Shape tracking and servoing of deformable objects

Sayed Mohammadreza Shetab Bushehri

UNIVERSITÉ CLERMONT AUVERGNE
ÉCOLE DOCTORALE SCIENCES POUR L'INGÉNIEUR
CLERMONT AUVERGNE INP, INSTITUT PASCAL
DOCTORAL THESIS

Shape Tracking and Servoing of Deformable Objects

Author:
Mohammadreza
Shetab-Bushehri

pour obtenir le grade de
Docteur d'Université

Sera soutenue le 11/07/2023 devant le jury composé de :

Gonzalo López-Nicolás	Rapporteur	Professor, Universidad de Zaragoza
Helder Araujo	Rapporteur	Professor, University of Coimbra
Brahim Tamadazte	Rapporteur	Chargé de recherche, Institute of Intelligent Systems and Robotics (ISIR)
Thierry Chateau	Président du jury	Professeur des universités, Université Clermont Auvergne
Claire Dune	Examinateur	Maître de conférences, Université de Toulon
Miguel Aranda	Examinateur	Chercheur postdoctoral, Universidad de Zaragoza
Youcef Mezouar	Directeur	Professeur des universités, Clermont Auvergne INP-SIGMA
Erol Özgür	Co-encadrant	Maître de conférences, Clermont Auvergne INP-SIGMA

Declaration of Authorship

I, Mohammadreza Shetab-Bushehri, declare that this thesis titled, Shape Tracking and Servoing of Deformable Objects and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this university or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

Abstract

In the past few decades, robotics research has gained momentum due to robots' ability to perform dangerous or monotonous tasks that are difficult for humans. The majority of the research in this field has focused on manipulating rigid objects that maintain their shape during the manipulation process. However, there are numerous objects that undergo significant deformation during manipulation, known as deformable objects (DOs). Our surroundings contain various types of DOs, including human tissue, clothing, cables, metal sheets, and vegetables. This abundance of DOs presents enormous potential for developing new robotic applications that could greatly improve our daily lives. However, the manipulation of DOs presents a considerable number of challenges due to their highly complex and dynamic nature. To tackle these challenges, two general directions of research have been introduced and developed: shape tracking and shape servoing of DOs. Shape tracking involves inferring the current shape of DOs at each instant. Shape servoing is concerned with controlling DOs' shape to a desired shape or trajectory.

Over the past few years, researchers have suggested several approaches for tracking and servoing DOs. However, these methods have significant drawbacks, such as being limited to a single form of the object, requiring precise mechanical parameters of the object, being imprecise due to the use of many simplifications, computationally intensive, and not real-time. Additionally, these methods can be susceptible to sensor noise and occlusions.

This thesis introduces several new approaches to address the existing challenges of tracking and servoing DOs. Our proposed approaches cover a wide range of object forms (linear, thin-shell, volumetric) and geometries, and work with both 2D or 3D sensors. Firstly, we present two approaches for handling shape tracking and servoing of thin-shell objects. The proposed shape tracking method is based on monocular 2D vision and is wide-baseline, real-time (up to 30 fps), robust against occlusions and against video cuts. The proposed shape servoing approach is based on a geometrical model called As-Rigid-As-Possible (ARAP), and does not require knowledge of the object's mechanical characteristics. It is fast and avoids using a Jacobian from data collected while the robots are in motion that is susceptible to noise. Next, we introduce a novel unified shape tracking-servoing approach based on 3D vision. In this approach, we form a lattice around the object and bind the lattice to the object using geometrical constraints. We track and servo the lattice instead of the object. This decouples the runtime complexity from the object's geometric complexity, which makes the whole tracking-servoing process much faster without requiring any specialized hardware. Using a lattice enables our approach to generalize to objects of any geometry and form, including linear, thin-shell, and volumetric objects. The proposed approach incorporates ARAP, and thus does not require knowledge of the object's mechanical properties. We validate the efficiency of the proposed approaches through various experiments with various objects.

Keywords. Shape tracking, shape servoing, elastic deformable objects, ARAP model.

Résumé

Ces dernières décennies, la recherche en robotique a gagné en importance grâce à la capacité des robots à effectuer des tâches dangereuses ou monotones pour les humains. La plupart des recherches dans ce domaine se sont concentrées sur la manipulation d'objets rigides qui conservent leur forme pendant le processus de manipulation. Cependant, il existe de nombreux objets qui subissent une déformation significative lors de leur manipulation. Ces objets sont appelés objets déformables (DO). Notre environnement contient divers types de DO, tels que les organes humains, les vêtements, les câbles, les feuilles métalliques et les légumes. Cette abondance de DO présente un énorme potentiel pour développer de nouvelles applications robotiques qui pourraient grandement améliorer notre vie quotidienne. Cependant, la manipulation de DO présente un nombre considérable de défis en raison de leur nature hautement complexe et dynamique. Le suivi de la forme implique de comprendre la forme courante des DOs à chaque instant. La commande de la forme concerne l'asservissement de la forme des DO vers une forme ou une trajectoire souhaitée.

Au cours des dernières années, les chercheurs ont proposé plusieurs approches pour relever les défis de suivi et de commande de la forme des DO. Cependant, ces méthodes présentent des inconvénients importants, tels que la limitation à une seule forme de l'objet, la nécessité de connaître les paramètres mécaniques précis de l'objet, l'imprécision due à l'utilisation de nombreuses simplifications, le coût calculatoire et la lenteur qui ne permet pas un traitement en temps réel. De plus, ces méthodes sont sensibles aux bruits des capteurs et aux occultation.

Dans cette thèse, nous présentons plusieurs nouvelles approches pour relever les défis liés au contrôle et au suivi des objets déformables (DOs). Nos approches proposées couvrent une large gamme de formes d'objets (linéaires, à coques minces, volumétriques) et utilisent à la fois des données issues de capteurs 2D et 3D en entrée. Tout d'abord, nous présentons deux solutions pour la gestion du suivi et du contrôle de la forme des objets à coque mince. Le suivi de forme est basé sur une vision monoculaire à large entraxe, en temps réel (jusqu'à 30 ips), laquelle est robuste contre les occultation et les coupures de vidéo, et laquelle est facile à utiliser. Notre approche de contrôle de la forme est basée sur un modèle géométrique appelé « As-rigid-as-possible » (ARAP), qui ne nécessite pas de connaissance préalable des caractéristiques mécaniques de l'objet. Elle est rapide et évite d'utiliser une jacobienne qui est vulnérable aux bruits provenant des données collectées sur une fenêtre temporelle durant laquelle les robots sont en mouvement. Nous introduisons ensuite une approche de suivi et de contrôle de la forme basée sur une vision 3D. Dans cette approche, nous formons une grille autour de l'objet et nous les contraignions ensemble à l'aide de contraintes géométriques. Nous suivons et contrôlons la grille au lieu de l'objet. Cela découple la complexité d'exécution de la complexité géométrique de l'objet, ce qui rend l'ensemble du processus de suivi et de contrôle beaucoup plus rapide sans nécessiter l'utilisation de matériel spécialisé. L'utilisation d'une grille rend également notre approche capable de généralisation à des objets de toute forme et géométrie, y compris les objets linéaires, à coque mince et volumétriques. Cette approche intègre ARAP, et ne nécessite donc pas de connaissance des propriétés mécaniques de l'objet. Nous validons l'efficacité de nos méthodes par le biais de diverses expériences en simulé et en réel avec différents objets.

Mots clés. Suivi de forme, contrôle de forme, objets déformables élastiques, modèle ARAP.

Dedicated to those who have fallen, but never lost determination

Acknowledgements

I would like to express my deepest gratitude and appreciation to everyone who has played a role in the completion of my thesis.

First and foremost, I am immensely grateful to my supervisors, Youcef Mezouar and Erol Özgür, for their guidance, expertise, and unwavering support throughout this research journey. Their valuable insights, constructive feedback, and constant encouragement have been instrumental in shaping this work.

Secondly, I want to express my gratitude to the thesis reviewers, Gonzalo López-Nicolás, Helder Araujo, and Brahim Tamadazte, for agreeing to review this dissertation. I also sincerely thank Thierry Chateau, Miguel Aranda, Claire Dune, Gonzalo López-Nicolás, Helder Araujo, and Brahim Tamadazte for accepting to be members of the thesis defense jury.

I also want to extend a special and heartfelt thank you to Miguel Aranda. Beyond being a dear friend, Miguel has been an invaluable source of knowledge and support throughout this entire process. I am truly grateful for his guidance, patience, and the countless hours he dedicated to helping me overcome challenges and refine my work.

I am also grateful to Yiannis Karayiannidis for inviting me for a three-month research mobility at Chalmers University of Technology in Sweden. I want to thank everyone there for their warm welcome and assistance, particularly Rita Laezza, with whom I made significant progress on the project and gained valuable knowledge.

Lastly, I would like to extend my heartfelt thanks to my family, friends, and colleagues in the lab for their love, understanding, and constant encouragement.

*****

This work was supported by project SOFTMANBOT, which received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 869855.

My mobility to Chalmers University of Technology in Sweden has received funding from ERASMUS + Stages and the project I-SITE Clermont of CAP 20-25.

Contents

1 Introduction **1**
- 1.1 Shape tracking . . . 3
- 1.2 Shape servoing . . . 7
- 1.3 Contributions . . . 8
- 1.4 Publications . . . 9
- 1.5 Thesis outline . . . 11

2 State of the art **13**
- 2.1 Introduction . . . 13
- 2.2 Shape modeling . . . 13
- 2.3 Shape tracking . . . 18
- 2.4 Shape servoing . . . 27

3 `ROBUSfT`: a Real-Time Monocular Shape-from-Template pipeline **33**
- 3.1 Introduction . . . 33
- 3.2 ROBUSfT . . . 34
- 3.3 Fake Realistic Experiment (`FREX`) . . . 37
- 3.4 `myNeighbor` . . . 39
- 3.5 Experimental Results . . . 49
- 3.6 Conclusion and future work . . . 56

4 As-Rigid-As-Possible Shape Servoing **57**
- 4.1 Introduction . . . 57
- 4.2 Problem formulation . . . 57
- 4.3 Deformation Jacobian computation . . . 59
- 4.4 Control law . . . 62
- 4.5 Shape tracking . . . 62
- 4.6 Experiments . . . 63
- 4.7 Conclusion and future work . . . 67

5 Lattice-based shape tracking and servoing **69**
- 5.1 Introduction . . . 69
- 5.2 The approach description and notation . . . 70
- 5.3 Tracking pipeline . . . 72
- 5.4 Servoing pipeline . . . 76
- 5.5 Experiments . . . 81
- 5.6 Conclusion and future work . . . 91

6 Conclusion and perspectives **93**
- 6.1 Shape tracking . . . 93
- 6.2 Shape servoing . . . 94

Bibliography . . . 95

List of Figures

1.1 DOs manipulation examples . . . 2
1.2 A visual representation of SfT [BGC+15]. . . . 5

2.1 A flexible rod modeled using FEM. The rod is discretized into an irregular tetrahedral mesh. . . . 14
2.2 A mass-spring system, (a) the system of particles and their interconnections, (b) The usage of mass-spring systems in cloth simulation [LBOK13]. 15
2.3 Several results of PBD simulations presented in [MHHR07], (a) a piece of cloth is torn apart by an attached cube, (b) demonstration of stable self-collision and response, (c) cloth stripes are attached via one-way interaction to static rigid bodies at the top and via two-way constraints to rigid bodies at the bottom. . . . 16
2.4 An example of modeling deformation using meshless shape matching. A beam is deformed by drawing it from a point at its top. The deformation is simulated while one, two, and five clusters of particles are considered. . 16
2.5 Modeling an Armadillo using ARAP presented in [SA07]. . . . 17
2.6 The reconstruction result of the method presented in [FBA18] on a deformed Spiderman poster. (a) The image of the deformed Spiderman poster. (b) The visualization of the error. (c) Estimation. (d) The groundtruth. . . . 19
2.7 The reconstruction result of Particle-SfT presented in [ÖB17]. For all the cases, the 3D reconstructions (black circles) of Particle-SfT are shown with the ground-truth shapes (green dots). The reconstruction error is written below each case. (a) Cushion dataset with non-planar template. (b-d) Real elastic datasets. For each case, the left image shows the template image and the right image shows the input image. (b) A foot size 41 wearing a sock for a foot size 37. (c) A pocket filled by a bottle of water and a magazine. (d) A piece of fabric that has large lateral stretching and a mild central pushing deformation. . . . 20
2.8 Shape tracking results presented in [ACRM+20]. Top: three images of a deforming paper sheet with a checkerboard pattern. Three tracked points are marked in blue in each image. Bottom: inferred 3D shapes of the images shown on top using all the 80 points (green) and using only the three tracked points (blue). . . . 21
2.9 Shape tracking results presented in [CB15] for a deforming juice bottle and a rugby ball. . . . 22
2.10 Shape reconstruction results of three different objects (a paper, a doll, and a shoe) presented in [FJPCP+22]. The network was trained specifically for these objects. . . . 23
2.11 Shape tracking results presented in [TT22] for (a) a rope and (b) a t-shirt. . 24
2.12 Shape tracking results presented in [PLS15] for (a) a pizza-like and (b) a cylindrical bar object. . . . 24

2.13 Shape servoing results presented in [FMC+18]. A comparison between the desired deformation planned in the Sofa environment scene and the deformation controlled on the real object is shown. 28
2.14 Shape servoing results presented in [KFB+21]. Two points on a linear object are derived toward their desired positions. 29
2.15 Shape servoing results presented in [ACRM+20]. Left to right: the initial image, an intermediate image, the final image, and RMS of the estimated shape servoing error. 29
2.16 (a) Shape servoing using contours presented in [NAL18], (b)-(c) dual-arm manipulation of a thick and a thin rope presented in [ZNF+18]. 30
2.17 Dual-arm manipulation of a rope presented in [LKM20]. (a) A failed case due to the large deformation of the desired shape, (b) A successful case by defining multiple intermediary desired shapes. 31

3.1 Overview of ROBUSfT. 35
3.2 Implementation of ROBUSfT on the CPU and GPU. A pure CPU implementation is also provided. 38
3.3 Flowchart of FREX. 39
3.4 Flowchart of myNeighbor. 40
3.5 Two sample results of the steps for synthetic data experiments. The first row is an experiment with 100 matches and a mismatch percentage of 30%. The second row is an experiment with 1000 matches and a mismatch percentage of 30%. The first and second columns represent $\mathcal{I}_F$ and $\mathcal{I}$ with correct matches in green and mismatches in red. The third column is the result of *Step I*. The wrongly chosen mismatches are shown in red. The fourth column is the result of *Step II*. The mismatches along with a small percentage of correct matches are removed. The fifth column is the separation of the estimated correct matches and the estimated mismatches from *Step III*. The transferred meshes $\widehat{M}_1$, $\widehat{M}_2$, and $\widehat{M}_3$ are shown in orange, yellow, and cyan for the three steps. 41
3.6 Histogram of MF values for three sample synthetic data experiments with 1000 matches and 30%, 60% and 90% of correct matches. 44
3.7 Results of applying the first two steps of the algorithm myNeighbor in synthetic data experiments in three different scenarios; Dense (1000 matches), Moderate (200 matches), and Sparse (50 matches). Each curve is the average result of 1000 trials. The first row gives n_s and AoS from *Step I* for two different values of MF_{th}. The second row gives the results of *Step II* in comparison to the results of *Step I* with $MF_{th} = \text{mean}(MF)$. 46
3.8 ROC curves resulting from the algorithm myNeighbor in synthetic data experiments in three scenarios; Dense (1000 matches), Moderate (200 matches), and Sparse (50 matches). Each point is the average result of 1000 trials calculated with a specific value of $d_{3_{th}}$. 46
3.9 Performance evaluation of our mismatch removal method myNeighbor in comparison to the state-of-the-art methods using the FREX protocol. The first row shows the Aruco template and three selected images (14, 47, 60) of the deformation of the printed Aruco template. The following rows show five datasets of generated scenes with the texturemap in the first column, three generated images corresponding to the first row in the next columns, and the ROC curves of the mismatch removal algorithms in the last column. For each of the images $\widehat{M}_3$ from myNeighbor is overlaid. . . . 47

3.10 Applying `myNeighbor` on four real cases: a cushion, a Spiderman poster, a shoe sole, and an elastic shirt. The first column shows the texturemaps. The second column shows *Step I*. All the matches are shown in this column while the selected matches in *Step I* are shown in green. These selected matches are transferred to column three that shows *Step II*. In this column, those matches which are chosen as possible mismatches are shown in red. The last column is the distinction between the estimated correct matches (in green) and the estimated mismatches (in red) in *Step III*. The meshes $\widehat{M}_1$, $\widehat{M}_2$, and $\widehat{M}_3$ are overlaid to illustrate the computed warps. 49
3.11 Comparing the accuracy of the 3D shape inference methods with Particle-SfT with three datasets obtained by `FREX`. The 3D shape inference methods are Brunet et al. [BBH14], Chhatkuli et al. [CPB14], Bartoli et al. [BGC+15], Ostlund et al. [ÖVNF12], and Salzmann et al. [SF10]. 50
3.12 Comparison between the results of applying `ROBUSfT` and the integrated methods, i.e., Famouri et al. [FBA18] and Ngo et al [NÖF15] on the public dataset provided in [VSSF12]. (a) Mean absolute 3D error between the inferred shape and the ground truth. (b) Execution time in milliseconds. . 51
3.13 Comparison between the depth map resulted from applying `ROBUSfT` and the deep object-generic monocular reconstruction methods, i.e., DenseDepth [AW18] and BTS [LHKS19], on one frame of the dataset provided in [VSSF12]. To make a fair comparison, the depth RMSE is calculated using the pixels from the segmentation mask. This segmentation mask is estimated in the registration step of `ROBUSfT`. 52
3.14 Evaluating `ROBUSfT` in three real data experiments; a Spiderman poster, a chopping mat, and a t-shirt. The texturemaps of the templates are shown in the first column. For each case, four images are shown. Below each frame, the reconstructed 3D shape of the deforming object with the estimated 3D coordinates of the estimated correct matches (red particles) as well as their ground truth (green particles) are shown. The 2D projections of the 3D inferred shapes are also overlaid on the image. For each image, the median Euclidean distance between the estimated 3D coordinates of the estimated correct matches and their ground truth is given below the reconstructed shape. 54
3.15 Evaluating `ROBUSfT` in a real data experiment with two robotic arms; soft constraints are applied to bind the constrained mesh points to the grippers. Each row shows three images: the original camera view, the projection of the 3D reconstructed mesh on the camera view, and the 3D reconstructed mesh with the robots in the RViz environment. 55

4.1 Flowchart of the proposed shape servoing scheme. Online components in solid blue lines, offline components in dashed orange lines. 58
4.2 Definition of a rigid set coupled with the gripper. The four mesh nodes forming the rigid set are encircled in yellow. 59
4.3 A summary of estimating procedure of $\mathbf{J}_A(\mathbf{s})$. 60
4.4 View of our experimental setup during testing of bi-arm visual shape servoing of a rubber tire tread with an overhead monocular camera. 63
4.5 Results for the seven tasks. For each we show, from left to right: image of the undeformed object; initial, desired and final shape (camera images on top, images from an external camera on bottom); initial, intermediate and final shapes represented in RViz; and evolution of error (RMS of $\mathbf{e}$) over time. 66

4.6 Comparison between the poses of the grippers in the desired (green lines) and the final (red lines) shapes in Tasks 4 (top) and 5 (bottom). In Task 4 the grippers were displaced manually. In Task 5, the right gripper rotated due to slippage during the task. 67

5.1 Flowchart of the proposed shape tracking-servoing approach. In each frame, the tracking pipeline takes the captured point cloud in the current frame and $\mathbf{s}$ (and thus $\mathbf{p}$) from the previous frame as the input and estimates $\mathbf{s}^c$ (and thus $\mathbf{p}^c$) in the current frame. $\mathbf{s}^c$ and $\mathbf{R}_l^o$ are sent to the servoing pipeline where an analytical expression for a deformation Jacobian is obtained. We use this Jacobian in a control law to send proper commands to the robots to guide $\mathbf{s}^c$ toward $\mathbf{s}^*$ and consequently $\mathbf{p}^c$ toward $\mathbf{p}^*$. 71
5.2 Lattice $\mathbf{s}$ encapsulating the point cloud $\mathbf{p}$ of a shoe sole at its rest shape. We also generate a tetrahedral mesh over the lattice nodes. 72
5.3 The results of different steps of the tracking pipeline in one frame of tracking a shoe sole. $\mathbf{p}^c$ and $\mathbf{s}^c$ are shown as green and cyan dots. The dark blue lines represent the tetrahedral mesh over $\mathbf{s}^c$. 73
5.4 Aligning the object with the reference shape at the beginning of tracking. To facilitate the process of alignment, we visualize the reference shape in 3D space and its projection in the 2D image. Top row: partially consistent alignment before starting tracking, bottom row: inferring the correct shape after several frames after triggering the tracking pipeline. 74
5.5 A graphical representation of the ICP-like algorithm in [PLS15] for two points in $\mathbf{p}_v^c$ (shown in blue) and several points in $\mathbf{d}_f$ (shown in green) surrounding them. Each red point is the corresponding point of a point in $\mathbf{p}_v^c$ in 3D space. 74
5.6 Our experimental setup with two Franka robots manipulating a foam octagonal cylinder. An overhead 3D camera provides the input for our approach. 82
5.7 Tasks with a cable and in-plane deformations. The plotted points represent the mean coordinates of the lattice nodes belonging to the cross-sections of the lattice along the cable length. Green nodes: current lattice shape, red points: desired lattice shape. Top row: full shape servoing, bottom row: partial shape servoing. 84
5.8 Task with a cable and out-of-plane deformations. Green: cable's current shape, red: cable's desired shape. 84
5.9 Tasks with a thin-shell object: an A4 paper. Top row: task with full shape servoing toward a desired shape with large deformation. Second row: task with partial shape servoing. Two separated regions (one-fifth of the paper) are servoed. Third row: task with partial shape servoing. One forth of the A4 paper is servoed. Only translational velocities of the robots are updated. Fourth row: task with partial shape servoing with a slightly thicker A4 paper. Two-fourths middle part of the object is servoed. The grippers are displaced while setting the initial shape. 85
5.10 Tasks with a thin-shell object: a convoluted foam. Top row: task with full shape servoing. Bottom row: task with partial shape servoing. Two small separated regions of the foam are servoed. 86
5.11 Tasks with a volumetric object; a bulky shoe sole. Top row: task with large deformation. Bottom row: task with a desired shape with a considerably different view of the object with respect to the initial shape. In this task, the object is rotated and flipped by the shape servoing approach. 86

5.12 Tasks with a volumetric object; a foam octagonal cylinder. Top row: task with merely twist. Bottom row: task with twist and bending deformation at the same time. 87
5.13 The simulated experiment where our servoing approach drives the lattice (in blue) towards a desired shape (in red). The desired shape is an unfeasible shape with different dimensions. 88
5.14 The simulated experiment where the shape servoing approach drives the lattice (in blue) to a desired shape (in red) with the same dimensions; once with rotation and once without rotation. 89
5.15 Uniform lattice vs non-uniform lattice over the geometry of a dinosaur [ZSGS12]. 89
5.16 Servoing a concave lattice toward the desired shape. 90
5.17 Comparison between ARAP-SS and our proposed shape servoing pipeline. Left: initial and desired shapes which are identical for both approaches. Right: shape servoing RMS error over time belonging to both approaches. 90
5.18 Tasks for driving a thick A4 paper through its singular shape, i.e., a flat shape. First row: successful task, deforming the paper from an upward-curved shape to a downward-curved shape. Second row: unsuccessful task, deforming the paper from a downward-curved shape to an upward-curved shape (deforming against gravity). Two last rows: successful task, defining two intermediary desired shapes for performing the failed task of the second row.) 91

6.1 Dual-arm ABB YuMi robot manipulates a rope on a table. A fixed Intel RealSense camera provides a top-view of the workspace. The field of view of the camera is shown in the top right corner, with the overlay of the current shape tracking in green and the desired shape in red. 95
6.2 Comparison between final and desired shapes. Note how the offline RL policy succeeded in inverting the curvature, while the shape servoing appraoch [Ber13] remained at a local minimum. 95

List of Tables

2.1 A comparison between the state-of-the-art monocular SfT methods. 25

3.1 Comparison of the average run-time of the mismatch removal algorithms for processing all the images of all the datasets. 49
3.2 Comparison between the average 3D error resulted from applying ROBUSfT and the DNN-based SfT methods, i.e., HDM-Net [GSVS18], IsMo-GAN [SGTS19], DeepSfT [FJPCP+22], and RRNet [FJPCP+21] on 50 frames of the public dataset provided in [VSSF12]. 53

4.1 Parameters of the shape servoing tasks. 64
4.2 Performance evaluation of shape change prediction. 66

5.1 Parameters of the shape servoing tasks. 83

Glossary

ARAP As-Rigid-As-Possible. v, vii, xv, xix, 17, 18, 22, 27, 28, 57–62, 64, 65, 67, 69, 70, 75–81, 88, 90, 93, 94

DO deformable object. v, vii, xv, 1–9, 11, 13, 14, 17, 18, 22, 23, 25–27, 30, 31, 33, 34, 57, 67, 69, 70, 72, 81, 88, 91–94

DOF Degrees of Freedom. 57, 58, 60–63, 70, 76, 80, 81

EDO elastic deformable object. 3

FEM Finite Element Method. xv, 13–15, 18, 24, 25, 27, 79

ICP Iterative Closest Point. 23

NRSfM Non-Rigid-Shape-from-Motion. 5

PBD Position-based Dynamics. xv, 15–18, 27, 41, 93

RL Reinforcement Learning. xix, 94, 95

SfM Shape-from-Motion. 5

SfT Shape-from-Template. xv, xvii, xxi, 5, 6, 8, 17, 18, 20–22, 25, 33, 36, 37, 50–53, 55, 57, 93

SVD Singular Value Decomposition. 61, 77, 78

Chapter 1

Introduction

In recent decades, researchers have taken an interest in robotic manipulation due to the ability of robots to perform tasks that are dangerous or tedious for humans. Most research in this area has focused on manipulating rigid objects, where the object's shape remains constant during manipulation. There are several well-established methods for addressing the challenges of manipulating rigid objects. These methods have enabled robots to interact with objects in a stable and predictable manner, allowing for tasks such as object pick and place, assembly, and tool use. In contrast to rigid objects, there are many objects that undergo considerable deformation during manipulation. These objects are referred to as deformable objects (DOs). Our environment is abundant with different types of DOs, including clothes, human tissue, vegetables, metal sheets, cables, etc. This abundance of DOs, coupled with the dearth of well-established robotic manipulation techniques for DOs, creates immense potential for the development of novel robotic applications that can significantly improve our daily lives. Figure 1.1 showcases the primary areas where these applications can be utilized to offer effective substitutes for manual work or to assist human operators. In the following, we will briefly describe each of these areas.

- *Agriculture.* In agriculture, crop harvesting requires developing robots that can handle delicate and soft objects, such as fruits and vegetables, without causing damage.
- *Surgery.* In surgery, robots can help to stabilize and expedite movements while manipulating organs, blood vessels, and tissues and consequently reduce the risk of complications, leading to better patient outcomes.
- *Production.* Robots can enable accurate detection and control of flexible materials, such as leather and fabric, to ensure accurate positioning and holding during production processes such as assembly and sewing.
- *Packaging.* Robots can pack and transport goods such as food using materials such as bags and pouches.
- *Housework.* Robots can help the elderly and disabled individuals with housework tasks including cleaning, cooking, etc.
- *Healthcare.* Robots can be used in rehabilitation therapies to help patients regain control of their limbs.

Despite all these applications, robotic manipulation of DOs presents challenges. Unlike rigid objects, the shape of DOs can change during manipulation, making DOs more difficult to track. In addition, DOs often exhibit intricate and unpredictable behaviors, making it difficult to anticipate their response to external forces and thus to control them. For instance, when a robot tries to grasp a piece of cloth, the material can easily slip or

(a) Agriculture (b) Surgery

(c) Production (d) Packaging industry

(e) Housework (f) Healthcare

FIGURE 1.1: DOs manipulation examples

deform, which makes it challenging to maintain a stable grip or achieve a specific deformation. In comparison to the well-established methods for robotic manipulation of rigid objects, the study of robotic manipulation of DOs is relatively recent. The complexity of the flexible and ever-changing nature of DOs necessitates the creation of unique and inventive solutions for their effective manipulation.

The robotic manipulation of DOs is a multifaceted research area, comprising three key domains: shape modeling, shape tracking, and shape control. In the following, we describe each area.

- **Shape modeling.** DOs modeling is an active research area in computer graphics, computer vision, and robotics. A primary difficulty in this area is the trade-off between modeling complexity and the computational resources required for real-time simulation. A realistic DO model based on physical parameters can be computationally expensive and unsuitable for real-time simulation [MM07]. Consequently,

researchers in this field often develop simpler models with properties that can approximate the behavior of DOs. Shape modeling has been used to provide a primary constraint in tracking and control of DOs.

- **Shape tracking.** Tracking DOs involves determining the shape of DOs at each instant. Accurately tracking a DO can be challenging due to several factors, including the high number of degrees of freedom, occlusions, local deformations, and variations in deformation types [BBH14]. Shape tracking is typically performed using various sensors, such as cameras, force sensors, and strain sensors among which 2D and 3D cameras are the most widely-used.

- **Shape control.** Controlling the shape of a DO refers to the process of actively manipulating the shape of the object toward a specific desired shape using robotic systems. This process is commonly known as shape servoing in the literature [NAL18]. A precise and robust shape tracking method is crucial for successful shape servoing, as it enables robots to accurately comprehend the object's shape at each instant and manipulate it effectively. Controlling the shape of DOs is a complex and challenging task due to their unpredictable and nonlinear deformation behavior.

In this thesis, we present several approaches for shape tracking and servoing of DOs. The proposed approaches tackle the current problems in these areas.

Assumptions. The following assumptions are made throughout this thesis:

- **Assumption on the deformation type.** We focus on elastic deformable objects (EDOs). An EDO undergoes changes in its shape due to applied external forces while having the ability to recover its original shape once the external forces are removed. Common examples of EDOs include tubes, cables, metal sheets, rubber layers, sponges, and shoe soles. To simplify the terminology, we use the term **"deformable object (DO)"** or **"object"** to refer to EDO in the rest of the thesis.

- **Assumptions on the object forms.** We define the object forms as:

  - **Linear.** Objects that have one dimension that is significantly longer than the other dimensions such as wires, rods, pipes, and cables.
  - **Thin-shell.** Objects with an infinitesimal thickness, such as cloth and paper.
  - **Volumetric.** Objects with non-negligible thickness, such as sponges or cushions.

In the following sections of this chapter, we overview the current status and primary challenges associated with shape tracking and servoing of DOs. We will then present our contributions and explain how they address these challenges.

1.1 Shape tracking

1.1.1 Challenges

Shape tracking of DOs is challenging. This stems from the following reasons:

- *Non-rigid motion:* DOs undergo non-rigid motions. Consequently, different parts of the object move in different ways.

- *Occlusions:* DOs can be occluded by itself or by other objects or both.

- *Noise:* Most off-the-shelf sensors such as cameras provide noisy measurements. Furthermore, their calibration typically involves modeling uncertainties.
- *Illumination changes:* Changes in lighting conditions can affect the appearance of DOs, making it difficult to track their shape accurately.
- *Computational complexity:* Shape tracking of DOs is computationally expensive which makes it difficult to be used in real-time. This limits its usage in many applications.
- *Initialization:* A good initial estimate of the shape of DOs at the beginning of the tracking process is crucial for certain shape tracking methods. However, providing and automating such an estimate is difficult.

Various shape tracking methods have been proposed in the literature. We approach these methods from two points of view, i.e., convergence and sensor type.

1.1.2 Shape tracking from convergence point of view

From the convergence point of view, shape tracking methods are divided into two main categories: short-baseline and wide-baseline. In the short-baseline case, the input is a continuous video sequence. The inferred 3D shape from one frame is used as an initial guess for the next frame. Although this method makes it easier to track the object, it fails if the object goes outside the field of view or the initial guess is too different from the 3D shape in the next frame due to large deformation between two consecutive frames. Furthermore, the tracking process requires an initial known shape, which necessitates an initialization process. On the other hand, in the wide-baseline case, the input image is processed individually and the convergence is achieved without an initial guess. This is a more challenging problem, but it is also more robust as it does not rely on the assumption of small camera motion and small object deformation with respect to the initial guess.

1.1.3 Shape tracking from sensor type point of view

In terms of the type of sensors used in the tracking process, there are two primary categories of methods in the literature. The first category relies solely on 2D images captured by a single 2D camera, known as a monocular camera, to infer the objects' shapes. The next category employs the use of 3D sensors, such as 3D cameras, Time-of-Flight (ToF) cameras, and structured light systems, to gather information about the objects' shapes. Both categories have their own advantages and limitations, and the choice of a sensor depends on the specific requirements and constraints of the task at hand. In the following, we discuss both categories.

1.1.3.1 Shape tracking using a monocular 2D camera

Shape tracking of DOs using a monocular 2D camera requires reconstruction of the shape at each frame. This is because, using monocular cameras, only a 2D representation of the object is accessible. In monocular shape reconstruction, the camera's perspective projection introduces the challenge of recovering the shape's depth from a 2D image. This challenge becomes extremely difficult for DOs. This is because a 2D image only captures the projection of a 3D object onto the image plane, resulting in an infinite number of possible 3D shapes that could produce the same 2D image. As a result, monocular shape reconstruction relies heavily on computer algorithms to interpret the depth of the DO from limited information contained in the 2D images such as shading, texture, silhouettes, contours, and motion. Additionally, monocular shape reconstruction may be affected by

factors such as camera viewpoint, perspective distortion, and occlusions, which makes it even more challenging. To address this problem, two general solutions have been proposed in the literature:

- **Shape-from-Template** (SfT) [BGC+15] is the generic name for a set of methods that perform the monocular 3D reconstruction of DOs using a template of the object as a known data. SfT takes a single image as the input and reconstructs the shape of the object in 3D.
- **Non-Rigid-Shape-from-Motion** (NRSfM) [BHB00, DB08, RFA11, THB08, VA12] is an extension of SfM [HZ03] which is a well-established method used to reconstruct rigid objects from monocular images. SfM utilizes the inter-image visual motion information to reconstruct the 3D shape of an object. However, the rigidity assumption in SfM limits its application to DOs as the deformation of the object between two images cannot be expressed only in terms of the camera rotation and translation. To address this limitation, NRSfM has been proposed. NRSfM replaces the rigidity constraint with a deformation model. It takes multiple images as input and produces the 3D shape of the object for each image [Par17].

In this thesis, we focus on a SfT-based solution. Figure 1.2 provides a visual representation of SfT.

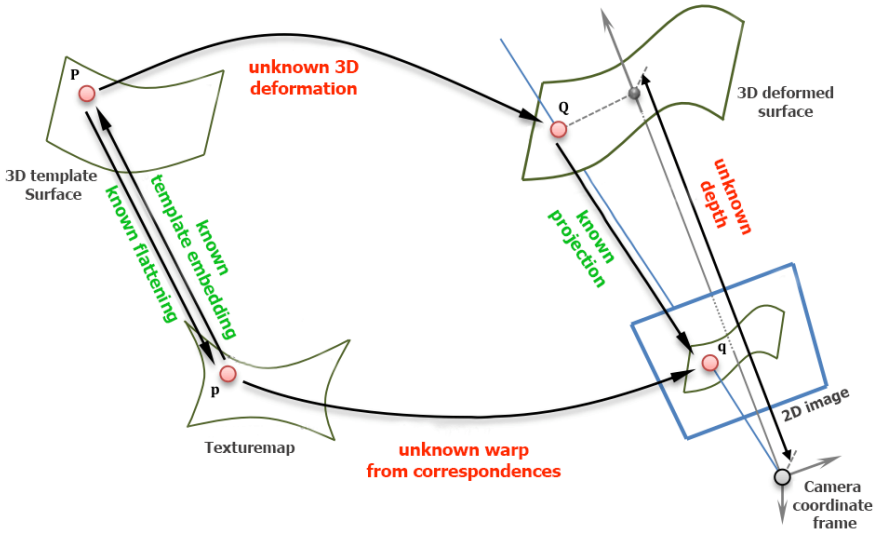


FIGURE 1.2: A visual representation of SfT [BGC+15].

SfT relies on a template consisting of a 3D reference shape of the object at its rest shape (such as a triangulated mesh), a texturemap that assigns colors to the mesh's facets, and a deformation prior. A 3D scanner, RGB-D sensor, or SfM can be used to obtain the 3D reference shape and the texturemap. Isometry is the most widely used deformation prior in SfT [SF09,SMNLF08,PHB11]. Isometry is characterized by the preservation of geodesic distances between any two points on the object during deformation. This concept can be viewed as local rigidity, and it is a good approximation for most natural objects that undergo near-isometric deformations. Isometry is relatively easy to model mathematically. It has been shown that using isometry in SfT can uniquely determine the depth of each object point [BGC+15].

The two primary challenges in SfT are registration and 3D shape inference, both of which should be addressed using a single image as the input. Registration involves establishing correspondences between the input image and the texturemap, while 3D shape inference is the process of inferring depth information and reconstructing the object's shape in 3D space. Following this, we categorize existing SfT methods into two groups: shape inference methods and integrated methods. Shape inference methods only address the 3D shape inference challenge [SMNLF08,SF09,BGC+15,CPBC16,FBA18,BBH14, ÖB17,ACRM+20]. In fact, they require other algorithms for handling registration, which involves finding correspondences between the texturemap and the image. Registration can be performed at each frame independently or at a series of consecutive frames. In the former case, correspondences between the texturemap and the image are first found using feature descriptors such as SIFT [Low04], and then mismatch removal algorithms are used to remove the wrong correspondences [TCC+12, PB12, FBA18]. In the latter case, feature tracking algorithms like [Suh09] are used. The majority of 3D shape inference methods are wide-baseline. However, they barely run in real-time. Furthermore, a complete solution with registration shall be even slower. In contrast to shape inference methods, integrated methods address both the registration and 3D shape inference challenges at the same time [SF10,ÖVNF12,NÖF15,CB15,CBBC16,LYYA+17]. The majority of integrated methods are short-baseline meaning that they require an initialization close to the solution. In addition, they often fail against occlusions and fast motions. Once failed, they need to be reinitialized. In addition to the two mentioned categories of methods, Deep Neural Network (DNN) based methods, as the third group has been recently introduced. DNN-based methods address both the registration and 3D shape inference challenges [PAP+18, GSVS18, SGTS19, FJPCP+22, FJPCP+21]. DNN-based methods are wide-baseline and run in real-time. However, they are object-specific. They require a massive amount of training data and proper computational resources for each new object. These make them difficult to consider as a general and ready-to-use solution.

1.1.3.2 Shape tracking using a 3D sensor

The use of 3D sensors can provide more accurate and comprehensive information about the objects' shapes. This is because 3D sensors provide a richer source of information compared to monocular cameras. With 3D sensors, the captured data consists of point clouds that inherently contain depth information. This added dimensionality allows for more accurate and robust shape tracking since the depth information can be used to calculate the precise location of objects in 3D space. However, the applications may be limited by the constraints of size, cost, and accuracy of the 3D sensors, specifically, in delicate cases such as surgery. Furthermore, the accuracy of these sensors can be affected by various factors such as lighting conditions, occlusions, and range limitations. 3D cameras (RGBD) are the mostly-used types of 3D sensors in the literature.

Various methods have been proposed for shape tracking of DOs using 3D cameras. These methods mainly involve registering the object's point cloud in 3D to the point cloud captured by the 3D camera. This registration is performed using regularization terms to preserve the local geometrical structure and global topology [CR00,CR03,MSCP06, MS10, TT22], or by using physical models [MT93, SLHA13, PLS15]. The latter approach is more precise but more computationally complex. In general, shape tracking methods based on 3D cameras are generally more robust than those using monocular cameras, but they require specific, powerful hardware and face challenges such as sensor noise. Additionally, these methods are short-baseline and require precise initialization at the beginning of the frame series.

1.2 Shape servoing

1.2.1 Challenges

Shape servoing is a task that poses various challenges. Not only does it inherit the challenges of shape tracking, which serves as an input to the servoing process, but there are additional challenges that need to be overcome including:

- *Under-actuation:* Due to their inherent flexibility, DOs typically have a much larger number of degrees of freedom than robotic systems, leading to a highly under-actuated control problem that can be challenging to tackle.
- *Non-linearity:* The relationship between the shape of an object and its control parameters is often nonlinear, making it challenging to derive a precise control strategy.
- *Sensitivity to initial conditions:* Small errors in the initial shape of the object can lead to large errors in the final shape, making it challenging to achieve accurate control.
- *Computational complexity:* Many shape servoing algorithms require significant computational resources, making it difficult to implement them in real-time systems.
- *Heterogeneity:* The properties of deformable objects vary from one object to another, making it challenging to design a unified and general controller for all objects.
- *Uncertainty:* In many real-world scenarios, there is uncertainty about the object's shape and properties, making it challenging to design robust control strategies.

Various shape servoing methods have been introduced to address these challenges. We shortly review these approaches from two points of view, i.e., deformation model and object form.

1.2.2 Shape servoing from deformation model point of view

Shape servoing approaches can be broadly categorized into three categories: model-based, model-free, and learning-based. Each category has its advantages and disadvantages. Model-based approaches incorporate a model to anticipate the object deformation under exerted force or displacement. The majority of these approaches are based on mechanical deformation models [DBPC18, FMC+18, KFB+21, SMEDC+20]. These shape servoing approaches are accurate but computationally expensive and require knowledge of mechanical parameters. Model-free approaches [NALRL13, NALRL14, NAL18, HSP18, ZNF+18, AWH+19, LKM20, ZNAPC21] use online sensor measurements to estimate a deformation Jacobian, but are sensitive to sensor noise. Learning-based approaches [MJD18, HHS+19, SFP+19, JAT20, TCKH21, LK21, HDZAL+22] require high-quality data and have limited applications.

1.2.3 Shape servoing from object general form point of view

Most of the shape servoing approaches are designed to handle one particular form of the object, i.e., linear [KFB+21, ZNF+18, LKM20, QMZ+21, BM14, SMBB20, LLJ22, WZZ+22, AACR+22, AALN+22, APCR+22], thin-shell [Ber13, MDRB20, HSP18, ZNAPC21, HHS+19, SBAMÖ22], and volumetric [TCKH21, FMC+18, ZNAPC21, HHS+19]. This is due to the distinct deformation characteristics and particular assumptions and simplifications that can be made for each one of these object forms. Among all of the presented shape servoing approaches, there are several that can be applied to two or three different forms of

the object [MDRB20, ZNAPC21, HHS+19]. These approaches, however, are merely tested on simple scenarios where the initial and desired shapes of the object are almost identical. In summary, the current literature lacks a shape servoing method that can effectively handle objects with diverse forms and characteristics.

1.3 Contributions

In this thesis, we present our contributions to both shape tracking and shape servoing of DOs. We list our contributions as follows:

Contribution 1. In Chapter 3, we present our contributions to the field of monocular shape tracking, which are summarized as follows:

- **Contribution to SfT.** We propose `ROBUSfT`, a complete real-time robust SfT pipeline for monocular 3D shape tracking of isometrically deforming thin-shell objects with matchable appearance. With the proposed CPU-GPU architecture, `ROBUSfT` can track up to 30 fps using 640×480 images on off-the-shelf hardware. It does not require initialization and implements tracking-by-detection. `ROBUSfT` is wide-baseline and robust to occlusions, object being out of field-of-view, large deformations and fast motions. To apply to a new object, all it needs is a template of that object. It, thus, does not require any training or fine-tuning and is directly usable in many industrial applications and research studies. `ROBUSfT` outperforms the state-of-the-art methods in challenging datasets.

- **Contribution to mismatch removal.** In the registration part of SfT, we introduce `myNeighbor`, a novel mismatch removal algorithm. It handles deforming scenes and a large percentage of mismatches. It is lightning fast, reaching 200 fps. It outperforms the existing mismatch removal algorithms in both performance and execution speed.

- **Contribution to experimental validation.** We design a novel type of validation procedure, called Fake Realistic Experiment (`FREX`). With just one run, it produces a large number of synthetic scenes featuring an object undergoing isometric deformation under various conditions. The scenes come with 2D and 3D ground truth, making it easy to test, evaluate, train and validate new algorithms that deal with isometrically deforming objects, such as removing mismatches, registering 2D images and inferring 3D shapes. Unlike other artificially generated scenes of isometrically deforming surfaces, the images generated by `FREX` are the result of real object deformations. `FREX` is very simple to set up. All that is needed is a piece of paper with a set of Aruco markers printed on it.

- **Contribution to public usability.** We release `ROBUSfT` as an out-of-the-box tool, in the form of a C++ library with a comprehensive tutorial for public use. The code, the tutorial, and a supplementary video of our experiments can be found at `https://github.com/mrshetab/ROBUSfT`.

