3D Shape Deformation Measurement and Dynamic Representation for Non-Rigid Objects under Manipulation

by

Angel Valencia

Thesis submitted in partial fulfillment of the requirements for
the M.A.Sc. degree in Electrical and Computer Engineering

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

Dexterous robotic manipulation of non-rigid objects is a challenging problem but necessary to explore as robots are increasingly interacting with more complex environments in which such objects are frequently present. In particular, common manipulation tasks such as molding clay to a target shape or picking fruits and vegetables for use in the kitchen, require a high-level understanding of the scene and objects. Commonly, the behavior of non-rigid objects is described by a model. Although, well-established modeling techniques are difficult to apply in robotic tasks since objects and their properties are unknown in such unstructured environments.

This work proposes a sensing and modeling framework to measure the 3D shape deformation of non-rigid objects. Unlike traditional methods, this framework explores data-driven learning techniques focused on shape representation and deformation dynamics prediction using a graph-based approach. The proposal is validated experimentally, analyzing the performance of the representation model to capture the current state of the non-rigid object shape. In addition, the performance of the prediction model is analyzed in terms of its ability to produce future states of the non-rigid object shape due to the manipulation actions of the robotic system. The results suggest that the representation model is able to produce graphs that closely capture the deformation behavior of the non-rigid object. Whereas, the prediction model produces visually plausible graphs when short-term predictions are required.

Acknowledgements

I would like to thank my supervisor, Prof. Pierre Payeur, for letting me to freely explore fascinating topics during my research, for trusting me and giving me the opportunity to work together these last two years. I would also like to thank the committee members of this thesis: Prof. David Mould and Prof. Jochen Lang for their insighful and constructive comments, substantially improving the content of this thesis.

I would like to thank my labmates at SMART laboratory for their company, support and advice throughout my studies.

I would especially like to thank my family, my mom Valeria, my grandparents Bertha and Manuel, my sister Nicolle and my brother Gustavo, who have supported me unconditionally despite the distance.

Finally and most importantly, I would like to express my entire gratitude to Vanessa, my partner in this journey to study abroad. This work could not have been possible without her constant emotional support, helping me find joy and peace during the hard times.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Nomenclature

Abbreviations

ANN	Artificial Neural Network
AR	Augmented Reality
BC-GNG	Batch Continual Growing Neural Gas
C-GNG	Continual Growing Neural Gas
CCD	Charged-Couple Device
DoF	Degrees of Freedom
FDM	Finite Difference Method
FEM	Finite Element Method
GNG	Growing Neural Gas
GNN	Graph Neural Network
GPR	Gaussian Process Regression
HSV	Hue Saturation Value
IN	Interaction Network
ISP	Image Signal Processor
KDE	Kernel Density Estimation
KLT	Kanade-Lucas-Tomasi
KNN	K-Nearest Neighbors
MLP	Multilayer Perceptron
MSE	Mean Square Error
MSS	Mass Spring System
NURBS	Non-Uniform Rational Basis Spline
PBD	Position Based Dynamics

PCA	Principal Component Analysis
QSlim	Quadric Based Simplification
RANSAC	Random Sample Consensus
RGB-D	Red Green Blue-Depth
RMSE	Root Mean Square Error
ToF	Time-of-Flight
TSDF	Truncated Signed Distance Function
VPU	Vision Processing Unit

Chapter 1

Introduction

1.1 Context

There is an ambition towards integrating robots into everyday environments, initially motivated by the automation of exhaustive and dangerous tasks which will allow humans to devote themselves to jobs that require more of their analytical skills and creativity. Another perhaps more interesting conception is the potential for collaborative work, in which humans amplify and augment their capabilities through robot assistance [1]. Therefore, the need to develop a high-level understanding of the environment in order to interact properly is evident. A particular skill that is ubiquitous for human beings and commonly carried out effortlessly is manipulation. The ability to perform complex actions involving an extensive set of objects with different shapes, textures and sizes is still far beyond the capabilities of state-of-the-art robots and remains as an open problem in research.

