous Systems*, 112:72–83, 2019. 37, 46

[TWT18] Te Tang, Changhao Wang, and Masayoshi Tomizuka. A framework for manipulating deformable linear objects by coherent point drift. *IEEE Robotics and Automation Letters*, 3(4):3426–3433, 2018. 36, 46

[TZL+12] Jing Tong, Jin Zhou, Ligang Liu, Zhigeng Pan, and Hao Yan. Scanning 3d full human bodies using kinects. *IEEE Transactions on Visualization and Computer Graphics*, 18(4):643–650, 2012. 76

[UWH+20] Naveen Kumar Uppalapati, Benjamin Walt, Aaron J Havens, Armeen Mahdian, Girish Chowdhary, and Girish Krishnan. A berry picking robot with a hybrid soft-rigid arm: Design and task space control. In *Robotics: Science and Systems*, page 95, 2020. 22

[VEBR18] Rok Vrabic, John Ahmet Erkoyuncu, Peter Butala, and Rajkumar Roy. Digital twins: Understanding the added value of integrated models for through-life engineering services. *Procedia Manufacturing*, 16:139–146, 2018. Proceedings of the 7th International Conference on Through-life Engineering Services. 68

[VSM+21] Rok Vrabic, Gasper Skulj, Andreja Malus, Dominik Kozjek, Luka Selak, Drago Bracun, and Primoz Podrzaj. An architecture for sim-to-real and real-to-sim experimentation in robotic systems. *Procedia CIRP*, 104:336–341, 2021. 54th CIRP CMS 2021 - Towards Digitalized Manufacturing 4.0. 68

[WKL+19] Angelina Wang, Thanard Kurutach, Kara Liu, Pieter Abbeel, and Aviv Tamar. Learning robotic manipulation through visual planning and acting. *arXiv preprint arXiv:1905.04411*, 2019. 37, 46

[WMC+20] Sebastian Weiss, Robert Maier, Daniel Cremers, Rudiger Westermann, and Nils Thuerey. Correspondence-free material reconstruction using sparse surface constraints. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4686–4695, 2020. 38

[WWY+15] Bin Wang, Longhua Wu, KangKang Yin, Uri M Ascher, Libin Liu, and Hui Huang. Deformation capture and modeling of soft objects. *ACM Trans. Graph.*, 34(4):94–1, 2015. 34

[WY23] Taohan Wang and Yuji Yamakawa. Edge-supervised linear object skeletonization for high-speed camera. *Sensors*, 23(12), 2023. 77

[WZR+18] Jimmy Wu, Bolei Zhou, Rebecca Russell, Vincent Kee, Syler Wagner, Mitchell Hebert, Antonio Torralba, and David MS Johnson. Real-time object pose estimation with pose interpreter networks. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6798–6805. IEEE, 2018. 29

[WZX17] Kangkan Wang, Guofeng Zhang, and Shihong Xia. Templateless non-rigid reconstruction and motion tracking with a single rgb-d camera. *IEEE Transactions on Image Processing*, 26(12):5966–5979, 2017. 31

[XCB+22] Zhenjia Xu, Cheng Chi, Benjamin Burchfiel, Eric Cousineau, Siyuan Feng, and Shuran Song. Dextairity: Deformable manipulation can be a breeze. In *Proceedings of Robotics: Science and Systems (RSS)*, 2022. 13, 21

[XSNF17] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. *arXiv preprint arXiv:1711.00199*, 2017. 29

[YGL+19] Jingyu Yang, Daoliang Guo, Kun Li, Zhenchao Wu, and Yu-Kun Lai. Global 3d non-rigid registration of deformable objects using a single rgb-d camera. *IEEE Transactions on Image Processing*, 28(10):4746–4761, 2019. 31

[YMH17] S Robin Yosafat, Carmadi Machbub, and Egi Muhammad Idris Hidayat. Design and implementation of pan-tilt control for face tracking. *2017 7th IEEE International Conference on System Engineering and Technology (ICSET)*, pages 217–222, 2017. 41

[YVK21] Hang Yin, Anastasia Varava, and Danica Kragic. Modeling, learning, perception, and control methods for deformable object manipulation. *Science Robotics*, 6(54), 2021. 27

[ZFH16] Antonio Zea, Florian Faion, and Uwe D. Hanebeck. Tracking elongated extended objects using splines. In *2016 19th International Conference on Information Fusion (FUSION)*, pages 612–619, 2016. 77

[ZNAPC20] Jihong Zhu, David Navarro-Alarcon, Robin Passama, and Andrea Cherubini. Vision-based manipulation of deformable and rigid objects using subspace projections of 2d contours. 2020. 44, 46, 47, 50, 57, 105, 107, 126

[ZNF+18] Jihong Zhu, Benjamin Navarro, Philippe Fraisse, André Crosnier, and Andrea Cherubini. Dual-arm robotic manipulation of flexible cables. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 479–484, 2018. 42, 46

[ZNI+14] Michael Zollhöfer, Matthias Nießner, Shahram Izadi, Christoph Rehmann, Christopher Zach, Matthew Fisher, Chenglei Wu, Andrew Fitzgibbon, Charles Loop, Christian Theobalt, et al. Real-time non-rigid reconstruction using an rgb-d camera. *ACM Transactions on Graphics (ToG)*, 33(4):1–12, 2014. 33

[ZPC21] Simon Zimmermann, Roi Poranne, and Stelian Coros. Dynamic manipulation of deformable objects with implicit integration. *IEEE Robotics and Automation Letters*, 6(2):4209–4216, 2021. 13, 39, 40, 46, 104

[ZSG+18] M. Zollhöfer, P. Stotko, A. Görlitz, C. Theobalt, M. Nießner, R. Klein, and A. Kolb. State of the Art on 3D Reconstruction with RGB-D Cameras. *Computer Graphics Forum (Eurographics State of the Art Reports 2018)*, 37(2), 2018. 76

[ZWWL19] Fangxun Zhong, Yaqing Wang, Zerui Wang, and Yun-Hui Liu. Dual-arm robotic needle insertion with active tissue deformation for autonomous suturing. *IEEE Robotics and Automation Letters*, 4(3):2669–2676, 2019. 22

[Žl08] Leon Žlajpah. Simulation in robotics. *Mathematics and Computers in Simulation*, 79(4):879–897, 2008. 5th Vienna International Conference on Mathematical Modelling/Workshop on Scientific Computing in Electronic Engineering of the 2006 International Conference on Computational Science/Structural Dynamical Systems: Computational Aspects. 104