Contribution 2. In Chapter 4, we present a novel shape servoing approach for thin-shell DOs based on the As-Rigid-As-Possible deformation model. Compared with existing model-based shape servoing approaches, we do not use any mechanical model of the object's deformation. Instead, we only need to define a geometric mesh that represents the object's surface during the task with sufficient accuracy. Our estimated deformation

Jacobian is not sensitive to sensor noise and does not need initialization as in the model-free shape servoing approaches. Finally, in contrast to learning-based shape servoing approach, our approach does not require any training. All it needs is a simple mesh of the object. In fact, the main advantages of our approach are its simplicity and generality, meaning that it can be applied to a wide variety of objects without regulating any parameters.

Contribution 3. In Chapter 5, we present a general unified tracking-servoing approach for DOs that can handle any object form (linear, thin-shell, volumetric) with any geometry. Our approach involves simplifying the deformation of objects by considering a lattice surrounding them and subsequently tracking and servoing the lattice instead of the object. The main characteristics of our approach are listed as follows:

- Our approach does not require any mechanical parameter of the object to be known. Instead, we only use a point cloud that represents the object's surface with sufficient accuracy.
- Our approach does not use the texture of the object. It only employs the point cloud captured in each frame by a 3D camera as the input. This brings the advantage to our approach that it works with objects without specific textures.
- We present a novel analytical expression of the deformation Jacobian. This avoids the need for numerical approximations.
- Our approach has full control over the object's deformation in 3D space. This means that we can start servoing the object from an initial shape, translate, rotate, and deform the object toward a desired shape that is characterized by a totally different visible part of the object in comparison to the visible part of the object at the initial configuration. To our knowledge, this is the first approach that is proven to handle such scenarios in practice.
- The idea of using a lattice makes tracking and servoing much faster as we deal with the deformation equations for the lattice and not the object. This makes the execution speed of the approach to a high extent independent of the object's geometric complexities. As a result, our approach runs fast and is needless of any specific hardware. The execution speed of our tracking-servoing approach reaches 20-30 FPS during our experiments without any parallelization using only CPU.
- The servoing pipeline can servo the whole or a part of the object. The definition of the servoed regions of the objects and its implementation in the servoing pipeline is easy and straightforward.
- Our approach is easily scalable in terms of increasing the number of manipulating robots. Moreover, increasing the number of robots does not impose considerable additional execution time on the servoing approach.

1.4 Publications

Journal papers:

- As-Rigid-as-Possible Shape Servoing.
  Shetab-Bushehri, Mohammadreza and Aranda, Miguel and Mezouar, Youcef and Özgür, Erol.

IEEE Robotics and Automation Letters vol.7, no.2, pp.3898–3905, 2022.
doi: 10.1109/LRA.2022.3145960
Video 1: https://www.youtube.com/watch?v=1w2tbgjLrUst=89s
Video 2: https://www.youtube.com/watch?v=2pzof0dEnMst=2s

- Lattice-based shape tracking and servoing of elastic objects.
  Shetab-Bushehri, Mohammadreza and Aranda, Miguel and Mezouar, Youcef and Özgür, Erol.
  Submitted to IEEE Transaction on Robotics (T-RO), 2022.
  Project website: https://sites.google.com/view/tracking-servoing-approach/home
  Video: https://www.youtube.com/watch?v=h4A2bgAKrMUt=1s

- ROBUSfT: Robust Real-Time Shape-from-Template, a C++ Library.
  Shetab-Bushehri, Mohammadreza and Aranda, Miguel and Mezouar, Youcef, and Bartoli, Adrien, and Özgür, Erol.
  Submitted to Image and Vision Computing, 2022.
  Video: https://www.youtube.com/watch?v=5tFIf7-eEKg

- 3-D shape control of linear deformable objects using an adaptive Lyapunov-based scheme.
  Omid Aghajanzadeh, Mohammadreza Shetab-Bushehri, Miguel Aranda, Juan Antonio Corrales Ramon, Christophe Cariou, Roland Lenain, and Youcef Mezouar.
  Submitted to Control Engineering Practice journal. 2022.
  Video: https://www.youtube.com/watch?v=FlEMAy_IcZo

Conference papers:

- As-Rigid-as-Possible Shape Servoing.
  Shetab-Bushehri, Mohammadreza and Aranda, Miguel and Mezouar, Youcef and Özgür, Erol.
  IEEE International Conference on Robotics and Automation (ICRA), 2022.
  Video 1: https://www.youtube.com/watch?v=1w2tbgjLrUst=89s
  Video 2: https://www.youtube.com/watch?v=2pzof0dEnMst=2s

- Optimal shape servoing with task-focused convergence constraints.
  Giraud, Victor H. and Padrin, Maxime and Shetab-Bushehri, Mohammadreza and Bouzgarrou, Chedli and Mezouar, Youcef and Özgür, Erol.
  IEEE International Conference on Intelligent Robots and Systems (IROS), 2022.
  Video: https://www.youtube.com/watch?v=7wFasZ7imEY

Workshops:

- Offline Reinforcement Learning for Shape Control of Deformable Linear Objects from Limited Real Data.
  Laezza, Rita, Shetab-Bushehri, Mohammadreza, Özgür, Erol, Mezouar, Youcef and Karayiannidis, Yiannis.
  IEEE International Conference on Intelligent Robots and Systems (ICRA), 3rd Workshop on Representing and Manipulating Deformable Objects, 2023.

Patents:

- Système Robotique Bi-Bras de confection de pneumatiques.
  Shetab-Bushehri, Mohammadreza and Giraud, Victor and Roca-Filella, Nicolas and Dettorre, Jean-Marie.

1.5 Thesis outline

This thesis is organized into six chapters. In the first chapter, we provided an overview of the challenges and solutions related to shape tracking and shape servoing of DOs. The following chapters will delve into these topics in greater detail. Chapter 2 provides an overview of the state-of-the-art and limitations in shape tracking and shape servoing of DOs. Our three contributions are detailed in Chapters 3 (shape tracking), 4 (shape servoing), and 5 (unified shape tracking-servoing), each with its own notation specific to that chapter. The thesis concludes in Chapter 6, where we also discuss potential future research directions.

Chapter 2

State of the art

2.1 Introduction

This chapter explores the current research on shape modeling, shape tracking, and shape servoing of DOs. We first review the existing works on shape modeling of DOs. We then examine the existing shape tracking methods using monocular and 3D cameras. Next, we review the state-of-the-art studies on shape servoing. Each section concludes with a discussion on the constraints of each category of methods and opportunities for potential future enhancements.

2.2 Shape modeling

The goal of shape modeling is to develop efficient and accurate methods for predicting the deformation of objects in a way that can be simulated, visualized, and analyzed. The study of modeling DOs has a long history and numerous applications including computer graphics, virtual surgery, and cloth animation [GM97, NMK+06, MM07, BMM15, ZCD+22]. However, due to the highly dynamic behavior of DOs, it remains a major challenge to incorporate their physical properties into models suitable for real-time simulation. In robotics, while selecting a suitable model, the most important factors to consider include computational complexity, physical accuracy, simplicity, and intuitiveness. The model must be able to be used in real-time perception and manipulation while being easy to implement [ARGF+20]. Although significant advancements have been made in the field, it is still challenging to find a comprehensive model that can effectively handle all the complexities associated with DOs, leading to a variety of models in the literature. We divide the models into two main categories:

- **Physics-based deformation models.** Physical laws and mechanical equations govern the object's deformation behavior.
- **Geometry-based deformation models.** Geometric constraints, such as isometry or inextensibility, govern the object's deformation behavior.

In the following, we describe each category of models in more detail.

2.2.1 Physics-based deformation models

Physics-based models are based on fundamental mechanical principles and laws. These models aim to capture the mechanical deformation behavior of DOs under external forces. The finite element method (FEM) and particle systems are the two most popular approaches to solve physics-based deformation models. While FEM requires the mechanical parameters of the object to be known, methods based on particle systems require

one or several parameters to be tuned to match the object's mechanical properties. We describe each of these approaches in more detail below.

2.2.1.1 Finite element method

The finite element method (FEM) is the most popular physics-based approach that directly solves an object's mechanical equations. This approach can handle a wide range of material behaviors, such as nonlinearity, anisotropy, and viscoelasticity, with high accuracy. FEM divides the object into finite elements and solves the mechanical deformation equations on these elements. An example is shown in Figure 2.1. FEM has been utilized

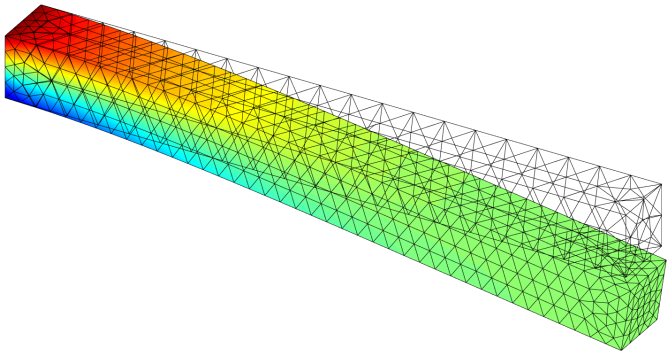


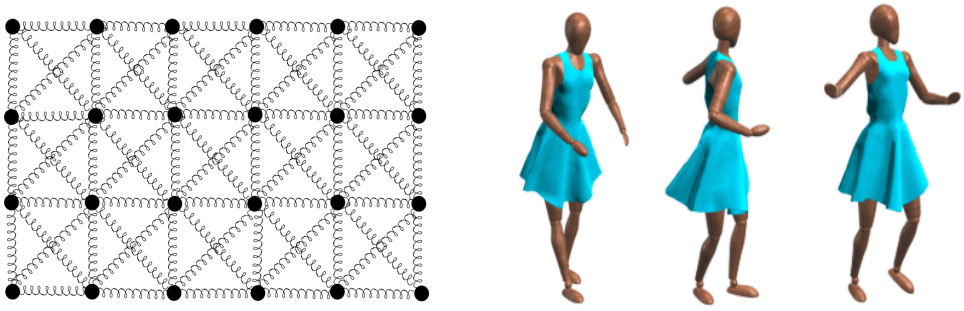
FIGURE 2.1: A flexible rod modeled using FEM. The rod is discretized into an irregular tetrahedral mesh.

in various robotics tasks, including tracking and manipulation of DOs, due to its ability to create realistic simulations and predict complex deformations [ESP92, PCLS18, SKM19, FSS+14]. The quality of the volumetric mesh has a significant impact on the outcome of FEM. Objects with irregular topology, thin parts, or details can generate ill-shaped elements, which are unstable and reduce the accuracy of the stiffness matrix. Increasing the mesh resolution is one solution, but it leads to a higher number of elements, which can decrease performance. Additionally, FEM relies on the mechanical properties of the object, such as its Young's modulus, Poisson's ratio, and friction parameters. FEM can be computationally intensive, but this can be mitigated by using linear FEM [MSJT08]. However, this approach has the limitation of only being able to simulate small deformations. Alternatively, co-rotational FEM and other methods can also be used to reduce computational complexity [MG04].

2.2.1.2 Mass-spring system

The mass-spring systems consist of a group of particles connected by springs. In these systems, particle movement follows Newton's second law of motion. The motion of connected particles is influenced by different forces, including the spring and damping forces. An example of a system of particles and their interconnections is shown in Figure 2.2.a. The object deformability can be determined by tuning the stiffness and damping coefficients defined in the system.

Mass-spring systems are a popular method commonly used in predicting and tracking object states during robotic manipulation due to their computational efficiency and intuitive nature [SLHA13]. However, stability is a well-known issue in these methods. Furthermore, it is difficult to tune the spring constants to obtain the desired deformation



(a) (b)

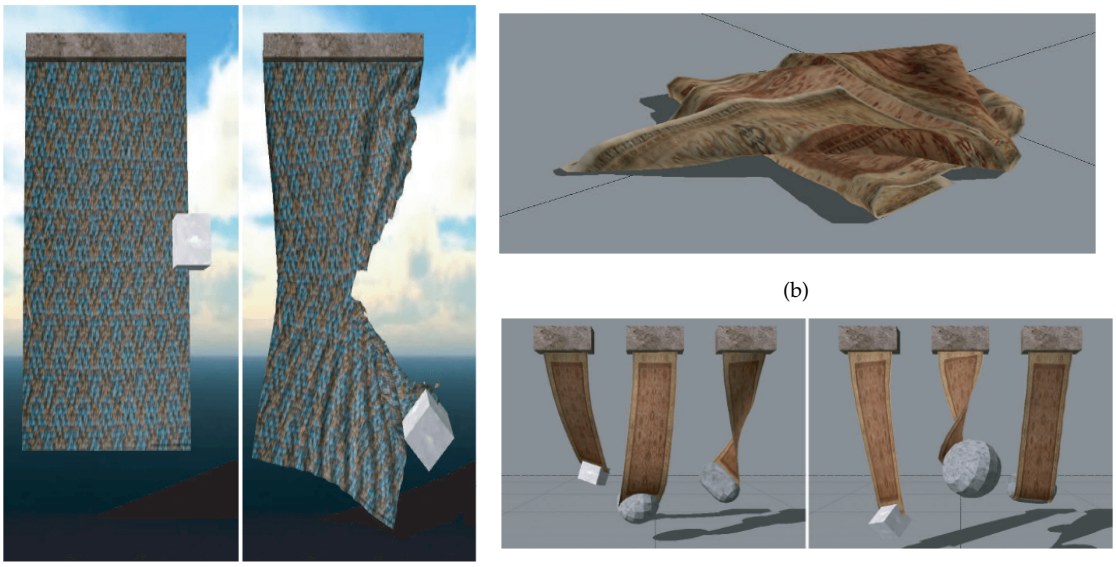
FIGURE 2.2: A mass-spring system, (a) the system of particles and their interconnections, (b) The usage of mass-spring systems in cloth simulation [LBOK13].

behavior according to material properties. To address this tuning problem, learning algorithms and reference solutions have been proposed [BSSH04, MS08, ARW17]. Moreover, mass-spring systems cannot directly simulate volumetric effects in their basic formulation, such as volume conservation. To address this limitation, other energy formulations have been introduced [THMG04]. Additionally, the placement of springs affects the behavior of a mass-spring systems. To compensate for this effect, virtual springs have been added [BC00]. Recently, a new method has been proposed that integrates more complex mechanical behaviors, such as viscoelasticity, non-linearity, and incompressibility, by introducing extra elastic forces into the traditional mass-spring systems [XLL18]. Mass-spring systems are widely used in computer games and animation, especially for materials with possible strong stretching resistance and weak bending resistance like cloth [LBOK13] (see Figure 2.2.b).

2.2.1.3 Position-based dynamics

In the previous section, we explained mass-spring systems which are force-based, i.e., based on given forces, the velocities and positions of particles are determined by a time integration scheme. In contrast, position-based dynamics (PBD) compute the positions directly by applying geometrical constraints in each simulation step [MHHR07]. In PBD, the objects are represented by a set of particles and a set of constraints. For each constraint, a stiffness parameter is introduced which defines the strength of the constraint in a range from zero to one. This gives a user more control over the elasticity of a body. In each simulation step, the predicted particles' positions are iteratively corrected such that they satisfy the internal and external position/force constraints. Finally, the corrected positions are used to update the positions and the velocities.

PBD is highly suitable for interactive environments, owing to its speed, stability, and controllability. By making minor adjustments to its structure, it can effectively simulate a range of physical phenomena, including self-collision, collisions with other bodies, independent bending and stretching, and tearing. However, PBD is generally not as accurate as FEM but still provides visual plausibility. As a result, the main application areas of position-based dynamics are virtual reality, computer games, and special effects in movies and commercials. Figure 2.3 demonstrates several results of PBD simulations.



(a) (b) (c)

FIGURE 2.3: Several results of PBD simulations presented in [MHHR07], (a) a piece of cloth is torn apart by an attached cube, (b) demonstration of stable self-collision and response, (c) cloth stripes are attached via one-way interaction to static rigid bodies at the top and via two-way constraints to rigid bodies at the bottom.

2.2.1.4 Meshless shape matching

Meshless shape matching [MHTG05] can be considered as a particular type of PBD. Here, we discuss it separately due to its unique structure and applications. This model represents the object as a set of particles that are not connected to each other. Due to the lack of connectivity information, when subjected to external forces, the particles tend to rearrange themselves into a configuration that does not resemble the object's original shape. To address this issue, meshless shape matching divides the particles into different clusters. Then, for each cluster of particles, it computes an optimal linear transformation between the current and original shapes. This transformation is divided into rotational (rigid) and symmetric (deformation) parts. To adjust the deformability and stiffness of the model, two hyperparameters can be fine-tuned.

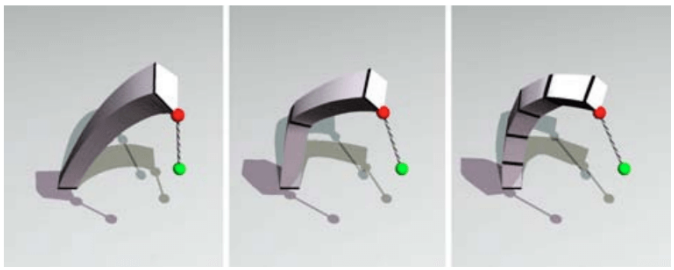


FIGURE 2.4: An example of modeling deformation using meshless shape matching. A beam is deformed by drawing it from a point at its top. The deformation is simulated while one, two, and five clusters of particles are considered.

Meshless shape matching has gained popularity due to its simplicity, efficiency, and scalability in modeling deformable objects without the need for a mesh [MHTG05]. It has

been used in various interactive graphical applications [TYGP13, MMCK14], including modeling the deformation of human body parts [ZGZ+08, SM14] and for robotic manipulation tasks [CBEK16, GPIK17]. One key advantage of meshless shape matching is its ability to simulate volumetric objects while preserving their topological shape, in contrast to mass-spring systems and PBD, which struggle with volumetric object simulation due to the high computational cost for large 3D systems of particles and also volumetric constraints such as Poisson's ratio. Figure 2.4 provides an example of modeling deformation using meshless shape matching.

2.2.2 Geometry-based deformation models

Geometry-based models are based on the geometric properties of an object such as lengths, angles, and curvatures. As a result, these models do not require the mechanical parameters of an object. Compared to physics-based models, geometry-based models are typically faster but less precise. As such, they are often used in computer graphics and animation to create visually appealing deformations rather than to precisely simulate the physical behavior of an object. As-Rigid-As-Possible is a well-known example of geometry-based models. In the following sections, we will provide a more detailed explanation of this model.

2.2.2.1 As-Rigid-As-Possible

The core idea of As-Rigid-As-Possible (ARAP) [SA07] is to simulate geometrically the tendency of an object to preserve local rigidity. The model is based on an energy measure that expresses the deviation from rigidity as the sum of deviations in the local regions of the object. Stable shapes of the object correspond to the local minima of that measure. ARAP has proven powerful and popular in diverse applications. It has

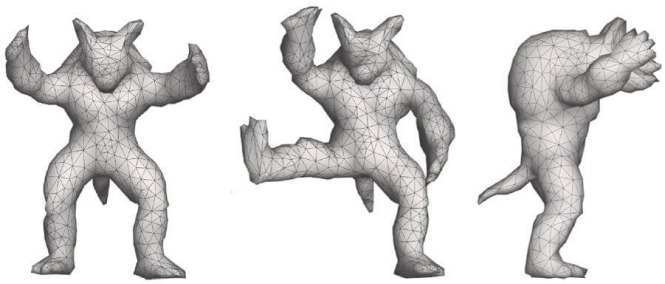


FIGURE 2.5: Modeling an Armadillo using ARAP presented in [SA07].

seen widespread use in computer graphics for shape interpolation, editing, and animation [ACOL00, SA07, LG15]. ARAP has also been employed as deformation constraint in SfT [CBBC16, PPBC15, FJPCP+22, FJPCP+21] and as regularization prior during tracking of nonrigid scenes [NFS15, DDF+17]. [HZSP18, HHS+19] have used ARAP for regularization and inference of occluded regions during deformable object tracking under robotic manipulation. [HALN+22] has utilized ARAP to model the deformation of a flexible sheet carried by multiple robots. An example of modeling movements of an Armadillo using ARAP is illustrated in Figure 2.5.

2.2.3 Discussion

We highlighted examples of commonly used approaches for simulating DO's deformation behavior. Approaches based on physics-based deformation models directly solve

or simulate the mechanical equations governing the object's deformation, which results in high precision but slow computational speed. These approaches require knowledge of the object's mechanical parameters or tuning a set of parameters to match them. On the other hand, approaches based on geometry-based deformation models simulate the object based on its geometric properties, without requiring knowledge of its mechanical parameters. This results in faster computation but less precision. One point that should be noted here is selecting an appropriate model should be based on the specific requirements of the application. For instance, geometry-based models and physics-based models using particle systems are often used for real-time shape tracking of general objects [CBBC16, PPBC15, ACRM+20, FJPCP+22, FJPCP+21], whereas FEM is preferred for accurate tracking of organs in surgical applications [BWG+99].

In this thesis, we utilized PBD in a monocular tracking pipeline in Chapter 3. We also used ARAP to estimate a deformation Jacobian for shape servoing of DOs in Chapter 4. In Chapter 5, we employed a specific formulation of ARAP for tracking objects of any shape and derived an analytical expression for the deformation Jacobian to control a DO's shape towards a desired shape.

2.3 Shape tracking

Tracking the 3D shape of DOs is essential in various fields such as augmented reality [PLF08,HDBC14], computer-assisted surgery [HPR+09,CCB11,MBC12,CBBC16,LPBM20], and robotics [LWC+14,FSS+14,ACRM+20]. Here, we categorize the existing shape tracking methods based on the type of sensor used, i.e., monocular 2D cameras [BGC+15, PPBC15,CPBC16,FBA18,ÖB17,ACRM+20,NÖF15,CB15,CBBC16,LYYA+17,PAP+18,GSVS18, SGTS19,FJPCP+22,FJPCP+21,HXR+18] and 3D cameras [PLS15,TT22,NFS15,TA16,CZLN20]. This categorization is due to the different challenges that arise while using each sensor. While a 3D camera directly accesses the scene's depth, depth estimation poses another challenge in shape tracking using monocular cameras, i.e., estimating depth from 2D features. In the following section, we will discuss each category of studies in more detail.

2.3.1 Shape tracking using a monocular 2D camera

In this section, we focus on SfT methods. An overview of SfT methods has been presented in Section 1.1.3.1. As discussed, SfT is a category of monocular shape registration and inference methods in which a template of DO is known. The template contains certain priors and constraints among which the most common ones are the object's 3D rest shape, texturemap, deformation law, and camera intrinsics. Furthermore, SfT can be broken down into two main parts: registration and 3D shape inference. Following this, the existing SfT methods are divided into two categories: shape inference methods and integrated methods. The former methods only cover the 3D shape inference part [SMNLF08, SF09, BGC+15, CPBC16, FBA18, BBH14, ÖB17, ACRM+20], while the latter methods cover both the registration and 3D shape inference parts [SF10,ÖVNF12,NÖF15,CB15,CBBC16, LYYA+17]. Another category of methods that has been recently introduced is Deep Neural Network (DNN) based SfT methods [PAP+18,GSVS18,SGTS19,FJPCP+22,FJPCP+21]. These methods cover both the registration and 3D shape inference parts. In the remainder of this chapter, we provide a comprehensive review of the different categories of SfT methods. For each category, we detail the underlying assumptions, key characteristics, and potential limitations.

2.3.1.1 SfT-Shape inference methods

These methods cover the 3D shape inference part. They assume the registration was previously handled [PB12,CMB14,TCC+12,PB12,FBA18,PLF08,FBA18]. The registration involves finding correspondences between the texturemap and the image, which can either be achieved independently for each frame or tracked over consecutive frames. Feature

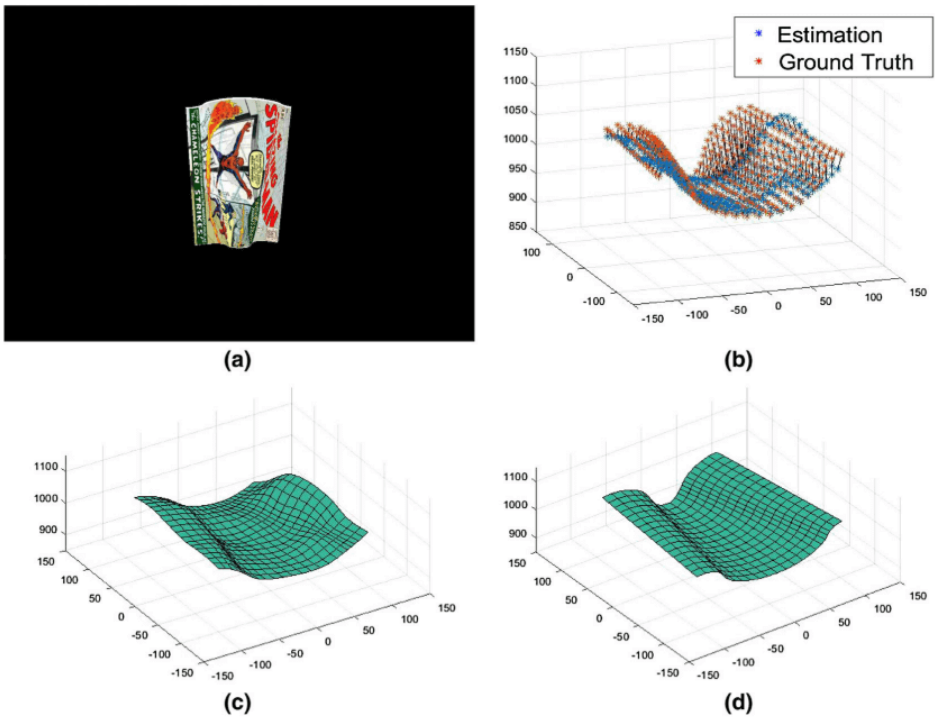


FIGURE 2.6: The reconstruction result of the method presented in [FBA18] on a deformed Spiderman poster. (a) The image of the deformed Spiderman poster. (b) The visualization of the error. (c) Estimation. (d) The groundtruth.

descriptors such as SIFT [Low04] are commonly used to find correspondences between the texturemap and the image. Mismatch removal algorithms are then employed to eliminate wrong correspondences [TCC+12, PB12, FBA18]. Additionally, feature tracking algorithms like [Suh09] can be applied when tracking correspondences over consecutive frames. Three general groups are found in existing 3D shape inference methods; *(i)* methods using a convex relaxation of isometry called inextensibility [SMNLF08,SF09,BBH14], *(ii)* methods using local differential geometry [BGC+15,CPBC16,FBA18], and *(iii)* methods minimizing a global non-convex cost function [BBH14,ÖB17,ACRM+20]. The methods in *(iii)* are the most precise ones but also computationally expensive and they require initialization. The first two groups of methods are often used to provide an initial guess for the third group.

In the first group, Salzmann et al. [SMNLF08] suggested a closed-form solution to non-rigid 3D surface registration by solving a set of quadratic equations accounting for inextensibility. Later, they replaced equality constraints with inequality and thus sharp deformations could be better recovered [SF09]. Brunet et al. [BBH14] formulated two shape inference methods based on point-wise and continuous surface models as Second Order Cone Programs (SOCP).

In the second group, Bartoli et al. [BGC+15] showed that in addition to keypoint 2D coordinates in the image, their first-order differential structure can be used to estimate the depth. Instead of calculating a warp globally, which is time-consuming, Famouri et al. [FBA18] estimated the depth locally for each match pair with respect to both local texture and neighboring matches. In each frame, the most recognizable matches were selected based on offline training. The execution speed of their algorithm is claimed to be up to 14 fps only for the 3D shape inference. An inferred shape of a deformed Spiderman poster along with its groundtruth is presented in Figure 2.6.

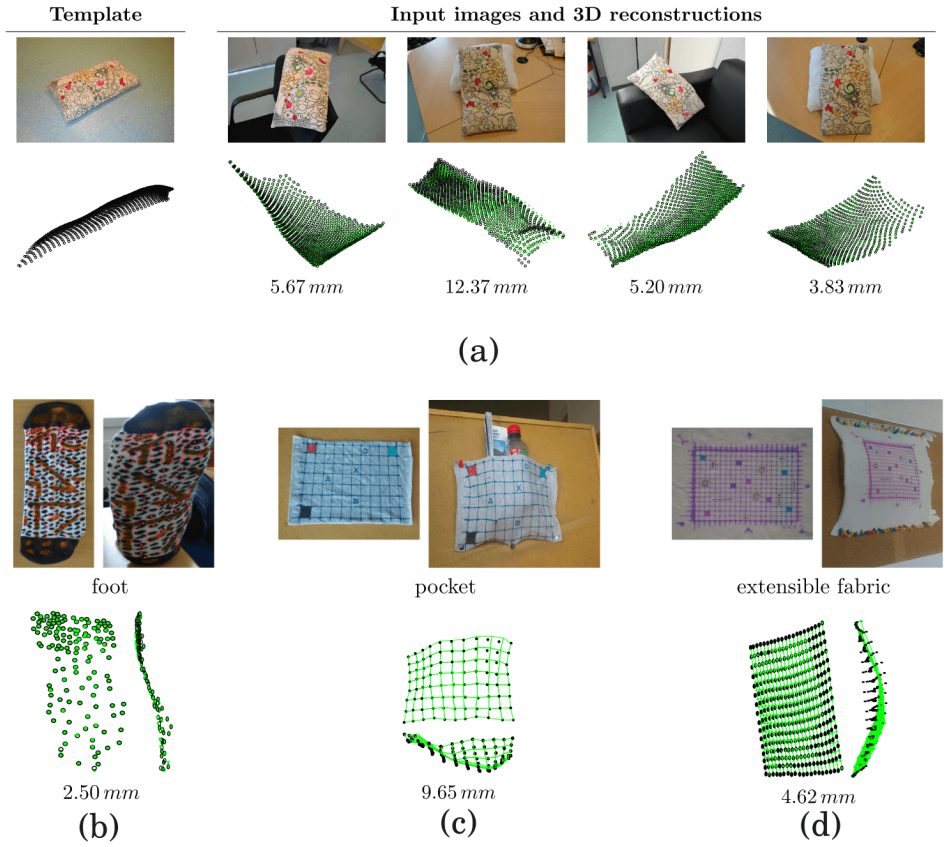


FIGURE 2.7: The reconstruction result of Particle-SfT presented in [ÖB17]. For all the cases, the 3D reconstructions (black circles) of Particle-SfT are shown with the ground-truth shapes (green dots). The reconstruction error is written below each case. (a) Cushion dataset with non-planar template. (b-d) Real elastic datasets. For each case, the left image shows the template image and the right image shows the input image. (b) A foot size 41 wearing a sock for a foot size 37. (c) A pocket filled by a bottle of water and a magazine. (d) A piece of fabric that has large lateral stretching and a mild central pushing deformation.

In the third group, Brunet et al. [BBH14] proposed a refining isometric SfT method by reformulating the isometric constraint and solving it as a non-convex optimization problem. The method required a reasonably accurate 3D shape of the deforming surface as the initializing guess. Özgür and Bartoli [ÖB17], developed Particle-SfT, which handles isometric and non-isometric deformations. A particle system is guided by deformation and reprojection constraints which are applied consecutively to the particle mesh. Similar to [BBH14], this algorithm needs an initial guess for the 3D position of the particles, however, for [ÖB17], sensitivity to this initial guess is very low. The closer the guess to the

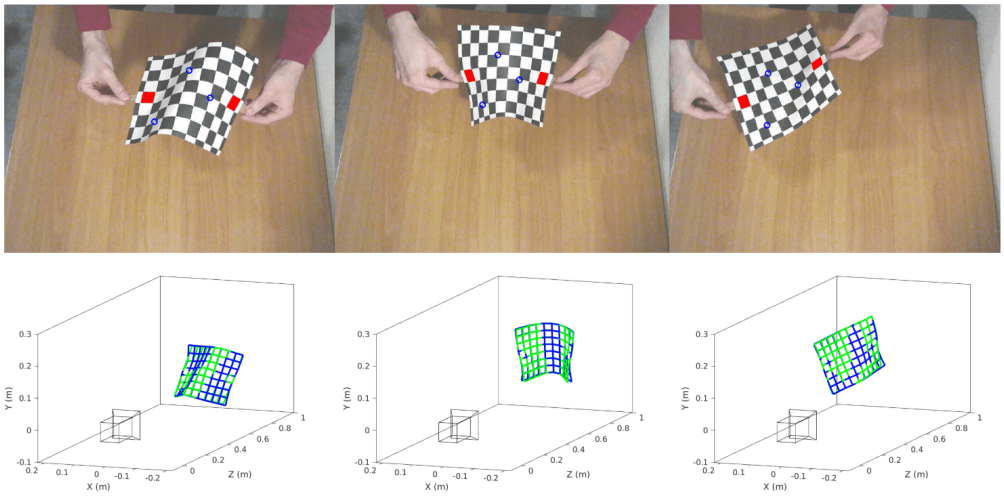


FIGURE 2.8: Shape tracking results presented in [ACRM+20]. Top: three images of a deforming paper sheet with a checkerboard pattern. Three tracked points are marked in blue in each image. Bottom: inferred 3D shapes of the images shown on top using all the 80 points (green) and using only the three tracked points (blue).

true 3D shape, the faster the convergence. The authors in [ÖB17] tested their method in various real experiments as shown in Figure 2.7. Aranda et al. [ACRM+20] improved this algorithm in terms of execution speed and occlusion resistance and used that in real-time shape servoing of isometrically deforming objects. They used the 3D shape estimated in one frame as the initial guess for the next frame and thus improved the convergence speed of the algorithm to a great extent. They showed that their algorithm can track a paper sheet covered with markers being manipulated by a robotic arm. To this end, they only needed to track a handful of markers. Knowing the 3D coordinates of several mesh points also has a significant effect on the convergence speed of the algorithm. The tracking results of this algorithm are illustrated in Figure 2.8.

2.3.1.2 SfT-Integrated methods

These methods handle registration and 3D shape inference at the same time. They minimize a non-convex cost function in order to align the 3D inferred shape with image features. These features can be local [ÖVNF12, NÖF15] or at the pixel-level [CB15, CBBC16].

Ostlund et al. [ÖVNF12] and later Ngo et al. [NÖF15] used the Laplacian formulation to reduce the problem size by introducing control points on the surface of the deforming object. The process of removing mismatches was performed iteratively during optimization by projecting the 3D estimated shape on the image and disregarding the correspondences with higher reprojection errors. Using this procedure, they could reach up to 10 fps using 640×480 input images and restricting the maximum number of template and image keypoints to 500 and 2000, respectively.

As for pixel-level alignment, Collins and Bartoli [CB15] introduced a real-time SfT algorithm which could handle large deformations and occlusions and reaches up to 21 fps. They combined extracted matches with physical deformation priors to perform shape inference. The results of this method in two different experiments with real objects are illustrated in Figure 2.9. Collins et al. [CBBC16] later extended this algorithm and used

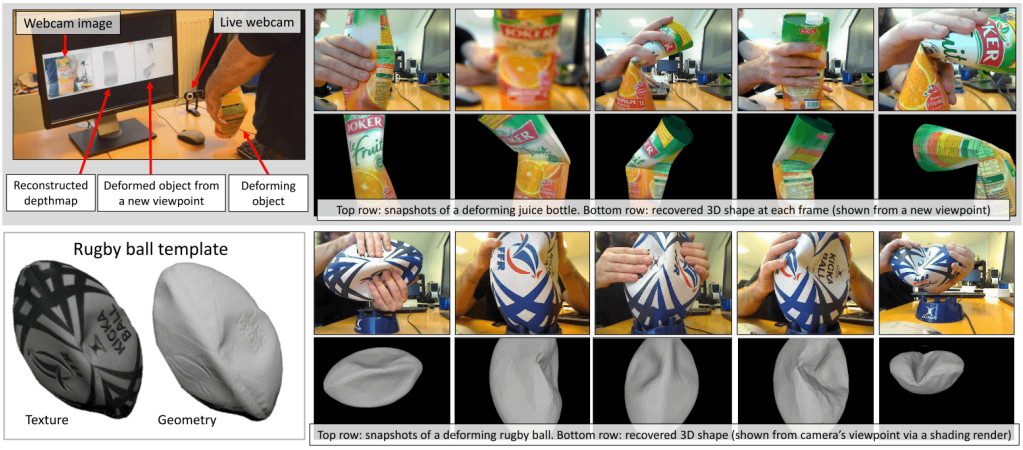


FIGURE 2.9: Shape tracking results presented in [CB15] for a deforming juice bottle and a rugby ball.

it for tracking organs in laparoscopic videos. For achieving better performance, they also exploited organ boundaries as a tracking constraint.

In general, integrated methods are fast and can handle large deformation. Their main drawback, however, is to be short-baseline. In case of tracking failure, they should be re-initialized precisely with a wide-baseline method. This restricts their usage to video streams.

2.3.1.3 DNN-based SfT methods

DNN-based SfT methods have been introduced in recent years, which coincides with the tendency to use deep learning to solve many computer vision problems. These methods are wide-baseline, fast, and cover both the registration and 3D shape inference parts [PAP+18, GSVS18, SGTS19, FJPCP+22, FJPCP+21]. We group these methods based on their type of output, which may be sparse or dense. The methods of the first group represent the SfT solution as the 3D coordinates of a regular mesh with a predefined size [PAP+18, GSVS18, SGTS19]. The usage of these methods is limited to thin-shell objects with rectangular shapes. The second group of methods gives a pixel-level depthmap as output [FJPCP+22, FJPCP+21]. They also apply a post-processing step based on ARAP [SA07] to the resulting depthmap. This step recovers the whole object, including the occluded parts, as a mesh. The method in [FJPCP+22] reconstructs the shape of the object with different geometries and texturemaps that the network was trained for. Several results of this method are presented in Figure 2.10. In [FJPCP+21], the proposed method can be applied to objects with new texturemaps unseen to the network. The geometry of the objects is, nevertheless, limited to flat paper-like shapes. All the aforementioned methods in this category are object-specific. This means that they merely work for the object that they were trained for. An exception is [FJPCP+21], as it works for unseen texturemaps but the applicability is still limited to flat rectangular objects. On the other hand, in order to use the DNN-based methods for a new object, the network should be fine-tuned for it. This demands proper computational resources and potentially a huge amount of training data, which are challenging to collect for DOs.

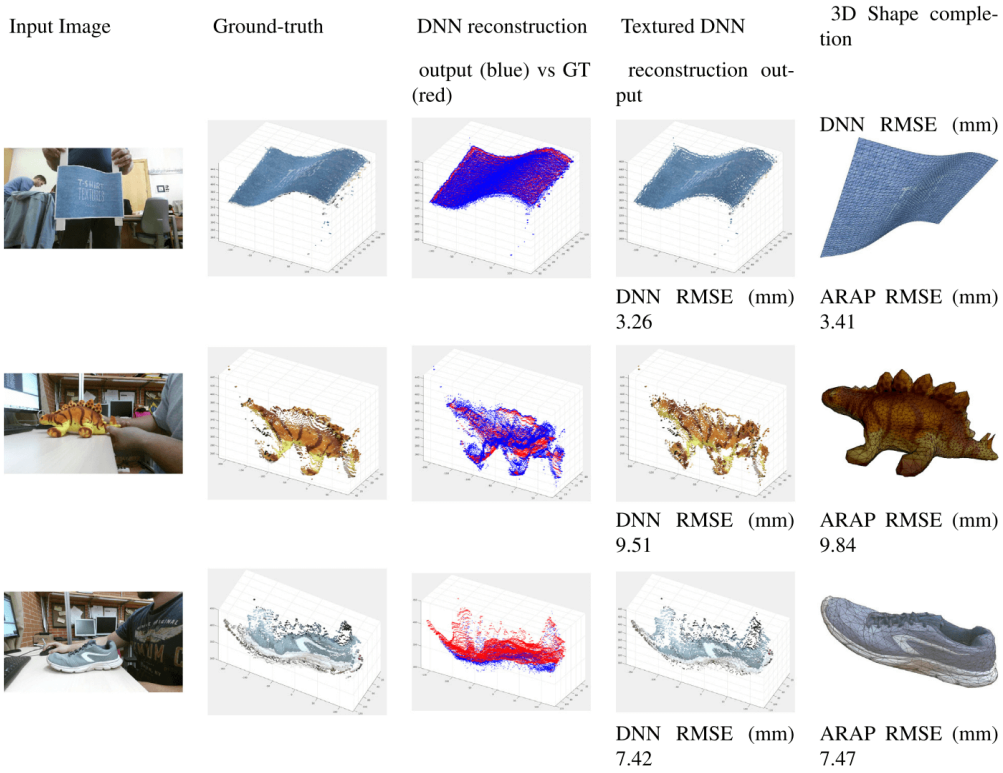


FIGURE 2.10: Shape reconstruction results of three different objects (a paper, a doll, and a shoe) presented in [FJPCP+22]. The network was trained specifically for these objects.

2.3.1.4 Shape tracking using a 3D camera

When using 3D cameras for shape tracking, the captured data consists of point clouds and images. The problem of tracking objects from point clouds involves assigning correspondence between two point clouds and finding the transformation between them. The challenge that might occur in this registration is that the object perceived by 3D cameras is usually represented by a point cloud, which lacks distinguishable features and thus it would be difficult to have explicit correspondence between the captured point cloud and the object point cloud. For objects with identifiable appearances, such as specific colors or rich textures, the captured image by the 3D camera can be utilized to aid in the registration process. This can be achieved through filtering the background's point cloud (using the object's color) or by establishing correspondences between the object's texturemap and the captured image. Shape tracking using a 3D camera can be further complicated by other common issues such as hardware limitations and occlusions, which can result in missing point clouds and noise in the captured data.

The registration of rigid objects has been extensively studied, with Iterative Closest Point (ICP) [BM92] being a well-known solution that assigns binary correspondence between two point sets by the closest distance criterion. To adapt this approach to DOs, the deformation of the object must be taken into account. This has been handled by proposing one-to-many relaxations and utilizing Gaussian mixture models (GMM) to calculate correspondence by probability distribution [CR00]. Thin plate spline-robust point matching (TPS-RPM) registration [CR03] and coherent point drift (CPD) regularization [MSCP06, MS10] are examples of approaches that parameterize the non-rigid transformation. Later, [TT22] introduced the structure-preserved registration (SPR) algorithm

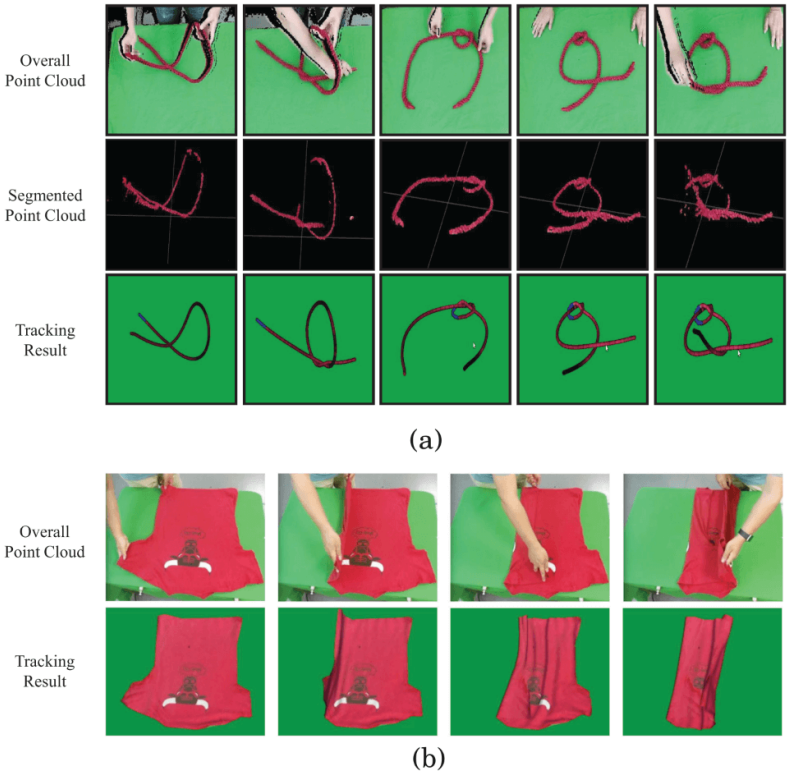


FIGURE 2.11: Shape tracking results presented in [TT22] for (a) a rope and (b) a t-shirt.

which further improves tracking robustness. The proposed method includes regularization terms to preserve local structure and global topology, and a parallel simulation step where an impedance controller drives a virtual object to approach the estimated shape. A series of real-time tracking experiments on linear and thin-shell objects were performed. The results are shown in Figure 2.11.