In simple manipulation tasks, such as pick and place, several simplifications can be made, either on the object or tool used. In contrast, dexterous manipulation increases the difficulty of the task because, in principle, it adds more complexity to the tool employed, which goes from being a simple parallel gripper to an anthropomorphic hand with several degrees of freedom and three or more contact points [2]. On the other hand, it is still possible to simplify the problem by assuming that objects involved can be described as rigid structures. Unfortunately, this assumption does not hold when objects exhibit deformation behaviours, which further increase the complexity by requiring an equally general solution but for a less constrained problem. In particular, it is now necessary to acquire sufficient

knowledge of the object's physics, typically, through the formalization of a model capable of monitoring and predicting its state over time, which will allow to properly plan its grasp and manipulation. Additionally, since deformations are only observed when interacting with the environment, the estimation of such models requires associating sensory signals with robot's actions over time [3].

Most recent advances in dexterous manipulation are mainly attributed to the exploration of learning techniques in simulated environments and transferring this knowledge to the real-world. This approach is attractive because of the relatively easy access to large amount of data generated in simulated environments and faster execution time, thus being only restricted by the available computing resources. It has shown remarkable results in terms of the level of dexterity achieved [4]. Nevertheless, these applications typically involve manipulation of rigid objects with simple shapes which a simulator is capable to provide, yet adjustments into the real-world are often needed which can be a non-trivial process to do. The problem is further complicated under situations when non-rigid objects are present, since existing simulators are not sophisticated enough to provide realistic models [5]. Even so, early attempts to model deformation in dexterous manipulation have mainly adopted classical physics-based approaches, extensively studied in computer simulation, thus providing well-established techniques in this area. Nonetheless, objects are often treated to be of homogeneous composition when using classical methods due to the substantial dependency on the material parameters in these models, which are normally provided in advance since they are supposed to be known and measurable. On the contrary, it is rarely possible to estimate the parameters reliably when handling non-homogeneous materials and unpractical to provide them in advance for every new object encountered in the environment.

Real environments are filled with a vast amount of non-rigid objects (e.g. food, organs, textiles). Conversely, their research has not received the same level of attention as rigid objects. For that reason, there is currently a gap in the level of maturity of the technology as opposed to the rigid variant. This encourages the development of alternative methods that can potentially capture the complexity of the deformations on non-rigid objects without the dependency on simulators or manual tuning of parameters.

1.2 Objectives

This thesis aims to develop a modeling approach, entirely from sensor data, to estimate and predict the state of non-rigid objects under robotic manipulation. Information about objects and their properties is assumed unknown and there is no dependency on simulators or predefined material parameters. In this way, a sensing and modeling framework is proposed to learn the 3D shape deformation[1] of non-rigid objects. The specific objectives include:

- Develop a shape representation model that estimates the current state of the non-rigid object with the ability to capture the deformation behavior using shape tracking and motion analysis.
- Develop a dynamics prediction model that estimates the future states of the non-rigid object using the current shape representation and the manipulation actions of the robotic system.
- Study the integration of the shape representation and dynamics prediction models to learn the deformation of non-rigid objects using sensor data.

The outcomes of this thesis contribute to the development of modern robotic solutions with more autonomy and dexterity for the manipulation of non-rigid objects in real-world unstructured environments.

1.3 Outline

The rest of the thesis is organized as follows: Chapter 2 provides an overview of the most relevant approaches for sensing and modeling the deformation of non-rigid objects from the literature. Chapter 3 describes the shape representation model developed in this research. Chapter 4 describes the integration of the dynamics prediction model. Finally, Chapter 5 presents conclusions and future directions.

[1] '3D shape deformation' or simply 'deformation' from here, unless stated otherwise.

Chapter 2

Related Work

This section provides an overview of methodologies from the literature for modeling the deformation of non-rigid objects. In particular, attention is given to techniques that integrate sensor data in their approaches. Recent comprehensive surveys on the broader scope of manipulation of non-rigid objects are available in [6] and [7], the latter was prepared as part of this thesis.

2.1 Sensing

In the context considered here, sensing is responsible for capturing information about changes of an object's shape along with providing an appropriate representation to use in subsequent processes. In the following sections, current approaches are discussed in terms of the sensing modality used, that is: vision (2.1.1), tactile (2.1.2) or a combination of both (2.1.3). The discussion is further expanded by examining many aspects of design [8], such as the technology and application, localization and interaction in the environment, and data integration. A summary of the characteristics of the sensing strategies is provided in Table 2.1.

2.1.1 Vision

Vision sensing is mostly employed to estimate the shape of a non-rigid object through extraction of surfaces or feature points. Initially, 2D applications used color cameras for the

characterization of linear and planar objects, and then were extended into 3D applications for volumetric objects by incorporating depth sensors. Moreover, some image processing techniques are applied in order to detect the object of interest, which is often simplified by using specific domain knowledge (e.g. color and depth filtering). In particular, this assumption is justified when the environment is controlled and isolated, capturing only interactions between an object and a gripper, thus facilitating the detection process.

Güler *et al.* [9] developed a 2D tracking system applied for the tuning of a position based model. The setup consists of a Logitech HD Pro C920 Webcam that captures a side view of a mechanical screw while pushing a non-rigid object. The object is manually segmented from the image, and then an optical flow algorithm is used to produce a discrete representation of the shape with particles while keeping track of them during the pushing action. Similarly, Hui *et al.* [10] proposed a 2D tracking system but applied it for a material classification task. In this setup, a three-fingered Barrett hand manipulates the object and a Kinect sensor captures color and depth data, though the latter is only used to detect the object of interest via segmentation. This process consists of applying Random Sample Consensus (RANSAC) and K-Nearest Neighbors (KNN) search of k-d trees algorithms in the point clouds to determine several clusters and keeping the one that contains the object. Then, the segmented points are projected back into the image plane for subsequent analysis.

Jordt *et al.* [11] proposed a 3D tracking system for the fitting of a geometric surface. The setup is capable of handling high spatial and temporal resolution by synchronizing several sensors, a SR4000 ToF camera with a Charged-Couple Device (CCD) camera to acquire color and depth data at high frame rate, and a laser line scanner with two CCD cameras to acquire data at high resolution. Object detection is done via classical color-based segmentation. Later in [12], the entire setup is replaced by a single Kinect sensor and used in a tracking system similar to their previous work. However, both approaches are not tested in a robotic scenario. Fugl *et al.* [13] adopted the tracking concept and included a Finite Difference Method (FDM) model to estimate the object's material properties, in which the reference values are taken by a PASCO PS-2189 force sensor and a 6 Degrees of Freedom (DoF) Universal robot arm. Leizea *et al.* [14] also proposed a 3D tracking system for tuning of a Finite Element Method (FEM) model. The setup consists of a 7 DoF Mitsubishi robot with an indenter attached to the gripper, a Kinect sensor that acquires color and depth data and an MCR Anton Paar Physica rheometer to measure material

properties. They combined both visual and material properties to initialize the model and thus estimate the deformation. Additionally, the gripper is equipped with a Mini40 sensor to measure the force exerted by the robot when interacting with the object.

On the other hand, Lin *et al.* [15] presented a strategy to pick-up non-rigid objects while resting on a table. The setup consists of a Barrett hand and a 3D laser scanner from NextEngine. In this application, there is no continuous shape tracking. Instead, the object is initially scanned and converted into a tetrahedral mesh using CGAL[1] library, and later reduced using MeshLab[2] software.

2.1.2 Tactile

Tactile sensing is mainly employed for the estimation of physical properties on the object, such as material parameters and forces exerted by robotic hand. Commonly, raw tactile data from custom sensors is processed to obtain a better representation, either as a vector or array image, and then standard image processing techniques can be used to extract features [16].

Mira *et al.* [17] proposed a grasping strategy which analyzes tactile data to guarantee a safe grasp. In this setup, two 7 DoF Mitsubishi manipulators are used. One robot is equipped with a Shadow five-finger hand with a Tekscan sensor able to record pressure levels throughout an array distributed in the finger sections and the palm. On the other robot, a Kinect sensor is used but only to recognize the object at the beginning of the execution. The same configuration is used by Delgado *et al.* [18] who proposed another grasping strategy. Later, it is updated in [19] by replacing the previous robots with two Kuka LWR arms equipped with two Shadow hands. One robot keeps the Teskcan sensor whereas the Kinect sensor is replaced with a Biomimetic Tactile sensor from SynTouch. For this work, the combination of pressure, vibrations and temperature information along contact points is used to create a new representation for tactile images based on a Gaussian mixture model. Zaidi *et al.* [20] adopted the same setup as in [19]. However, they proposed an alternative grasping strategy. In this case, tactile data is used to extract the information about the applied contact force from the robotic hand to the object.

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

[2] http://www.meshlab.net/

2.1.3 Multi-Modal

Several of the methodologies that study non-rigid objects require the estimation of both physical and shape properties. Therefore, these approaches combine a variety of the sensors and processing techniques previously described in their frameworks.