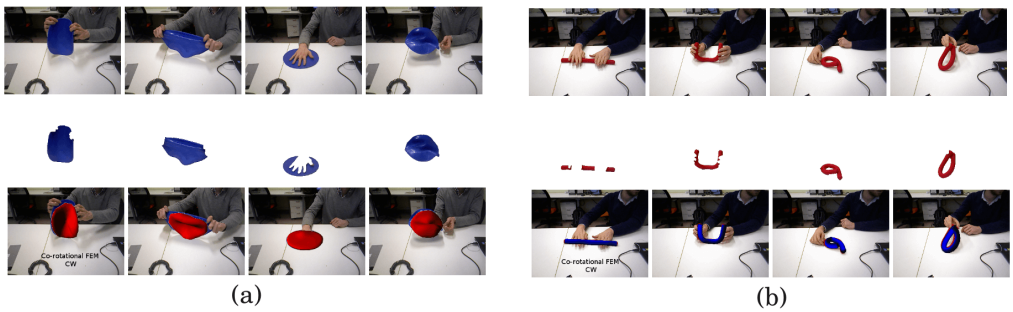


FIGURE 2.12: Shape tracking results presented in [PLS15] for (a) a pizza-like and (b) a cylindrical bar object.

Physical models have also been incorporated into registration methods to improve accuracy, such as a second-order dynamic model for multi-body objects [MT93] and the use of physics engines to minimize bending energy [SLHA13]. Finally, [PLS15] used FEM for tracking elastic objects. Figure 2.12 presents the result of their real-time tracking

method on a pizza-like and a cylindrical bar object. To solve the FEM equations in real-time, the code runs on a GPU.

2.3.1.5 Discussion

We provided an overview of the existing studies on shape tracking of DOs. Our discussion was divided into two main categories based on the type of sensor used, i.e., monocular or 3D cameras, and we highlighted the advantages and disadvantages of each category.

On monocular camera solutions. Typically, monocular cameras are utilized for shape tracking when the object being tracked is well-textured, and 3D sensors cannot be used due to task conditions. In the context of shape tracking using monocular cameras, we focused on SfT methods. SfT methods comprise two main challenges: registration and 3D shape inference. We thus categorized SfT methods based on each of the challenges they aim to tackle. Additionally, we explored a third category of SfT methods that is based on DNN. A summary of the main features of SfT methods in each category is presented in Table 2.1. Typically, SfT methods do not encompass the complete tracking pipeline and

Category	Method	Registration	Real-time	Wide-baseline	General geometry	Needless of training for new objects	Public access code
Shape inference methods	Salzmann et al. [SMNLF08]	×	NA	✓	✓	✓	×
	Brunet et al. [BBH14]	×	×	✓	✓	✓	✓
	Bartoli et al. [BGC+15]	×	NA	✓	✓	✓	✓
	Ozgur et Bartoli [ÖB17]	×	×	✓	✓	✓	×
	Famouri et al. [FBA18]	×	✓	✓	✓	✓	✓
	Aranda et al. [ACRM+20]	×	✓	✓	✓	✓	×
Integrated methods	Ostlund et al. [ÖVNF12]	✓	✓	×	✓	✓	×
	Ngo et al. [NÖF15]	✓	✓	×	✓	✓	×
	Collins and Bartoli [CB15]	✓	✓	×	✓	✓	×
	Collins et al. [CBBC16]	✓	✓	×	✓	✓	×
DNN-based methods	Pumarola et al. [PAP+18]	✓	×	✓	×	×	×
	Golyanik et al. [GSVS18]	✓	✓	✓	×	×	×
	Fuentes-Jimenez et al. [FJPCP+22]	✓	✓	✓	✓	×	×
	Shimada et al. [SGTS19]	✓	✓	✓	×	×	×
	Fuentes-Jimenez et al. [FJPCP+21]	✓	✓	✓	×	✓	×

TABLE 2.1: A comparison between the state-of-the-art monocular SfT methods.

often rely on other algorithms. While some SfT methods do cover the entire pipeline, they are either short-baseline or primarily based on deep neural networks. The DNN-based SfT methods are object-specific, necessitating the retraining of the network for each new object. The substantial data required for this process is particularly challenging to obtain for DOs.

On 3D camera solutions. As for shape tracking methods using 3D cameras, the main challenge reduces to point registration since we have access to point clouds that include depth information. Besides, the possible distinct appearance of the object (specific color or rich texturemap) along with the captured image can also be employed to improve this registration. Consequently, these methods are generally more robust than those using monocular cameras. However, the downside of these methods is that they usually require specific, powerful hardware for analyzing the point cloud. Furthermore, filtering the point cloud belonging to the object can be challenging due to various factors such as noise from 3D cameras, occlusions and self-occlusions, and the presence of other objects in the scene.

On the limitations of shape tracking in robotic applications. Despite the significant progress made in shape tracking of DOs, the application of shape tracking methods in the robotic field has been limited for which we can list several reasons:

- The codes of the deformable object tracking methods in the literature are not open access, or if they are, they are challenging to adapt to a new problem.
- These methods are primarily object-specific, meaning that plenty of adjustments should be made to apply them to a new object. In the case of methods based on physics-based models, these adjustments might include setting or tuning the mechanical parameters of the object [FBA18, CB15, NÖF15, CBBC16, PLS15]. Regarding the neural-network-based methods, these adjustments include creating a new synthetic dataset of the new object of interest and training the whole network from the beginning with that new synthetic dataset, followed by a possible fine-tuning step [PAP+18, GSVS18, SGTS19, FJPCP+22, FJPCP+21].
- Many of the presented tracking methods in the literature are specified to a particular object form including linear [TT22] or thin-shell [BGC+15, CPBC16, FBA18, HXR+18, NFS15, TA16]. As a result, they cannot be used as a general tracking solution to different object forms.
- As these tracking methods are presented in the field of computer vision, they have not exploited the common priors existing in the field of robotics that can be considerably helpful throughout the tracking period, such as the known coordinates of the robotic grippers.

As a result of the mentioned challenges, researchers in the field of robotics have turned to track a simplified representation of objects instead of the whole shape. This simplified representation can be a handful of points on the surface of the object [NALRL13, NALRL14, FMC+18, SFP+19, ACRM+20, MDRB20, AALN+22] or the point cloud of the visible part of the object that remains almost the same throughout the manipulation [HSP18, HHS+19, TCKH21].

On novelty of the thesis in shape tracking. This thesis introduces two shape tracking methods: `ROBUSfT` (Chapter 3), based on a monocular camera, and lattice-based shape tracking (Chapter 5), based on a 3D camera. `ROBUSfT` overcomes many challenges of existing monocular shape tracking methods, such as incomplete coverage of the pipeline, being short baseline, and the need for object-specific training. `ROBUSfT` is also released as a user-friendly C++ library. Lattice-based shape tracking simplifies object deformation by considering a lattice around the object. This results in lower execution complexity and eliminates the need for specialized hardware. It is also capable of handling various forms of objects. Furthermore, both proposed shape tracking methods in this thesis offer several features that make them suitable for robotics applications. With these methods, there is no need for any specific ad-hoc adjustment to work with a new object. Furthermore, both proposed methods support constraints regarding known grippers' coordinates, making them well-suited for robotic applications. More details on these features will be provided in Chapters 3 and 5.

2.4 Shape servoing

This section provides a review of the state-of-the-art shape servoing approaches. We categorize these approaches into model-based, model-free, and learning-based. Model-based approaches use a model to facilitate shape servoing, leveraging knowledge of objects' behavior under deformation. These models were discussed in Section 2.2. On the other hand, model-free shape servoing does not require any prior information on object deformation. Finally, learning-based shape servoing exploits learning techniques for performing the servoing tasks. The following subsections discuss these categories in detail.

2.4.1 Model-based shape servoing

Deformation models play a crucial role in shape servoing of DOs by providing knowledge of the objects' behavior under deformation. Physics-based and geometry-based models are among the most commonly used deformation models in the literature of shape servoing. Physics-based models offer high precision and provide an accurate model for shape servoing in delicate situations. Geometry-based models, on the other hand, are computationally efficient and can handle the real-time requirements of robotic shape servoing.

Shape servoing based on physics-based models. As for shape servoing based on physics-based models, FEM is the most widely used in the literature [DBPC18, FMC+18, KFB+21, SMEDC+20]. In this context, [DBPC18] presented an open-loop simulation-based control methodology wherein the desired deformations are directly mapped to joint angle commands. [FMC+18] used FEM in an open-loop control approach for in-hand soft object manipulation. The approach involves constructing an FEM model of the object and updating it in real-time to account for changes in the object's shape due to manipulation. The FEM model is used to estimate the deformation of the object in response to external forces and the resulting deformation is used to control the manipulation of the object. The approach is validated through experiments on a robotic manipulator performing grasping and twisting tasks. This is shown in Figure 2.13. [KFB+21] introduced a closed-loop controller employing a computationally-efficient FEM that exploits a partition of mesh nodes. The authors applied their approach to a linear object. This is illustrated in Figure 2.14. In [SMEDC+20], the authors used only force feedback along with FEM to servo a volumetric object. [LLJ22] presented a dynamic model that simulated stretching, bending, and twisting deformation in linear objects. The authors exploited this model to deploy a linear object onto a plane using both single-arm and dual-arm setups. They utilize a model-based predictive control (MPC) approach to track a desired trajectory for a linear object. The control objective is to minimize the difference between the desired and actual positions and velocities of the linear object. The MPC controller takes into account the object's dynamic model, constraints on the end effector's motion, and actuator limitations. The proposed approach is evaluated through simulations and experiments.

Shape servoing based on geometry-based models. Unlike physics-based models, geometry-based models simulate object deformation considering only geometrical constraints. This makes these models independent of the mechanical parameters of the object. [ACRM+20] drove the object toward the desired shape by defining intermediary shapes. The intermediary shapes were calculated by applying PBD [MHHR07] as the deformation model. This approach has been tested in a single-arm servoing scenario with a piece of paper. This is illustrated in Figure 2.15. In Chapter 4, we propose to employ ARAP to calculate a deformation Jacobian for full shape servoing of thin-shell objects. Later, [GPSB+22,

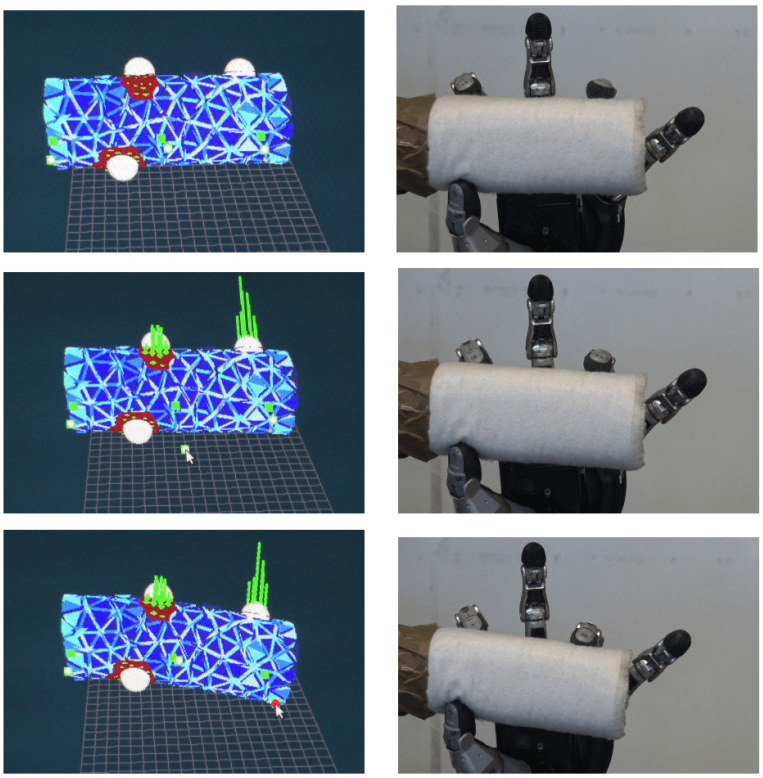


FIGURE 2.13: Shape servoing results presented in [FMC+18]. A comparison between the desired deformation planned in the Sofa environment scene and the deformation controlled on the real object is shown.

APCR+22] exploited the same deformation Jacobian with an optimal controller. Recently, [AALN+22] proposed a novel offline Jacobian to be used in servoing linear objects in 2D space. This Jacobian was based on the As-Similar-As-Possible (ASAP) deformation model which is a simplified version of ARAP. In this deformation model, the object has a tendency to preserve its original shape up to a similarity transformation of that shape. In Chapter 5, we introduce a novel shape servoing approach called Lattice-based shape servoing, which is based on ARAP. This approach works by servoing a lattice around the object towards the desired shape instead of the object itself. This approach can provide a generalized solution that can work with objects of any form and geometry.

2.4.2 Model-free shape servoing

Model-free shape servoing can be considered as the most studied category of approaches in the literature. In contrast to model-based shape servoing, in model-free shape servoing, no prior information on the object's deformation is required. We divide these approaches into two main categories: sensor-based deformation Jacobian, and geometric heuristics.

Shape servoing with sensor-based deformation Jacobian. As for the first category of approaches, online sensor measurements are employed to estimate a deformation Jacobian used for shape servoing [NALRL13, NALRL14, NAL18, HSP18, ZNF+18, AWH+19, LKM20, ZNAPC21]. These sensor measurements are taken by a 2D or 3D camera while

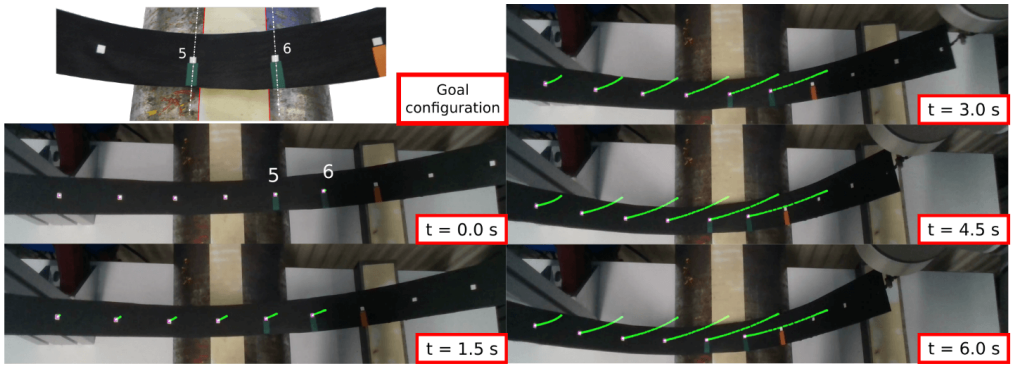


FIGURE 2.14: Shape servoing results presented in [KFB+21]. Two points on a linear object are derived toward their desired positions.

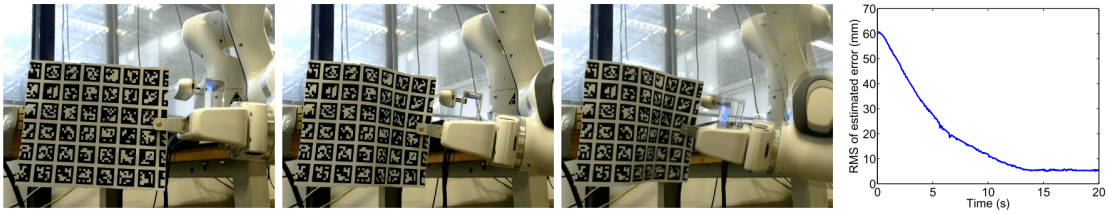


FIGURE 2.15: Shape servoing results presented in [ACRM+20]. Left to right: the initial image, an intermediate image, the final image, and RMS of the estimated shape servoing error.

the object is manipulated by robots. This Jacobian is then used to control a simplified representation of the object, i.e., several sampled points on object's surface [NALRL13, NALRL14, HSP18, AWH+19], or object's contour [NAL18, ZNF+18, ZNAPC21] in the image space. Two examples are presented in Figure 2.16. In general, requiring the robot's motion for calculating the Jacobian makes these approaches more complex, and sensitive to noise.

Shape servoing with geometric heuristics. While many approaches in the literature use sensor-based deformation Jacobian, only a few studies have explored the use of geometric heuristics [Ber13, MDRB20]. These approaches rely on a heuristic called diminishing rigidity to estimate the deformation Jacobian. While this approach is fast and has shown promise in simulation and real experiments with simple deformations, its lack of a deformation model limits its practicality in more complex scenarios.

2.4.3 Learning-based shape servoing

In recent years, the field of deformable object manipulation has been aligned with the growing trend toward using learning-based approaches. Different studies have been conducted in this field [MJD18, HHS+19, SFP+19, JAT20, TCKH21, LK21, HDZAL+22] among which Reinforcement Learning (RL) was the most widely-used approach [MJD18, JAT20, LK21, HDZAL+22]. Despite the relative success of these approaches, they still suffer from significant limitations. First, they are specified to a particular type of object such as linear [LK21, HDZAL+22] or thin-shell [MJD18, SFP+19, JAT20] or just a surface of a

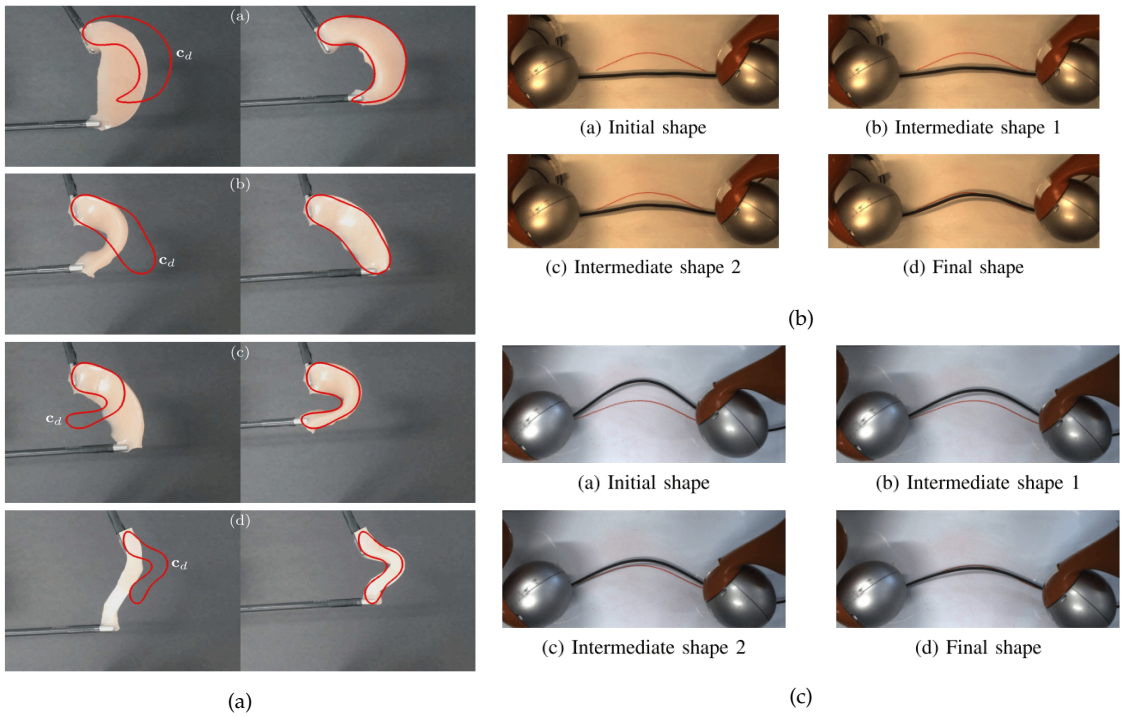


FIGURE 2.16: (a) Shape servoing using contours presented in [NAL18], (b)-(c) dual-arm manipulation of a thick and a thin rope presented in [ZNF+18].

volumetric object [HHS+19,TCKH21]. Furthermore, they have been mostly tested in simulation, and if in real, the deformations have been simple [HHS+19, SFP+19, TCKH21]. Except in [HHS+19] where an online learning process was used to train the agent (i.e., robot) in real, the other studies trained their agents in simulation with objects of specific mechanical parameters. This makes them suffer from the well-known sim-to-real gap. In addition, training the agent for a new object requires a lot of data to provide, which is a tedious task for deformable objects.

2.4.4 Discussion

We discussed different categories of servoing approaches. Model-based shape servoing incorporates a model to anticipate the object deformation under manipulation. Shape servoing approaches based on mechanical models are reliable for predicting object deformation, but they can be computationally expensive and depend on the intrinsic mechanical properties of the object, which vary from one object to another. Shape servoing approaches based on geometrical models, on the other hand, can handle a large range of objects and are fast enough to meet the real-time requirements of robotic shape servoing. Model-free shape servoing approaches do not require prior knowledge of object deformation, but they are prone to sensor noise. Learning-based shape servoing is a recent category that requires high-quality data to perform servoing tasks and has limited applications so far.

One point that should be noted here is that most of the approaches in the literature can merely servo the DOs to the shapes not that different from the initial shape and thus cannot handle large deformations. To overcome this in [LKM20], intermediary shapes were defined to guide the object toward the desired shape which is largely different from the initial shape. This is shown in Figure 2.17.

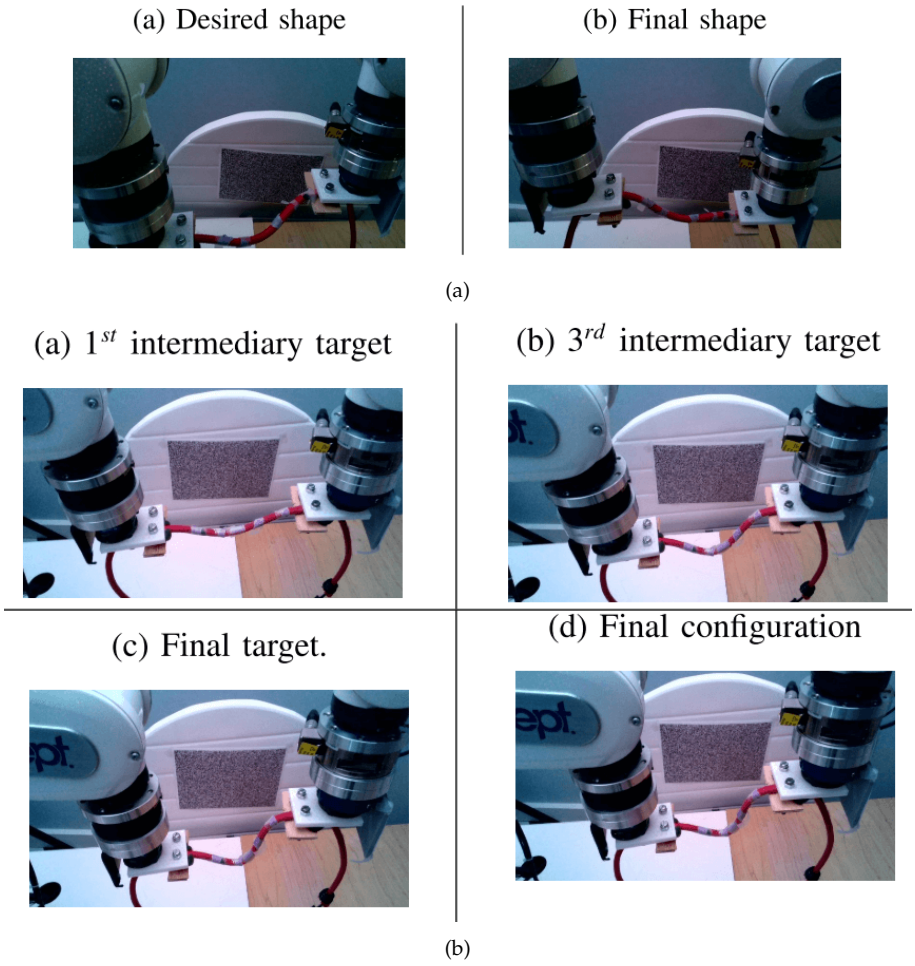


FIGURE 2.17: Dual-arm manipulation of a rope presented in [LKM20]. (a) A failed case due to the large deformation of the desired shape, (b) A successful case by defining multiple intermediary desired shapes.

Another point is that, as discussed in Section 1.2, the existing shape servoing approaches have mainly focused on one form of DOs, i.e., linear [KFB+21, ZNF+18, LKM20, QMZ+21, BM14, SMBB20, LLJ22, WZZ+22, AACR+22, AALN+22, APCR+22], thin-shell [Ber13, MDRB20, HSP18, ZNAPC21, HHS+19, SBAMÖ22], and volumetric [TCKH21, FMC+18, ZNAPC21, HHS+19], due to their distinct deformation characteristics and specific assumptions. While some shape servoing methods have been tested on two or three object forms [MDRB20, ZNAPC21, HHS+19], they are limited to simple scenarios with almost aligned initial and desired shapes. Indeed, the literature lacks a general shape servoing approach capable of being exploited as a universal solution for all forms of DOs and having full control over the whole object in 3D space. In Chapter 5, we introduce a novel approach to shape servoing that overcomes this limitation by forming and servoing a lattice around the object.

Chapter 3

ROBUSfT: a Real-Time Monocular Shape-from-Template pipeline

3.1 Introduction

In this chapter, we focus on the monocular shape tracking of DOs, particularly SfT methods. As discussed in Section 2.3.1, SfT methods in the literature suffer from one or several limitations including not addressing both challenges of monocular shape tracking (namely, registration and shape inference), being short-baseline, failing against occlusions and video cuts, being limited to small deformations, being slow, and being object-specific. In addition, the state-of-the-art SfT methods in the literature either have not provided their codes or if they have, their code is difficult to use. This is because these codes are not presented as out-of-the-box tools, depend on many prerequisite libraries, and need plenty of adjustments to be used. Furthermore, the existing SfT methods do not consider the robotic-related constraints which limits their usage in robotic cases.

These limitations are our motivation for the work in this chapter. We propose ROBUSfT, a complete SfT pipeline for monocular 3D shape tracking of isometrically deforming thin-shell DOs. As discussed in Section 1.1.3.1, isometry is the most widely used deformation prior in SfT [SF09, SMNLF08, PHB11]. We thus use this prior as many natural objects around us deform isometrically, such as paper and cloth.

Our proposed pipeline, ROBUSfT, overcomes all the existing limitations of the state-of-the-art. It addresses both registration and shape inference challenges efficiently. It works in real-time (up to 30 fps) and handles large deformations, partial occlusions, and discontinuity in video frames. ROBUSfT outperforms existing methods in precision and execution speed, as shown in our experiments. To apply to a new DO, all it needs is a template of that DO. Therefore, it does not require any training or fine-tuning, and its application is instantaneous. ROBUSfT is the result of putting existing and novel algorithms together under a novel CPU-GPU architecture. The choice of the algorithms and their distribution between the two processing units are performed in order to optimize efficiency. We release ROBUSfT as a publicly available out-of-the-box C++ library. The code, a tutorial on how to use it, and a supplementary video of the experiments can be found at https://github.com/mrshetab/ROBUSfT.

In the registration part of ROBUSfT, we proposed myNeighbor, a novel mismatch removal algorithm. It works based on the preservation of the local topology modeled by the neighborhood structure of matches. myNeighbor runs very fast (up to 200 fps) and shows high performance even with a high percentage of mismatches and outperforms the existing methods.

As the final contribution presented in this chapter, we introduce Fake Realistic Experiment (FREX), a novel experimental validation framework. A single execution of FREX

provides a large collection of synthetic scenes of an isometrically deforming DO in various conditions, with known 2D and 3D ground truth which can be used to evaluate, compare, train, and validate new algorithms regarding isometrically deforming DOs such as mismatch removal, 2D image registration, and 3D shape inference. In contrast to other artificially generated scenes of an isometrically deforming surface, the generated images in FREX are the result of real object deformations. It is also extremely easy to set up. All it needs is a paper on which a set of Aruco markers are printed.

Chapter outline. We explain ROBUSfT in Section 3.2. This includes describing the structure of the pipeline, i.e., offline and online parts, key steps, and implementation architecture. Section 3.3 presents FREX. Section 3.4 describes myNeighbor, conducts a series of experiments, and evaluates the results of myNeighbor in comparison to previous work. We validate ROBUSfT in Section 3.5. This is done through FREX and real data experiments and comparing the results with previous work. Finally, Section 3.6 concludes and suggests future work.

3.2 ROBUSfT

3.2.1 Overview of the pipeline

The overview of our pipeline is presented in Figure 3.1. The pipeline is divided into two sections: offline and online. The offline section focuses on the creation of the template, while the online section involves four key steps: keypoint extraction and matching, mismatch removal, warp estimation, and 3D shape inference. The input for the first step is the image obtained directly from the camera. Keypoints are extracted and matched with those previously extracted from the template's texturemap. The mismatch removal algorithm, myNeighbor, is then applied to detect and eliminate mismatches. The list of correct matches is used to estimate a warp between the template's texturemap and the image. This warp transforms the registered mesh of the template to the image space, which is used as input for the 3D shape inference algorithm. This process is repeated for each image, allowing for tracking-by-detection. The offline and online sections of the pipeline will be discussed in further detail and a fast implementation of the pipeline will also be provided.

3.2.2 Offline section: creating a template

We create a template for the surface of the deforming object that we want to track. We call this surface the tracking surface. The template of the tracking surface consists of the following elements:

- M_T: the triangular mesh covering the tracking surface at rest shape.
- $\mathcal{P}$: the texturemap of the tracking surface.
- M: the alignment of M_T to $\mathcal{P}$.

The first step in creating the template is to generate the 3D model of the tracking surface. The 3D model is in fact the textured 3D geometry of the tracking surface in real dimensions in rest shape. We form M_T by triangulating this 3D geometry. The resolution of M_T should be high enough to be well aligned to the shape of the tracking surface. The next step is to take an image from the 3D model of the tracking surface while it is positioned perpendicular to the camera's optical axis in a simple texture-less background. In this

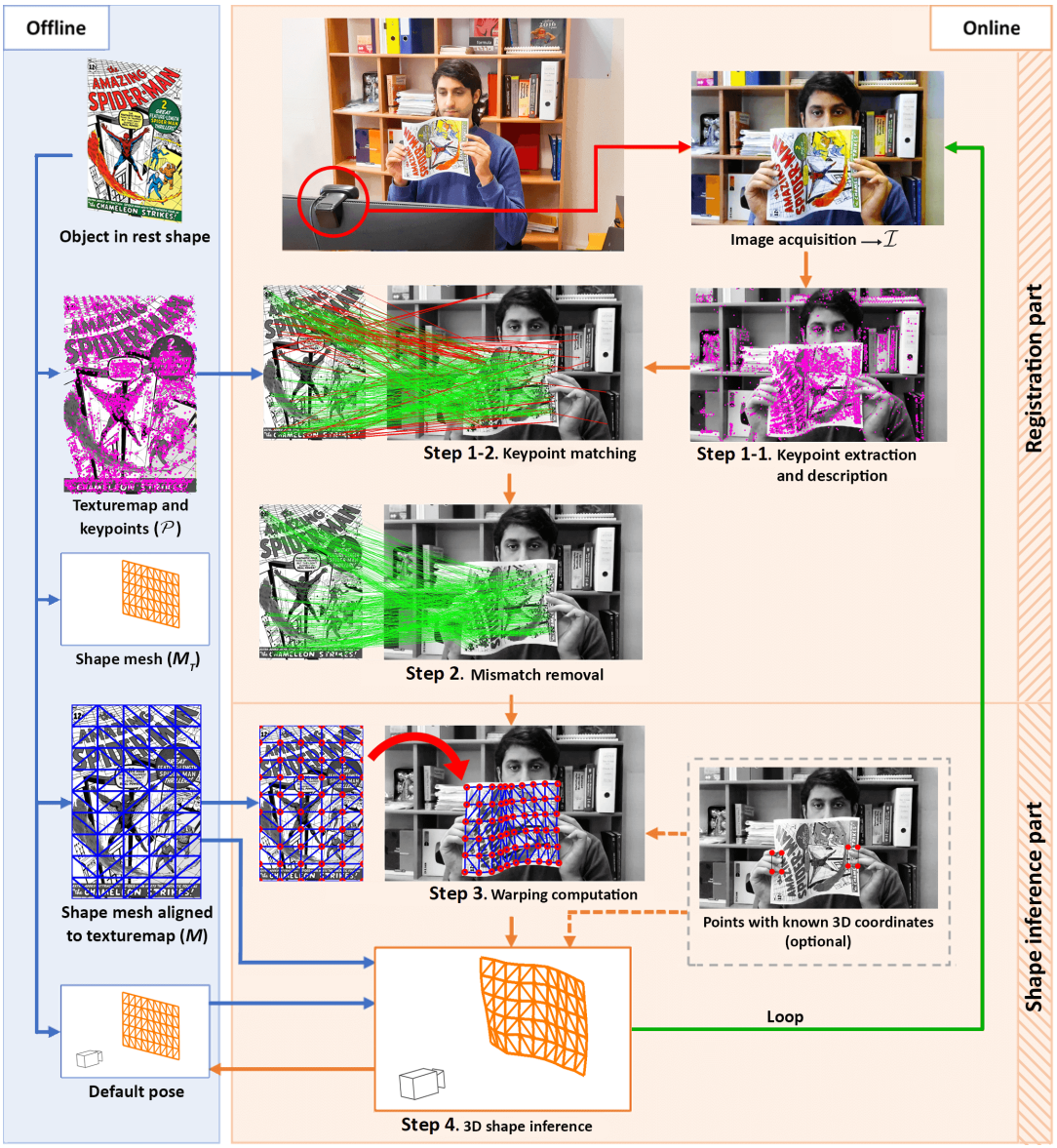


FIGURE 3.1: Overview of ROBUSfT.

image, $\mathcal{P}$ is formed by the projection of the texture of the tracking surface and M by the projection of M_T. For simple rectangular thin-shell objects like a piece of paper, the whole process is straightforward. For other objects, including thin-shell objects with arbitrary shape, such as a shoe sole, and also volumetric objects, 3D reconstruction software like Agisoft Photoscan [Agi] can be used.

Next, we extract keypoints on $\mathcal{P}$. These keypoints will be matched with the ones that will be extracted from the input image in the online section. We use SIFT [Low04] for extracting keypoints but any other feature descriptor could be swapped in. As the final step, we initialize the pose of M_T in 3D space. This initial pose can be arbitrarily chosen as it will be used only once by Step 4 of the online section of the pipeline for the first input image. It will then be replaced by the inferred 3D shape in the next images.

3.2.3 Online section: shape tracking

Step 1: keypoint extraction and matching. The first step of the online section of the pipeline is to extract keypoints in the input image $\mathcal{I}$. To do so, we use the PopSift library [GCH18], which is a GPU implementation of the SIFT algorithm. We then match these keypoints with the ones that were previously extracted from $\mathcal{P}$ by comparing descriptors, using winner-takes-all and Lowe's ratio test. Inevitably, a number of mismatches will be formed between $\mathcal{P}$ and $\mathcal{I}$. The mismatch points in $\mathcal{I}$ can be located on the surface of the deforming object or even in the background. This is shown as red lines in the *Matching* step of Figure 3.1. These mismatches will be eliminated in *Step 2* thanks to `myNeighbor` which can cope with a large percentage of mismatches. As a result, in this step, the images coming from the camera can be used directly without pretraining either on the image for segmenting the object from the background, or on the matches for preselection of the most reliable ones.

Step 2: mismatch removal. To remove the possible mismatches introduced in *Step 1*, a new mismatch removal algorithm, `myNeighbor`, was developed. The main principle used in this algorithm is the preservation of the neighborhood structure of correct matches on a deforming object. In other words, if all of the matches were correct, by deforming the object, the neighbor matches of each match should be preserved. On the contrary, mismatches lead to differences in the neighboring matches of each matched point in $\mathcal{I}$ in comparison to $\mathcal{P}$. This was used as a key indication to detect and remove mismatches. The whole process of `myNeighbor` is explained in Section 3.4.

Step 3: warp estimation. We use the estimated correct matches to estimate a warp W between $\mathcal{P}$ and $\mathcal{I}$. We then use W to transfer M to $\mathcal{I}$ and form $\widehat{M}$. The mesh points in $\widehat{M}$ will be used as sightline constraints in the 3D shape inference algorithm in *Step 4*. The precision of warping depends on the number of matches, their correctness, and their distribution all over $\mathcal{P}$. Warp W can be estimated in the most precise way if all the matches are correct between $\mathcal{P}$ and $\mathcal{I}$. However, due to the smoothing nature of the warping algorithms, the transferring process can cope with a small percentage of mistakenly selected mismatches. It should be noted that W cannot be extremely precise in areas without matches. As a result, in these areas, the shape of $\widehat{M}$ might not be aligned well to the shape of the deforming object in $\mathcal{I}$. This is worse when the matchless area is located near the boundaries of $\mathcal{P}$ as the alignment cannot be guided by the surrounding matches. Hence, in order to use just well-aligned transferred mesh points of $\widehat{M}$ as the input for the 3D shape inference step, an assessment is performed over all of the mesh points and only the qualified ones are passed to *Step 4*. For this, we check M cell-by-cell. Only the mesh vertices for cells containing at least one correct match will be qualified as salient mesh points. The indices of these mesh points and their coordinates in $\widehat{M}$ are passed to *Step 4*. The other mesh points are disregarded.

Representing and estimating W can be done with two well-known types of warp, the Thin-Plate Spline (TPS) [Boo89] and the Bicubic B-Spline (BBS) warps [RSH+99], which we both tested. The former is based on radial basis functions while the latter is formulated on the tensor-product. Having the same number of matches as input, the TPS warp proved to be more precise than the BBS warp; nevertheless, its execution time rises exponentially with increasing number of matches. The execution time, however, remains almost constant for the BBS warp regardless of the number of matches. Thus, considering the criterion of fast execution of the code, the BBS warp was chosen as the warp function in this step and also in the mismatch removal step discussed in Section 3.4.

Step 4: 3D shape inference. We use Particle-SfT [ÖB17]. We particularly use the version improved for real-time shape tracking presented in [ACRM+20]. In this algorithm, a

particle system is defined from the points and edges in M_T. Then, the sightline and deformation constraints are applied consecutively on the particles until they converge to a stable 3D shape. As described in [ACRM+20], in order to increase the convergence speed of the algorithm, the stable 3D shape for an image is used as the initial guess for the next image. It should be noted that Particle-SfT can work even without a close initial guess. If the object is invisible in one or several images, the last inferred 3D shape can be used as the initial guess for the upcoming frame containing the object. This results in a slightly longer computation time in that image. For the next upcoming images the normal computation time is resumed. This capability brings about two of the major advantages of our pipeline, which are being wide-baseline and robust to video discontinuities.

As mentioned in [ACRM+20], one of the optional input data that can significantly improve the convergence of Particle-SfT is the existence of 3D known coordinates of one or several particles. This is shown in Figure 3.1. The known 3D coordinates can be fixed in space, or can move on a certain trajectory. The latter happens when the deforming object is manipulated by tools with known poses in 3D space like robotic grippers.

3.2.4 Implementation

In order to optimize the implementation of `ROBUSfT`, it was coded in C++ in two parallel loops: one on the GPU, and one on the CPU. The GPU loop handles keypoint extraction in the images. These keypoints are transferred to the CPU loop where the rest of the steps of the pipeline are taken. This is shown in Figure 3.2. We also provide a pure CPU implementation of the library. Any arbitrary resolution can be considered for the captured images, nevertheless, we obtained the best performance by using 640×480 images. The code is tested on a Dell laptop with an Intel Core i7 2.60 GHz CPU and a Quadro T1000 GPU. The code, a tutorial on how to use it, and a supplementary video of our experiments are provided at `https://github.com/mrshetab/ROBUSfT`.

3.3 Fake Realistic Experiment (FREX)

We introduce a novel experimental protocol, which we used for evaluating `myNeighbor` and `ROBUSfT` in comparison to the state-of-the-art methods. A single execution of this protocol provides a large collection of scenes of an isometrically deforming object in various conditions, with known 2D and 3D ground truth. This collection can be used to evaluate, compare, train, and validate new algorithms regarding isometrically deforming objects such as mismatch removal, 2D image registration, and isometric 3D shape inference. In contrast to other artificially generated scenes of an isometrically deforming surface, the generated images in our protocol are the result of real object deformations. Being formed of successive images with continuous deformation, it can also be used for algorithms which exploit feature and shape tracking. In addition, object occlusion and invisibility can be easily simulated, by dropping frames or pasting an occluder.

The protocol flowchart is shown in Figure 3.3. First, we form the *Aruco template* by randomly distributing a set of Aruco markers all over a blank image. We then print the Aruco template on a standard A4 paper. These markers should be big enough to be recognizable by the user's camera in the desired distance. In order to improve recognition, there should be white space between the markers on the paper. In our experiments, we used 100 markers with a width of 1.4 cm. The OpenCV library was used to identify the markers. These markers were recognizable by a 720p RGB camera from an approximate distance of 0.6m. The next step is to deform the printed Aruco template in front of the camera. In each frame, the 2D and 3D coordinates of the markers' centers are estimated.

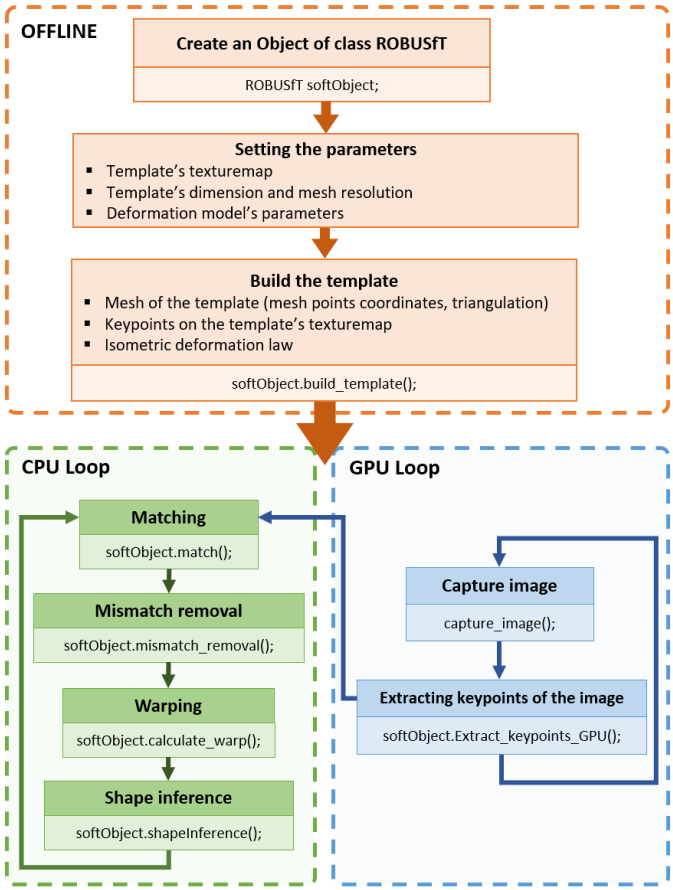


FIGURE 3.2: Implementation of ROBUSfT on the CPU and GPU. A pure CPU implementation is also provided.

Because each marker has its own unique id, they can be used as correspondences between the Aruco template and each image of the video. We exploit the 2D coordinates of these recognized correspondences to estimate a warp with which we can transfer an arbitrary texturemap to the video image space. This is done firstly by resizing the arbitrary texture to the size of the Aruco template. In order to keep the aspect ratio of the arbitrary texturemap, white margins can also be added before resizing. Then, an inverse warping process with bilinear interpolation is used to transfer the pixel color information from the arbitrary texturemap to their corresponding pixels in the video images. The whole procedure results in a scene with the arbitrary texturemap being deformed exactly on top of the Aruco template. It is also possible to add further modifications; for instance, one can transfer the arbitrary texturemap to another scene with any different background. Besides, as in [VSSF12], an artificial lighting can also be added to form different variations of the scene.

For evaluating algorithms, one can use the 2D and 3D ground truth estimated in each frame of the video. Regarding the 2D ground truth, the estimated warp can be used to identify the 2D corresponding point of each pixel of the arbitrary texturemap in the image. As for the 3D ground truth, one can exploit the 3D estimated coordinates of the Aruco markers in each frame which can be achieved using the OpenCV library.

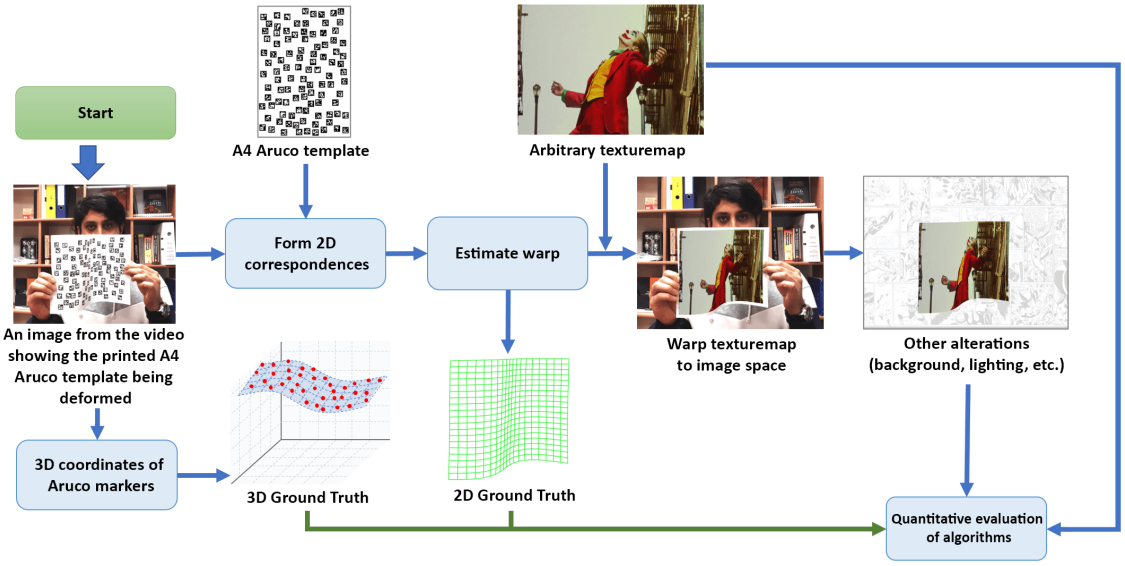


FIGURE 3.3: Flowchart of FREX.