Arriola-Rios *et al.* [21] integrated tactile and visual data to develop a learning framework to predict deformations of non-rigid objects for pushing actions. The setup consists of a DAQ-FT-GAMA force sensor and a color firewire camera that records perpendicularly to the scene. Visual data is used for a 2D tracking system. They proposed an algorithm called linear snake, which represents the contour as a polygon in which the vertices act as control points adjusted following the deformation, whereas, forces are directly recorded during the continuous pushing actions. Cretu *et al.* [22] combined visual and tactile information to model the deformation using a Multilayer Perceptron (MLP) neural network. Initially, data is collected using a setup described in [23], which consists of a three-fingered Barrett hand equipped with PPS RoboTouch tactile pads and a point Grey Research Flea 2 firewire camera. Visual data is used for 2D shape tracking whereas tactile data is processed to transform from raw measurements to applied forces. The tracking system consists of training a Growing Neural Gas (GNG) neural network to cluster regions of an image based on color and spatial information, given as Hue Saturation Value (HSV) components and coordinates of pixels, respectively. The output of the GNG represents the object of interest and background. In this way, the object is filtered by extracting its contour using a Sobel edge detector. Afterwards, another GNG is trained for sampling purposes, which reduces the number of contour points while preserving details in regions where local deformations occur.

Later, Tawbe and Cretu [24] extended the previous approach from 2D to 3D tracking. In this case, data from a Kinect sensor is segmented to obtain a mesh that represents the object of interest. A RANSAC algorithm is used to identify flat surfaces and eventually remove them from the point clouds. In addition, missing data due to occlusion with the robotic tool is filled using Meshmixer[3] software. The remaining meshes are selectively reduced by using a variation of the QSlim algorithm which simplifies mesh regions that are not in the neighborhood where the interaction occurs. Afterwards, the new mesh is

[3] http://www.meshmixer.com/

clustered according to the distance with respect to the original mesh, then a stratified sampling technique is employed to only retain a subset of data.

Frank *et al.* [25] presented a method to estimate elastic properties by simulating the deformation of a non-rigid object, with visual and tactile information that is acquired from multiple views and combined. The setup consists of a 7 DoF robot manipulator built with Schunk Powercube modules and equipped with a Schunk-FTCL-050 force/torque sensor integrated into the gripper through a wooden stick to minimize occlusions. In the same gripper is attached either a Bumblebee stereo or a PMD-[vision]-O3 Time-of-Flight (ToF) camera. The former is employed for textured objects and the latter for uniformly colored objects since better depth results are obtained. Later, they updated the vision system replacing the stereo camera by a Kinect sensor [26]. This time, the deformation models are learned by the robot and applied to a robotic navigation system.

Petit *et al.* [27] presented a framework that integrated visual and tactile data to estimate material parameters and applied forces by integrating a physics-based model and a data registration process. The setup consists of a Kuka LWR arm equipped with a force sensor in its gripper and an Asus Xtion RGB-D sensor. Force data is used to initially estimate the unknown material parameters of the object measured as Young's modulus and Poisson ratio. Visual data is processed according to a previous work [28], in which a graph cut algorithm is applied to segment the depth data in order to create point clouds of the object of interest, and then scanning the object by moving the sensor around. Gil *et al.* [29] adopted a similar framework as [18], which consists of a Shadow robot hand equipped with a Tekscan tactile sensor on the fingertips and a Kinect camera with an eye-in-hand configuration. In this approach, however, visual data from the Kinect sensor is used to detect and track the deformation while the force measurement is used as a complement in the case that a grasp adjustment is needed. Caccamo *et al.* [30] proposed a system for modeling deformation of elastic surfaces. The setup consisted of a PrimeSense RGB-D camera and a Kinova Jaco24 robotic arm equipped with a three fingered Kinova KG-3 gripper that carried a 3D OptoForce force sensor. RGB-D images are converted to point clouds and then preprocessed to obtain a reduction of the resolution using a crop box. In addition, a statistical filter is applied for outlier removal. On the other hand, the tactile sensor produces force-position vectors that are combined to generate tactile point clouds.