3.4 myNeighbor

We describe myNeighbor, our novel mismatch removal algorithm. It works based on two main principles:

- Given a sample set of correct matches between an image of a textured surface and another image of that surface undergoing a deformation, one can estimate a sufficiently accurate transfer function between the images such that the correctness of all the matches can be judged. Consequently there is no need to remove all the mismatches.
- This sample set of correct matches can be extracted from the images considering that in reality, under a deformation, the neighborhood structure among the points on a deforming surface is preserved.

We show that by using these two principles, the mismatches can be detected and removed in a fast and efficient way. The proposed algorithm is illustrated in Figure 3.4. It consists of three steps. First, a set of matches which are highly probable to be correct are selected. This selection is done by forming two triangulations using match points, one in $\mathcal{P}$ and one in $\mathcal{I}$, and then choosing matches with high similarity in the list of their neighbors. Second, a small percentage of possible mismatches among the selected matches are identified and removed. This is done by transferring the selected match points from $\mathcal{P}$ to $\mathcal{I}$ and then removing those with large distances from their correspondences in $\mathcal{I}$. Third, we transfer all the match points from $\mathcal{P}$ to $\mathcal{I}$ using a warp estimated based on the clean set of selected matches from the second step. The distance between the transferred template match points and their correspondences in $\mathcal{I}$ is used as the criterion to distinguish estimated mismatches from estimated correct matches.

In order to analyze the performance of myNeighbor and calibrate the parameters in the different steps, we used synthetic data experiments. In the following section, we describe the design of these experiments. Afterwards, we describe in detail the different steps of myNeighbor.

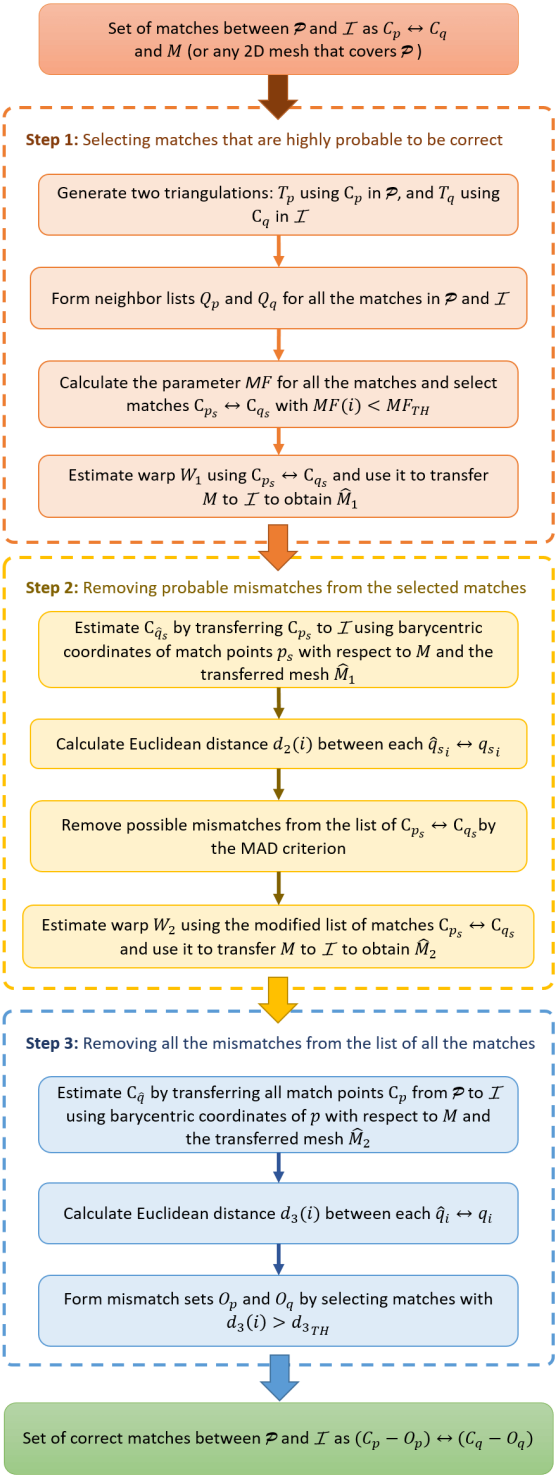


FIGURE 3.4: Flowchart of `myNeighbor`.

3.4.1 Synthetic data experiments for calibrating parameters

These experiments are conducted by synthetically forming two images of a mesh M_T and a series of matches between the two images. The first image shows M_T in its flat rest shape with all its keypoints on it. We call this image $\mathcal{I}_F$. In $\mathcal{I}_F$, the keypoints can be considered as the extracted keypoints from $\mathcal{P}$ and the 2D mesh is equivalent to M.

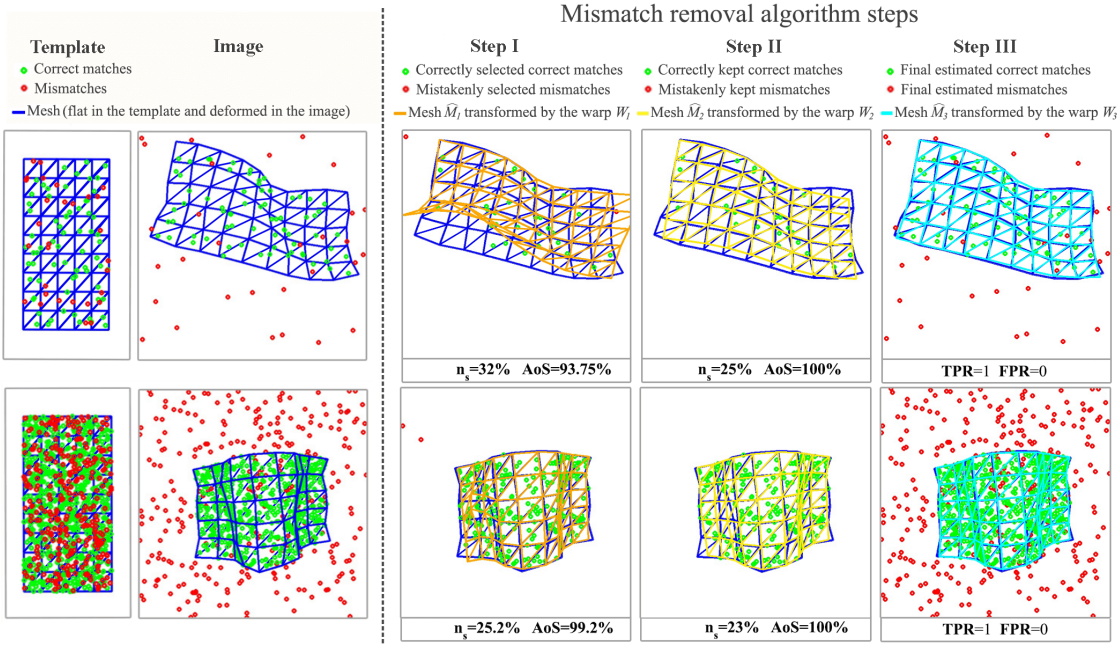


FIGURE 3.5: Two sample results of the steps for synthetic data experiments. The first row is an experiment with 100 matches and a mismatch percentage of 30%. The second row is an experiment with 1000 matches and a mismatch percentage of 30%. The first and second columns represent $\mathcal{I}_F$ and $\mathcal{I}$ with correct matches in green and mismatches in red. The third column is the result of *Step I*. The wrongly chosen mismatches are shown in red. The fourth column is the result of *Step II*. The mismatches along with a small percentage of correct matches are removed. The fifth column is the separation of the estimated correct matches and the estimated mismatches from *Step III*. The transferred meshes $\widehat{M}_1$, $\widehat{M}_2$, and $\widehat{M}_3$ are shown in orange, yellow, and cyan for the three steps.

The second image simulates $\mathcal{I}$ and shows M_T having undergone a random 3D deformation. We call this deformed mesh M_G. The keypoints in this image can be positioned in their correct locations on the mesh (correct matches) or being displaced in the image area (mismatches).

We consider M_T as a regular triangular mesh with 10×6 points in 3D space. In order to deform M_T, we use the same method as in [ACRM+20]. This is done by applying two 3D deformations containing random translations and rotations to two mesh cells at both sides of M_T. The deformation is calculated in an iterative process based on PBD [MHHR07, BMO+14]. As for generating keypoints, we first randomly place keypoints in the inner area of M in $\mathcal{I}_F$. In order to create the matches between $\mathcal{I}_F$ and $\mathcal{I}$, we then transfer the keypoints from $\mathcal{I}_F$ to $\mathcal{I}$ using a three-step process: calculating barycentric coordinates of the keypoints in M, transferring the keypoints to the 3D deformed mesh using the barycentric coordinates and the new 3D mesh points of the deformed M_T, and eventually projecting the transferred keypoints to $\mathcal{I}$. To generate mismatches, an arbitrary percentage of the transferred keypoints were corrupted by randomly distributing them all over the area of $\mathcal{I}$. Two samples of the generated images for 100 and 1000 matches each with 30% mismatches can be observed in the two first columns of Figure 3.5.

3.4.2 Methodology

The algorithm `myNeighbor` is applied on N_m matches denoted as $C_p \leftrightarrow C_q$ between $\mathcal{P}$ and $\mathcal{I}$, with:

$$C_p = \{p_1, ..., p_{N_m}\}, \quad p_i = (x_i, y_i) \tag{3.1}$$

$$C_q = \{q_1, ..., q_{N_m}\}, \quad q_i = (u_i, v_i) \tag{3.2}$$

A pair (p_i, q_i) of points with the same index forms a match $p_i \leftrightarrow q_i$. We define the set of correct matches S_{in} as the collection of matches $p_i \leftrightarrow q_i$ where p_i and q_i point to the same location on the deforming surface in $\mathcal{P}$ and $\mathcal{I}$. On the contrary, when the pointing locations of the match points are different, they are categorized as mismatches S_{out}. The goal of `myNeighbor` is to form and remove the subsets $O_p \subset C_p$ and $O_q \subset C_q$ which have the largest possible number of matches belonging to S_{out} and smallest possible number of matches belonging to S_{in}. We explain the steps of our algorithm to fulfill this goal.

3.4.2.1 Step I – Neighbor-based correct match selection

We select subsets $C_{p_s} \subset C_p$ and $C_{q_s} \subset C_q$ which are highly probable to form correct matches. We start by defining W_G as the ground truth warp between $\mathcal{P}$ and $\mathcal{I}$ that can transfer all the match points C_p from $\mathcal{P}$ to their correct locations in $\mathcal{I}$. With this definition, we have the set of correct matches S_{in} as:

$$S_{in} = \{(p_i, q_i) \,|\, i \in R\}, \tag{3.3}$$

where:

$$R = \{i \,|\, \|W_G(p_i) - q_i\| < \epsilon\}, \tag{3.4}$$

where ϵ is a very small positive number. Warp W_G is an unknown composition of isometric deformation and perspective projection mappings. The isometric deformation mapping preserves the geodesic distances among the points and their topological structure on the object's surface. However, with the addition of perspective projection mappings, only the topological structure of points remains preserved in visible areas. This implies that by applying W_G, the neighborhood structure among the points on the object in $\mathcal{P}$ and $\mathcal{I}$ should be preserved. We exploit this characteristic of W_G to estimate $\widehat{R}$ as the set of indices of highly probable correct matches $C_{p_s} \leftrightarrow C_{q_s}$. To do so, first, we form two Delaunay triangulations, $T_p = D(C_p)$ in $\mathcal{P}$, and $T_q = D(C_q)$ in $\mathcal{I}$. Then, for each match i, we calculate two sets of first-order neighbors $Q_p(i)$ and $Q_q(i)$ in $\mathcal{P}$ and $\mathcal{I}$, respectively. We then define the *Mismatch Factor* (MF) criterion for match i as:

$$MF(i) = \frac{|Q_p(i) \cup Q_q(i) - Q_p(i) \cap Q_q(i)|}{|Q_p(i) \cup Q_q(i)|} \times 100 \tag{3.5}$$

For each match, MF represents the difference in the neighbor points between $\mathcal{P}$ and $\mathcal{I}$ as a percentage. Ideally, we expect that for all the matches $MF = 0$, which implies that there is no difference in the neighbors of each match during a deformation. However, in practice, there are two reasons which rather put MF values in a range from 0 to 100: the presence of mismatches and variations in triangulation. The presence of mismatches can affect the value of MF in two ways. First, when the match point i in $\mathcal{I}$ is a mismatch and thus located in a wrong location. And second, when the match point i in $\mathcal{I}$ is a correct match but one, several, or all of its neighbors are mismatches. Both of these cases result in different neighbors in $\mathcal{I}$ in comparison to $\mathcal{P}$. As for the two triangulations, it should be noted that even in the absence of mismatches, the neighborhood structures in T_p and T_q do not necessarily coincide. This is because of surface deformation, change in viewpoint, and occlusions.

Calculating MF for all the matches, we can have a fair estimation regarding the state of the matches. The lower values of $MF(i)$ indicate that the match i is surrounded by similar matches in $\mathcal{P}$ and $\mathcal{I}$ and has a higher probability to be placed in its correct location and thus be a correct match. On the contrary, the higher values of $MF(i)$ can stem from the wrong location of the match i in comparison to its neighbors which strengthens the possibility of it being a mismatch. The basic idea in this step is to form $C_{p_s} \leftrightarrow C_{q_s}$ by selecting pairs of highly probable correct matches $p_s \leftrightarrow q_s$. This is done by choosing the matches with lower values of MF. We examined the validity of this reasoning by evaluating three different synthetic data experiments, each with 1000 matches and different rates of correct matches (30%, 60%, and 90%). Figure 3.6 shows the histogram of MF for each case. We observe that the dispersion of MF spans a wider range as the value of the correct match rate grows. For higher numbers of correct matches, there are more similarities in the neighbor lists of each match and, consequently, MF decreases. Furthermore, regardless of the values of the correct match rate, the majority of the mismatches are accumulated in the top bins of the graphs that correspond to higher values of MF. This is shown in more detail for the case with the correct match percentage of 30% by expanding the last two bins of the graph in Figure 3.6.a. This validates our prior reasoning that by selecting the matches with MF below a certain threshold MF_{th}, we can have a set of matches which are highly probable to be correct. To quantify the appropriateness of this selection, we define two criteria, based on the following two quantities. The first quantity is n_s, which is the percentage of the selected matches compared to the total number of matches:

$$n_s = \frac{|C_s|}{N_m} \times 100, \tag{3.6}$$

where $C_s = \{(p_i, q_i) \,|\, i \in \widehat{R}\}$ is the set of selected matches. The second quantity is AoS, which is the Accuracy of Selection, defined as:

$$AoS = \frac{|C_s \cap S_{in}|}{|C_s|} \times 100. \tag{3.7}$$

Our goal is to choose the value of MF_{th} in the way that we have both of these criteria to be as high as possible, which means selecting a high percentage of matches with high accuracy. However, practically, these two criteria work in reverse. By choosing a higher value for MF_{th}, more matches are selected (higher n_s) but with less accuracy (lower AoS) and vice versa. In order to choose the proper value for MF_{th}, we analyzed the behavior of these two criteria for a series of synthetic data experiments.

We consider three scenarios for these experiments based on the number of matches, i.e., Dense, Moderate, and Sparse with in turn 1000, 200, and 50 total number of matches. The experiments were done in a wide range of correct match percentages (10% to 100%) for each scenario. Two different values of the criterion MF_{th} were studied; *mean* and $0.9 \times mean$ where *mean* is the mean of all MF values in each experiment. The results are presented in Figure 3.7.a and 3.7.b. Each point in the graph is the average result of 1000 trials. The first point that should be noted here is that, generally, the proposed match selection method in this step is more reliable as the number of total matches grows. This can be deduced by comparing the higher values of AoS in the Dense case with the ones in the Moderate and Sparse cases. As for choosing MF_{th}, it should be noted that setting $MF_{th} = 0.9 \times mean$ leads to higher values of AoS in comparison to the case with $MF_{th} = mean$. Nevertheless, as shown in Figure 3.7.a, this sacrifices a high percentage of matches by dropping n_s significantly, which is undesirable. Hence, in this step, we choose *mean* as the value of MF_{th} and form $\widehat{R}$ as the set of indices of probable correct matches. While this choice implies a higher number of selected mismatches (lower AoS), we note that these mismatches can be removed in *Step II*.

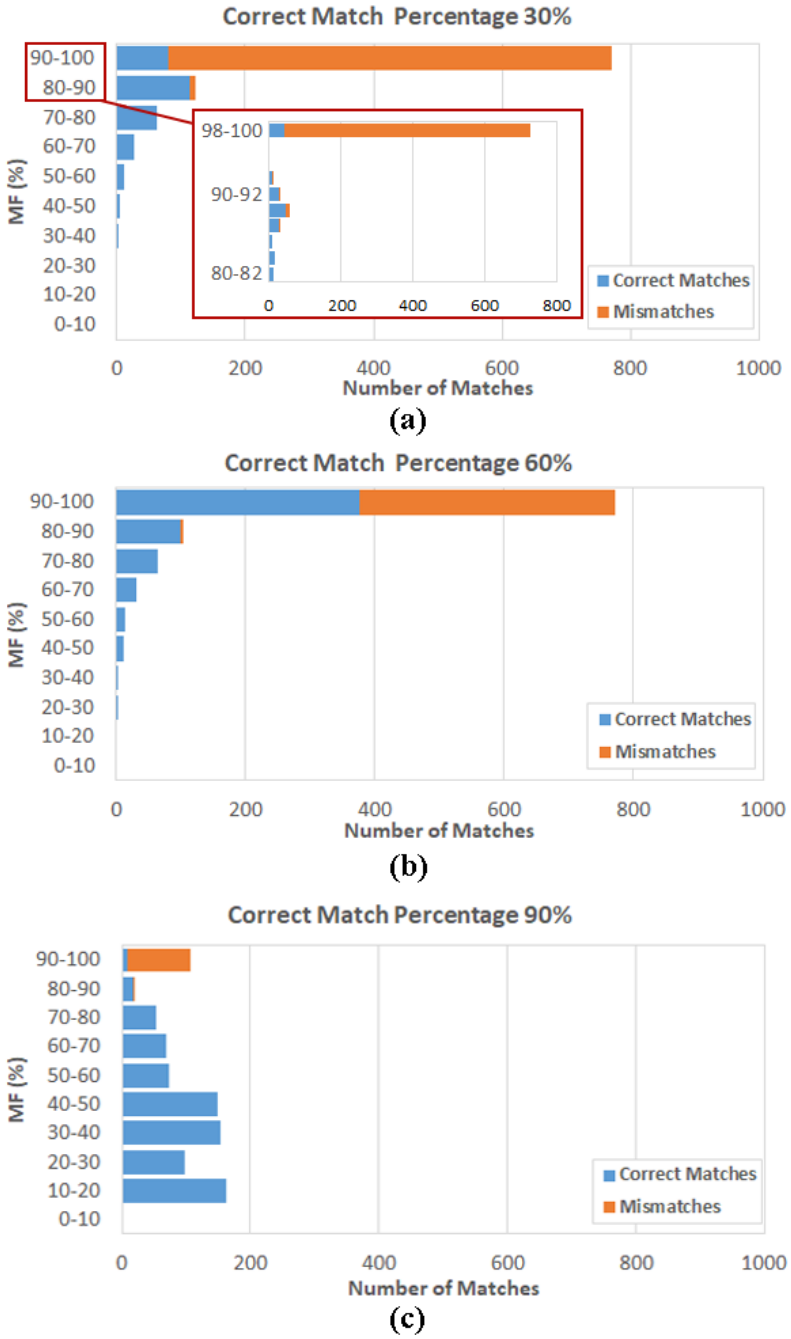


FIGURE 3.6: Histogram of MF values for three sample synthetic data experiments with 1000 matches and 30%, 60% and 90% of correct matches.

As the final operation in this step, we estimate the warp W_1 between $\mathcal{P}$ and $\mathcal{I}$ using the selected matches $C_{p_s} \leftrightarrow C_{q_s}$. We then exploit this warp to transfer M to $\mathcal{I}$. We call this new mesh $\widehat{M}_1$. As can be seen in the third column of Figure 3.5, the mesh $\widehat{M}_1$ (shown in orange) may not be totally faithful to the deformation of M_G in $\mathcal{I}$, which is due to the inaccuracies in the calculation of the warp W_1. This stems from two main reasons; the existence of mismatches in our selection (shown as red dots), and the insufficient number of correct matches in some areas. In the next step, we exploit the transferred mesh $\widehat{M}_1$ to remove the possible remaining mismatches from the selected matches.

3.4.2.2 Step II – Removing mismatches from the list of selected matches

We remove the possible mismatches from the selected matches $C_{p_s} \leftrightarrow C_{q_s}$. We first form the set $C_{\hat{q}_s}$ by transferring C_{p_s} to $\mathcal{I}$. This is done by finding the barycentric coordinates of each selected match $p_{s_i} \in C_{p_s}$ with respect to M and applying them on the transferred 2D mesh $\widehat{M}_1$ from *Step I*. We then use the following decision criterion to identify and remove possible mismatches one by one from the selected matches $C_{p_s} \leftrightarrow C_{q_s}$:

$$\Big| d_2(i) - \text{median}\big(\{d_2(j)\}\big) \Big| \geqslant 2.5\,\text{MAD}, \tag{3.8}$$

where $d_2(i) = \|\hat{q}_{s_i} - q_{s_i}\|$ with $i \in \widehat{R}$. MAD (Median of Absolute Deviations from Median) is calculated as:

$$\text{MAD} = k\ \text{median}\Big(\Big\{ \Big| d_2(i) - \text{median}\big(\{d_2(j)\}\big) \Big| \Big\} \Big), \tag{3.9}$$

where $k = 1.4826$ is a constant number. The values of d_2 are relatively larger for mismatches in comparison to correct matches. This stems from two reasons. First, the small percentage of mismatches compared to the great majority of correct matches coming from *Step I* and thus smaller influence of mismatches in the estimation of warp W_1. Second, the inconsistent location of mismatches in $\mathcal{P}$ and $\mathcal{I}$. The decision criterion in equation (3.8) is chosen due to the distribution type of d_2, with the presence of just a small percentage of large values among the majority of small values. Figure 3.7.c and d illustrate the result of this step. As can be seen, unlike the previous strategy of choosing a smaller MF_{th}, this method results in improvement of AoS without losing a considerable percentage of selected matches. This can be clearly observed by comparing n_s in Figures 3.7.a and c.

As the last operation in this step, warp W_2 is calculated using the purified selected matches $C_{p_s} \leftrightarrow C_{q_s}$. This warp is then used to transfer M to the image space and form $\widehat{M}_2$. The result of removing possible mismatches in this step along with the transferred mesh $\widehat{M}_2$ are shown in the fourth column of Figure 3.5. As can be observed, in comparison to $\widehat{M}_1$, $\widehat{M}_2$ has a better compliance to M_G.

3.4.2.3 Step III – Extracting mismatches from the list of all the matches

In this step, we exploit the transferred mesh $\widehat{M}_2$ to extract the mismatches $O_p \leftrightarrow O_q$ from the total matches $C_p \leftrightarrow C_q$. The process is similar to *Step II* except that this time all of the matches are checked. We first transfer the template match points C_p to the image space and form the set $C_{\hat{q}}$. This is done by calculating barycentric coordinates of all the match points C_p with respect to M and applying them on the new transferred mesh $\widehat{M}_2$. We define the following decision criterion to detect and remove mismatches:

$$d_3(i) = \|\hat{q}_i - q_i\| \geqslant d_{3_{th}} \tag{3.10}$$

Unlike *Step II* where we used the MAD criterion to remove just a small rate of mismatches, this time we use a constant threshold $d_{3_{th}}$. This is due to the higher percentage of mismatches compared to *Step II*. In order to make this distinction method more robust, we consider $d_{3_{th}}$ as the multiplication of a sample length l_s and a constant coefficient α_s. The sample length l_s is a measure of the size of the object in the image in pixels and is calculated as the average distance between all the mesh points in the transferred mesh $\widehat{M}_2$. To choose a proper value for the constant coefficient α_s, a series of synthetic data experiments with the same three scenarios as before (Dense, Moderate, and Sparse) and four different correct match rates was performed. The results are presented as ROC (Receiver Operating Characteristic) curves in Figure 3.8.a-c. Each point represents the average TPR

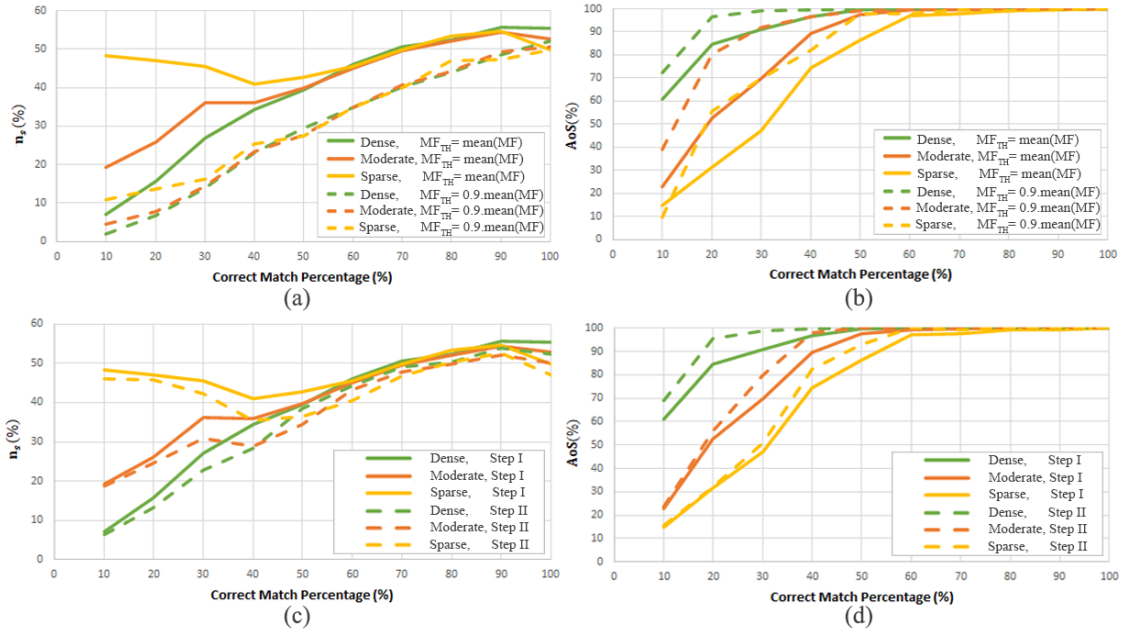


FIGURE 3.7: Results of applying the first two steps of the algorithm `myNeighbor` in synthetic data experiments in three different scenarios; Dense (1000 matches), Moderate (200 matches), and Sparse (50 matches). Each curve is the average result of 1000 trials. The first row gives n_s and AoS from *Step I* for two different values of MF_{th}. The second row gives the results of *Step II* in comparison to the results of *Step I* with $MF_{th} = \text{mean}(MF)$.

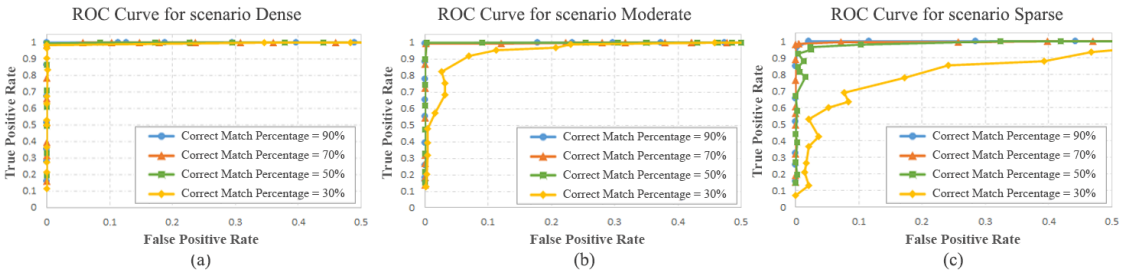


FIGURE 3.8: ROC curves resulting from the algorithm `myNeighbor` in synthetic data experiments in three scenarios; Dense (1000 matches), Moderate (200 matches), and Sparse (50 matches). Each point is the average result of 1000 trials calculated with a specific value of $d_{3_{th}}$.

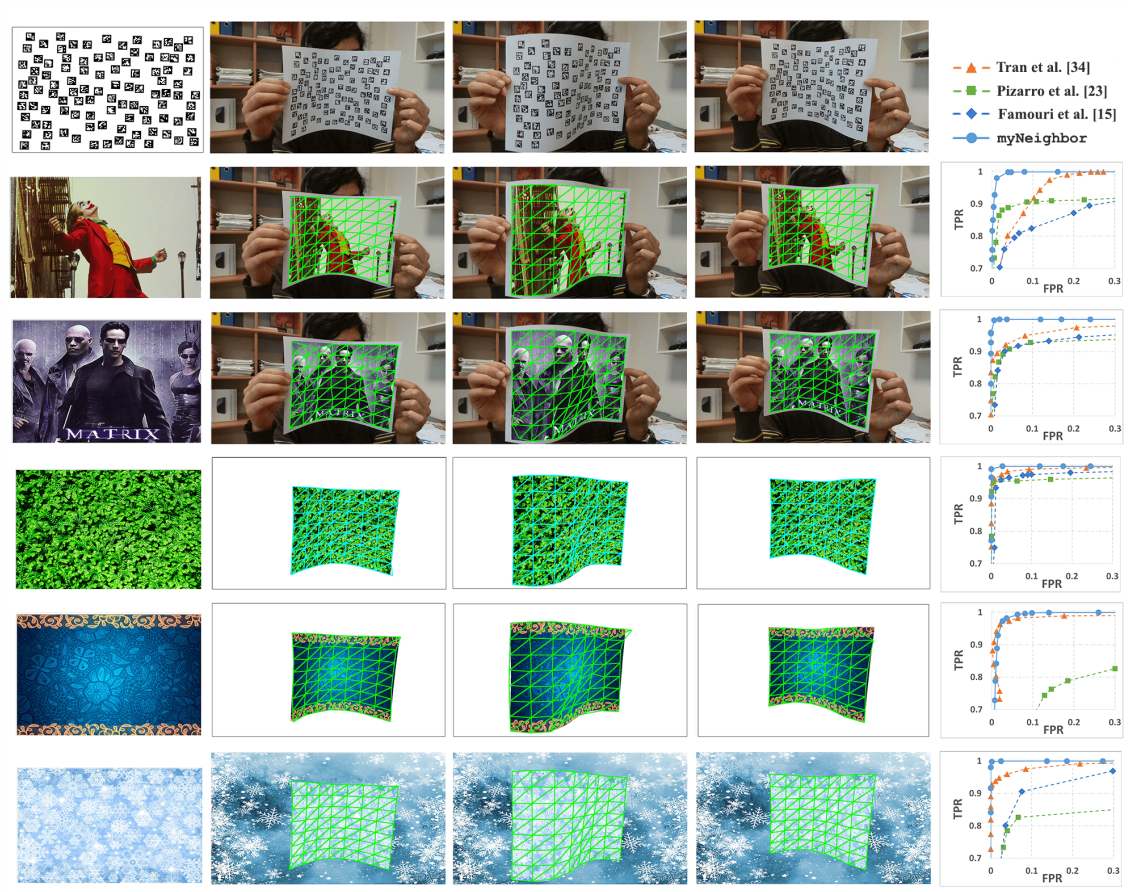


FIGURE 3.9: Performance evaluation of our mismatch removal method myNeighbor in comparison to the state-of-the-art methods using the FREX protocol. The first row shows the Aruco template and three selected images (14, 47, 60) of the deformation of the printed Aruco template. The following rows show five datasets of generated scenes with the texturemap in the first column, three generated images corresponding to the first row in the next columns, and the ROC curves of the mismatch removal algorithms in the last column. For each of the images $\widehat{M}_3$ from myNeighbor is overlaid.

(True Positive Rate) versus the average FPR (False Positive Rate) computed in 1000 trials using a specific value of α_s in the range of $[0, 1]$. TPR is calculated as the number of selected true mismatches over the number of all true mismatches, and FPR is calculated as the number of true correct matches mistakenly selected as mismatches over the number of all true correct matches. Ideally, all the mismatches should be discarded (TPR=100%) without discarding any correct matches (FPR=0%). Hence, the most favorable α_s in a single ROC curve is the one that results in the maximum possible TPR leaving the FPR below a reasonable value. We choose $\alpha_s = 0.15$ which keeps TPR above 90% while FPR remains below 10% for most of the cases. The last column of Figure 3.5 illustrates the estimated correct matches (in green) and the estimated mismatches (in red) for each case. We also use the estimated correct matches to estimate warp W_3 and transfer M to $\mathcal{I}$ and form $\widehat{M}_3$ (shown in cyan). As can be seen, there is a high compliance between $\widehat{M}_3$ and M_G. It should be noted that estimating W_3 and $\widehat{M}_3$ is not necessary in myNeighbor and we merely estimate them just to visually present the effectiveness of the algorithm in removing the mismatches. However, considering myNeighbor as a step in ROBUSfT, due to the fact that the final estimated correct matches are passed from this step to *Step 3* of ROBUSfT which is warping, W_3 and $\widehat{M}_3$ can also represent W and $\widehat{M}$ in the warping step,

respectively.

3.4.3 Mismatch removal results

In this section, we demonstrate the efficiency of myNeighbor by evaluating its performance through various tests. We first compare the results of the algorithm with the state-of-the-art algorithms in the literature by testing them through FREX. The experiment includes 60 frames of continuous deformation of the Aruco template in front of the camera. Five datasets were generated in this experiment each with an arbitrary texture with a challenging pattern. Three different types of backgrounds were also considered for these five cases, specifically two original backgrounds, two white backgrounds, and a background with a pattern similar to one of the texturemaps. We apply all the mismatch removal algorithms on all datasets. For each dataset, the corresponding arbitrary texture was used as the texturemap for the mismatch removal algorithms. The matches between the texturemap and each image of the dataset are extracted using SIFT. The results are presented in Figure 3.9. The first row illustrates the Aruco template and also three selected original images of its deformation in front of the camera. The lower rows represent the five datasets generated by FREX. Each row shows the arbitrary texture of the dataset in the first column, the three selected generated images, and eventually the resulting ROC curves for all the mismatch removal algorithms on the dataset. In the ROC curves, for a certain algorithm and a certain dataset, each point is the average value of TPR and FPR over all 60 images of that dataset using a specific value for the threshold used in the algorithm. As can be seen, in all cases, our algorithm outperforms the other algorithms. In order to show the performance of our algorithm visually, for each dataset, we overlaid $\widehat{M}_3$ for the three selected frames. As can be observed, the transferred meshes are visually well-aligned to the 2D deformed shape of the object. In some cases, a small number of irregularities can be observed in certain areas (for example in the Matrix poster). This is because of the presence of a small number of mismatches in our list of estimated correct matches and the lack of matches in those areas. As for comparing the execution speed of different mismatch removal algorithms, the process run-times for all the frames of all datasets were averaged and tabulated in Table 3.1. It shows that our algorithm is faster than the others. It should be however noted that our algorithm is implemented in C++ while the others are in Matlab.

After validating the efficiency of myNeighbor in comparison to the state-of-the-art algorithms in the literature, we evaluate its performance in real cases. To this end, we applied our algorithm to four real deforming objects as shown in Figure 3.10. We chose these cases in such a way that each one is challenging in a special way. The cases include a cushion with non-smooth surface and severe deformation, a Spiderman poster deformed in a scene with background covered with almost the same posters, a shoe sole with an almost repetitive texture, and a shirt with elastic deformation. The texturemaps are shown in the first column of Figure 3.10. The second to fourth columns show the results of *Step I* to *Step III* of myNeighbor. In each step, the alignment of the corresponding transferred mesh to the 2D shape of the deforming object can be considered as an indication of the correctness and abundance of the estimated correct matches. Like in the synthetic data experiments, this alignment improves progressively in different steps of our algorithm. One point that should be noted here is that the shirt (the last case in Figure 3.10) is elastic. We exert a non-isometric deformation on it by pulling from both sides, and myNeighbor still works. This is due to the fact that we did not make any assumption regarding isometry. In fact, the only assumption that we made is the preservation of neighborhood structure in the deforming object. As a result, myNeighbor also works with non-isometric deformations which preserve neighborhood structure.

Method	Average run-time (s)
`myNeighbor`	0.0139
Tran et al. [TCC+12]	0.0206
Pizarro et al. [PB12]	1.8925
Famouri et al. [FBA18]	0.0171

TABLE 3.1: Comparison of the average run-time of the mismatch removal algorithms for processing all the images of all the datasets.

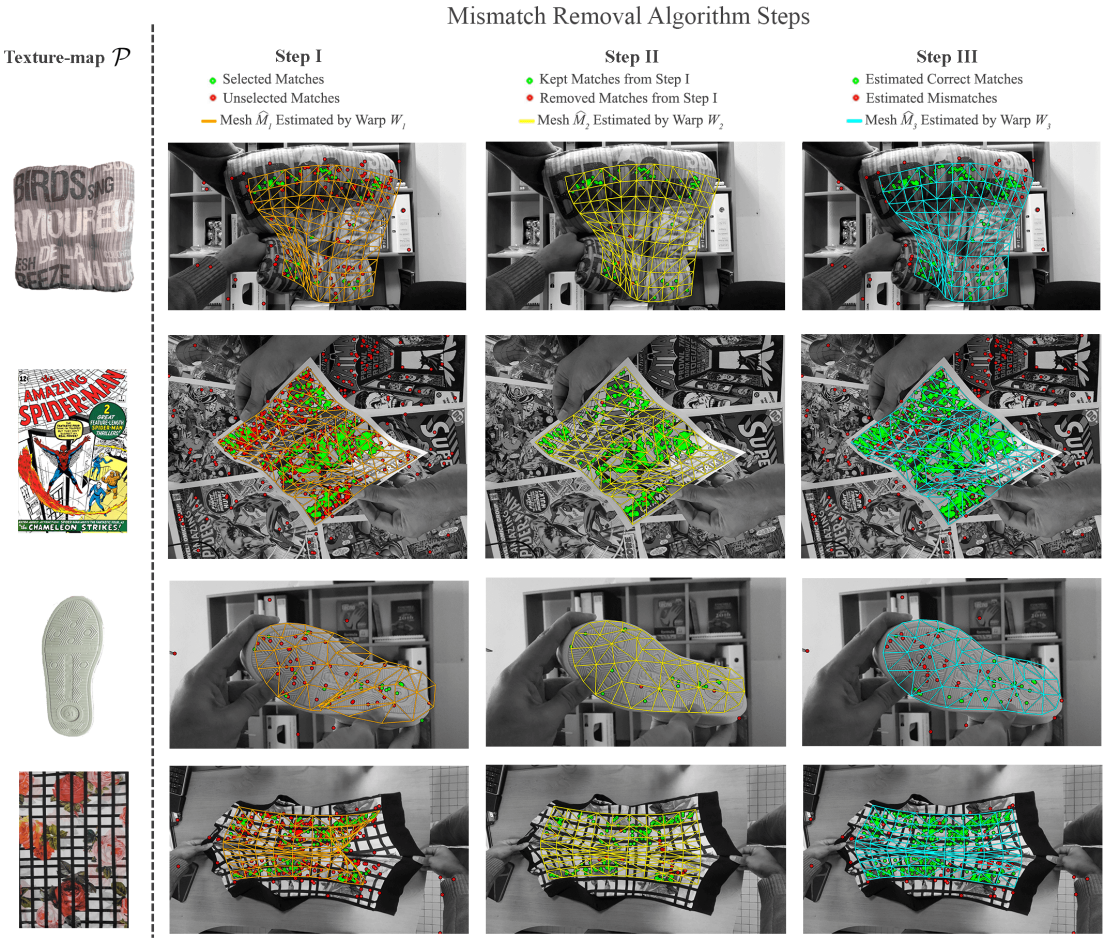


FIGURE 3.10: Applying `myNeighbor` on four real cases: a cushion, a Spiderman poster, a shoe sole, and an elastic shirt. The first column shows the texturemaps. The second column shows *Step I*. All the matches are shown in this column while the selected matches in *Step I* are shown in green. These selected matches are transferred to column three that shows *Step II*. In this column, those matches which are chosen as possible mismatches are shown in red. The last column is the distinction between the estimated correct matches (in green) and the estimated mismatches (in red) in *Step III*. The meshes $\widehat{M}_1$, $\widehat{M}_2$, and $\widehat{M}_3$ are overlaid to illustrate the computed warps.

3.5 Experimental Results

We evaluate the performance of `ROBUSfT` on different deforming objects in various conditions. We divide this section into two main parts; first, comparing the results with the state-of-the-art methods and then evaluating `ROBUSfT` in several other challenging cases.

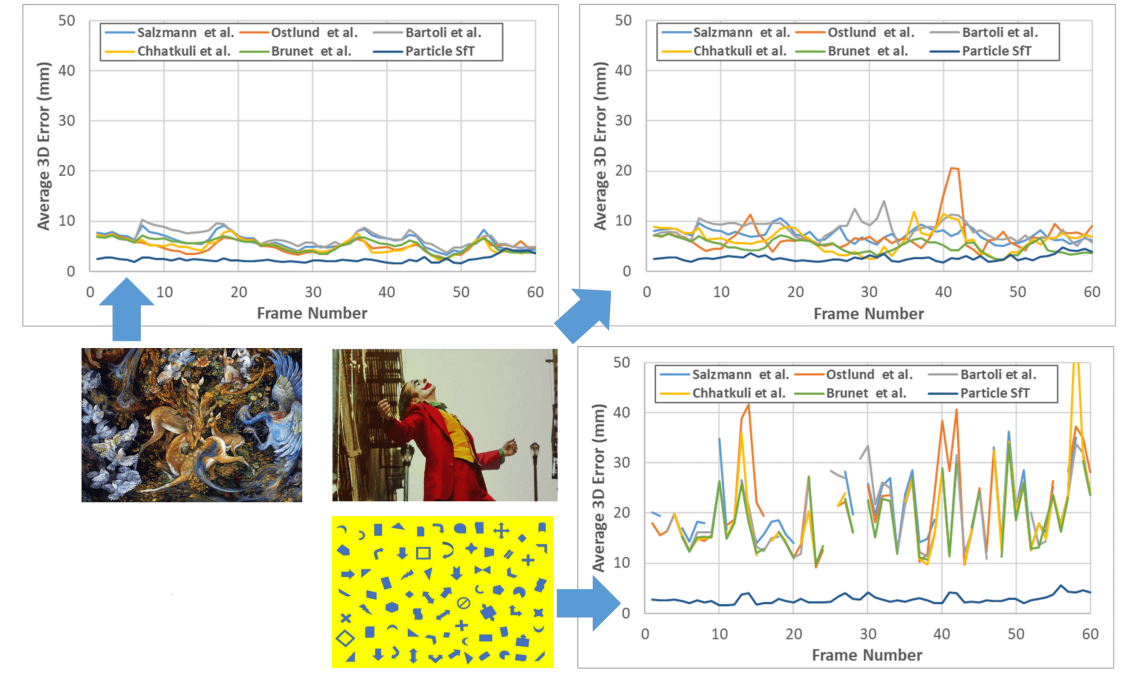


FIGURE 3.11: Comparing the accuracy of the 3D shape inference methods with Particle-SfT with three datasets obtained by FREX. The 3D shape inference methods are Brunet et al. [BBH14], Chhatkuli et al. [CPB14], Bartoli et al. [BGC+15], Ostlund et al. [ÖVNF12], and Salzmann et al. [SF10].

3.5.1 Comparison to the state-of-the-art methods

We compare ROBUSfT with the state-of-the-art methods through two different tests. The first test is conducted among the shape inference methods (G1). The second test is carried out among the integrated methods (G2) and the DNN-based methods (G3).

Comparison to G1 methods. We use FREX to conduct the first test. To this end, the same 60 images of deforming Aruco marker paper sheet are used. We create three different datasets using three arbitrary texturemaps and apply a white background to all the scenes. The arbitrary texturemaps include a painting, the Joker poster, and a paper sheet filled with basic geometric shapes. These images are shown in Figure 3.11. In each dataset, we compare the result of the last two steps of ROBUSfT (warp estimation and 3D shape inference) with five other shape inference methods from Brunet et al. [BBH14], Chhatkuli et al. [CPB14], Bartoli et al. [BGC+15], Ostlund et al. [ÖVNF12], and Salzmann et al. [SF10]. A similar comparison was made in [ÖB17] on another dataset. However, in [ÖB17], a random 3D shape was used as the initial guess for Particle-SfT algorithm in each image of the video; in contrast, we use the 3D inferred shape of the object in each image as the initial guess for the next image. In each dataset, the matches between $\mathcal{P}$ and each image are extracted using SIFT. We then separate the correct matches and use them as the input for all the methods. If required by a shape inference method, a BBS warp is estimated based on these correct matches and used as the input to that shape inference method. The results for all three datasets are presented in Figure 3.11 as the average 3D error between the 3D inferred shapes and the ground truth. As can be observed, Particle-SfT provides the lowest value of 3D error in comparison to the other methods. This is more apparent in the datasets with lower number of matches. In the last dataset, there are several discontinuities in the 3D error graph of state-of-the-art methods. This

is due to the failure of shape inference in those images of the video by those methods. Particle-SfT, however, succeeds to infer the 3D shape of the object in all of the images with a reasonable error.

Comparison to G2 and G3 methods. For the second test, we ran ROBUSfT on the public dataset provided in [VSSF12]. The dataset includes the 2D correspondences as well as 3D Kinect data of 193 consecutive images of a deforming paper. The paper is planar and no occlusion appears in the series of images. We compared our results with the integrated and DNN-based methods. This is shown in Figures 3.12 and 3.13, and Table 3.2. The integrated methods include Famouri et al. [FBA18] and Ngo et al. [NÖF15]. Here, we use the results of applying these methods on all the dataset frames provided in [FBA18]. As can be observed in Figure 3.12, ROBUSfT is more precise in most of the frames. It is also faster than the compared methods. It should be noted that ROBUSfT used directly images as the

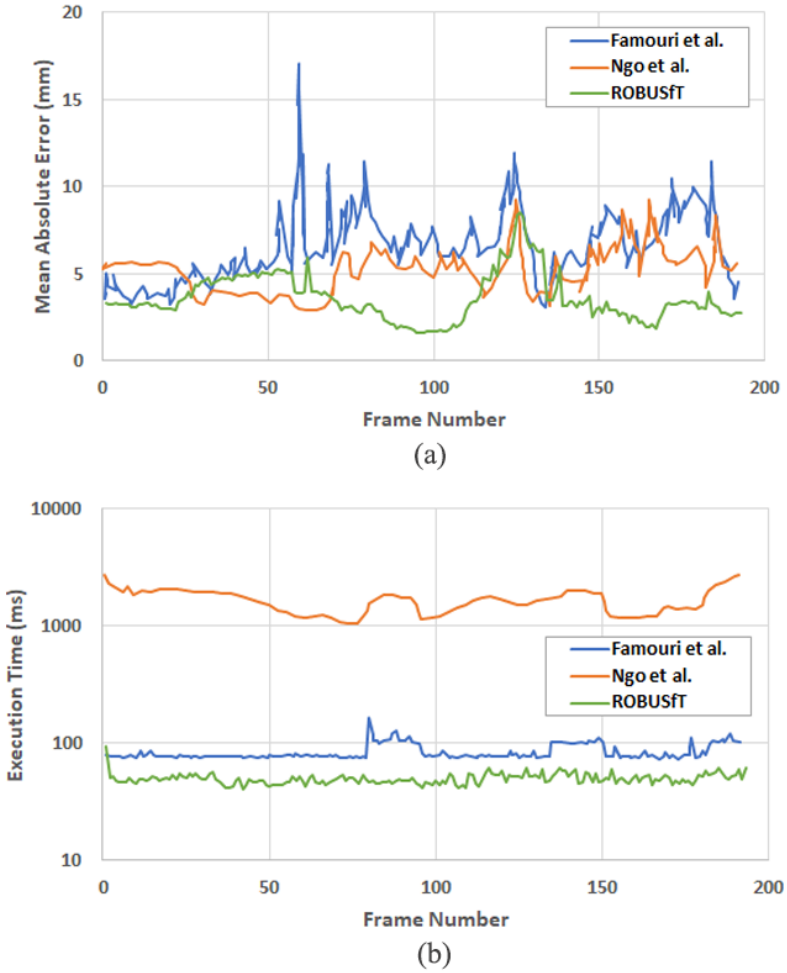


FIGURE 3.12: Comparison between the results of applying ROBUSfT and the integrated methods, i.e., Famouri et al. [FBA18] and Ngo et al [NÖF15] on the public dataset provided in [VSSF12]. (a) Mean absolute 3D error between the inferred shape and the ground truth. (b) Execution time in milliseconds.

input and covered the whole process from extracting keypoints to 3D shape inference. In contrast, the two integrated methods used the already available correspondences in the dataset. Next, we compare ROBUSfT with two deep object-generic monocular reconstruction methods, i.e., DenseDepth [AW18] and BTS [LHKS19]. The importance of these

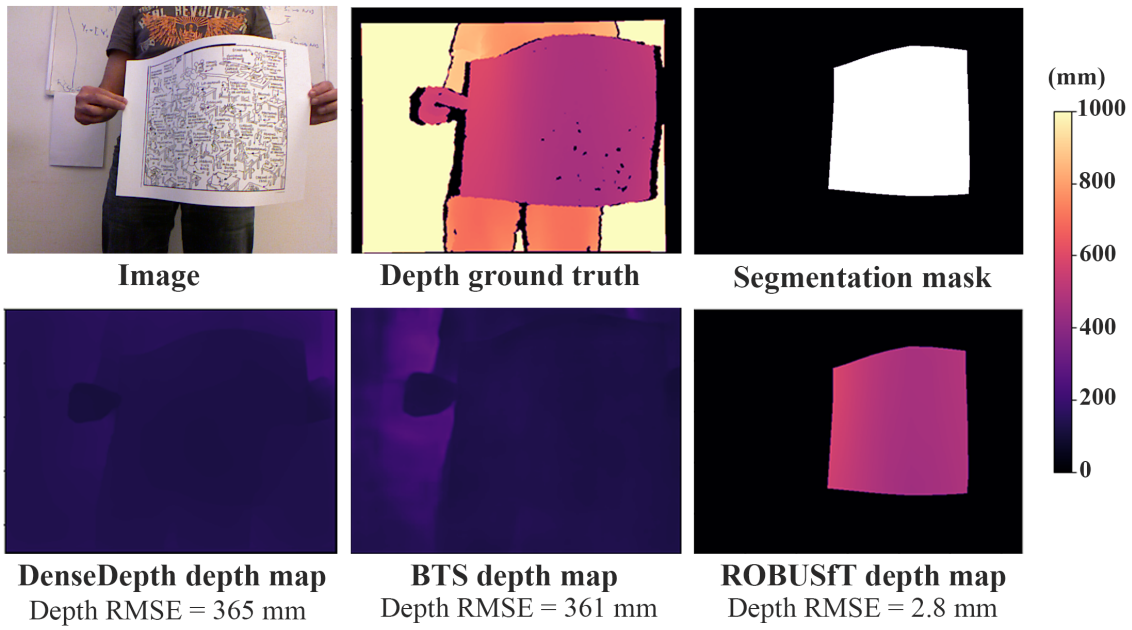


FIGURE 3.13: Comparison between the depth map resulted from applying ROBUSfT and the deep object-generic monocular reconstruction methods, i.e., DenseDepth [AW18] and BTS [LHKS19], on one frame of the dataset provided in [VSSF12]. To make a fair comparison, the depth RMSE is calculated using the pixels from the segmentation mask. This segmentation mask is estimated in the registration step of ROBUSfT.

methods is that, similar to ROBUSfT, they do not need to be trained for any new object. For both methods, we use the model pre-trained with the NYUDepth dataset [SHKF12], which is a collection of RGB-D images from indoor scenes with various common objects. We compare the resulting depth maps from ROBUSfT and these two methods. This comparison is shown for one frame of our test dataset in Figure 3.13. To make a fair comparison, the depth error is computed only for the object's surface and not the whole scene. This is done by considering the pixels from the segmentation mask which is estimated in the registration step of ROBUSfT. As can be observed, the deep object-generic monocular reconstruction methods are unable to predict the depth of the object. This is also discussed in [FJPCP+22] as they showed that these methods require fine-tuning to be able to estimate the deformation of specific objects. Another issue with these methods is the lack of registration. In other words, these methods only provide the depth map and not the shape of the object in 3D space. Consequently, an additional registration method should be used to register the object to the depth map so that the 3D shape of the object can be estimated. Our last comparison is with the DNN-based SfT methods. In contrast to the deep object-generic monocular reconstruction methods, DNN-based SfT methods are trained for specific objects using synthetic and real datasets and are capable of handling both registration and reconstruction. The compared methods include HDM-Net [GSVS18], IsMo-GAN [SGTS19], DeepSfT [FJPCP+22], and RRNet [FJPCP+21]. The comparison is shown in Table 3.2. We compare the average errors of applying these methods on 50 frames of the dataset [VSSF12] presented in [FJPCP+21] to the average error of applying ROBUSfT on the same frames. As can be seen, ROBUSfT is more precise than DNN-based SfT methods. One point that should be noted here is that these results are achieved while, as it is explained in [FJPCP+21], the DNN-based SfT methods are already trained on this dataset. ROBUSfT, however, does not need any prior training. In terms of execution speed, several of the DNN-based SfT methods are faster than ROBUSfT. We,

Method	Average error (mm)	Average execution time (ms)
ROBUSfT	3.56	50
HDM-Net [GSVS18]	17.92	40
IsMo-GAN [SGTS19]	15.91	96
DeepSfT [FJPCP+22]	6.97	49
RRNet [FJPCP+21]	8.63	16

TABLE 3.2: Comparison between the average 3D error resulted from applying ROBUSfT and the DNN-based SfT methods, i.e., HDM-Net [GSVS18], IsMo-GAN [SGTS19], DeepSfT [FJPCP+22], and RRNet [FJPCP+21] on 50 frames of the public dataset provided in [VSSF12].

however, note that in this test, we use a serial CPU-GPU architecture instead of a parallel one. This is done to make sure that the captured image that we analyze and the ground truth that we compare to are for the same image. This consequently reduces the execution speed of our code compared to the parallel architecture. In conclusion, considering generalizability, efficiency, and execution speed, ROBUSfT achieves better performance in comparison to the state-of-the-art methods.

3.5.2 Evaluation of ROBUSfT

Evaluation on daily objects. We first evaluate the efficiency of ROBUSfT in three real cases. These cases are shown in Figure 3.14. The tested objects are a Spiderman poster, a chopping mat, and a t-shirt. In each case, the object is deformed in front of a 3D camera with which we capture both RGB image and the depth of each point on the object. We use the measured depth as ground truth for evaluating the reconstructed 3D shape. We use the Intel RealSense D435 depth camera and built-in libraries for aligning the depth map to the RGB image. For each case, four images of the experiment are shown in Figure 3.14. In the first case, we set the resolution of the camera to 640×480. In the second and third cases, we increased it to 1280×720 due to the insufficient number of detected keypoints using the previous resolution. Below each image, the reconstructed 3D shape of the deforming object along with the 3D coordinates of the estimated correct matches (red particles) as well as their ground truth (green particles) are shown. The 3D coordinates of the estimated correct matches are estimated by calculating their barycentric coordinates in $\mathcal{P}$ with respect to M and applying these coordinates on the 3D reconstructed mesh of the object. The number written below each frame is the median distance between the reconstructed 3D coordinates of the estimated correct matches and their ground truth. The median is chosen due to the probable existence of mismatches among the list of estimated correct matches. In 3D space, the ground truth of these mismatches can be located in the background and not on the object itself. This significantly increases the 3D shape error. Using the median gives a better estimate of the 3D shape error considering the existence of this small percentage of mismatches with large 3D errors.

As can be observed, the pipeline succeeds to infer the 3D shape of the object in all of the cases. This success is more visible in the second and third cases due to the relative scarcity of keypoints and existence of repetition in their patterns. Regarding the Spiderman poster case, it should be noted that there are self-occlusions in the first and third illustrated images. In these images, the 3D shape of the object in the occluded areas is estimated by the deformation constraints implemented in Particle-SfT. These constraints preserve the geodesic distance between each pair of mesh points as its initial value in M_T. Regarding the runtime, using the parallel architecture and 640×480 captured frames as the input (as in the Spiderman poster case), the execution speed reaches 30 fps.

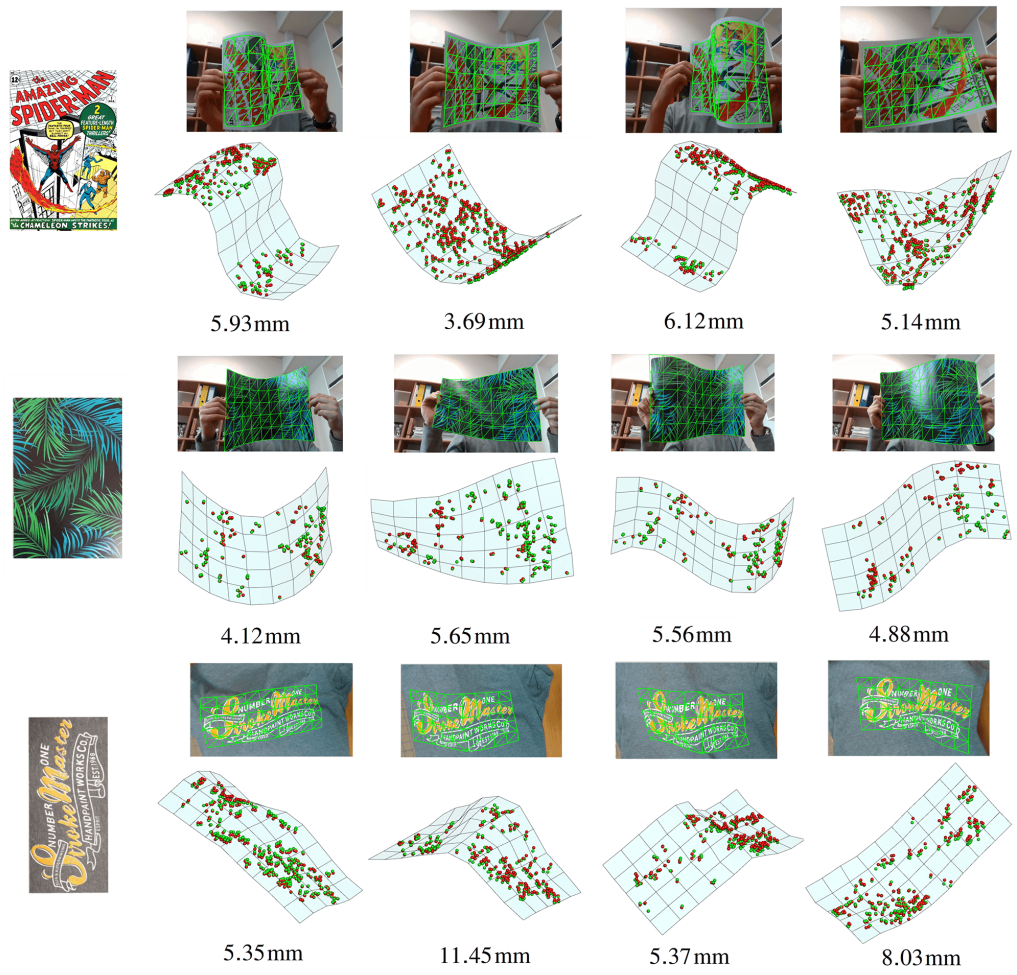


FIGURE 3.14: Evaluating ROBUSfT in three real data experiments; a Spiderman poster, a chopping mat, and a t-shirt. The texturemaps of the templates are shown in the first column. For each case, four images are shown. Below each frame, the reconstructed 3D shape of the deforming object with the estimated 3D coordinates of the estimated correct matches (red particles) as well as their ground truth (green particles) are shown. The 2D projections of the 3D inferred shapes are also overlaid on the image. For each image, the median Euclidean distance between the estimated 3D coordinates of the estimated correct matches and their ground truth is given below the reconstructed shape.

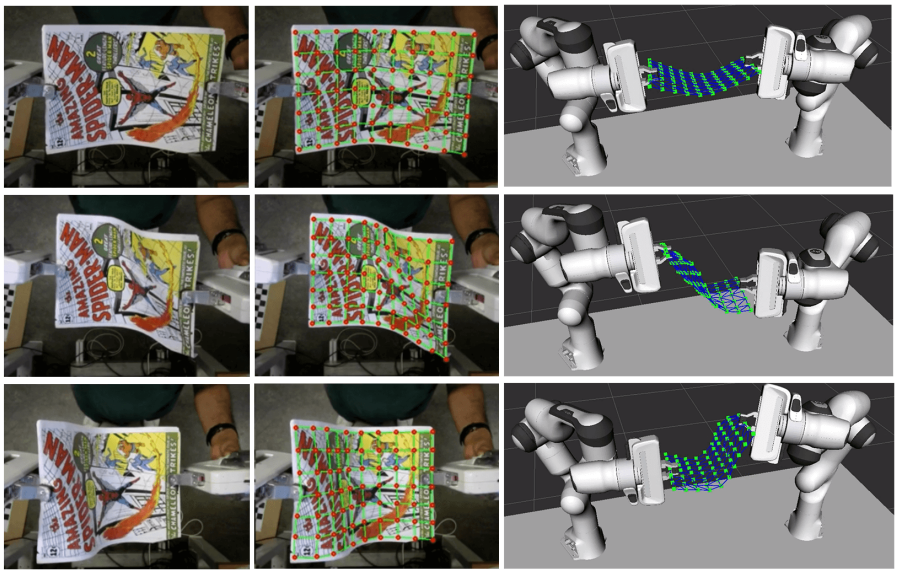


FIGURE 3.15: Evaluating ROBUSfT in a real data experiment with two robotic arms; soft constraints are applied to bind the constrained mesh points to the grippers. Each row shows three images: the original camera view, the projection of the 3D reconstructed mesh on the camera view, and the 3D reconstructed mesh with the robots in the RViz environment.

Evaluation on a robotic use case. The last experiment is a practical use case with robots. The experiment aims at highlighting the advantage of using known 3D coordinates in ROBUSfT. As mentioned in *Step 4* and shown in Figure 3.15, these known coordinates are an optional input to the last step of ROBUSfT. Their usage can increase the robustness of the tracking process. The setup of this experiment is the same as in Chapter 4 of this thesis, where we applied ROBUSfT in a robotic case, specifically, controlling the shape of deformable objects. The setup consists of two robotic arms grasping and manipulating the Spiderman poster from both sides and a top camera facing the manipulation area. The 3D positions of the two robotic grippers are known in camera coordinates thanks to the known pose of each gripper in the robots' coordinate frames and also the external calibration between the robots and the camera. For each gripper, we consider the closest mesh point to the gripper as a constrained mesh point. These mesh points should be bound to their corresponding gripper and move with it. This binding is performed using a soft constraint. In this soft constraint, for each gripper, a sphere with a small radius centered at the gripper's 3D position is considered. Then, in each iteration of Particle-SfT, if the corresponding mesh point is outside this sphere, it will be absorbed to the closest point on the sphere surface. This soft constraint has two main advantages over rigidly binding the constrained mesh points to the grippers: first, it lets the position-based dynamic equations in Particle-SfT that preserve the distances between the mesh points be applied on the constrained mesh points, which leads to a smoother reconstructed shape. Second, it allows us to cope with small possible errors in robot-camera calibration. In fact, a wrong robot-camera calibration leads to a wrong transfer of the grippers' 3D coordinates to the camera coordinate frame which eventually results in wrong coordinates of the constrained mesh points. By using the soft constraint and considering a sphere rather than a rigid bind, we give a certain degree of flexibility to the constrained mesh points to move in close proximity to the gripper's coordinates. This can compensate for slightly inaccurate coordinates of the grippers.

3.6 Conclusion and future work

We have proposed ROBUSfT, a publicly available C++ library that can effectively track the 3D shape of an isometrically deforming object using a monocular 2D camera. ROBUSfT outperforms the state-of-the-art methods. The proposed pipeline addresses the well-known challenges in this area. These challenges include ambiguities in inferring the 3D shape of the deforming object from a single 2D image and real-time implementation. We have introduced myNeighbor, a novel mismatch removal algorithm for deforming objects, which works based on the preservation of the neighborhood structure of matches. We validated the efficiency of myNeighbor in comparison to the state-of-the-art algorithms in numerous experiments. In order to compare ROBUSfT and myNeighbor with the state-of-the-art methods in the literature, we have presented a novel type of experimental protocol called FREX (Fake Realistic Experiment). This protocol is executed once, but it provides a large number of resulting scenes of an isometrically deforming object in various conditions with 2D and 3D ground truth. This collection can be used to evaluate, compare, and validate algorithms regarding isometrically deforming objects. In addition, the provided 2D and 3D ground truth may be used for training learning-based algorithms. In contrast to other artificially made scenes of an isometrically deforming surface, the generated images in our protocol are the result of real isometric deformations.

Possible directions for future work include exploiting the silhouette of the object in the image for improving 3D shape inference in challenging cases such as weakly-textured objects and extending ROBUSfT to volumetric objects.

Chapter 4

As-Rigid-As-Possible Shape Servoing

4.1 Introduction

In the previous chapter, we proposed an SfT method for shape tracking of isometrically deforming thin-shell DOs. In this chapter, we focus on shape servoing. Specifically, we propose a shape servoing approach based on ARAP for thin-shell DOs. In this approach, we use ARAP to estimate a deformation Jacobian. We then propose a control scheme based on this Jacobian. The proposed scheme drives the object to a desired 3D shape using as feedback the object's measured current 3D shape. The scheme is simple to implement, and it avoids some typical requirements in existing work: specifically, we do not need to know the object's mechanical deformation parameters, and we avoid using a Jacobian that is computed from data collected over a time window while the robots are in motion, as this approach is known to be susceptible to noise. We test the proposed scheme in bi-arm shape servoing experiments with a variety of deformable objects of different material (paper, rubber, plastic). Tracking of the deformable object's surface in our experiments is performed using `ROBUSfT` from Chapter 3. The experimental results validate the practicality of our scheme. A **video** of our experiments can also be found at `https://youtu.be/1w2tbgjLrUs`.

Chapter outline. We present the problem formulation and main assumptions in Section 4.2. The process of computing the deformation Jacobian including the usage of ARAP is described in Section 4.3. Section 4.4 describes the control law based on this Jacobian and discusses the stability. The shape tracking method used in the experiments is explained in Section 4.5. We validate our control scheme in Section 4.6. Finally, Section 4.7 concludes this chapter and suggests future directions.

4.2 Problem formulation

The shape servoing problem we address consists of manipulating a DO in a feedback control loop to drive it to a desired shape. A flowchart of our proposed solution can be found in Figure 4.1. We consider a set of robots $\mathcal{M} = \{1, ..., m\}$ whose grippers grasp the object. A sensor (2D camera) viewing the object is used to provide the feedback. We assume the relative poses of the camera frame and robots' frames are all known, as well as the robots' kinematic models. We want to design a control scheme that will compute a 6-DOF velocity input associated with every gripper frame: $\mathbf{v}_i = [\boldsymbol{v}_i^T, \boldsymbol{\omega}_i^T]^T$, where we stack the velocities for all in a column vector of length $6m$: $\mathbf{v} = [\mathbf{v}_1^T, ..., \mathbf{v}_m^T]^T$.

We define the shape of the object as the shape of its surface. We use a triangular surface mesh to represent this shape. The resolution of the mesh is chosen such that

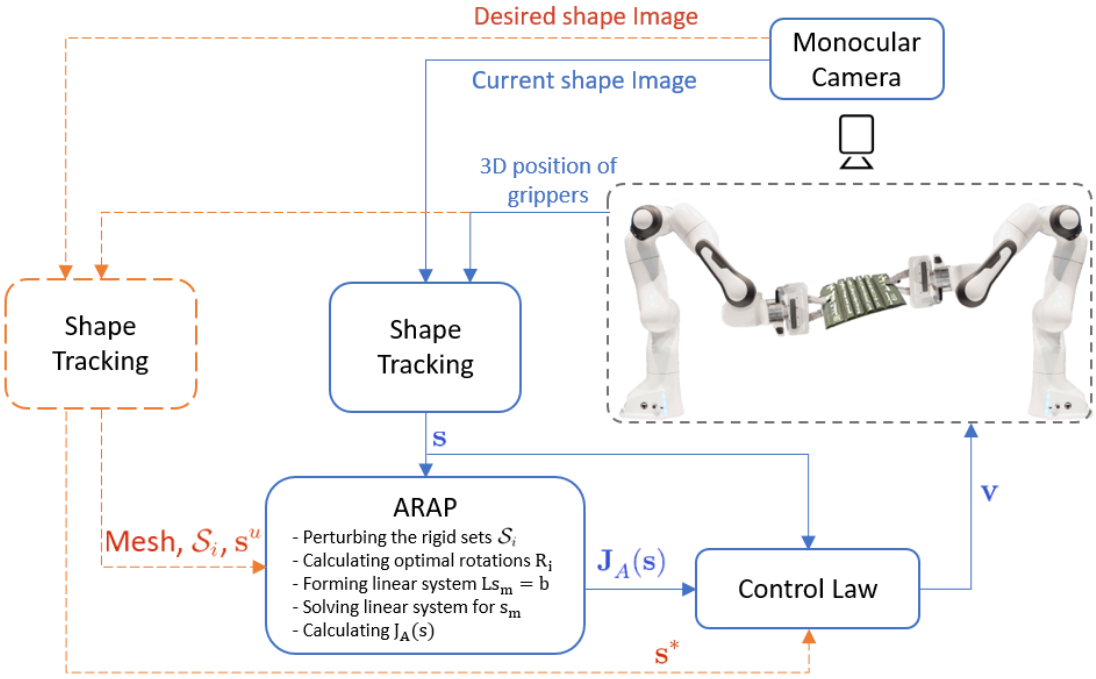


FIGURE 4.1: Flowchart of the proposed shape servoing scheme. Online components in solid blue lines, offline components in dashed orange lines.

it represents with sufficient accuracy the geometry of the surface both in its initial and desired shapes. We call the set of mesh nodes $\mathcal{N} = \{1, ..., n\}$. Henceforth, by *shape* we refer to the 3D positions of the nodes of the chosen mesh. These positions are expressed in the camera frame. We define different shapes to be used in the control scheme:

- Current shape, $\mathbf{s}$.
- Desired shape, $\mathbf{s}^*$, which defines the control goal.
- Undeformed shape, $\mathbf{s}^u$, used in the ARAP model.
- Initial shape, $\mathbf{s}^0 = \mathbf{s}(t = 0)$.

Each of them has size $3n \times 1$ and is a stacking of the node positions. More precisely, the current position of node i is named $\mathbf{s}_i$, and we define $\mathbf{s} = [\mathbf{s}_1^T, ..., \mathbf{s}_n^T]^T$ (analogously for the other shapes). $\mathbf{s}^*$ and $\mathbf{s}^u$ are assumed to be known before starting the servoing task. The measurement of $\mathbf{s}$ obtained from the camera at every time instant is available to the control scheme. We formulate the shape servoing task in terms of the following error:

$$\mathbf{e} = \mathbf{s} - \mathbf{s}^*. \tag{4.1}$$

Driving this error to zero means the task has been completed.

4.2.1 Assumptions on object and robots

We assume the following regarding the object's behavior and its interaction with the robots:

- The robot controller can set exactly the 6-DOF velocity at which the grippers move at every instant. The grippers grasp the object firmly throughout the task. The object's shape stays statically stable (i.e., in quasi-static equilibrium), and reacts

smoothly to the robot motions; i.e., infinitesimally, the change of shape under gripper displacements can be modeled by a deformation Jacobian $\mathbf{J}_O(\mathbf{s})$ [Ber13,NAYW+16, NAL18]. Therefore, the following relation holds:

$$\dot{\mathbf{s}} = \mathbf{J}_O(\mathbf{s})\mathbf{v}. \tag{4.2}$$

- The object has a tendency to resist deformation and to maintain local rigidity, to the extent allowed by the external forces acting on it. This corresponds to an elastic (not plastic) behavior.
- The desired shape is a shape that the object can take (i.e., it is *feasible*). It is also *reachable*, *(i)* with the available actuation (i.e., for the number of grippers used and for their specific placement on the object) in the robots' shared workspace, and *(ii)* by a monotonic decrease of the shape error $\mathbf{e}$, starting from the initial shape.

4.3 Deformation Jacobian computation

Our idea is to compute, using the ARAP model, an approximation of the unknown Jacobian at the current shape $\mathbf{J}_O(\mathbf{s})$. Specifically, we compute numerically a Jacobian $\mathbf{J}_A(\mathbf{s})$ by simulating the ARAP model. The procedure is described next.

4.3.1 Finite-difference estimation

To compute the Jacobian, we need to link the robot gripper frame with the ARAP mesh. To do this, for each robot i we define a *rigid set* $\mathcal{S}_i \subset \mathcal{N}$, with $|\mathcal{S}_i| = n_{\mathcal{S}_i}$ such that the nodes contained in $\mathcal{S}_i$ are rigidly coupled with i's gripper. A simple way, which we used in our

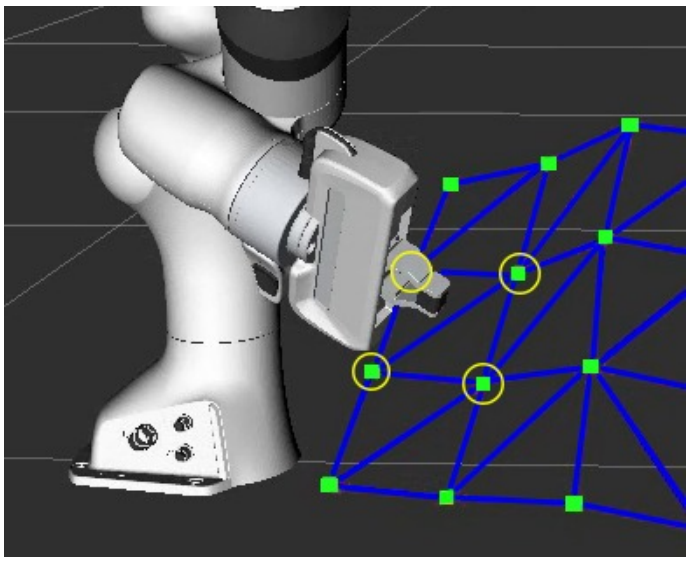


FIGURE 4.2: Definition of a rigid set coupled with the gripper. The four mesh nodes forming the rigid set are encircled in yellow.

tests, to define $\mathcal{S}_i$ is to choose it as the set of nearest neighbors: i.e., the four nodes that surround the gripper i (see an illustration in Figure 4.2). This set can be identified from the knowledge of $\mathbf{s}^0$ and the gripper position in 3D. As $\mathcal{S}_i$ contains at least three nodes whose positions are not aligned, we can associate a Cartesian frame, $\mathcal{F}_i$, to it. To do so, let us define the positions of the nodes in $\mathcal{S}_i$ as $\mathbf{s}_{\mathcal{S}_i} \in \mathbb{R}^{3n_{\mathcal{S}_i}}$. Then, we establish a mapping

$f_i : \mathbb{R}^{3n_{\mathcal{S}_i}} \rightarrow SE(3)$. This mapping gives the frame's pose $\xi_i = f_i(\mathbf{s}_{\mathcal{S}_i})$. Conversely, the positions of the nodes for the frame's pose are obtained as: $\mathbf{s}_{\mathcal{S}_i} = f_i^{-1}(\xi_i)$.

We parameterize the motion of $\mathcal{F}_i$ by 6 DOF (3 translational, 3 rotational). Starting from the current shape $\mathbf{s}$, we simulate a perturbation of one DOF. This is the standard first step in finite-difference estimation. It gives us a new ξ_i. We compute the positions of the nodes $\mathbf{s}_{\mathcal{S}_i} = f_i^{-1}(\xi_i)$, and run the simulation of ARAP (see Section 4.3.2). This gives the new stable shape of the object after the perturbation. Using this new stable shape we apply standard forward finite differences to estimate the column of $\mathbf{J}_A(\mathbf{s})$ corresponding to the perturbed DOF using equation 4.2. Repeating this procedure for all DOF (6 for every $\mathcal{S}_i$) gives the full matrix. Figure 4.3 presents a summary of estimating procedure of $\mathbf{J}_A(\mathbf{s})$.

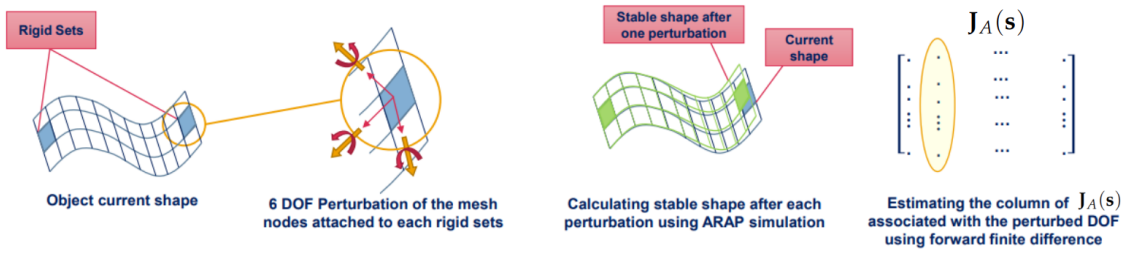


FIGURE 4.3: A summary of estimating procedure of $\mathbf{J}_A(\mathbf{s})$.

In experiments, we also test a reduced, translation-only, Jacobian. This is computed by considering a single mesh node (i.e., a rigid set having only one element), and the 3 translational DOF only. This has the advantage of greater simplicity and is appropriate when the object's region moving rigidly with the gripper is small (e.g., a small contact area on a slack object). Section 4.6 includes more details.

4.3.2 ARAP simulation

We implemented ARAP following the algorithm of Sorkine and Alexa [SA07]. ARAP relies on an energy E that expresses the deviation from rigidity. A *cell* is defined for each node in the mesh, comprising its first-order neighbors. We call this neighborhood $\mathcal{N}_i$ for cell i. Then the global energy is formulated as the sum of energies over all cells, i.e., $E = \sum_{i \in \mathcal{N}} E_i$, where:

$$E_i = \sum_{j \in \mathcal{N}_i} w_{ij} ||\mathbf{s}_i - \mathbf{s}_j - \mathbf{R}_i(\mathbf{s}_i^u - \mathbf{s}_j^u)||^2. \tag{4.3}$$

Each w_{ij} is a scalar encoding the connection between nodes i and j in the mesh. E_i expresses the deviation from rigidity (i.e., from the undeformed shape) of the cell i. This is because $\mathbf{R}_i$ is computed as the optimal rotation that minimizes E_i for given $\mathbf{s}^u$, $\mathbf{s}$. Therefore, E_i measures only the non-rigid component of the difference between the two cell shapes. ARAP works by computing a shape for the full object that corresponds to a minimum of E. This is explained next.

4.3.2.1 Optimal rotations

ARAP requires computing the optimal rotation $\mathbf{R}_i$ for every cell $i \in \mathcal{N}$. For this we define $\mathbf{e}_{ij} = \mathbf{s}_i - \mathbf{s}_j$, $\mathbf{e}_{ij}^u = \mathbf{s}_i^u - \mathbf{s}_j^u$, and the covariance matrix:

$$\mathbf{S}_i = \sum_{j \in \mathcal{N}_i} w_{ij} \mathbf{e}_{ij}^u \mathbf{e}_{ij}^T = \mathbf{P}_i^u \mathbf{D}_i \mathbf{P}_i^T, \tag{4.4}$$

where $\mathbf{P}_i^u$ and $\mathbf{P}_i$ contain respectively $\mathbf{e}_{ij}^u$ and $\mathbf{e}_{ij}$, arranged in columns for $j \in \mathcal{N}_i$, and $\mathbf{D}_i$ is a diagonal matrix that contains the w_{ij} for $j \in \mathcal{N}_i$. Then one computes the Singular Value Decomposition (SVD) of the matrix, i.e., $\mathbf{S}_i = \mathbf{U}_i \mathbf{\Sigma}_i \mathbf{V}_i^\intercal$. From this, the optimal rotation is obtained:

$$\mathbf{R}_i = \mathbf{V}_i \, \mathrm{diag}(1, 1, \det(\mathbf{V}_i \mathbf{U}_i^\intercal)) \, \mathbf{U}_i^\intercal. \tag{4.5}$$

Due to the requirement of computing an SVD for each $\mathbf{R}_i$, this is the most computationally expensive part of our implementation. We avoid some unnecessary calculations by realizing that after perturbing a given DOF of a given rigid set (Section 4.3.1), $\mathbf{R}_i$ changes only for the cells i that contain a node in the rigid set that has been perturbed. Therefore, for the remaining cells, $\mathbf{R}_i$ only needs to be computed once, for the current shape $\mathbf{s}$ before perturbation.

4.3.2.2 Linear system solution

ARAP divides the nodes into two groups: *handled* and *free*. The handled nodes are those whose position is fixed directly by external constraints. The main idea of ARAP is that the object will remain as rigid as possible under these externally fixed positions. Concretely, the stable shape of the object in quasi-static equilibrium after the motion will be at a local minimum of the energy E. For this, one enforces the gradient $\partial E / \partial \mathbf{s}_i$ for every free node i to be zero. This results in this equation for a free node i:

$$\sum_{j \in \mathcal{N}_i} w_{ij}(\mathbf{s}_i - \mathbf{s}_j) = \sum_{j \in \mathcal{N}_i} \frac{w_{ij}}{2}(\mathbf{R}_i + \mathbf{R}_j)(\mathbf{s}_i^u - \mathbf{s}_j^u). \tag{4.6}$$

On the other hand, the positions of the handled nodes are defined as fixed. In our case, the handled nodes are the nodes belonging to the rigid sets being perturbed. We fix their positions as those computed after perturbation (see Section 4.3.1), which we call $\mathbf{c}_i$ of size $n_{\mathcal{S}_i} \times 3$ for rigid set i. Considering the equations for all nodes gives a linear system expression: $\mathbf{L}\mathbf{s}_m = \mathbf{b}$, where $\mathbf{L}$ is an $n \times n$ size Laplacian matrix, $\mathbf{s}_m$ is the shape to be computed (i.e., the new stable shape after perturbation), expressed in $n \times 3$ format, and $\mathbf{b}$ contains the right-hand side of (4.6). The handled node constraints are included as $rows_{\mathcal{S}_i}(\mathbf{s}_m) = \mathbf{c}_i$ for every rigid set i. The linear system expression consists of an individual system for each of the three coordinates. Since our meshes are relatively small (tens of nodes), solving this linear system does not have a relevant effect on the overall computation time. We employ a standard least-squares solution method via the pseudoinverse of the system matrix.

ARAP uses an alternating minimization strategy where the shape that locally minimizes E is found by iterating the described procedure: i.e., the optimal rotations (Section 4.3.2.1) are recomputed with the new $\mathbf{s}$ and the linear system is solved again. In [SA07], 2-3 iterations of the alternating minimization are used for a scenario where a user interactively moves the handled nodes in a visualization application. In our case, we use small motions of the handled nodes to estimate the Jacobian. For this reason, we use a single iteration. That is to say, we run each of the two steps (computation of optimal rotations and solution of linear system) once. In practice, this provides accurate enough results and a fast computation time.

4.3.3 Discussion

An interesting fact to highlight is that we compute the Jacobian using only the current shape $\mathbf{s}$, with no need for dynamic estimators. Moreover, the ARAP model is perfectly suited to the quasi-static scenario we consider because this model computes the new

stable shape directly without having to simulate, and keep track of, velocities of the mesh nodes. The Jacobian computation is fast enough for real-time shape servoing: our non-optimized implementation runs at 20 Hz or more, for two robots (12 DOFs), with meshes ranging from 24 to 70 nodes. More details are given in Section 4.6.

4.4 Control law

We propose the following proportional control law, based on the ARAP Jacobian computed as explained in Section 4.3:

$$\mathbf{v} = -\lambda \, \mathbf{J}_A^{\dagger}(\mathbf{s}) \, \mathbf{e}, \tag{4.7}$$

where λ is a positive scalar gain and $\mathbf{J}_A^{\dagger}(\mathbf{s})$ is the pseudoinverse of the ARAP Jacobian. Note that $\mathbf{J}_A(\mathbf{s})$ was computed with respect to motions of $\mathcal{F}_i \; \forall i \in \mathcal{M}$, which are Cartesian frames defined to represent the object's rigid sets. Therefore $\mathbf{v}$ are velocities to be applied to these rigid set frames. We transform for each robot i the velocity of the rigid set frame to the velocity of the gripper frame rigidly coupled with it (Section 4.3.1). Note that for every robot i the poses of these two frames are known at each instant. The resulting gripper Cartesian velocities are sent to the robots. We assume the rigid sets are fixed (i.e., a robot is coupled with the same region of the object throughout the task). However, note that we do not assume a specific fixed geometry of the coupling between the gripper and the object. Therefore, an important fact to highlight is that our control scheme is robust to minor changes, such as moderate slippage, of the grasping conditions during servoing.

4.4.1 Stability discussion

The control law (4.7) is a classical one in visual servoing. Our control scheme is underactuated. For typical mesh sizes the number of features to be controlled ($3n$) clearly exceeds the number of actuated degrees of freedom ($6m$). Therefore, global stability (i.e., convergence to the desired shape from any initial condition) cannot be guaranteed when using (4.7). As discussed in, e.g., [CH06], with this control law the system is stable locally around the equilibrium $\mathbf{s}^*$ if $\mathbf{J}_A^{\dagger}(\mathbf{s})$ and $\mathbf{J}_O(\mathbf{s})$ have full rank and the product $\mathbf{J}_A^{\dagger}(\mathbf{s}) \, \mathbf{J}_O(\mathbf{s})$ is a positive definite matrix.

Two important facts to note are: *(i)* $\mathbf{J}_A^{\dagger}(\mathbf{s}) \, \mathbf{J}_O(\mathbf{s})$ can be positive definite as long as $\mathbf{J}_A(\mathbf{s})$ approximates $\mathbf{J}_O(\mathbf{s})$ not too coarsely [CH06]. Therefore, precise knowledge of the Jacobian $\mathbf{J}_O(\mathbf{s})$ is not needed. *(ii)* We are just approximating the object's instantaneous reaction to forces, not its behavior over a long time horizon. Doing the latter would be challenging without knowledge of a deformation model for the specific object being manipulated. Considering these two facts and under the assumptions made (Section 4.2.1), it is reasonable to use ARAP to represent, via $\mathbf{J}_A(\mathbf{s})$, the object's instantaneous tendency to preserve rigidity when subjected to forces. ARAP's principle is to approximate geometrically the behavior of real physical objects, and its ability to do so has been extensively validated in diverse applications. We verify the stability and suitability of the scheme in our experiments.

4.5 Shape tracking

To measure $\mathbf{s}$, we employ `ROBUSfT` from Chapter 3. We use $\mathbf{s}^u$ as the 3D template of the object. We also use an overhead photo of each object while being undeformed as the template's texturemap. The known 3D positions of the grippers, which are coupled with the object, are used as constraints in the inference of $\mathbf{s}$. As explained in Chapter 3, this

makes the inference more precise and robust. The process of applying these constraints is to first identify in the camera image, the closest mesh node to each gripper. Then, during tracking, the 3D position of that node can be defined as equal to the known 3D position of that gripper in the camera frame. However, due to the possible errors in robot-camera relative calibration, we use a *soft constraint* instead. In this soft constraint, for each gripper, we consider a sphere with a small radius centered at the 3D position of the gripper. The constraint is that if the corresponding mesh node is outside this sphere, it will be absorbed to the closest point on the surface of the sphere. As explained in Chapter 3, the tracking process is performed at a high frame rate (around 30 fps) to be used with our servoing scheme in real-time. It should be noted that shape tracking and shape servoing are executed on two separate computational units in a parallel architecture (see Section 4.6.1).

4.6 Experiments

In this section, we present the experimental results. A video of our experiments can also be found at `https://youtu.be/1w2tbgjLrUs`.

4.6.1 Experimental setup

We validate the effectiveness of our proposed scheme through different tasks conducted with two Franka Emika robot arms each with 7 DOF. $\mathbf{s}$ would be equal to the 3D shape of the deformed object inferred by `ROBUSfT`. The input for `ROBUSfT` is provided by a calibrated Logitech C270 webcam installed on top of the two robots facing downward. We also externally calibrate the camera with the two robots using Moveit Hand-eye calibration plugin. The entire setup is shown in Figure 4.4. Both the shape tracking and

FIGURE 4.4: View of our experimental setup during testing of bi-arm visual shape servoing of a rubber tire tread with an overhead monocular camera.

shape servoing codes were written in C++ and run on a Dell laptop with an Intel Core i7 CPU, an NVIDIA Quadro T1000 GPU and 16GB RAM. The shape tracking and shape

TABLE 4.1: Parameters of the shape servoing tasks.