	Robot				Sensor				
		Gripper			Vision		Tactile		
	Manipulator	One-Finger	Two-Finger	Multi-Finger	2D	3D	Force	Pressure	Biomimetic
Fugl *et al.* [13]	✓	✓				✓			
Frank *et al.* [26]	✓	✓				✓	✓		
Leizea *et al.* [14]	✓	✓				✓			
Petit *et al.* [27]	✓	✓				✓	✓		
Lin *et al.* [15]			✓			✓			
Güler *et al.* [9]		✓			✓				
Caccamo *et al.* [30]	✓	✓				✓	✓		
Hui *et al.* [10]				✓		✓			
Cretu *et al.* [22]				✓	✓		✓		
Tawbe & Cretu [24]		✓				✓	✓		
Arriolla-Rios & Wyatt [21]		✓			✓		✓		
Gil *et al.* [29]	✓			✓		✓		✓	
Mira *et al.* [17]	✓			✓				✓	
Delgado *et al.* [19]	✓			✓					✓

Table 2.1: Characteristics of the surveyed strategies for sensing the deformation of non-rigid objects.

2.2 Modeling

In the context of non-rigid objects, modeling typically consists of the definition of a law of evolution for the shape representation [31]. In the following sections, state-of-the-art approaches are further discussed in terms of a proposed systematic categorization, that is: physics-based (2.2.1), geometry-based (2.2.2) and learning-based (2.2.3) models. A summary of the characteristics of the modeling strategies is provided in Table 2.2.

2.2.1 Physics-based

This category includes approaches that model the deformation of a non-rigid object following the law of physics, such as Newtonian dynamics. Early approaches that explore physics-based models applied on interactive robotic environments are demonstrated by Pai *et al.* [32, 33]. A linear model defined by Green's function method is used to estimate the dynamics of the object's boundary. More recently, Fugl *et al.* [34] proposed a 3D model to describe linear deformations based on the Navier-Cauchy equations and considering an elastic and isotropic material. The mechanical response is formulated from Hooke's law using the Navier-Cauchy equations. Also, some boundary conditions are defined considering the case of robotic manipulation since the model is used for simulation only. An FDM is formulated for the Navier-Cauchy equations to be approximated with a discrete set of algebraic equations. Similarly, another model that describes linear deformations is presented by Fugl *et al.* [13] based on a discretized Euler-Bernoulli model and also developed for an elastic and isotropic material. The physics model is able to compute the deformation curve as a function of Young's modulus, the mesh geometry and the gripper pose. Afterwards, a comparison between the deformed model and the input data is synthesized by an error function.

Frank *et al.* [26] proposed a 3D model built on a previous work [25] which consists in the estimation of the elasticity properties, described by Young's modulus and the Poisson ratio considering an elastic and isotropic material. The data from the sensors is used to formalize a linear FEM of the discretized tetrahedral mesh, which relates the external forces acting on the nodes of the mesh and the consecutive displacement with the elasticity parameters via the stiffness matrix. Then, the FEM is initialized with a given stiffness matrix and the problem is solved by comparing the observed and simulated displacement and minimizing their difference. The robotic surgery application of Leizea *et al.* [14] used the acquired color and depth information as the input to a non-linear FEM model to simulate the deformation behavior. As a preliminary step, the system needs to compute the density, Young's Modulus and Poisson ratio parameters of an elastic and isotropic material. Additionally, a set of key-points that relates the input data with a 3D mesh of tetrahedrons is also defined. Afterwards, the model is initialized as a Saint-Venant-Kirchhoff formulation within a FEM structure using a tetrahedral mesh that is updated in

every frame. This process is done by matching selected key-points with the input raw point clouds to find associations between them and discarding those key-points that present no deformations. In this way, the FEM model uses the previous association to displace the mesh until it captures the current state of deformation of the object.

Petit *et al.* [28, 35, 27] applied a set of techniques to model the deformation of elastic and isotropic material by combining point cloud representation of an object of interest with a linear FEM model of a tetrahedral mesh. This method uses a previously segmented point cloud which is applied to a rigid Iterative Closest Point (ICP) algorithm to estimate a transformation from the point cloud to the mesh. Afterwards, a registration procedure is performed by computing external forces exerted by the point cloud with respect to the nodes of the mesh and associating them with the internal forces computed from the visual and force data. Thus, the estimation of the deformations consists in solving a dynamic system of linear ordinary differential equations involving the internal and external forces. Later, Ficuciello *et al.* [36] used the FEM model estimation of [27] and included a deformation control strategy. Thus, this approach minimizes the error between the observed and desired shape and provides as output the contact forces between the fingers of the hand and the object, which are converted into position trajectories of the fingertips.