Task	1	2	3	4	5	6	7
Mesh Type (Regular/Irregular)	R	R	R	R	R	R	I
Mesh Nodes	7x10	4x6	6x8	6x8	6x8	6x8	40
Control Computation (Hz)	20	30	24	24	24	24	26

servoing are implemented as two separate ROS nodes that communicate in parallel with each other. We also implement the control of the robots in ROS using Cartesian velocity control.

4.6.2 Shape servoing tasks

We report the results of 7 shape servoing tasks involving 4 distinct objects: a Spiderman poster, a place mat, a piece of tire tread and a shoe sole. The general information of each task including the tested object, initial, desired, and final shapes of the object, several snapshots of the evolution of the task, and finally the error (RMS of $\mathbf{e}$) graph are shown in Figure 4.5. The process of each task starts by firstly setting the desired shape of the object. This is done by manually moving the two arms while grasping the object. We infer and store this desired shape using our shape tracking pipeline. This desired shape is indicated as a red mesh in the RViz visualization shown in Figure 4.5. Afterwards, we manually move the two robot arms to set the deforming object in its initial shape. We then start the task. This current shape is indicated as a blue mesh with green nodes in the RViz representation. In order to avoid reflexes (which abort the motion) in the robots caused by sudden and non-smooth movements, we increase the gain progressively at the beginning of the robots' movement. Moreover, we saturate the translational and rotational velocities sent to the robots.

To validate the generality of the proposed scheme, the defined tasks cover diverse materials with different stiffness, various deformations, and several mesh sizes. We also evaluate the behavior under effects such as uncertainty of the grasping and unstable shape tracking conditions, as described below. The main parameters of each task are tabulated in table 4.1. In the following, we explain the tasks in more detail.

Task 1. This task aims to deform a Spiderman poster printed on an A4 paper by just applying translation to the robot grippers without any rotation (see Section 4.3.1). To this end, in each side of the paper, we select only one mesh node as rigid set. As a result, ARAP shape servoing controller merely updates the translational velocity of the robots. In order to achieve a reasonably low error at the end of task, we tried to keep the rotational pose of the grippers unchanged during setting both the desired and initial shapes of the Spiderman poster.

Task 2. This task is conducted with a place mat that is made out of plastic which is stiffer than paper. In this task, in contrast to Task 1, we start from a highly deformed initial shape to a slightly deformed desired shape. The desired shape is both translated and rotated in different directions with respect to the initial shape.

Task 3. This task is carried out with the place mat but with a downward-curved desired shape and a denser mesh. This illustrates that the servoing scheme is not tied to a specific mesh for a given object. One can use different meshes as long as they capture the object's geometry precisely enough.

Task 4. This task is also performed with the place mat with almost the same initial and desired shapes as Task 2. The major change that we made is displacing grippers at the initial shape with respect to their locations when the desired shape was formed. Specifically, we displaced each gripper to its neighbor rigid set on the place mat in inverse directions with respect to each other. We then run the control scheme and try to reach the desired shape using these two new rigid sets.

Task 5. The object used in this task is a piece of tire tread from a heavy vehicle. In order to make it recognizable by our shape tracking pipeline, we added some texture on its surface by writing on it using a white marker.

Task 6. This task is also performed with the tire tread but with an upward-curved desired shape.

Task 7. The shoe sole made of rubber used in this task is the stiffest among our deforming objects. We use an irregular triangular mesh to represent this object in both shape tracking and shape servoing algorithms. In order to make the shoe sole detectable by our shape tracking pipeline, we highlighted the available grooves on it and also drew some new lines on its surface using a black pen.

4.6.3 Results and Discussion

The proposed ARAP shape servoing scheme is able to accurately accomplish all the tasks covering different materials, desired shapes, and mesh sizes. As shown in the RViz representations in Figure 4.5, in each task, the controller moves the robots gradually to conform the shape of the deforming object to the desired shape. The shape RMS error for each task is illustrated in the last column of the Figure 4.5. As can be observed, some noise is present in a few of the shape RMS error graphs. This noise is introduced by the shape tracking pipeline. One could apply a temporal averaging to the positions of the object mesh nodes between consecutive frames in the shape tracking pipeline. This could diminish the noise and provide a more stable 3D inferred shape. However, our goal here is to show that our ARAP shape servoing scheme is robust against these noises and converges successfully in their presence.

Another point that should be noted is that the control scheme can be robust to changes of the grasping configuration. To clarify this, we illustrate in Figure 4.6 the pose of the grippers in the desired and final shapes of Task 4 and Task 5 with green and red dashed lines, respectively. In Task 4, the relative poses of the grippers and their corresponding rigid sets are changed manually before starting the task. In Task 5, in turn, the right gripper rotated due to the slippage during the task while remaining in the same rigid set. Despite these changes in the relative pose of the grippers with respect to their grasping points, the shape servoing scheme managed to accomplish both tasks thanks to guiding the rigid sets under a closed control loop. As long as the grippers remain coupled with the guiding rigid sets, the deforming object can converge to the desired shape.

We note that if the desired shape is unreachable or infeasible, the scheme converges to a shape corresponding to a local minimum of the error. Finally, we compare the prediction capability of our ARAP deformation Jacobian and the diminishing rigidity (DR) Jacobian used in [Ber13, MDRB20]. For this we use some of the less noisy data from our servoing tasks. As comparison metric we use the cosine similarity between the measured $\dot{\mathbf{s}}$ vector and each of the two predicted ones. The closer the values of this metric to 1, the better the alignment between the two vectors. From table 4.2, the ARAP Jacobian

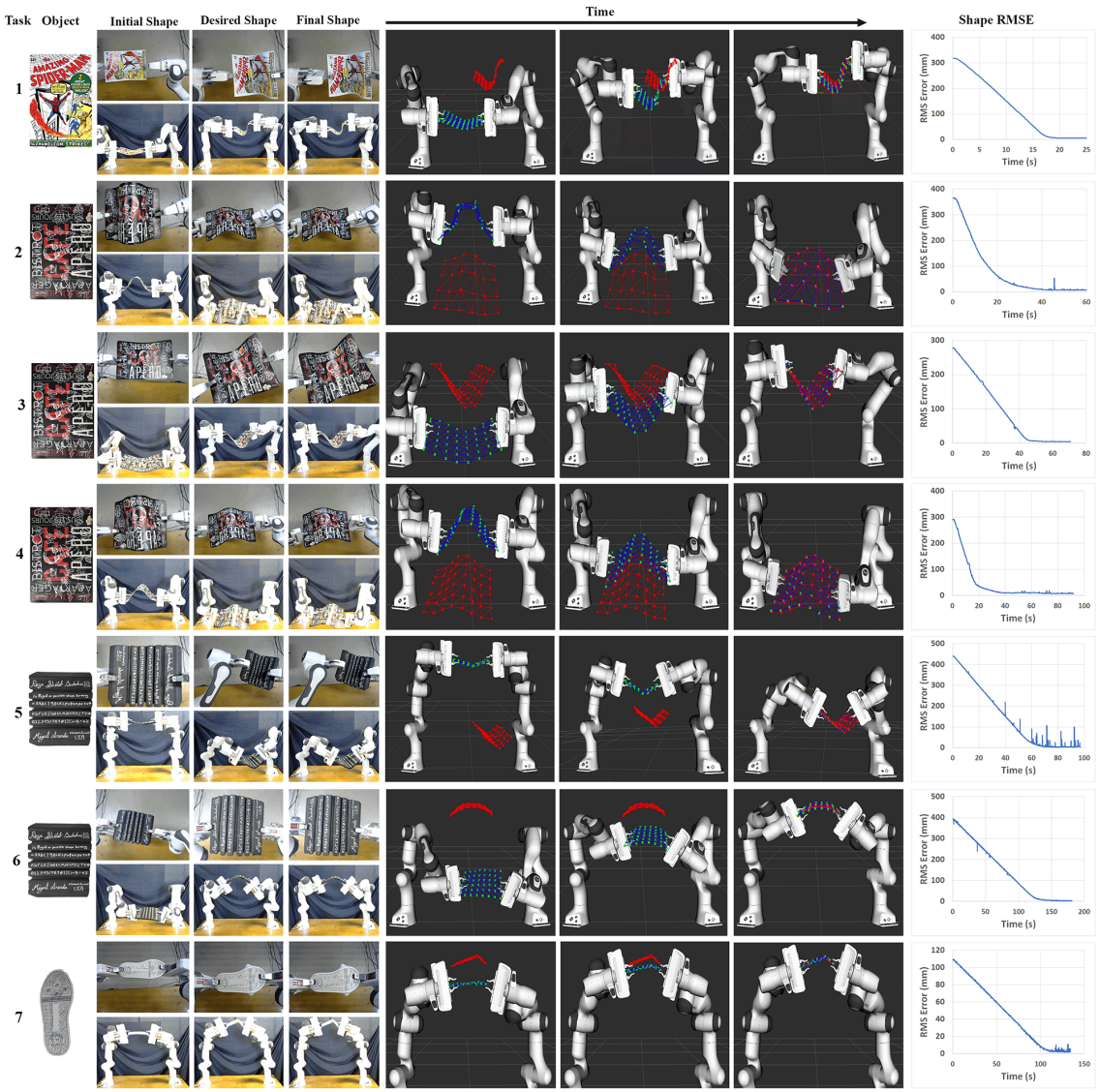


FIGURE 4.5: Results for the seven tasks. For each we show, from left to right: image of the undeformed object; initial, desired and final shape (camera images on top, images from an external camera on bottom); initial, intermediate and final shapes represented in RViz; and evolution of error (RMS of **e**) over time.

TABLE 4.2: Performance evaluation of shape change prediction.

Task	1		3		5	
Jacobian	Avg	Std	Avg	Std	Avg	Std
ARAP	0.839	0.045	0.816	0.052	0.890	0.029
DR	0.663	0.388	0.822	0.144	0.728	0.273

exhibits better accuracy (higher averages, Avg) and stability (lower standard deviations, Std) overall.

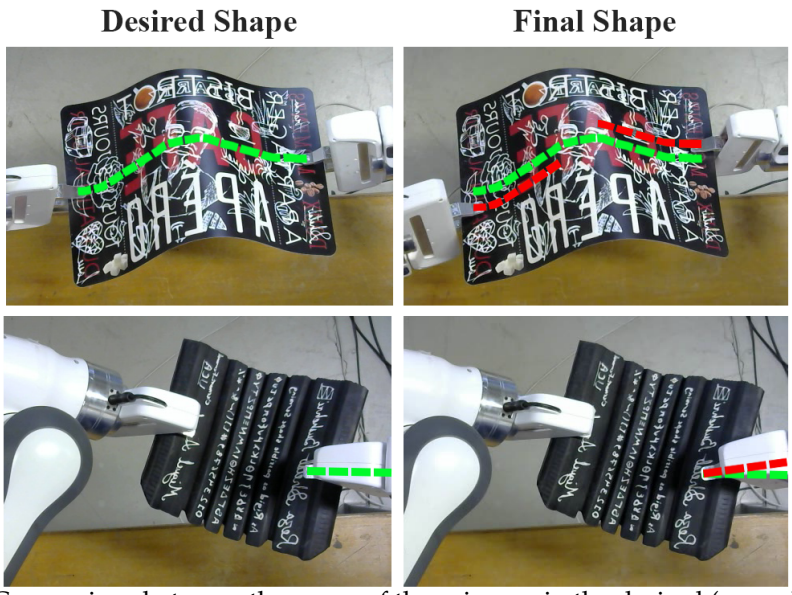


FIGURE 4.6: Comparison between the poses of the grippers in the desired (green lines) and the final (red lines) shapes in Tasks 4 (top) and 5 (bottom). In Task 4 the grippers were displaced manually. In Task 5, the right gripper rotated due to slippage during the task.

4.7 Conclusion and future work

The proposed shape servoing scheme based on ARAP is effective and simple, without requiring any prior knowledge of the mechanical parameters of the object. Our experiments have shown that it works well with a wide range of DOs. However, this approach has its limitations. For example, the servoing is limited to thin-shell objects or surfaces of volumetric objects. Moreover, the effect of gravity is not considered in this work, which may be necessary for low-stiffness objects like cloth. To extend this work, in the next chapter, we introduce lattice-based shape servoing, which is a novel approach that can handle any form and geometry of DOs. We also address the issue of partial shape servoing, where only a specific part of the object needs to be controlled to achieve the desired shape.

Chapter 5

Lattice-based shape tracking and servoing

5.1 Introduction

In Chapter 3, we proposed a monocular shape tracking method for isometrically deforming thin-shell DOs, and in Chapter 4, we introduced a shape servoing approach based on ARAP for thin-shell DOs. However, these methods suffer from the limitation of being applicable only to thin-shell DOs. In fact, this is a well-known limitation in the literature of DOs, as most shape tracking and servoing studies are focused on a specific form of DO, i.e., linear, thin-shell, or volumetric. This is due to the different deformation characteristics and simplifications made for each object form. While some approaches handle multiple object forms, they are often limited to simple scenarios with small deformations [MDRB20, ZNAPC21, HHS+19]. This is our motivation for this work. In this chapter, we propose a novel general unified tracking-servoing approach that overcomes this limitation by addressing different forms of DOs. The approach is composed of a tracking pipeline and a servoing pipeline. The tracking pipeline tracks the shape of the DO in 3D at every instant. The servoing pipeline employs the tracking data to drive the object to a desired shape. Our approach is *general* as it can handle linear, thin-shell and volumetric DOs, not being limited to a specific form of object. It is *unified* as we solve jointly the two problems, tracking and servoing.

The central idea of our approach is to form a 3D lattice around the object. This lattice is bound to the object by geometrical constraints. We track and servo this lattice instead of the object itself. This is done by applying visual and deformation constraints on the lattice. We use ARAP as the deformation model. We present an analytical expression of the deformation Jacobian of the ARAP model. This deformation Jacobian, which we use on our lattice, is our model of how the robot motions deform the lattice. We obtain the Jacobian's expression by rearranging the ARAP deformation equation and exploiting modeling tools presented in [SCOL+04, KFB+21, AALN+22]. Our Jacobian expression is directly applicable to both the whole object (full shape), or a region of it (partial shape). This gives us the useful ability to perform both full or partial shape servoing. We propose a robotic control law based on this Jacobian. Using this control law, we servo the lattice and thus the object toward the desired shape. The inputs to our approach are the point cloud of the object's surface in its rest shape and the point cloud captured by a 3D camera in each frame.

The idea of exploiting a lattice to manage object's deformation has been previously utilized in manipulating shapes in computer graphics when the shape's representative mesh is too high-resolution to be directly manipulated [ZSGS12]. We use the same idea to transfer the object's deformation to the lattice and simplify the deformation. This is reasonable as for many objects, despite all the details on their surface (which are represented by a detailed mesh), the object cannot be deformed in more than several mostly

simple ways. These deformations can, thus, be effectively represented by the lattice encapsulating the object. In fact, we solve the deformation equations for the lattice instead of the object. At the same time, we use geometrical constraints between the lattice and the object as binding constraints. This, concretely, decouples the runtime complexity of the approach from the objects' geometric complexity and makes the whole tracking-servoing process run much faster. Furthermore, using a lattice makes our approach capable of being generalized for any form of the object (linear, thin-shell, volumetric). This is due to the fact that we develop our tracking-servoing approach for a 3D lattice that is formed in the same way for any object geometry.

The experimental results validate the practicality of our proposed shape tracking-servoing approach for a large variation of DOs. The **videos** of our experiments are available at `https://sites.google.com/view/tracking-servoing-approach`.

Chapter outline. We start with a general description of our approach and the notation used in Section 5.2. The tracking pipeline and all its steps are detailed in Section 5.3. We then derive the analytical formulation of the Jacobian and explain its usage in the shape servoing control law in Section 5.4. The effectiveness of our approach is validated through a series of experiments, presented in Section 5.5, where we also compare our results with those of ARAP shape servoing from Chapter 4. We also showcase a failed case of our approach and propose a solution for addressing it. Finally, we conclude this chapter in Section 5.6 and suggest future research directions.

5.2 The approach description and notation

The problem that we address is tracking a DO with any general form (linear, thin-shell, and volumetric) and manipulating it by robotic arms in a feedback control loop so that its shape gradually conforms to a desired shape. To formulate this problem, we use the following notation throughout the chapter:

- Scalars: italic lower-case letters.
- Vectors: bold lower-case letters.
- Matrices: bold upper-case letters.
- Sets: calligraphic letters.

Note that for some developments we represent vectors of points as two-dimensional arrays (i.e., with one column for each spatial dimension), following references [ZSGS12, SA07]. We also define flattening a vector as transposing the rows of the vector and stacking them together in a one-column vector. For example, given a vector $\mathbf{w}$ of dimension $m \times n$, we define the flattened $\mathbf{w}$ as $\mathbf{w_f}$ of dimension $mn \times 1$.

We start by considering a set of robots $\mathcal{M} = \{1, 2, ..., m\}$ that firmly grasp the object during manipulation. We also assume that the forward kinematic models of the robots are known. We use a 3D camera as the sensor providing the input point cloud data. The relative poses between the camera and the robots are assumed to be known. Our goal is to track the object at each instant and introduce a control scheme that computes the 6-DOF velocity vectors associated with all the robots' grippers stacked in a column vector of length $6m$: $\mathbf{v} = [\mathbf{v}_1^\intercal, ..., \mathbf{v}_m^\intercal]^\intercal$. Figure 5.1 illustrates the flowchart of our proposed approach. In the following sections, we will provide a comprehensive explanation of each component in this figure.

We use a point cloud of the object's surface to represent the object's shape. This point cloud is generated when the object is undeformed, i.e., at its rest shape. We use the

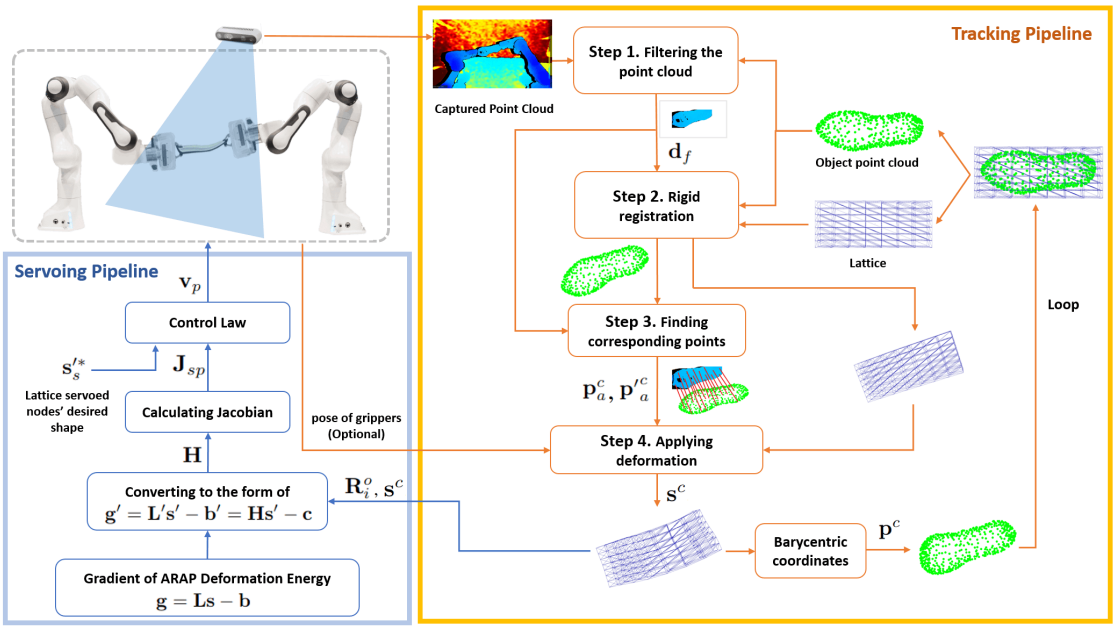


FIGURE 5.1: Flowchart of the proposed shape tracking-servoing approach. In each frame, the tracking pipeline takes the captured point cloud in the current frame and $\mathbf{s}$ (and thus $\mathbf{p}$) from the previous frame as the input and estimates $\mathbf{s}^c$ (and thus $\mathbf{p}^c$) in the current frame. $\mathbf{s}^c$ and $\mathbf{R}_i^o$ are sent to the servoing pipeline where an analytical expression for a deformation Jacobian is obtained. We use this Jacobian in a control law to send proper commands to the robots to guide $\mathbf{s}^c$ toward $\mathbf{s}^*$ and consequently $\mathbf{p}^c$ toward $\mathbf{p}^*$.

$n \times 3$ vector $\mathbf{p}$ to represent this point cloud. The resolution of this point cloud should be fine enough to represent the object's geometry with sufficient accuracy throughout the tracking period. This will be explained more in Sec. 5.3. We, then, define the lattice $\mathbf{s}$, a $n_l \times 3$ vector, encapsulating $\mathbf{p}$ where n_l is the total number of lattice nodes. The lattice $\mathbf{s}$ contains $\mathbf{p}$. Specifically, the metric length of the lattice $\mathbf{s}$ in each spatial dimension is chosen slightly larger than the length of the object $\mathbf{p}$. As the lattice will represent the deformation of the object, its orientation should be set in a way that the lattice axes be aligned with the principal axes that the object bends around (as an example see the lattice around the shoe sole in Figure 5.1). Furthermore, the resolution of $\mathbf{s}$ is selected high enough in each direction to be able to represent the deformation of the object in that direction effectively. Typically n_l can be chosen much smaller than n, i.e., the lattice is a compact representation of the object that allows for faster processing. We generate a tetrahedral mesh over the lattice nodes in $\mathbf{s}$. An example of the formed lattice and the tetrahedral mesh around a shoe sole is illustrated in Figure 5.2. We form the set $\mathcal{T}_j$ for each point $\mathbf{p}_j$ in $\mathbf{p}$ as the indices of lattice nodes belonging to the tetrahedral cell encapsulating $\mathbf{p}_j$. We can, then, express $\mathbf{p}_j$ as a linear combination of lattice nodes in $\mathcal{T}_j$ using barycentric coordinates $\alpha_{i,j}$ as follows:

$$\sum_{\forall i \in \mathcal{T}_j} \alpha_{i,j} \mathbf{s}_i = \mathbf{p}_j. \tag{5.1}$$

This equation serves as a geometrical constraint between $\mathbf{p}$ and $\mathbf{s}$. In the rest of the chapter, by the shape of object and lattice, we refer to the 3D coordinates of the points in $\mathbf{p}$ and $\mathbf{s}$, respectively. We also define the following superscripts that will henceforth be used to specify the state of different shapes:

- Current shape c
- Desired shape $*$

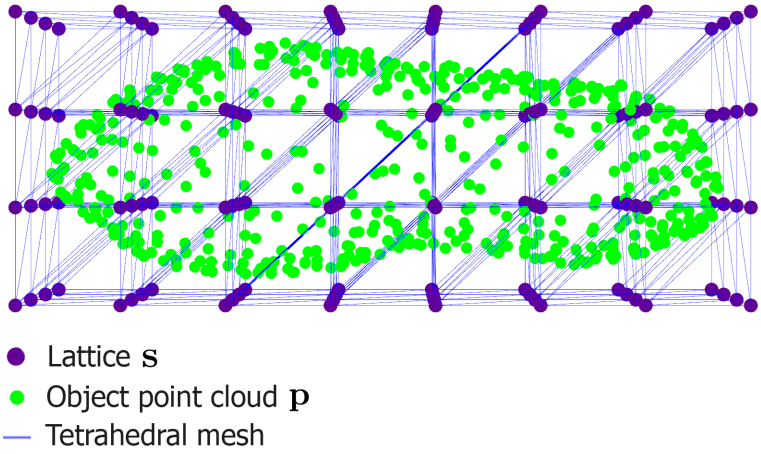


FIGURE 5.2: Lattice $\mathbf{s}$ encapsulating the point cloud $\mathbf{p}$ of a shoe sole at its rest shape. We also generate a tetrahedral mesh over the lattice nodes.

- Undeformed shape u

In the following sections, we describe the proposed tracking and servoing pipelines.

5.3 Tracking pipeline

Our proposed tracking pipeline is able to track a 3D DO in real-time. It comprises 4 steps, namely: capturing and filtering point cloud, rigid registration, finding corresponding points, and finally, applying the deformation model. The first three steps of the pipeline are inspired by [PLS15]. Figure 5.3 illustrates the results of different steps of the tracking pipeline in tracking a shoe sole.

The pipeline tracks the shape of the object in each frame from a known initial shape in 3D space. Knowing the initial shape of the object at the beginning of tracking is common in the state-of-the-art [CB15, PLS15, NÖF15, TT22]. We set the object and lattice initial shapes ($\mathbf{p}^c$ and $\mathbf{s}^c$) by rigidly transforming $\mathbf{p}^u$ (and thus $\mathbf{s}^u$) to an initial pose (i.e., a reference shape) in 3D space visible from the camera. For starting the tracking pipeline, we approximately align the object with the reference shape and trigger tracking. The alignment can be done by hand or by robots while grasping the object. To facilitate the process of alignment, we visualize the reference shape in 3D space and its projection in 2D. It should be noted that this alignment only needs to be partially consistent. This means that even if the real object is slightly displaced and deformed with respect to the reference shape, the tracking pipeline is able to infer a correct $\mathbf{p}^c$ after several frames. This is mainly thanks to the second step of our pipeline, i.e., rigid registration. We will explain this further in Section 5.3.2. An example of this alignment is illustrated in Figure 5.4.

5.3.1 Step 1: Capturing and filtering point cloud

The pipeline starts by capturing and filtering the point cloud from a 3D camera in each frame. The purpose of filtering is to remove the points in the point cloud that do not belong to the object but to the surroundings including background, other objects in the scene, and grippers. The presence of these points in the point cloud might lead to disruption of the tracking pipeline, especially in Section 5.3.3 where correspondences will be found between the object and the captured point cloud. We perform filtering by merely

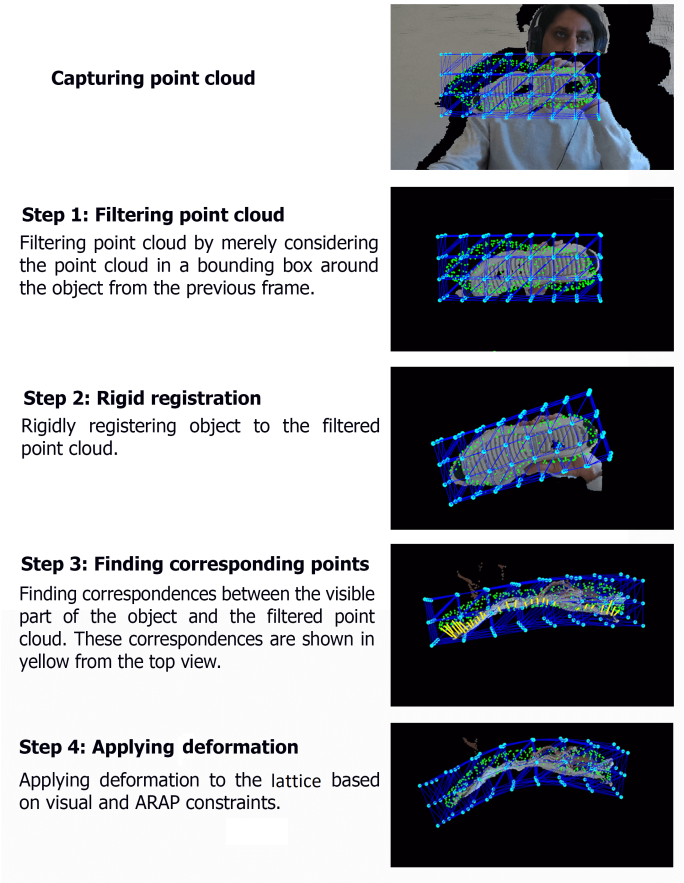


FIGURE 5.3: The results of different steps of the tracking pipeline in one frame of tracking a shoe sole. $\mathbf{p}^c$ and $\mathbf{s}^c$ are shown as green and cyan dots. The dark blue lines represent the tetrahedral mesh over $\mathbf{s}^c$.

considering the point cloud inside a bounding box around $\mathbf{p}$ from the previous frame. In our experiments, we consider the dimensions of the bounding box to be 1 cm larger than the dimensions of $\mathbf{p}$ in each direction. We call this filtered point cloud $\mathbf{d}_f$. We also reduce the size of $\mathbf{d}_f$ by sampling the points on a 5 mm square grid. In addition to using a bounding box to filter the captured point cloud, one can use 2D image filters. This is optional but can be advantageous when there is a significant difference between the pixel characteristics of the object and its surroundings, e.g., a dark object in a light background.

5.3.2 Step 2: Rigid registration

In this step, we rigidly register the points on the visible surface of $\mathbf{p}^c$, namely $\mathbf{p}^c_v$, to the points in $\mathbf{d}_f$. To this end, we exploit a classical rigid iterative closest point (ICP) algorithm. The output of this registration is rigid translation and rotation transformations that are applied on both $\mathbf{p}^c$ and $\mathbf{s}^c$. This step is essential to deal with rapid movements of the objects as it compensates for the rigid non-alignment between $\mathbf{p}^c$ and $\mathbf{d}_f$ in each frame. This is necessary to have a fair initialization for executing the next step of the tracking pipeline (i.e., finding corresponding points). $\mathbf{p}^c_v$ is determined in each frame through a two-step process. First, the normals of $\mathbf{p}^c$ are calculated. Second, the points in $\mathbf{p}^c$ with an angle of more than 90 degrees between the normal of that point in $\mathbf{p}^c$ and the sightline passing through that point are selected. With this procedure, the selected points are in the visible surface of the object. After rigidly registering the object, we update $\mathbf{p}^c_v$ to be used in the next step of the tracking pipeline.

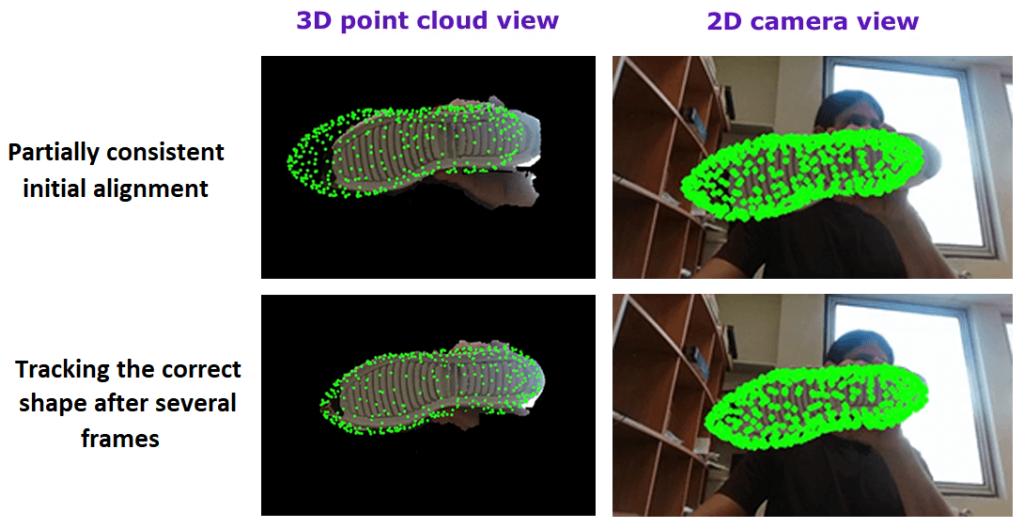


FIGURE 5.4: Aligning the object with the reference shape at the beginning of tracking. To facilitate the process of alignment, we visualize the reference shape in 3D space and its projection in the 2D image. Top row: partially consistent alignment before starting tracking, bottom row: inferring the correct shape after several frames after triggering the tracking pipeline.

5.3.3 Step 3: Finding corresponding points

In this step, we find correspondences between the points in $\mathbf{p}_v^c$ and $\mathbf{d}_f$. We use the ICP-like algorithm suggested in [PLS15]. In this algorithm, first, by employing K-d tree searches, nearest neighbor correspondences are determined, both from $\mathbf{p}_v^c$ to $\mathbf{d}_f$ and vice versa. Then, using these two sets of correspondences, a corresponding point in 3D space is computed for each point in $\mathbf{p}_v^c$. To better understand, we graphically depict this process in Figure 5.5 for two points in $\mathbf{p}_v^c$ (shown in blue) and several points in $\mathbf{d}_f$ surrounding them (shown in green). The process comprises two steps: first, we average the coordi-

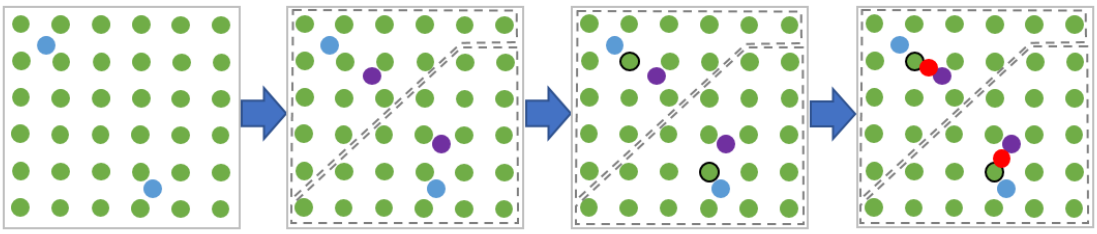


FIGURE 5.5: A graphical representation of the ICP-like algorithm in [PLS15] for two points in $\mathbf{p}_v^c$ (shown in blue) and several points in $\mathbf{d}_f$ (shown in green) surrounding them. Each red point is the corresponding point of a point in $\mathbf{p}_v^c$ in 3D space.

nates of all the points in $\mathbf{d}_f$ having the same nearest neighbors in $\mathbf{p}_v^c$. In our example, this divides the points in $\mathbf{d}_f$ in two groups which are shown inside two dashed contours. The resulting averaged coordinates of these two groups of points are shown as two purple points in Figure 5.5. Second, we average the resulting coordinates from the first step (the purple points) and the coordinates of the points in $\mathbf{d}_f$ which are the nearest neighbors of each point of $\mathbf{p}_v^c$ (the green points with the black outline). The result of the second step is shown as two red points in Figure 5.5. In this stage, we remove the points in $\mathbf{p}_v^c$ (the blue points) whose distance from their corresponding point (the corresponding red point) is more than a specific threshold. This threshold is determined based on the mean value and the standard deviation of the whole set of point-to-point distances. We name the vector of 3D coordinates of these remaining points in $\mathbf{p}_v^c$ (the blue points) as $\mathbf{p}_a^c$ and

the vector of their corresponding 3D coordinates (the red points) in 3D space as $\mathbf{p'}_a^c$. We indicate the number of points in $\mathbf{p}_a^c$ and $\mathbf{p'}_a^c$ with n_a.

5.3.4 Step 4: Applying deformation

In this step, we apply deformation to the inferred shape. This deformation is applied to the lattice and not to the object and the visual constraints related to the object are imposed as the constraints to this deformation. This way, we solve for the lattice shape $\mathbf{s}^c$ and not $\mathbf{p}^c$, which is a faster process as n_l is normally much smaller than n. It should be noted that $\mathbf{s}^c$ and $\mathbf{p}^c$ remain connected due to the spatial constraints in (5.1). We deform $\mathbf{s}^c$ by solving the following linear system, which is a modified version of the one suggested by [ZSGS12]:

$$(\mathbf{\Gamma L} + \mathbf{B}^\intercal \mathbf{B})\, \mathbf{s}^c = \mathbf{\Gamma b} + \mathbf{B}^\intercal \mathbf{t}. \tag{5.2}$$

Two series of constraints are considered in this equation:

- **ARAP constraints**. These constraints ensure that $\mathbf{s}^c$ and consequently $\mathbf{p}^c$ try to keep their local rigidity. These constraints are imposed by matrix $\mathbf{L}$ on the left-hand and vector $\mathbf{b}$ on the right-hand side of (5.2). $\mathbf{L}$ is the $n_l \times n_l$ Laplacian matrix of the tetrahedral mesh formed on the lattice, and $\mathbf{b}$ is a $n_l \times 3$ vector whose ith row is:

$$\mathbf{b}_i = \sum_{j \in \mathcal{N}_i} \frac{w_{ij}}{2} (\mathbf{R}_i + \mathbf{R}_j)(\mathbf{s}_i^u - \mathbf{s}_j^u), \tag{5.3}$$

  In this equation, $\mathcal{N}_i$ is defined as the set containing the first-order neighbors of node i in $\mathbf{s}$, w_{ij} is a scalar encoding the connection between nodes i and j in $\mathbf{s}$, and $\mathbf{R}_i$ is the optimal rotation matrix that conforms the nodes of the set $\mathcal{N}_i$ in $\mathbf{s}^u$ to the same nodes in $\mathbf{s}^c$ with least-squares error. $\mathbf{R}_i$ is computed using the singular value decomposition (for more details see [SA07]).

- **Object visual constraints**. These constraints try to minimize the error between the points in $\mathbf{p}_a^c$ and their corresponding points in $\mathbf{p'}_a^c$. They are imposed by $\mathbf{B}^\intercal \mathbf{B}$ on the left-hand and $\mathbf{B}^\intercal \mathbf{t}$ on the right-hand side of (5.2). $\mathbf{t}$ is a $n_a \times 3$ vector containing the 3D coordinates in $\mathbf{p'}_a^c$ as rows. $\mathbf{B}$ is a $n_a \times n_l$ matrix containing the constraints from (5.1) as rows for each row of $\mathbf{t}$.

We adjust the effectiveness of these two constraints using $\mathbf{\Gamma}$. We consider $\mathbf{\Gamma}$ as a diagonal matrix with the size of $n_l \times n_l$ to attribute different values to each node of $\mathbf{s}^c$. This is concerned with the ARAP constraints as they strongly try to rigidly maintain the shape of $\mathbf{s}^c$ and consequently $\mathbf{p}^c$. We, thus, set two different values for each diagonal element of $\mathbf{\Gamma}$; 0.1 for the nodes of $\mathbf{s}^c$ belonging to a tetrahedral cell that surrounds a point of $\mathbf{p}_a^c$, and 1 for other nodes. As a result, the nodes in $\mathbf{s}^c$ being subject to deformation by object visual constraints can flexibly move, and thus the points in $\mathbf{p}_a^c$ can be absorbed to their corresponding points in $\mathbf{p'}_a^c$. At the same time, the other nodes in $\mathbf{s}^c$ keep the local rigidity, and thus the general shape of the lattice can be maintained. Solving for $\mathbf{s}^c$ is not possible through a one-step solution. Instead as explained in [ZSGS12], an iterative flip-flop solution is used. In each flip-flop iteration: we calculate $\mathbf{R}_i$ (that is a function of $\mathbf{s}^c$) and use that in solving the linear system of (5.2) for $\mathbf{s}^c$. We consider (5.2) converged when the difference between the two successive calculated $\mathbf{s}^c$ is smaller than a certain value. In our experiments, the solution converges in 5-8 iterations. As $\mathbf{B}$ is sparse, we solve (5.2) for $\mathbf{s}^c$ using the conjugate gradient solver of Eigen library for sparse least-square problems.

Similar to the ARAP shape servoing method presented in Chapter 4, known coordinates of robotic grippers can also be incorporated into the shape tracking process. As explained, this enhances the robustness of tracking in robotic applications. In our approach, these constraints can be implemented within the tracking pipeline by using the known 3D coordinates of several lattice nodes. The indices of these lattice nodes are determined manually by the user before starting the tracking pipeline or automatically by selecting the closest lattice nodes to the known point in 3D space at the beginning of tracking. In both cases, at the beginning of the tracking pipeline, the relative 3D coordinates of the selected lattice nodes with respect to their corresponding 3D known points are saved. These relative coordinates along with the coordinates of the known points at each instant give us the 3D coordinates of the selected lattice nodes. As for the grippers, their poses in the camera frame can be computed knowing the configuration of the robot and the calibration between the robots and the camera. The implementation of this constraint can be performed by modifying (5.2) before solving it. This is done by removing corresponding rows and columns from the left-hand side and recalculating the right-hand side with the known 3D coordinates of the relevant lattice nodes. The use of these constraints is optional; they make tracking more precise and robust, specifically in scenarios with large deformations or the presence of self or external occlusions.

5.4 Servoing pipeline

In this section, we explain our shape servoing pipeline. The same as in ARAP shape servoing presented in Chapter 4, we use the ARAP deformation model to obtain a deformation Jacobian. The control law we propose, then, is based on this Jacobian. In Chapter 4, we computed the Jacobian numerically, by simulating perturbations in every DOF of the grippers that grip the object. In contrast, here we use an analytical expression of the Jacobian. The advantages of this are: the Jacobian we compute does not involve any numerical approximation, and its computation is fully scalable as the number of grippers grows. Another main difference is that we derive the formulation for the lattice and not the object. This not only generalizes the servoing formulation for any form of the object, but also decouples the runtime complexity of the servoing from the objects' geometric complexity. We use this Jacobian to propose a control law. Finally, we apply the control law on the lattice to guide $\mathbf{s}^c$ toward $\mathbf{s}^*$ and consequently $\mathbf{p}^c$ toward $\mathbf{p}^*$ thanks to the spatial constraints between the lattice and the object in (5.1). We present the different steps of the servoing pipeline in the following.

5.4.1 ARAP deformation Jacobian

The main ingredient of our shape servoing pipeline is the Jacobian that expresses how the infinitesimal gripper motions change the shape of the object, namely *deformation Jacobian*. Most shape servoing works [Ber13, NALRL14, NAL18, HSP18, DBPC18, KFB+21, ZNAPC21, SBAMÖ22, AALN+22] assume that the object's shape changes quasi-statically and that this change can be represented by such deformation Jacobian. In this chapter, we also make the same assumption. As in Chapter 4, the main principle of our servoing pipeline is approximating the true deformation Jacobian of the object by the deformation Jacobian of the ARAP model of the object. In particular, we present an analytic derivation of the deformation Jacobian of the ARAP model. We start with the main ARAP linear system

$$\mathbf{L}\mathbf{s} = \mathbf{b}, \tag{5.4}$$

which is (5.2) with only deformation constraints. The principle of ARAP's modeling [SA07] is that the object's shape (in our case the lattice's shape $\mathbf{s}$) in quasi-static equilibrium minimizes the ARAP deformation energy. In particular, this energy is at a local minimum under the existing constraints (e.g., regions of the object that are grasped and moved by a robot). The gradient of this energy with respect to $\mathbf{s}$ has the following form:

$$\mathbf{g} = \mathbf{L}\mathbf{s} - \mathbf{b}, \tag{5.5}$$

omitting multiplicative constants that are irrelevant to our purpose. $\mathbf{b}$ contains the $\mathbf{b}_i$ for each node i in $\mathbf{s}$, which can be written as in (5.3). As the stable shape $\mathbf{s}$ is a local minimum, one makes the gradient zero to find $\mathbf{s}$: this is what (5.4) expresses. In the standard ARAP formulation, the optimal rotations $\mathbf{R}_i$ in (5.3) depend on $\mathbf{s}$ via SVD. Note that in this section $\mathbf{s}$ represents any shape in the neighborhood of $\mathbf{s}^c$. This is because what we want to do is to compute the deformation Jacobian at $\mathbf{s}^c$. Then, our insight is that the optimal rotations for $\mathbf{s}$ will also be in the neighborhood of the optimal rotations for $\mathbf{s}^c$. Therefore, we can express the relative rotation between them as an infinitesimal rotation. We will show that this infinitesimal rotation can be parameterized as a linear function of the node positions, avoiding the SVD. With this linear parameterization, we will transform (5.5) in an expression where the dependency on $\mathbf{s}$ is fully linear. From this expression we will derive the deformation Jacobian.

We start by writing the rotation matrix $\mathbf{R}_i$ in (5.3) as the multiplication of a general rotation matrix $\mathbf{R}_i^o$ and an infinitesimal rotation matrix $\mathbf{R}_i^s$:

$$\mathbf{R}_i = \mathbf{R}_i^s \mathbf{R}_i^o. \tag{5.6}$$

$\mathbf{R}_i^o$ is the optimal rotation matrix that best conforms the nodes of the set $\mathcal{N}_i$ in $\mathbf{s}^u$ to the same nodes in $\mathbf{s}^c$. Note that this rotation matrix $\mathbf{R}_i^o$ is known for us: it is the last optimal rotation matrix computed by our tracking pipeline. We are computing the Jacobian for changes of shape in the neighborhood of the current shape. Therefore, for the Jacobian computation, $\mathbf{R}_i^o$ is considered a fixed matrix. $\mathbf{R}_i^s$, the infinitesimal rotation matrix, can be expressed as a linear function of $\mathbf{s}_i$ using the estimation suggested by [SCOL+04]. According to [SCOL+04], when $\mathbf{s}_i$ is in the local neighborhood of $\mathbf{s}_i^c$, one can write an approximation for the transformation matrix $\mathbf{T}_i$ between $\mathbf{s}_i^c$ and $\mathbf{s}_i$ that is a linear function of $\mathbf{s}_i$. The transformation matrix $\mathbf{T}_i$ is a 4×4 matrix including translation, rotation, and scale. Here, what we need is the rotation part of this transformation. We, consequently, isolate the rotation part of the $\mathbf{T}_i$ formulation presented in [SCOL+04] and use it as $\mathbf{R}_i^s$. We can thus write $\mathbf{R}_i^s$ as the following infinitesimal rotation matrix:

$$\mathbf{R}_i^s = \begin{pmatrix} 1 & -h_{i_3} & h_{i_2} \\ h_{i_3} & 1 & -h_{i_1} \\ -h_{i_2} & h_{i_1} & 1 \end{pmatrix}. \tag{5.7}$$

As this is a least-squares optimal rotation, h_{i_1}, h_{i_2}, and h_{i_3} can be computed from:

$$\begin{pmatrix} h_{i_1} \\ h_{i_2} \\ h_{i_3} \end{pmatrix} = (\mathbf{A}_i^\intercal \mathbf{A}_i)^{-1} \mathbf{A}_i^\intercal \mathbf{w}_i. \tag{5.8}$$

We consider that the reference shape for this least-squares computation is $\mathbf{s}^c$. Therefore, $\mathbf{A}_i$ and $\mathbf{w}_i$ can be written as:

$$\mathbf{A}_i = \begin{pmatrix} 0 & \mathbf{s}^c_{k_z} & -\mathbf{s}^c_{k_y} \\ -\mathbf{s}^c_{k_z} & 0 & \mathbf{s}^c_{k_x} \\ \mathbf{s}^c_{k_y} & -\mathbf{s}^c_{k_x} & 0 \\ \vdots & \vdots & \vdots \end{pmatrix}, k \in \{i\} \cup \mathcal{N}_i, \tag{5.9}$$

and

$$\mathbf{w}_i = \begin{pmatrix} \mathbf{s}_{k_x} \\ \mathbf{s}_{k_y} \\ \mathbf{s}_{k_z} \\ \vdots \end{pmatrix}, k \in \{i\} \cup \mathcal{N}_i. \tag{5.10}$$

$\mathbf{A}_i$ is a known $3d_i \times 3$ matrix (d_i is the number of elements in $\{i\} \cup \mathcal{N}_i$) composed of the elements from the current shape, and $\mathbf{w}_i$ is a $3d_i \times 1$ vector of unknowns comprising $\mathbf{s}_i$ and its first-order neighbors. By putting (5.9) and (5.10) in (5.8) and then in (5.7) we can have $\mathbf{R}_i^s$ as:

$$\mathbf{R}_i^s = \begin{pmatrix} 1 & -\mathbf{M}_{i_{3,*}}\mathbf{w}_i & \mathbf{M}_{i_{2,*}}\mathbf{w}_i \\ \mathbf{M}_{i_{3,*}}\mathbf{w}_i & 1 & -\mathbf{M}_{i_{1,*}}\mathbf{w}_i \\ -\mathbf{M}_{i_{2,*}}\mathbf{w}_i & \mathbf{M}_{i_{1,*}}\mathbf{w}_i & 1 \end{pmatrix}, \tag{5.11}$$

where $\mathbf{M}_i = (\mathbf{A}_i^\intercal \mathbf{A}_i)^{-1}\mathbf{A}_i^\intercal$ is a $3 \times 3d_i$ matrix and the subscript $(k, *)$ indicates the kth row of the matrix $\mathbf{M}_i$. We rearrange $\mathbf{R}_i^s$ as the following:

$$\mathbf{R}_i^s = \mathbf{I} + \begin{pmatrix} \mathbf{0}_{3d_i}\mathbf{w}_i & -\mathbf{M}_{i_{3,*}}\mathbf{w}_i & \mathbf{M}_{i_{2,*}}\mathbf{w}_i \\ \mathbf{M}_{i_{3,*}}\mathbf{w}_i & \mathbf{0}_{3d_i}\mathbf{w}_i & -\mathbf{M}_{i_{1,*}}\mathbf{w}_i \\ -\mathbf{M}_{i_{2,*}}\mathbf{w}_i & \mathbf{M}_{i_{1,*}}\mathbf{w}_i & \mathbf{0}_{3d_i}\mathbf{w}_i \end{pmatrix}, \tag{5.12}$$

where $\mathbf{I}$ is a 3×3 identity matrix, and $\mathbf{0}_{3d_i}$ is a row vector of zeros with the length of $3d_i$. Replacing (5.12) in (5.6) and then in (5.3) and rearranging the terms we have:

$$\mathbf{b}_i = \sum_{j \in \mathcal{N}_i} \frac{w_{ij}}{2} \mathbf{I}(\mathbf{R}_i^o + \mathbf{R}_j^o)(\mathbf{s}_i^u - \mathbf{s}_j^u) + \sum_{j \in \mathcal{N}_i} \mathbf{q}_i \mathbf{w}_i + \sum_{j \in \mathcal{N}_i} \mathbf{q}_j \mathbf{w}_j \tag{5.13}$$

where $\mathbf{q}_i$ and $\mathbf{q}_j$ are known $1 \times 3d_i$ and $1 \times 3d_j$ vectors that can be written as the following:

$$\begin{aligned} \mathbf{q}_i = \frac{w_{ij}}{2} (&\mathbf{u}_i[1](\mathbf{M}_{i_{3,*}} - \mathbf{M}_{i_{2,*}}) + \\ &\mathbf{u}_i[2](\mathbf{M}_{i_{1,*}} - \mathbf{M}_{i_{3,*}}) + \\ &\mathbf{u}_i[3](\mathbf{M}_{i_{2,*}} - \mathbf{M}_{i_{1,*}})), \end{aligned} \tag{5.14}$$

$$\begin{aligned} \mathbf{q}_j = \frac{w_{ij}}{2} (&\mathbf{u}_j[1](\mathbf{M}_{j_{3,*}} - \mathbf{M}_{j_{2,*}}) + \\ &\mathbf{u}_j[2](\mathbf{M}_{j_{1,*}} - \mathbf{M}_{j_{3,*}}) + \\ &\mathbf{u}_j[3](\mathbf{M}_{j_{2,*}} - \mathbf{M}_{j_{1,*}})). \end{aligned} \tag{5.15}$$

In these equations $\mathbf{u}_i = \mathbf{R}_i^o(\mathbf{s}_i^u - \mathbf{s}_j^u)$ and $\mathbf{u}_j = \mathbf{R}_j^o(\mathbf{s}_i^u - \mathbf{s}_j^u)$ are both 3×1 vectors, and $[k]$ signifies the kth element of vector $\mathbf{u}$.

On the right-hand side of (5.13), the first sum is a known and constant vector, while the second and third sums are linear functions of $\mathbf{w}_i$ and $\mathbf{w}_j$. One point that should be noted is that for solving the original formulation of ARAP in (5.4), as described in Section 5.3.4, a flip-flop solution is used. This solution comprises two steps: calculating optimal rotations and solving the ARAP linear system. In calculating optimal rotations, the directions of x, y, and z are dependent in the SVD. In solving the ARAP linear system, however, having the optimal rotation matrices calculated, one can solve (5.4) for each direction independently regardless of the other directions. In our new formulation of $\mathbf{b}$ in (5.13), we propagate the dependency between these directions (coming from the second and third sums) into the equation. This dependency stems from the usage of the estimation for rotation matrices (see equations (5.7) to (5.10)). In order to integrate this dependency with other terms in the original formulation of ARAP, we rewrite each term in (5.4). This is done by:

- *(i)* We define $\mathbf{s}'$ with the size of $3n_l \times 1$ as the flattened form of $\mathbf{s}$.
- *(ii)* We define $\mathbf{L}'$ as a $3n_l \times 3n_l$ matrix such that $\mathbf{L}' = \mathbf{L} \otimes \mathbf{I}$ where $\mathbf{I}$ is a 3×3 identity matrix and $\otimes$ denotes the Kronecker product.
- *(iii)* We define $\mathbf{b}'$ as the following:

$$\mathbf{b}' = \mathbf{c} + \mathbf{Q}\mathbf{s}'. \tag{5.16}$$

In this equation, $\mathbf{b}'$ with the size of $3n_l \times 1$ is the flattened form of $\mathbf{b}$ with the size of $n_l \times 3$. $\mathbf{c}$ is a $3n_l \times 1$ vector that represents the general form of the first sum of (5.13) consisting of n_l vectors of all lattice nodes (each of size 3×1) concatenated together. The multiplication of $\mathbf{Q}\mathbf{s}'$ includes all the elements in the second and third sums in (5.13). The same as in *(i)*, $\mathbf{s}'$ with the size of $3n_l \times 1$ is the flattened form of $\mathbf{s}$. $\mathbf{Q}$ is a $3n_l \times 3n_l$ known matrix that is initially filled with zeros and then the values from $\mathbf{q}_i$ and $\mathbf{q}_j$ will be added to their corresponding elements. Each consecutive three rows in $\mathbf{Q}$ belong to three directions of one lattice node. We set these three rows identical. This comes from the fact that we apply the constraints from the second and the third sums in (5.13) that depend on the three directions to each one of the directions identically. Each consecutive three columns of $\mathbf{Q}$ are dedicated to the three directions of each lattice node. The elements in $\mathbf{q}_i$ and $\mathbf{q}_j$ are summed with the elements in the rows corresponding to ith lattice node at their corresponding columns regarding their indices and their directions. The process of filling $\mathbf{Q}$ can be formulated by the following equation:

$$\mathbf{Q}(3(i-1)+r, 3(\mathcal{K}[k]-1)+c) = \sum_{j \in \mathcal{N}_i} \mathbf{q}[3(k-1)+c], \tag{5.17}$$

where $\mathbf{q} \in \{\mathbf{q}_i, \mathbf{q}_j\}$, $i \in \{1, ..., n_l\}$, $r, c \in \{1, 2, 3\}$ and

$$\begin{cases} k \in \{1, ..., d_i\}, \mathcal{K} = i \cup \mathcal{N}_i, & \text{if } \mathbf{q} = \mathbf{q}_i \\ k \in \{1, ..., d_j\}, \mathcal{K} = j \cup \mathcal{N}_j, & \text{if } \mathbf{q} = \mathbf{q}_j \end{cases} \tag{5.18}$$

Using flattening and (5.16), we can write (5.5) as:

$$\mathbf{g}' = \mathbf{L}'\mathbf{s}' - \mathbf{b}' = \mathbf{H}\mathbf{s}' - \mathbf{c}, \tag{5.19}$$

where $\mathbf{g}'$ with the size of $3n_l$ is the flattened form of $\mathbf{g}$, and $\mathbf{H} = \mathbf{L}' - \mathbf{Q}$ is a known $3n_l \times 3n_l$ matrix. We will define the deformation Jacobian in terms of velocities (i.e., time variations). Note that for this Jacobian computation, $\mathbf{H}$ and $\mathbf{c}$ are constant, as they depend on fixed quantities. Therefore, by taking derivative from (5.19) with respect to time, we have $\mathbf{H}\dot{\mathbf{s}}' = \dot{\mathbf{g}}'$.

The final step of our derivation is to obtain the deformation Jacobian from this equation. For this, we apply on our ARAP model an approach that is based on node partitioning. This approach has been previously applied in [KFB+21] on an FEM model, and in [AALN+22] on an offline geometrical model. In [KFB+21, AALN+22] this approach was defined for linear objects deforming in 2D; here, we widen that scope as we consider objects of arbitrary form (linear, thin-shell, volumetric) which deform in 3D.

We, first, categorize the nodes of the lattice into three sets; free, servoed, and gripped, having in turn n_f, n_s, and n_g elements such that $n_f + n_s + n_g = n_l$. Accordingly, we divide the flattened position vector of the lattice nodes $\mathbf{s}'$ into $\mathbf{s}'_f \in \mathbb{R}^{3n_f}$, $\mathbf{s}'_s \in \mathbb{R}^{3n_s}$, and $\mathbf{s}'_g \in \mathbb{R}^{3n_g}$. Likewise, we partition matrix $\mathbf{H}$. Under ARAP modeling, the gripped nodes

are moved externally, but the motions of the free and servoed nodes are determined only by the ARAP deformation energy. As the object is always in quasi-static equilibrium, its shape corresponds to a locally minimum energy. Therefore, for the free and servoed nodes the ARAP energy gradient has to be always zero. Hence, the time derivative of the gradient is also zero. Therefore, the expression $\mathbf{H}\dot{\mathbf{s}}' = \dot{\mathbf{g}}'$ above takes the following form:

$$\begin{pmatrix} \mathbf{H}_{gg} & \mathbf{H}_{gs} & \mathbf{H}_{gf} \\ \mathbf{H}_{sg} & \mathbf{H}_{ss} & \mathbf{H}_{sf} \\ \mathbf{H}_{fg} & \mathbf{H}_{fs} & \mathbf{H}_{ff} \end{pmatrix} \begin{pmatrix} \dot{\mathbf{s}}'_g \\ \dot{\mathbf{s}}'_s \\ \dot{\mathbf{s}}'_f \end{pmatrix} = \begin{pmatrix} \dot{\mathbf{g}}'_g \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix} \tag{5.20}$$

From (5.20), we can obtain the following expression linking the velocities of the gripped and the servoed nodes:

$$\dot{\mathbf{s}}'_s = \mathbf{J}_{sg}\dot{\mathbf{s}}'_g, \tag{5.21}$$

where

$$\mathbf{J}_{sg} = -(\mathbf{H}_{ss} - \mathbf{H}_{sf}\mathbf{H}_{ff}^{-1}\mathbf{H}_{fs})^{-1}(\mathbf{H}_{sg} - \mathbf{H}_{sf}\mathbf{H}_{ff}^{-1}\mathbf{H}_{fg}). \tag{5.22}$$

As in [KFB+21, YZL21, IMH05, AALN+22], we assume the matrices that have to be inverted are full-rank. This stems from the initial made assumption that the shape is fully constrained by the grippers. Thus, $\mathbf{J}_{sg}$ is the ARAP deformation Jacobian, which we have obtained analytically from the knowledge of the current shape $\mathbf{s}^c$, the optimal rotations $\mathbf{R}^o$, and the ARAP model parameters.

5.4.2 Control law for shape servoing

We use $\mathbf{J}_{sg}$ to propose a proportional control law to drive the positions of the servoed nodes of the lattice toward their desired values. We, first, define the servoing error as:

$$\mathbf{e}_s = \mathbf{s}'^c_s - \mathbf{s}'^*_s. \tag{5.23}$$

Using (5.21) and (5.23) we can write our proportional control law as the following:

$$\mathbf{v}_g = -k_g\mathbf{J}^\dagger_{sg}\mathbf{e}_s, \tag{5.24}$$

where k_g is a positive gain, $\dagger$ signifies the pseudoinverse, and $\mathbf{v}_g$ is the vector of translational velocities to be applied to the gripped lattice nodes. In a common robotic application, the object is controlled by a robotic gripper with 6 DOFs. Hence, we transfer the calculated translational velocities of the gripped lattice nodes to the 6-DOF velocities of their corresponding grippers. Note that the grippers firmly hold the object and, thus, are coupled with the lattice due to the constraints in (5.1). We categorize the gripped nodes corresponding to each gripper in the sets $\mathcal{G}$ such that $\sum_{l=1}^{m} |\mathcal{G}_l| = n_g$ where $|.|$ signifies the number of nodes in the set. We can, then, write the following equation between $\mathbf{v}_{p_l}$, the 6×1 velocity vector of the gripper l, and $\mathbf{v}_{g_l}$, the velocity vectors of the nodes in $\mathcal{G}_l$ stacked together.

$$\mathbf{v}_{g_l} = \mathbf{G}_{gp_l}\mathbf{v}_{p_l}, \tag{5.25}$$

where $\mathbf{G}_{gp_l}$ is the grasp matrix of the gripper l and can be written as:

$$\mathbf{G}_{gp_l} = \begin{pmatrix} 1 & 0 & 0 & 0 & r_{lj_z} & -r_{lj_y} \\ 0 & 1 & 0 & -r_{lj_z} & 0 & r_{lj_x} \\ 0 & 0 & 1 & r_{lj_y} & -r_{lj_x} & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}, j \in \mathcal{G}_l, \tag{5.26}$$

where r_{lj} represents the vector from the gripper l to the lattice node j at each instant. We extend (5.26) to write a general equation for all the grippers and their corresponding lattice nodes:

$$\mathbf{v}_g = \mathbf{G}_{gp}\mathbf{v}_p, \tag{5.27}$$

where $\mathbf{v}_g$ is a $3n_g \times 1$ vector containing the translational velocities of all the gripped nodes of the lattice stacked together, $\mathbf{v}_p$ is a $6m \times 1$ vector containing the velocities of all the grippers stacked together, and $\mathbf{G}_{gp}$ is the total grasp matrix with the size of $3n_g \times 6m$ assembled from all the grasp matrices $\mathbf{G}_{gp_l}$:

$$\mathbf{G}_{gp} = \begin{pmatrix} \mathbf{G}_{gp_0} & 0 & \cdots & 0 \\ 0 & \ddots & & \\ \vdots & & \mathbf{G}_{gp_l} & \\ 0 & & & \ddots \end{pmatrix}, l \in \mathcal{M}. \tag{5.28}$$

We finally define the control law:

$$\mathbf{v}_p = -k_p \mathbf{J}_{sp}^{\dagger} \mathbf{e}_s \tag{5.29}$$

where $\mathbf{J}_{sp} = \mathbf{J}_{sg}\mathbf{G}_{gp}$ is the Jacobian that relates the lattice servoed nodes to the grippers, and k_p is an arbitrary positive gain that can be the same or different from k_g. One can use a diagonal gain matrix instead of k_p to weight differently translation and rotation velocities.

In Chapter 4, we carried out shape servoing with a numerically computed Jacobian-based proportional control law. On the contrary, our new control law in (5.29) is based on an analytic formulation of the deformation Jacobian, is applicable on objects of all forms, and can be used for both full and partial shape servoing. This new control law allows an exponential decrease of the shape servoing error $\mathbf{e}_s$ towards zero if the ARAP deformation Jacobian approximates the object's true deformation Jacobian well. The practical performance of the control law is illustrated and discussed in detail in the experiments section.

5.5 Experiments

In this section, we validate the effectiveness of our approach through a diverse set of experiments covering various forms and materials of DOs. A video of these experiments can be found at `https://sites.google.com/view/tracking-servoing-approach`. The objects of interest comprise a linear object, i.e., a cable, two thin-shell objects including a sheet of A4 paper and a convoluted foam, and two volumetric objects including a shoe sole and a foam octagonal cylinder. We apply large deformations and do both full and partial shape servoing. Full shape servoing is when we servo all the lattice nodes (except for the gripped nodes), i.e., $n_f = 0$. Partial shape servoing is, however, when we servo a portion of the lattice nodes, i.e., $n_f \neq 0$. The ability to perform partial shape servoing is interesting in practice (e.g., for tasks where a specific part of the object has to be assembled on another object), and it highlights the versatility of our approach.

5.5.1 Experimental setup

The experiments are conducted using a dual-arm setup made of two Franka Emika robots each with 7 DOFs. A RealSense D435 camera facing the manipulation area provides the input for the tracking pipeline. We use the camera resolution 424×240 in all the

experiments except for the ones with linear objects in which we double this resolution to apply image filters (see Section 5.5.2). We use MoveIt Hand-eye calibration plugin to externally calibrate the camera with the robots. The setup is shown in Figure 5.6. The whole code is written in C++ and runs on a single ROS node on a Dell laptop with a 9th generation Intel Core i7 CPU. No parallelization is employed to run the ROS node. Our approach calculates the grippers' velocities and sends them to a Cartesian velocity control ROS node that controls each robot. We use Point Cloud Library (PCL) for handling point clouds.

FIGURE 5.6: Our experimental setup with two Franka robots manipulating a foam octagonal cylinder. An overhead 3D camera provides the input for our approach.

Depending on the complexity of the objects' geometry, we form the undeformed point cloud of each object $\mathbf{p}^u$ either by scanning them (using a Kinect and Skanect software) or by drawing simple shapes in Blender. We, then, form the lattice $\mathbf{s}^u$ around each object point cloud. In all cases, we consider the size of the lattice to be 1 cm larger than the size of the object in each direction. We rigidly transform the created $\mathbf{p}^u$, as the initial shape $\mathbf{p}^c$, somewhere in front of the camera where it is reachable by the robots. We, then, align the object with the initial shape $\mathbf{p}^c$ while being grasped by the robots. Note that a partial alignment is sufficient (see Sect. 5.3).

We, then, trigger the tracking pipeline. After the tracking pipeline successfully starts tracking the shape of the object, we activate the use of grippers' 3D coordinates in the tracking pipeline. We select the eight closest lattice nodes to each gripper as the lattice nodes with known coordinates to be used in the tracking pipeline. In this stage, we also select the gripped lattice nodes in the servoing pipeline. We use the same selected lattice nodes with the known coordinates in the tracking pipeline as the gripped lattice nodes. Next, we set the lattice desired shape $\mathbf{s}^*$. To this end, we manually deform the object by moving the robotic arms while grasping the object. This is a natural way of

TABLE 5.1: Parameters of the shape servoing tasks.

Task	Object Type	Object	Template creation	Object point cloud size	Lattice Dimension	Full/partial shape servoing	Main feature
T1.1	Linear	Cable	Blender	1734	15×3×3	f	In-plane deformation, large deformation
T1.2						p	In-plane deformation
T1.3						f	Out-of-Plane deformation
T2.1	Thin-shell	A4 Paper	Blender	1024	8×8×3	f	Large deformation
T2.2						p	Separated servoing regions
T2.3						p	Only translation
T2.4						p	Changing grippers' positions on the object
T2.5		Convoluted foam	Scan	5535	8×8×3	f	Only translation, large deformation
T2.6						p	Only translation, small separated servoing regions
T3.1	Volumetric	Shoe sole	Scan	502	8×4×4	f	Large deformation
T3.2						f	Severe rotation, change of view
T3.3		Foam octagonal cylinder	Blender	2352	8×4×4	f	Twist
T3.4						f	Twist + Bending deformation

defining the desired shape. One can also define the desired shape without the robots holding the object; our approach does not have any constraint in this respect. We store the lattice desired shape $\mathbf{s}^*$ which is corresponding to the object's desired shape $\mathbf{p}^*$. The next step is to manually move the robots to set the initial shapes of the lattice and the object. This is done in the same way as setting the desired shapes. Finally, we start the servoing pipeline to drive the lattice (and thus the object) from its initial shape to the desired shape. As a common occurrence in research robots, the robots' movement might be aborted by reflex errors. This is mainly due to sudden and non-smooth movements. To avoid these reflexes, we gradually increase the gain (from zero to a final constant value) at the beginning of the servoing tasks. We also saturate the velocities (translational and rotational) sent to the robots.

In total, we define thirteen tasks, each with specific features and challenges. Table 5.1 presents the main parameters of each task. We categorize the tasks based on the general form of the object under manipulation. The tasks' results are presented in Figures 5.7 to 5.12. In each figure, the elements corresponding to the current and desired shapes are visualized with green and red colors, respectively. Furthermore, the sections of the object or lattice belonging to the servoed regions are indicated in brighter colors while the ones belonging to the free regions (which are present in partial shape servoing) are indicated in darker colors. Finally, for each task, an RMSE graph (RMS of $\mathbf{e}_s$) illustrates the servoing error during the task. We set the control gain k_p as 0.1 in full shape servoing tasks and 0.05 in partial shape servoing tasks. We tune these gain values empirically. They allow us to obtain good performance while avoiding reflex errors, which cause the robots to stop moving. In the following, we explain the tasks in more detail.

5.5.2 Linear objects

For the experiments with linear objects, we use an electric cable with the length of 70 cm. As for creating $\mathbf{p}^u$, we form a cylinder with the same dimension as the cable. However, instead of considering the whole point cloud on the surface of this cylinder, we consider merely half of the points, i.e., a semi-cylinder. This is done to prevent the inferred object shape and the lattice from rotating around the longitudinal axis during tracking due to axial symmetry. We form a $15 \times 3 \times 3$ size lattice around $\mathbf{p}^u$ in a way that the lattice direction with 15 nodes is in line with the length of the cable. The cable is manipulated from its two ends by the two Franka robots. We define three tasks with this cable: two in-plane, and one out-of-plane. We put a board between the two robots on which we lay the cable. This board serves two purposes; first, the in-plane manipulations take place on the surface of this board, and second, we employ the difference in colors of the board and the cable to filter out unwanted captured point cloud coming from the board. The results of the tasks can be found in Figures 5.7 and 5.8. In Figure 5.7, the plotted points

represent the mean coordinates of the lattice nodes belonging to the cross-sections of the lattice along the cable length. In the following, we explain the tasks in more detail.

- *T1.1.* In this task, we fully servo the cable toward a shape with a large in-plane deformation. In order to make sure that the deformation remains in-plane, while setting the initial and the desired shapes, we keep the robots' grippers parallel to the board at a slight distance above the surface of the board. Furthermore, during servoing, we only send the velocity elements to the robots that keep the robot in the same distance with respect to the board, i.e., two translational velocities parallel to the board and the rotational velocity perpendicular to the board.
- *T1.2.* This task is similar to T1.1. The main difference is that we partially servo the cable. To this end, we divide the cable into 5 sections along its length and select the three middle sections as the servoed region of the object. We then select the lattice nodes encapsulating this region as the servoed nodes of the lattice.
- *T1.3.* This experiment aims to servo the cable through an out-of-plane deformation. To this end, we remove the constraints regarding keeping the grippers' relative pose with respect to the board. This is done by setting the grippers higher in comparison to the board's surface while defining the desired shape. This can be observed in Figure 5.8. We also send the full translational and rotational velocities to the robots.

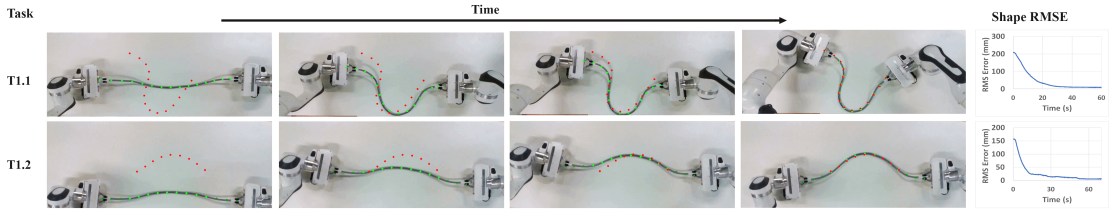


FIGURE 5.7: Tasks with a cable and in-plane deformations. The plotted points represent the mean coordinates of the lattice nodes belonging to the cross-sections of the lattice along the cable length. Green nodes: current lattice shape, red points: desired lattice shape. Top row: full shape servoing, bottom row: partial shape servoing.

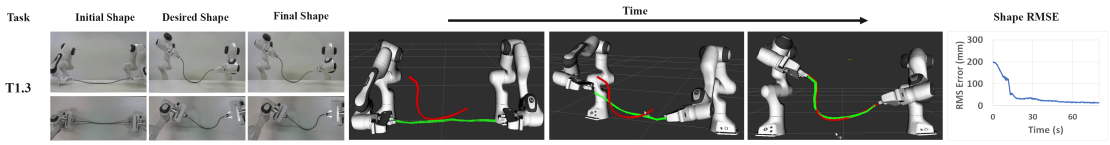


FIGURE 5.8: Task with a cable and out-of-plane deformations. Green: cable's current shape, red: cable's desired shape.

5.5.3 Thin-shell objects

The next experiments are conducted with thin-shell objects. Our objects of interest are a blank A4 paper and a convoluted foam. For both objects, we form a $8 \times 8 \times 3$ size lattice where the 8×8 side is aligned with the surface of the objects and the direction with 3 nodes is in line with the width of the objects. We start with the paper. We define four tasks which are explained in the following. Figure 5.9 presents the results of these tasks.

- *T2.1.* In this task, we fully servo the paper toward a desired shape with large deformation.

- *T2.2.* This task aims to do partial servoing with the paper. To this end, similarly to T1.2, we divide the paper into five sections along its longer dimension. We, then, select the lattice nodes encapsulating the second and the fourth sections of the paper as the servoed lattice nodes.
- *T2.3.* This task is similar to T2.2. There are, however, two main differences: first, the servoed region is smaller, i.e., one-fourth of the object, and second, we merely apply translation to the robots' grippers without any rotation. The latter is done by updating only the translational velocities of the robots.
- *T2.4.* In this task, we do partial shape servoing with two-fourths middle region of the paper as the servoed region. The important change in this task is that we significantly displace the grippers on the paper while setting the initial shape (each in the opposite corner of the paper) in comparison to the desired shape (both in the middle of the paper). We also update the grasped lattice nodes after setting the initial shape. This task is performed with a slightly thicker A4 paper so that the corners of the paper do not entirely loosen due to the large distance from the grippers.

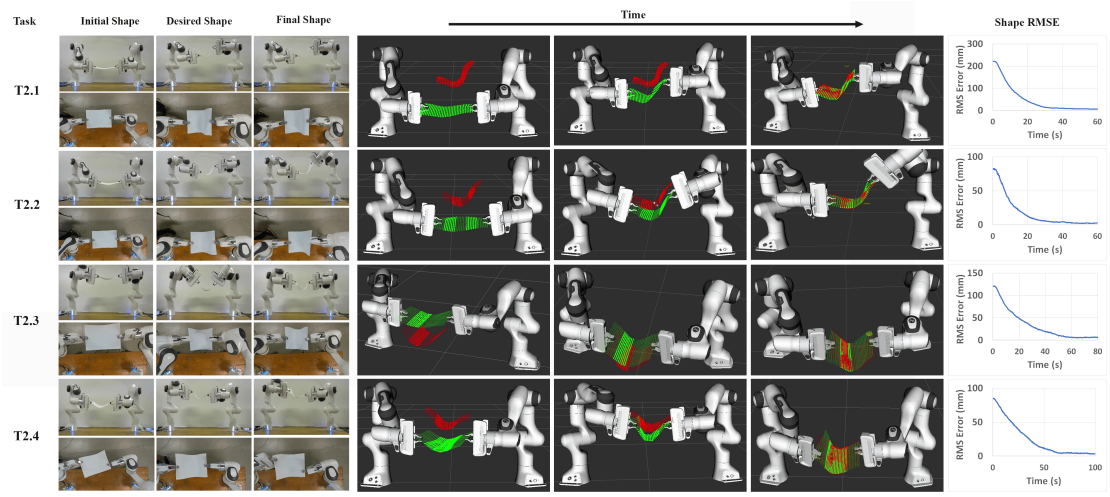


FIGURE 5.9: Tasks with a thin-shell object: an A4 paper. Top row: task with full shape servoing toward a desired shape with large deformation. Second row: task with partial shape servoing. Two separated regions (one-fifth of the paper) are servoed. Third row: task with partial shape servoing. One forth of the A4 paper is servoed. Only translational velocities of the robots are updated. Fourth row: task with partial shape servoing with a slightly thicker A4 paper. Two-fourths middle part of the object is servoed. The grippers are displaced while setting the initial shape.

The next series of experiments are with a convoluted foam that is widely used in packaging industry. The goal is to demonstrate that the surface of the thin-shell object should not necessarily be flat. This is thanks to the generality that using the lattice brings to our approach as we can track and servo objects of any geometry. Another point is that as the convoluted foam is thin and has a low stiffness, it does not follow the rotations of the grippers. We, thus, similarly to T2.3, update merely the translational velocities of the robots. We define two tasks with the convoluted foam which are described in the following. Figure 5.10 presents the results of these tasks.

- *T2.5.* In this task, we conduct full shape servoing toward a desired shape with a large deformation with respect to the initial shape.

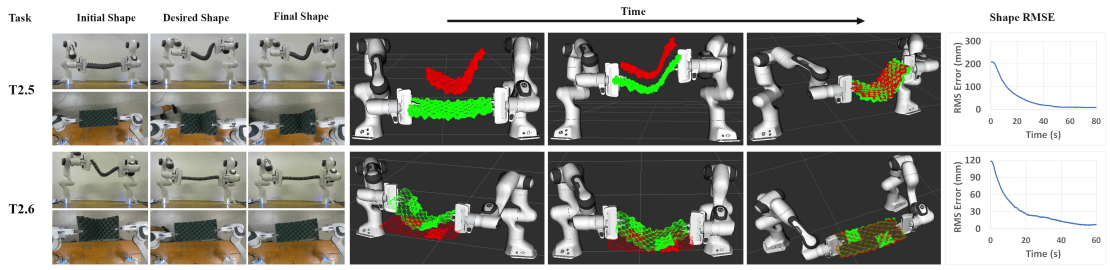


FIGURE 5.10: Tasks with a thin-shell object: a convoluted foam. Top row: task with full shape servoing. Bottom row: task with partial shape servoing. Two small separated regions of the foam are servoed.

- *T2.6.* This task aims to conduct a partial shape servoing with two small separated servoed regions of the convoluted foam. To this end, we initialy select the same servoed regions as in T2.2 and then inversely halve each region relative to the centerline of the convoluted foam. In contrast to the previous tasks, we define the servoed regions' desired shapes in a way that makes the convoluted foam undeformed at the end of the task.

5.5.4 Volumetric objects

The final set of experiments is carried out with two volumetric objects: a bulky shoe sole, and a foam octagonal cylinder. For both objects, we form an $8 \times 4 \times 4$ size lattice where the direction with eight nodes is in line with the longer direction of the objects. We start with the shoe sole. We define two tasks with the shoe sole which are explained in the following. Figure 5.11 presents the results of these tasks.

- *T3.1.* In this task, we fully servo the shoe sole toward a desired shape with a large deformation.
- *T3.2.* This task is similar to T3.1. The main difference is that the desired shape is defined with a severe rotation with respect to the initial shape. This can be observed in Figure 5.11. In fact, the shoe sole should be deformed and flipped during the servoing. Consequently, the part of the shoe sole that is visible in the desired shape is considerably different from the one in the initial shape.

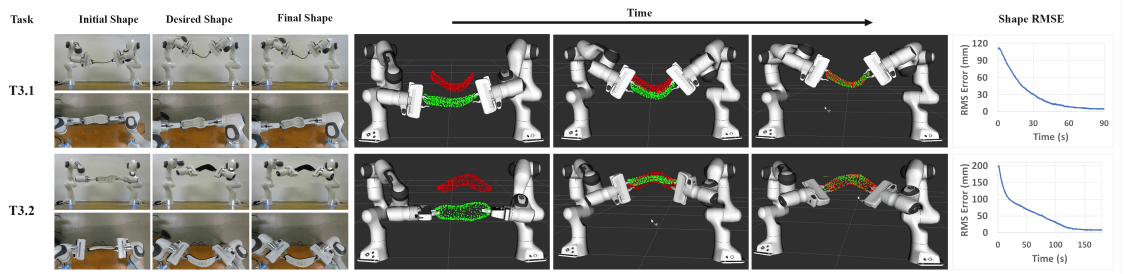


FIGURE 5.11: Tasks with a volumetric object; a bulky shoe sole. Top row: task with large deformation. Bottom row: task with a desired shape with a considerably different view of the object with respect to the initial shape. In this task, the object is rotated and flipped by the shape servoing approach.

As the last set of experiments, we define two tasks with the foam octagonal cylinder. These tasks include twisting the foam. Hence, in order to ensure that the foam follows

the rotations of the grippers, we select the foam to be relatively dense. This applies an intense rotational force to the robots during the servoing. We, thus, considerably decrease the saturation values of the grippers' velocities to avoid reflex errors in the robots. We also paint the foam lengthwise to better illustrate the applied twist. Figure 5.12 presents the results of these tasks. The defined tasks are described in the following.

- *T3.3.* In this task, the desired shape is set by applying a pure twist to the foam.
- *T3.4.* In this task, we apply twist and bending deformation at the same time.

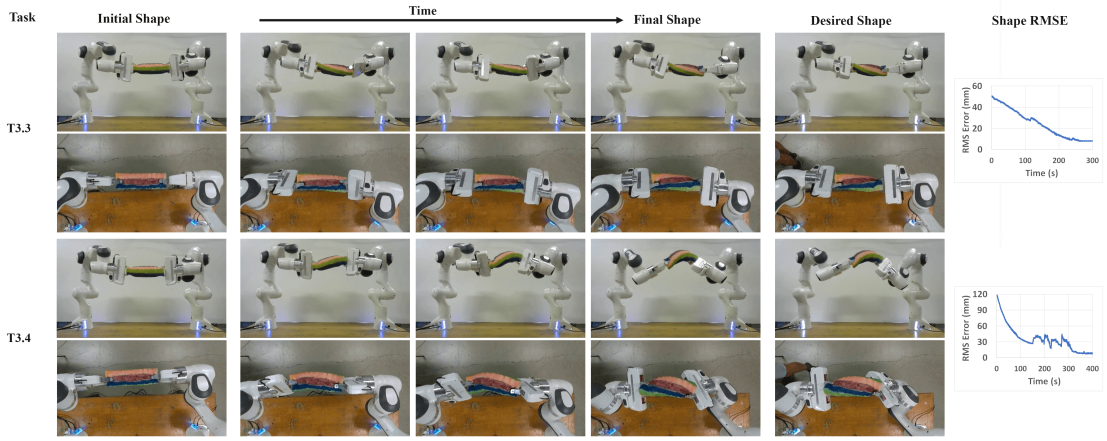


FIGURE 5.12: Tasks with a volumetric object; a foam octagonal cylinder. Top row: task with merely twist. Bottom row: task with twist and bending deformation at the same time.

5.5.5 Results and Discussion

As shown through various experiments, our proposed unified tracking-servoing approach is able to track different forms of the objects and fully and partially servo them toward largely deformed desired shapes. The servoing error graphs in the rightmost side of the Figures 5.7 to 5.12 verify the efficiency of our approach in different scenarios. The variety of objects' materials employed in these experiments confirms the robustness of our proposed approach in dealing with many of the elastic deformable objects around us without having a knowledge of their mechanical parameters. Using a 3D lattice makes it possible to use the same tracking and servoing approach for an object with any form. Furthermore, as shown, our approach can handle tasks with partial shape servoing with one or multiple servoed regions. Defining servoed regions is quite straightforward and is done by just specifying the corresponding servoed lattice nodes encapsulating the object's servoed regions. Another point that should be noted here is that our approach provides full 3D control over the shape of the object; i.e., the object can be simultaneously deformed, rotated and translated. This can be particularly observed in T3.2 where the visible side of the shoe sole in its desired shape is totally different in comparison to the one in its initial shape. Applying twist and bending deformation in T3.3 and T3.4 is another manifestation of this full 3D control. **To the best of our knowledge, no approach in the literature possesses this feature.** The servoing can be even performed in the existence of some noise in the tracking as can be seen in T3.4. Note that the perturbations seen in the middle part of this task are due to tracking noise, and not related to the servoing performance. Next, we discuss several more specific aspects of the implementation and performance of our proposed approach.

Unreachable shapes. There might be cases where the desired shape is unreachable. This can be due to three main reasons: (i) the intrinsic properties of the desired shape, i.e., it is not reachable by the current shape of the object from the current grasping points, (ii) the existing movement constraints in the robots, e.g, the desired shape is defined out of the shared workspace of the robots, and (iii) manipulation constraints, i.e., the interaction between the grippers and the DO is in a way that one or several degrees of freedom are practically lost. An example of the latter is when the object is too soft to follow rotations (T2.5 and T2.6). When dealing with these cases, our proposed servoing pipeline drives the object toward a shape with a small residual error that corresponds with a local minimum. This can be observed in T2.3, T2.5, and T2.6.

To better illustrate the behavior of our servoing approach when the desired shape is unreachable, we conduct two simulations. In these simulations, two robot grippers at both ends of the lattice servo the lattice towards the desired shape. In each time step, we simulate the lattice shape by solving equation 5.2 using ARAP deformation constraints and known 3D coordinates of gripped lattice nodes, without considering any object visual constraints. In the first simulation, we servo the lattice towards a desired shape with different dimensions, resulting in a non-zero residual error. This is shown in Figure 5.13. The simulated current and desired lattices are shown in blue and red, respectively, and tetrahedral meshes are added for better visualization. The second simulation involves

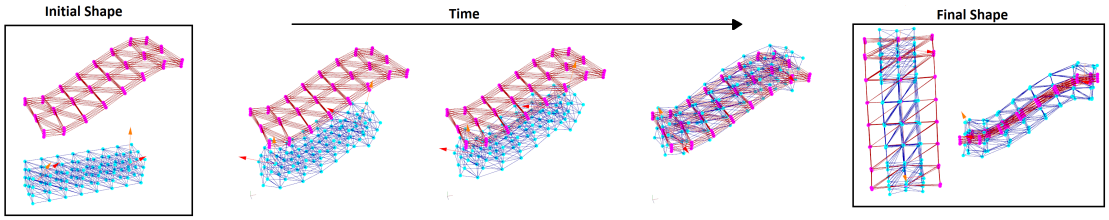


FIGURE 5.13: The simulated experiment where our servoing approach drives the lattice (in blue) towards a desired shape (in red). The desired shape is an unfeasible shape with different dimensions.

reaching a desired lattice shape using translations and rotations. In this configuration, the lattice's shape adapts well to the desired shape. We then attempt to reach the same shape using only translations, resulting in a non-zero residual error. This is shown in Figure 5.14.

Rigid motion. Our servoing pipeline does not make a distinction between rigid and nonrigid components of the shape servoing error. This is the common approach in the state-of-the-art [DBPC18, NAL18, Ber13]. Our extensive experiments are carried out in scenarios close to those of interest in real-world industrial applications, and they involve simultaneous rigid and nonrigid object motions. As our experimental results demonstrate, our servoing pipeline performs with efficiency and accuracy in these scenarios. In cases with a very large rigid motion of the object between the initial and desired shapes, it may be interesting to distinguish rigid from nonrigid components of the shape servoing error. This could enable finer control over the robotic arm motions and the evolution of the object's shape. In this respect, we see no hurdles to combining our approach with a specific module for rigid-body motion handling (which is a well-studied problem).

Computational cost. Regarding the execution speed of our approach, during the experiments, we reached 20-30 FPS for the whole process (tracking and servoing) without any parallelization using only CPU. This range is the same as in ARAP shape servoing from

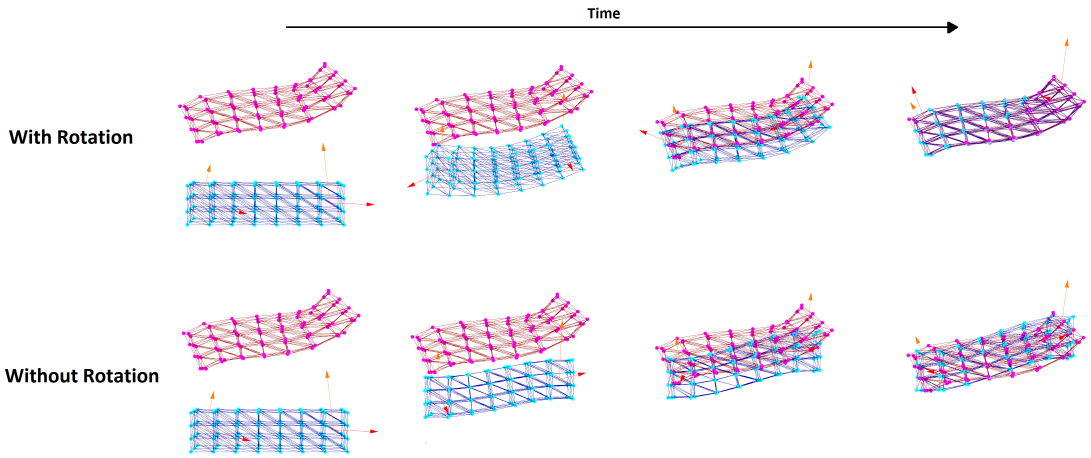


FIGURE 5.14: The simulated experiment where the shape servoing approach drives the lattice (in blue) to a desired shape (in red) with the same dimensions; once with rotation and once without rotation.

Chapter 4. The main difference is that our approach handles both tracking and servoing at the same time while in Chapter 4, the execution speed is reported only for servoing, and tracking is performed by a separate approach. One point that should be noted here is that in calculating the analytical Jacobian, the inversion of matrices in equation 5.22 would be costly when applied to a very large mesh. In our approach, however, this does not cause a problem as we calculate the deformation Jacobian for the lattice whose shape has much smaller size than the object.

Non-convex object geometries. In our experiments, we only considered objects with convex geometries, and we formed a uniform lattice around them. If the object has a non-convex geometry, using a uniform lattice can result in unnecessary geometric constraints between different parts of the object that are not directly connected. For instance, in the case of a doll toy, such constraints may arise between the hands and legs, leading to unrealistic deformations where the movement of one hand affects the position of a leg. To overcome this challenge, a method proposed in [ZSGS12] can be used, where unnecessary links in the lattice are disregarded. This can be achieved by creating a uniform lattice around the non-convex object and then removing all tetrahedral cells and their corresponding lattice nodes that lie entirely outside the input geometry. An example is illustrated in Figure 5.15. We note that our approach can be applied to the non-uniform

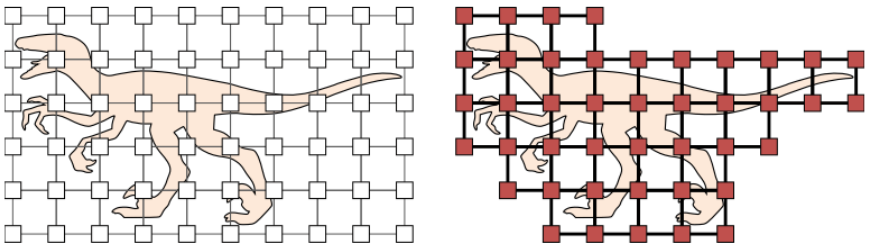


FIGURE 5.15: Uniform lattice vs non-uniform lattice over the geometry of a dinosaur [ZSGS12].

lattice without any modifications. To demonstrate it, we conducted a simulated experiment with a non-uniform lattice. To create a non-uniform lattice, we removed the middle part of a uniform lattice. Figure 5.16 displays snapshots of this experiment throughout

time. The results demonstrate the success of our approach in accomplishing the shape servoing task using the non-uniform lattice.

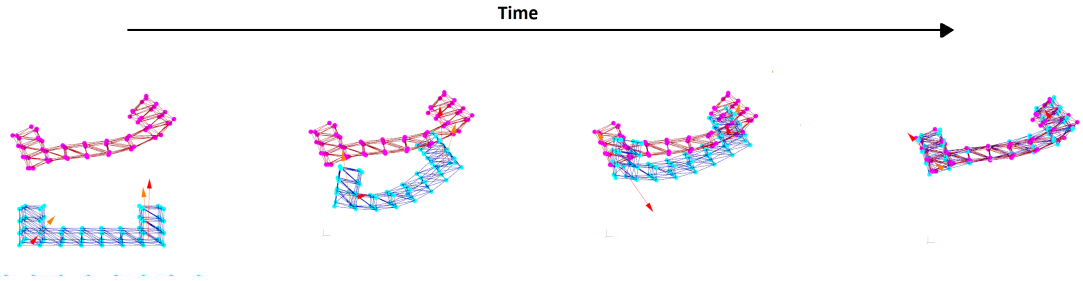


FIGURE 5.16: Servoing a concave lattice toward the desired shape.

5.5.6 Comparison with state-of-the-art

In this section, we compare our shape servoing pipeline with ARAP shape servoing from Chapter 4 through an experiment. Here we call ARAP shape servoing ARAP-SS for short. We design two identical tasks for the two approaches with the thick A4 paper from T2.4. The initial and the desired shapes are considered the same for the two tasks as can be seen in Figure 5.17. This is done by storing the robots' configuration for both initial and desired shapes and keeping the grippers' poses on the object unchanged in the two tasks.

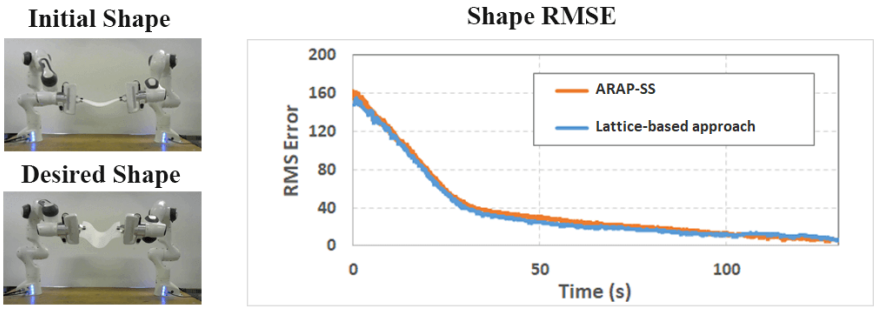


FIGURE 5.17: Comparison between ARAP-SS and our proposed shape servoing pipeline. Left: initial and desired shapes which are identical for both approaches. Right: shape servoing RMS error over time belonging to both approaches.

ARAP-SS was designed for thin-shell objects, as it was based on a surface (not volumetric) deformation model. Thus, we use only the nodes and interconnections on the outer surface of the lattice as the template to ARAP-SS. We thus servo only these lattice nodes from the initial to the final shape. In order to have a fair comparison, we do a partial shape servoing with the same outer lattice nodes with our proposed servoing pipeline. We also use the same gripped nodes and control gains in both approaches. Similarly to the previous tasks, we saturate the translational and rotational velocities sent to the robots. In the right side of Figure 5.17, the shape servoing errors of the two approaches are compared. As seen, no significant difference can be observed between these graphs. This validates the precision of our servoing approach in comparison to the precision (which is state-of-the-art) of ARAP-SS. This precision, along with other features of our proposed approach, including the analytical expression for Jacobian, having full 3D control over the object, and scalability, privilege our approach with respect to other existing approaches.

5.5.7 Failed cases

In this section, we present a failed case of our approach. Our failed case concerns driving the object through its singular shape, i.e., deforming it from an upward-curved to a downward-curved shape or vice versa. In general, this is a complicated task as it requires certain techniques that might differ from one object to another. We show this difficulty by defining tasks with the thick A4 paper from T2.4. These tasks can be observed in Figure

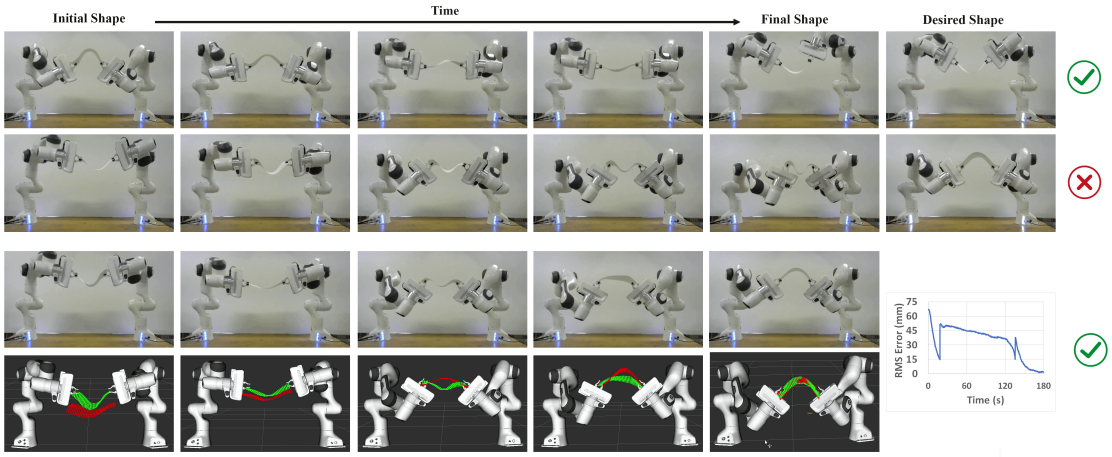


FIGURE 5.18: Tasks for driving a thick A4 paper through its singular shape, i.e., a flat shape. First row: successful task, deforming the paper from an upward-curved shape to a downward-curved shape. Second row: unsuccessful task, deforming the paper from a downward-curved shape to an upward-curved shape (deforming against gravity). Two last rows: successful task, defining two intermediary desired shapes for performing the failed task of the second row.)

5.18. We start with a task in which the initial shape is upward-curved, and the desired shape is downward-curved. This is shown in the first row of Figure 5.18. As indicated, the shape servoing pipeline can successfully drive the object through the flat shape, i.e., the singular shape. It should be noted that, for this task, the direction of gravity is favorable throughout the servoing. In order to make the task more challenging, this time, we try to start from a downward-curved shape and drive the object to an upward-curved shape. This is shown in the second row of Figure 5.18. As seen, despite the severe rotations applied to the paper by the robots, the paper cannot pass through the flat shape. This continues until the robots reach their rotational limits. We show that it is possible to solve this problem using a planning strategy. In particular, this can be done by firstly unwrapping the paper to a certain extent and then applying the required rotation to drive the paper through its singular shape. We applied this approach by defining two intermediary desired shapes for the paper: one nearly unwrapped downward-curved shape, and one nearly unwrapped upward-curved shape. We switch from one desired shape to the next one when the shape servoing RMS error becomes smaller than 15 mm. The last two rows of Figure 5.18 present this process. As seen, using this solution, the task can successfully be carried out.

5.6 Conclusion and future work

In this chapter, we presented a general unified shape tracking-servoing approach that is capable of deforming DOs toward a desired shape. Our approach has full control over the objects of any form (linear, thin-shell, volumetric) and geometry. Next, we mention several limitations of our approach. For objects with low stiffness (like a cloth), it might

be required to incorporate gravity which is not considered in our approach. Besides, reliably driving the object through a singular shape requires additional techniques. Furthermore, handling contact with the environment is not included in our approach. Finally, as we used our Jacobian with a Cartesian velocity controller, there is no guarantee that the robots will always maintain comfortable or safe configurations during the task. This might lead to the failure of the task. A direction for future work is to employ additional constraints for alleviating the last limitation. Considering contact with static objects in the scene is another possible direction of future work. Furthermore, as we use a 3D lattice for any object, it is possible to transfer deformation from one lattice to another and consequently from one object to another. This feature can be useful in motion transfer applications concerning DOs. We also consider using the idea of employing a lattice for improving generalization in reinforcement learning for DOs of any form. An interesting direction for future research would be to apply our proposed approach to control the deformation of an interior region of a volumetric object by deforming its surface. In our current work, we performed all experiments by considering both the interior and surface regions of the object as the servoing regions which were visible throughout the manipulation. However, our approach can be adapted for scenarios where the servoed region of the object is located completely in the interior and is not visible. An example is in surgical applications where an organ needs to be deformed by a tool to drive a non-visible tumor to a particular location. To validate this approach for such cases, specialized tracking techniques capable of tracking the interior region of the object would be needed. A phantom that is transparent and enables the visualization of its interior, with colored tumors embedded inside, could be a potential setup for validating this idea.

Chapter 6

Conclusion and perspectives

In this thesis, we presented solutions for both shape tracking and servoing of DOs. Specifically, we proposed two tracking solutions: one based on monocular vision and another using a 3D camera, and two servoing solutions: one for thin-shell objects and one for objects of any form and geometry, both based on the ARAP deformation model. We conclude this thesis in two sections: shape tracking and shape servoing. We highlight contributions and discuss limitations and possible future work for each section.

6.1 Shape tracking

We proposed two shape tracking methods: `ROBUSfT`, a complete SfT pipeline for monocular 3D shape tracking of isometrically deforming thin-shell objects, and lattice-based shape tracking (linked with lattice-based shape servoing) based on 3D camera.

Chapter 3 presents `ROBUSfT`, a novel CPU-GPU architecture that outperforms existing SfT methods in precision and execution speed. It works in real-time (up to 30 fps), handles large deformations, partial occlusions, and discontinuity in video frames. It is a wide-baseline method that requires only a template of the object, making it instantaneous and not requiring training or fine-tuning. However, `ROBUSfT` has limitations, such as requiring rich texture and being limited to thin-shell objects.

Possible improvements for `ROBUSfT` include leveraging object silhouettes in the image to improve 3D shape inference for weakly-textured objects, and exploring new deformation models to extend the approach to volumetric objects. The current PBD model used in `ROBUSfT` slows down when used with dense meshes. Meshless shape matching is a potential candidate for this thanks to its fast execution speed for modeling volumetric objects. Another direction for future work in SfT methods is to exploit Deep Neural Networks in a more efficient way. As discussed in Chapter 3, the current DNN-based methods are based on classical convolutional neural networks with the input of a single image of the object being deformed. These works require to be thoroughly retrained for a new unseen object [PAP+18, GSVS18, SGTS19, FJPCP+22, FJPCP+21]. One solution proposed in [FJPCP+21] is to feed the texturemap of the object into the network, making it agnostic to the texture and work for unseen texturemaps. However, this method is limited to a specific object form, such as an A4 paper. A possible direction for extending this work is to feed the 3D mesh of the object into the network and train it with many 3D meshes, making it agnostic to the shape of the object. This can be done using Graph Neural Network (GNN) based methods. Using these networks it is possible to consider the object as a mesh and employ the connections between the mesh nodes in the learning process. GNN-based methods show promising results in simulating complex physics using this mechanism [SGGP+20].

To overcome the limitation of being restricted to well-textured thin-shell objects, we proposed lattice-based shape tracking using 3D vision in Chapter 5. Lattice-based shape

tracking is a general method that can handle linear, thin-shell, and volumetric objects. The lattice-based approach decouples the runtime complexity from the geometric complexity of objects, resulting in faster tracking without the need for specialized hardware (running on CPU at around 30 fps). However, lattice-based shape tracking is currently short-baseline and requires initialization for tracking. Future work could involve integrating 2D vision to address the initialization problem and also handle more complex shapes and deformations such as an axisymmetrical cylinder.

6.2 Shape servoing

In Chapter 4, we proposed a simple and robust shape servoing approach for thin-shell DOs called ARAP shape servoing. Unlike existing work, our approach does not require knowledge of the object's mechanical deformation parameters or computation of Jacobian from data collected over a time window, which is prone to noise. We demonstrated the effectiveness of our approach in bi-arm shape servoing experiments involving various deformable objects made of different materials, including paper, rubber, and plastic.

Chapter 5 presented the lattice-based shape servoing approach, which inherits the advantages of ARAP shape servoing while addressing its limitations. Lattice-based shape servoing can handle objects of any form (linear, thin-shell, volumetric) and geometry. It also supports partial shape servoing. Unlike ARAP shape servoing, the computation of the Jacobian in lattice-based shape servoing does not involve any numerical approximation, making it more accurate and reliable. Additionally, the approach is fully scalable and can accommodate any number of grippers. We validated the efficacy of lattice-based shape servoing in various experiments involving objects of different forms, geometries, and materials.

Regarding the limitations of our proposed shape servoing approaches, for objects with low stiffness (like a cloth), it might be required to incorporate gravity which is not considered in our approaches. Another potential future direction is to consider handling contact with the environment, which is not currently addressed in our proposed approaches. Additionally, reliably driving the object through a singular shape may require additional techniques. For example, directly deforming a curved paper from a downward-curved to an upward-curved shape is not always possible, as the paper may get stuck in a flat shape. We address this issue in our experiments with a planning solution. Employing learning-based approaches to learn these techniques near the singular shapes can be another possible solution. Following this, we, in collaboration with the Chalmers University of Technology, employed offline reinforcement learning (RL) to servo a slack rope toward desired shapes, particularly to pass it through the flat shape. We used the setup shown in Figure 6.1 to collect data, and train and test the network. The rope is deformed on a table from its two ends using two grippers of the Yumi robot. We used our tracking pipeline from Chapter 5 to track the rope. We compared our approach with the shape servoing approach proposed by Berenson et al. [Ber13] for ten random downward-curved and upward-curved shapes. Our offline RL approach does a better job than [Ber13] in inverting the direction of the rope and driving it toward the desired shapes as it is shown in Figure 6.2. While the results presented in this work were not optimal, our learning-based approach provides a promising direction for future work to address the challenge of reliably driving DOs through singular shapes. Further research in this area could lead to improved techniques for deforming DOs, particularly in scenarios where conventional shape servoing methods may not suffice.

The last point that we discuss here is that the role of gripper positioning on the object is often overlooked in the literature of shape servoing. In most of the existing work,

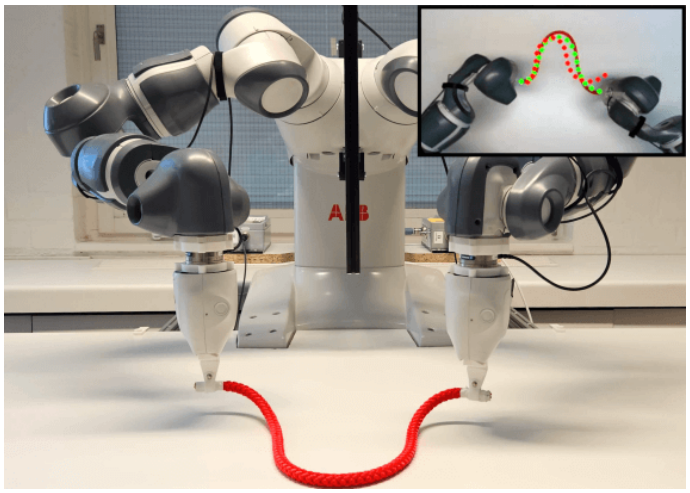


FIGURE 6.1: Dual-arm ABB YuMi robot manipulates a rope on a table. A fixed Intel RealSense camera provides a top-view of the workspace. The field of view of the camera is shown in the top right corner, with the overlay of the current shape tracking in green and the desired shape in red.

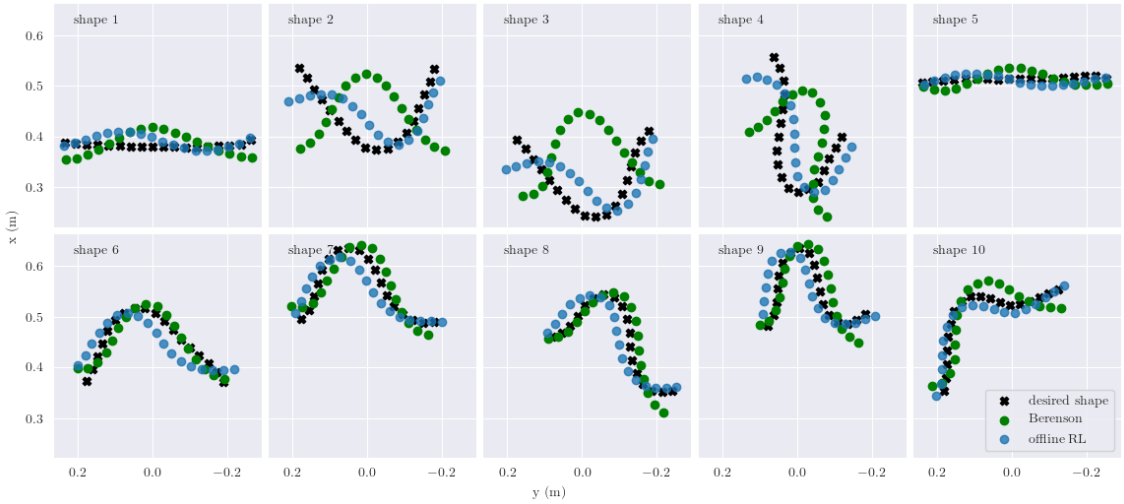


FIGURE 6.2: Comparison between final and desired shapes. Note how the offline RL policy succeeded in inverting the curvature, while the shape servoing appraoch [Ber13] remained at a local minimum.

the gripper positions are predefined and given as input to the shape servoing problem. However, a poor gripper setup may render the deformation task non-feasible. On the contrary, a proper gripper positioning can not only improve the shape servoing performance but may also be decisive for the task's success [CZLNA22, DFTSM22]. Therefore, it is crucial to consider the gripper positioning along with shape servoing approaches to guarantee the success of the task. This aspect could be a promising direction for future research in the field of shape servoing.

Bibliography

[AACR+22] Omid Aghajanzadeh, Miguel Aranda, Juan Antonio Corrales Ramon, Christophe Cariou, Roland Lenain, and Youcef Mezouar. Adaptive deformation control for elastic linear objects. *Frontiers in Robotics and AI*, 9:868459, 2022.

[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)*, 2022.

[ACOL00] Marc Alexa, Daniel Cohen-Or, and David Levin. As-rigid-as-possible shape interpolation. In *Proc. of the 27th Annual Conf. on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, page 157–164, 2000.

[ACRM+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.

[Agi] Agisoft. Agisoft PhotoScan. https://www.agisoft.com.

[APCR+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)*, 2022.

[ARGF+20] Veronica E Arriola-Rios, Puren Guler, Fanny Ficuciello, Danica Kragic, Bruno Siciliano, and Jeremy L Wyatt. Modeling of deformable objects for robotic manipulation: A tutorial and review. *Frontiers in Robotics and AI*, 7:82, 2020.

[ARW17] Veronica E Arriola-Rios and Jeremy L Wyatt. A multimodal model of object deformation under robotic pushing. *IEEE Transactions on Cognitive and Developmental Systems*, 9(2):153–169, 2017.

[AW18] Ibraheem Alhashim and Peter Wonka. High quality monocular depth estimation via transfer learning. *arXiv preprint arXiv:1812.11941*, 2018.

[AWH+19] F. Alambeigi, Z. Wang, R. Hegeman, Y. Liu, and M. Armand. Autonomous data-driven manipulation of unknown anisotropic deformable tissues using unmodelled continuum manipulators. *IEEE Robotics and Automation Letters*, 4(2):254–261, 2019.

[BBH14] Florent Brunet, Adrien Bartoli, and Richard I Hartley. Monocular template-based 3D surface reconstruction: Convex inextensible and nonconvex isometric methods. *Computer Vision and Image Understanding*, 125:138–154, 2014.

[BC00] David Bourguignon and Marie-Paule Cani. Controlling anisotropy in mass-spring systems. In *Computer Animation and Simulation 2000: Proceedings of the Eurographics Workshop in Interlaken, Switzerland, August 21–22, 2000*, pages 113–123. Springer, 2000.

[Ber13] Dmitry Berenson. Manipulation of deformable objects without modeling and simulating deformation. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4525–4532, 2013.

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

[BHB00] Christoph Bregler, Aaron Hertzmann, and Henning Biermann. Recovering non-rigid 3d shape from image streams. In *International Conference on Computer Vision and Pattern Recognition*, 2000.

[BM92] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Sensor fusion IV: control paradigms and data structures*, volume 1611, pages 586–606. Spie, 1992.

[BM14] Timothy Bretl and Zoe McCarthy. Quasi-static manipulation of a Kirchhoff elastic rod based on a geometric analysis of equilibrium configurations. *The International Journal of Robotics Research*, 33(1):48–68, 2014.

[BMM15] Jan Bender, Matthias Müller, and Miles Macklin. Position-based simulation methods in computer graphics. In *Eurographics (tutorials)*, page 8, 2015.

[BMO+14] Jan Bender, Matthias Müller, Miguel A Otaduy, Matthias Teschner, and Miles Macklin. A survey on position-based simulation methods in computer graphics. In *Computer Graphics Forum*, volume 33, pages 228–251. Wiley Online Library, 2014.

[Boo89] Fred L. Bookstein. Principal warps: Thin-plate splines and the decomposition of deformations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(6):567–585, 1989.

[BSSH04] Gérald Bianchi, Barbara Solenthaler, Gábor Székely, and Matthias Harders. Simultaneous topology and stiffness identification for mass-spring models based on fem reference deformations. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2004: 7th International Conference, Saint-Malo, France, September 26-29, 2004. Proceedings, Part II 7*, pages 293–301. Springer, 2004.

[BWG+99] Jeffrey Berkley, Suzanne Weghorst, Hayes Gladstone, Gregory Raugi, Daniel Berg, and Mark Ganter. Fast finite element modeling for surgical simulation. *Studies in health technology and informatics*, pages 55–61, 1999.

[CB15] Toby Collins and Adrien Bartoli. [POSTER] Realtime shape-from-template: System and applications. In *2015 IEEE International Symposium on Mixed and Augmented Reality*, pages 116–119, 2015.

[CBBC16] Toby Collins, Adrien Bartoli, Nicolas Bourdel, and Michel Canis. Robust, real-time, dense and deformable 3D organ tracking in laparoscopic videos. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 404–412, 2016.

[CBEK16] Sergio Caccamo, Yasemin Bekiroglu, Carl Henrik Ek, and Danica Kragic. Active exploration using gaussian random fields and gaussian process implicit surfaces. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 582–589. IEEE, 2016.

[CCB11] Toby Collins, Benoît Compte, and Adrien Bartoli. Deformable shape-from-motion in laparoscopy using a rigid sliding window. In *MIUA*, pages 173–178, 2011.

[CH06] F. Chaumette and S. Hutchinson. Visual servo control. I. Basic approaches. *IEEE Rob. & Aut. Mag.*, 13(4):82–90, 2006.

[CMB14] Toby Collins, Pablo Mesejo, and Adrien Bartoli. An analysis of errors in graph-based keypoint matching and proposed solutions. In *European Conference on Computer Vision*, pages 138–153. Springer, 2014.

[CPB14] Ajad Chhatkuli, Daniel Pizarro, and Adrien Bartoli. Stable template-based isometric 3D reconstruction in all imaging conditions by linear least-squares. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 708–715, 2014.

[CPBC16] Ajad Chhatkuli, Daniel Pizarro, Adrien Bartoli, and Toby Collins. A stable analytical framework for isometric shape-from-template by surface integration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(5):833–850, 2016.

[CR00] Haili Chui and Anand Rangarajan. A feature registration framework using mixture models. In *Proceedings IEEE workshop on mathematical methods in biomedical image analysis. MMBIA-2000 (Cat. No. PR00737)*, pages 190–197. IEEE, 2000.

[CR03] Haili Chui and Anand Rangarajan. A new point matching algorithm for non-rigid registration. *Computer Vision and Image Understanding*, 89(2-3):114–141, 2003.

[CZLN20] Ignacio Cuiral-Zueco and Gonzalo López-Nicolás. Rgb-d tracking and optimal perception of deformable objects. *IEEE Access*, 8:136884–136897, 2020.

[CZLNA22] Ignacio Cuiral-Zueco, Gonzalo López-Nicolás, and Helder Araujo. Gripper positioning for object deformation tasks. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 963–969. IEEE, 2022.

[DB08] Alessio Del Bue. A factorization approach to structure from motion with shape priors. In *International Conference on Computer Vision and Pattern Recognition*, 2008.

[DBPC18] S. Duenser, J.M. Bern, R. Poranne, and S. Coros. Interactive robotic manipulation of elastic objects. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3476–3481, 2018.

[DDF+17] Mingsong Dou, Philip Davidson, Sean Ryan Fanello, Sameh Khamis, Adarsh Kowdle, Christoph Rhemann, Vladimir Tankovich, and Shahram Izadi. Motion2fusion: Real-time volumetric performance capture. *ACM Trans. Graph.*, 36(6), 2017.

[DFTSM22] Cristiana De Farias, Brahim Tamadazte, Rustam Stolkin, and Naresh Marturi. Grasp transfer for deformable objects by functional map correspondence. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 735–741. IEEE, 2022.

[ESP92] Irfan A Essa, Stan Sclaroff, and Alex Pentland. A unified approach for physical and geometric modeling for graphics and animation. In *Computer Graphics Forum*, volume 11, pages 129–138. Wiley Online Library, 1992.

[FBA18] Mahmoud Famouri, Adrien Bartoli, and Zohreh Azimifar. Fast shape-from-template using local features. *Machine Vision and Applications*, 29(1):73–93, 2018.

[FJPCP+21] David Fuentes-Jimenez, Daniel Pizarro, David Casillas-Perez, Toby Collins, and Adrien Bartoli. Texture-generic deep shape-from-template. *IEEE Access*, 9:75211–75230, 2021.

[FJPCP+22] David Fuentes-Jimenez, Daniel Pizarro, David Casillas-Pérez, Toby Collins, and Adrien Bartoli. Deep shape-from-template: Single-image quasi-isometric deformable registration and reconstruction. *Image and Vision Computing*, 127:104531, 2022.

[FMC+18] F. Ficuciello, A. Migliozzi, E. Coevoet, A. Petit, and C. Duriez. FEM-based deformation control for dexterous manipulation of 3D soft objects. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4007–4013, 2018.

[FSS+14] Barbara Frank, Cyrill Stachniss, Rüdiger Schmedding, Matthias Teschner, and Wolfram Burgard. Learning object deformation models for robot motion planning. *Robotics and Autonomous Systems*, 62(8):1153–1174, 2014.

[GCH18] Carsten Griwodz, Lilian Calvet, and Pål Halvorsen. Popsift: A faithful SIFT implementation for real-time applications. In *Proceedings of the 9th ACM Multimedia Systems Conference*, MMSys '18, pages 415–420, New York, NY, USA, 2018. ACM.

[GM97] Sarah FF Gibson and Brian Mirtich. A survey of deformable modeling in computer graphics. Technical report, Technical report, Mitsubishi Electric Research Laboratories, 1997.

[GPIK17] Püren Güler, 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. IEEE, 2017.

[GPSB+22] Victor H. Giraud, Maxime Padrin, Mohammadreza Shetab-Bushehri, Chedli Bouzgarrou, Youcef Mezouar, and Erol Ozgur. Optimal shape servoing with task-focused convergence constraints. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022.

[GSVS18] Vladislav Golyanik, Soshi Shimada, Kiran Varanasi, and Didier Stricker. HDM-net: Monocular non-rigid 3d reconstruction with learned deformation model. In *International Conference on Virtual Reality and Augmented Reality*, pages 51–72, 2018.

[HALN+22] Rafael Herguedas, Miguel Aranda, Gonzalo López-Nicolás, Carlos Sagüés, and Youcef Mezouar. Multirobot control with double-integrator dynamics and control barrier functions for deformable object transport. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 1485–1491. IEEE, 2022.

[HDBC14] Nazim Haouchine, Jérémie Dequidt, Marie-Odile Berger, and Stéphane Cotin. Single view augmentation of 3D elastic objects. In *2014 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 229–236. IEEE, 2014.

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

[HHS+19] Z. Hu, T. Han, P. Sun, J. Pan, and D. Manocha. 3-D deformable object manipulation using deep neural networks. *IEEE Robotics and Automation Letters*, 4(4):4255–4261, 2019.

[HPR+09] Mingxing Hu, Graeme P Penney, Daniel Rueckert, Philip J Edwards, Fernando Bello, Roberto Casula, Michael Figl, and David J Hawkes. Non-rigid reconstruction of the beating heart surface for minimally invasive cardiac surgery. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 34–42. Springer, 2009.

[HSP18] Z. Hu, P. Sun, and J. Pan. Three-dimensional deformable object manipulation using fast online Gaussian process regression. *IEEE Robotics and Automation Letters*, 3(2):979–986, 2018.

[HXR+18] Marc Habermann, Weipeng Xu, Helge Rhodin, Michael Zollhöfer, Gerard Pons-Moll, and Christian Theobalt. NRST: Non-rigid surface tracking from monocular video. In *German Conference on Pattern Recognition*, pages 335–348, 2018.

[HZ03] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[HZSP18] Tao Han, Xuan Zhao, Peigen Sun, and Jia Pan. Robust shape estimation for 3D deformable object manipulation. *Communications in Information and Systems*, 18(2):107–124, 2018.

[IMH05] Takeo Igarashi, Tomer Moscovich, and John F Hughes. As-rigid-as-possible shape manipulation. *ACM Transactions on Graphics*, 24(3):1134–1141, 2005.

[JAT20] Rishabh Jangir, Guillem Alenya, and Carme Torras. Dynamic cloth manipulation with deep reinforcement learning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4630–4636, 2020.

[KFB+21] A Koessler, N Roca Filella, BC Bouzgarrou, L Lequievre, 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.

[LBOK13] Tiantian Liu, Adam W Bargteil, James F O'Brien, and Ladislav Kavan. Fast simulation of mass-spring systems. *ACM Transactions on Graphics (TOG)*, 32(6):1–7, 2013.

[LG15] Zohar Levi and Craig Gotsman. Smooth rotation enhanced as-rigid-as-possible mesh animation. *IEEE Transactions on Visualization and Computer Graphics*, 21(2):264–277, Feb. 2015.

[LHKS19] Jin Han Lee, Myung-Kyu Han, Dong Wook Ko, and Il Hong Suh. From big to small: Multi-scale local planar guidance for monocular depth estimation. *arXiv preprint arXiv:1907.10326*, 2019.

[LK21] Rita Laezza and Yiannis Karayiannidis. Learning shape control of elastoplastic deformable linear objects. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4438–4444, 2021.

[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.

[LLJ22] Naijing Lv, Jianhua Liu, and Yunyi Jia. Dynamic modeling and control of deformable linear objects for single-arm and dual-arm robot manipulations. *IEEE Transactions on Robotics*, 38(4):2341–2353, 2022.

[Low04] David G Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[LPBM20] Jose Lamarca, Shaifali Parashar, Adrien Bartoli, and JMM Montiel. Def-SLAM: Tracking and mapping of deforming scenes from monocular sequences. *IEEE Transactions on Robotics*, 37(1):291–303, 2020.

[LWC+14] Yinxiao Li, Yan Wang, Michael Case, Shih-Fu Chang, and Peter K Allen. Real-time pose estimation of deformable objects using a volumetric approach. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1046–1052. IEEE, 2014.

[LYYA+17] Qi Liu-Yin, Rui Yu, Lourdes Agapito, Andrew Fitzgibbon, and Chris Russell. Better together: Joint reasoning for non-rigid 3D reconstruction with specularities and shading. *arXiv preprint arXiv:1708.01654*, 2017.

[MBC12] Abed Malti, Adrien Bartoli, and Toby Collins. Template-based conformal shape-from-motion-and-shading for laparoscopy. In *International Conference on Information Processing in Computer-Assisted Interventions*, pages 1–10. Springer, 2012.

[MDRB20] Dale McConachie, Andrew Dobson, Mengyao Ruan, and Dmitry Berenson. Manipulating deformable objects by interleaving prediction, planning, and control. *Int. J. of Rob. Research*, 39(8):957–982, 2020.

[MG04] Matthias Müller and Markus H Gross. Interactive virtual materials. In *Graphics interface*, volume 2004, pages 239–246, 2004.

[MHHR07] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. *Journal of Visual Communication and Image Representation*, 18(2):109–118, 2007.

[MHTG05] Matthias Müller, Bruno Heidelberger, Matthias Teschner, and Markus Gross. Meshless deformations based on shape matching. *ACM transactions on graphics (TOG)*, 24(3):471–478, 2005.

[MJD18] Jan Matas, Stephen James, and Andrew J Davison. Sim-to-real reinforcement learning for deformable object manipulation. In *Conference on Robot Learning*, pages 734–743, 2018.

[MM07] Patricia Moore and Derek Molloy. A survey of computer-based deformable models. In *International Machine Vision and Image Processing Conference (IMVIP 2007)*, pages 55–66. IEEE, 2007.

[MMCK14] Miles Macklin, Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. Unified particle physics for real-time applications. *ACM Transactions on Graphics (TOG)*, 33(4):1–12, 2014.

[MS08] Dan Morris and Kenneth Salisbury. Automatic preparation, calibration, and simulation of deformable objects. *Computer methods in biomechanics and biomedical engineering*, 11(3):263–279, 2008.

[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.

[MSCP06] Andriy Myronenko, Xubo Song, and Miguel Carreira-Perpinan. Non-rigid point set registration: Coherent point drift. *Advances in neural information processing systems*, 19, 2006.

[MSJT08] Matthias Müller, Jos Stam, Doug James, and Nils Thürey. Real time physics: class notes. In *ACM SIGGRAPH 2008 classes*, pages 1–90. 2008.

[MT93] Dimitris Metaxas and Demetri Terzopoulos. Shape and nonrigid motion estimation through physics-based synthesis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(6):580–591, 1993.

[NAL18] D. Navarro-Alarcon and Y.-H. 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.

[NALRL13] D. Navarro-Alarcon, Y.-H. Liu, J.G. Romero, and P. Li. Model-free visually servoed deformation control of elastic objects by robot manipulators. *IEEE Transactions on Robotics*, 29(6):1457–1468, 2013.

[NALRL14] D. Navarro-Alarcon, Y.-H. Liu, J.G. Romero, and P. Li. On the visual deformation servoing of compliant objects: uncalibrated control methods and experiments. *International Journal of Robotics Research*, 33(11):1462–1480, 2014.

[NAYW+16] D. Navarro-Alarcon, H.M. Yip, Z. Wang, Y.-H. Liu, F. Zhong, T. Zhang, and P. Li. Automatic 3D Manipulation of Soft Objects by Robotic Arms with Adaptive Deformation Model. *IEEE Transactions on Robotics*, 32(2):429–441, 2016.

[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*, pages 343–352, 2015.

[NMK+06] Andrew Nealen, Matthias Müller, Richard Keiser, Eddy Boxerman, and Mark Carlson. Physically based deformable models in computer graphics. In *Computer graphics forum*, volume 25, pages 809–836. Wiley Online Library, 2006.

[NÖF15] Dat Tien Ngo, Jonas Östlund, and Pascal Fua. Template-based monocular 3D shape recovery using Laplacian meshes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(1):172–187, 2015.

[ÖB17] Erol Özgür and Adrien Bartoli. Particle-SfT: A provably-convergent, fast shape-from-template algorithm. *International Journal of Computer Vision*, 123(2):184–205, 2017.

[ÖVNF12] Jonas Östlund, Aydin Varol, Dat Tien Ngo, and Pascal Fua. Laplacian meshes for monocular 3D shape recovery. In *European Conference on Computer Vision*, pages 412–425. Springer, 2012.

[PAP+18] Albert Pumarola, Antonio Agudo, Lorenzo Porzi, Alberto Sanfeliu, Vincent Lepetit, and Francesc Moreno-Noguer. Geometry-aware network for non-rigid shape prediction from a single view. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4681–4690, 2018.

[Par17] Shaifali Parashar. *Image-based deformable 3D reconstruction using differential geometry and cartan's connections*. PhD thesis, Université Clermont Auvergne(2017-2020), 2017.

[PB12] Daniel Pizarro and Adrien Bartoli. Feature-based deformable surface detection with self-occlusion reasoning. *International Journal of Computer Vision*, 97(1):54–70, 2012.

[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. IEEE, 2018.

[PHB11] Mathieu Perriollat, Richard Hartley, and Adrien Bartoli. Monocular template-based reconstruction of inextensible surfaces. *International Journal of Computer Vision*, 95(2):124–137, 2011.

[PLF08] Julien Pilet, Vincent Lepetit, and Pascal Fua. Fast non-rigid surface detection, registration and realistic augmentation. *International Journal of Computer Vision*, 76(2):109–122, 2008.

[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.

[PPBC15] Shaifali Parashar, Daniel Pizarro, Adrien Bartoli, and Toby Collins. As-rigid-as-possible volumetric shape-from-template. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 891–899, 2015.

[QMZ+21] 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*, doi: 10.1109/TMECH.2021.3126383, 2021.

[RFA11] Christopher Russell, Jamila Fayad, and Lourdes Agapito. Energy based multiple model fitting for non-rigid structure from motion. In *International Conference on Computer Vision and Pattern Recognition*, 2011.

[RSH+99] Daniel Rueckert, Luke I Sonoda, Carmel Hayes, Derek LG Hill, Martin O Leach, and David J Hawkes. Nonrigid registration using free-form deformations: application to breast MR images. *IEEE Transactions on Medical Imaging*, 18(8):712–721, 1999.

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

[SBAMÖ22] Mohammadreza Shetab-Bushehri, Miguel Aranda, Youcef Mezouar, and Erol Özgür. As-rigid-as-possible shape servoing. *IEEE Robotics and Automation Letters*, 7(2):3898–3905, 2022.

[SCOL+04] Olga Sorkine, Daniel Cohen-Or, Yaron Lipman, Marc Alexa, Christian Rössl, and H-P Seidel. Laplacian surface editing. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, pages 175–184, 2004.

[SF09] Mathieu Salzmann and Pascal Fua. Reconstructing sharply folding surfaces: A convex formulation. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1054–1061. IEEE, 2009.

[SF10] Mathieu Salzmann and Pascal Fua. Linear local models for monocular reconstruction of deformable surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(5):931–944, 2010.

[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, 2019.

[SGGP+20] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International conference on machine learning*, pages 8459–8468. PMLR, 2020.

[SGTS19] Soshi Shimada, Vladislav Golyanik, Christian Theobalt, and Didier Stricker. IsMo-GAN: Adversarial learning for monocular non-rigid 3D reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 2876–2885, 2019.

[SHKF12] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. *ECCV (5)*, 7576:746–760, 2012.

[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. IEEE, 2019.

[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. IEEE, 2013.

[SM14] Kirill A Sidorov and A David Marshall. Learnt real-time meshless simulation. In *Computer graphics forum*, volume 33, pages 147–156. Wiley Online Library, 2014.

[SMBB20] Avishai Sintov, Steven Macenski, Andy Borum, and Timothy Bretl. Motion planning for dual-arm manipulation of elastic rods. *IEEE Robotics and Automation Letters*, 5(4):6065–6072, 2020.

[SMEDC+20] Jose Sanchez, Kamal Mohy El Dine, Juan Antonio Corrales, Belhassen-Chedli Bouzgarrou, and Youcef Mezouar. Blind manipulation of deformable objects based on force sensing and finite element modeling. *Frontiers in Robotics and AI*, 7:73, 2020.

[SMNLF08] Mathieu Salzmann, Francesc Moreno-Noguer, Vincent Lepetit, and Pascal Fua. Closed-form solution to non-rigid 3D surface registration. In *European Conference on Computer Vision*, pages 581–594. Springer, 2008.

[Suh09] Jae Kyu Suhr. Kanade-lucas-tomasi (klt) feature tracker. *Computer Vision (EEE6503)*, pages 9–18, 2009.

[TA16] Aggeliki Tsoli and Antonis A Argyros. Tracking deformable surfaces that undergo topological changes using an RGB-D camera. In *2016 Fourth International Conference on 3D Vision (3DV)*, pages 333–341, 2016.

[TCC+12] Quoc-Huy Tran, Tat-Jun Chin, Gustavo Carneiro, Michael S Brown, and David Suter. In defence of RANSAC for outlier rejection in deformable registration. In *European Conference on Computer Vision*, pages 274–287. Springer, 2012.

[TCKH21] Bao Thach, Brian Y Cho, Alan Kuntz, and Tucker Hermans. Learning visual shape control of novel 3D deformable objects from partial-view point clouds. *arXiv preprint arXiv:2110.04685*, 2021.

[THB08] Lorenzo Torresani, Aaron Hertzmann, and Christoph Bregler. Non-rigid structure-from-motion: Estimating shape and motion with hierarchical priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(5):878–892, 2008.

[THMG04] Matthias Teschner, Bruno Heidelberger, Matthias Muller, and Markus Gross. A versatile and robust model for geometrically complex deformable solids. In *Proceedings Computer Graphics International, 2004.*, pages 312–319. IEEE, 2004.

[TT22] Te Tang and Masayoshi Tomizuka. Track deformable objects from point clouds with structure preserved registration. *The International Journal of Robotics Research*, 41(6):599–614, 2022.

[TYGP13] Yuan Tian, Yin Yang, Xiaohu Guo, and Balakrishnan Prabhakaran. Haptic-enabled interactive rendering of deformable objects based on shape matching. In *2013 IEEE International Symposium on Haptic Audio Visual Environments and Games (HAVE)*, pages 75–80. IEEE, 2013.

[VA12] Samuel Vicente and Lourdes Agapito. Soft inextensibility constraints for template-free non-rigid reconstruction. In *European Conference on Computer Vision*. Springer, 2012.

[VSSF12] Aydin Varol, Appu Shaji, Mathieu Salzmann, and Pascal Fua. Monocular 3D reconstruction of locally textured surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(6):1118–1130, 2012.

[WZZ+22] Changhao Wang, Yuyou Zhang, Xiang Zhang, Zheng Wu, Xinghao Zhu, Shiyu Jin, Te Tang, and Masayoshi Tomizuka. Offline-online learning of deformation model for cable manipulation with graph neural networks. *IEEE Robotics and Automation Letters*, 7(2):5544–5551, 2022.

[XLL18] Lang Xu, Yuhua Lu, and Qian Liu. Integrating viscoelastic mass spring dampers into position-based dynamics to simulate soft tissue deformation in real time. *Royal Society open science*, 5(2):171587, 2018.

[YZL21] Mingrui Yu, Hanzhong Zhong, and Xiang Li. Shape control of deformable linear objects with offline and online learning of local linear deformation models. *arXiv preprint arXiv:2109.11091*, 2021.

[ZCD+22] Jihong Zhu, Andrea Cherubini, Claire Dune, David Navarro-Alarcon, Farshid Alambeigi, Dmitry Berenson, Fanny Ficuciello, Kensuke Harada, Jens Kober, Xiang Li, et al. Challenges and outlook in robotic manipulation of deformable objects. *IEEE Robotics & Automation Magazine*, 29(3):67–77, 2022.

[ZGZ+08] Bo Zhu, Lixu Gu, Jingsi Zhang, Zhennan Yan, Lei Pan, and Qiang Zhao. Simulation of organ deformation using boundary element method and meshless shape matching. In *2008 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 3253–3256. IEEE, 2008.

[ZNAPC21] Jihong Zhu, David Navarro-Alarcon, Robin Passama, and Andrea Cherubini. Vision-based manipulation of deformable and rigid objects using subspace projections of 2D contours. *Robotics and Autonomous Systems*, 142:103798, 2021.

[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. IEEE, 2018.

[ZSGS12] Michael Zollhöfer, Ezgi Sert, Günther Greiner, and Jochen Süßmuth. GPU based ARAP deformation using volumetric lattices. In *Eurographics (Short Papers)*, pages 85–88, 2012.