Lin *et al.* [15, 37] presented a squeeze grasping approach in which the deformation is described by analyzing the displacements of the points in contact with the object. Based on a previous approach [38] but adapted for a 3D representation, they formulated a linear FEM model of tetrahedral mesh for an elastic and isotropic object. Similarly, a set of external forces are applied to the nodes of the mesh displacing their location when deformation occurs, although, the influence of gravity is also included by combining the proportional contribution of the mass of each tetrahedron. Moreover, Duenser *et al.* [39] proposed to use a FEM model to characterize the deformation of elastic objects. Their formulation also establishes a sensitivity analysis to calculate a Jacobian function, which is required as a measure of changes in robot's joint angles and the corresponding shape deformation. This optimization speeds up the generation of control signals for their model, and thus is able to perform shape control in real-time scenarios.

On the other hand, the grasping strategy of Zaidi *et al.* [20] considered known isotropic objects represented as a non-linear Mass Spring System (MSS) in a tetrahedral mesh, in which lumped masses and non-linear springs are attached to the nodes and edges respec-

tively, as described in [40]. The description of the global and contact areas of deformation are based on the tracking of the positions of the nodes by solving the dynamic equation of Newton's Second law.

2.2.2 Geometry-based

This category includes approaches that model the deformation as a geometric surface problem, in which displacements are analyzed directly instead of relying on forces (as in Section 2.2.1). A parametric representation known as Non-Uniform Rational Basis Spline (NURBS) is used by Jordt *et al.* [11] to describe and store the surface of an object. First, they extract a set of 2D Kanade-Lucas-Tomasi (KLT) features in the low resolution image data to create a dense 2D deformation map that is then fitted into a 2D NURBS function. The 3D surface deformation function is created using the previous 2D NURBS, the depth data and a global pose descriptor forming a 3D NURBS. Finally, the deformation sequence is propagated to a high resolution mesh by mapping the vertex position in 3D to the corresponding NURBS coordinates. Afterwards, Jordt and Koch [12] improved their model by removing the necessity to explicitly detect tracking features. Instead, the algorithm defines a 3D mesh from the initial frame and registers it into a NURBS surface function, such that the mesh deformation is manipulated by the NURBS control points. Then, it is fitted to the current measurements by minimizing an error function between the mesh and the color and depth data.

With the intention of improving the stability issues of physics-based models, Güler *et al.* [9] used a Position Based Dynamics (PBD) approach to simulate the deformation. The technique, originally presented in [41], represents the object as a set of particles with some initial configurations. In this approach, there is no requirement of connectivity information between particles as opposed to mesh-based methods. Thus, an optical flow-based algorithm is used to spread and keep track of a set of particles on real object over consecutive frames. In principle, the model is able to simulate several levels of deformation, however only a 2D object experimenting shear and stretch-like deformations is considered, which is controlled with a parameter. Later in [42], this formulation is updated to include a FEM model for generating ground truth data of various deformation parameters. The robustness of the system improves by adding quadratic deformation (i.e. twist and bend). Similarly,

volume conservation during deformation is also included with a new control parameter, which is estimated by matching the PBD and the FEM deformation models. The latter is built upon Young's modulus and the Poisson ratio for a linear elastic and isotropic material. Experiments show that by adding this new control parameter, it is possible to relate with the elasticity parameter of the FEM model. In complement, Caccamo *et al.* [30] proposed to train a set of Gaussian Process Regression (GPR) to estimate the deformability of the object, and then integrate that information into the PBD methodology in order to produce the simulation of the deformation, which is applied to an elastic heterogeneous object.

Gil *et al.* [29] presented a grasping strategy that uses color and depth data and geometric information to be able to model the shape deformation of a planar object. This method, originally described in [43], consists in clustering the point clouds into subsets named patches, characterized by the size and number of points contained. Then, the curvature maps between every point of a patch are estimated by analyzing the eigenvalues of the points in a neighborhood using Principal Component Analysis (PCA). Furthermore, a curvature histogram is defined and represents the distribution of the variation. It is used to detect singular points, which are those with maximum curvature values. Finally, by combining both formulations the system is able to detect deformations by finding the critical points where variations occur. As can be seen, tactile information is not part of the modelling process, instead a vision algorithm is proposed for a grasp planner. Furthermore, the latter deformation descriptor is used in Mateo *et al.* [44] as a part of their proposed surface supervision method that includes volumetric object representation. Additionally, the curvature surface approach is formalized as Curvatures Skeletons and described in detail in [45]. Alternatively, Hui *et al.* [10, 46] used an implicit geometric representation for tracking the contour of a non-rigid object. The proposed approach is based on the fast implementation of the level set method, and takes as input the previously segmented image of the object of interest transformed to the YUV color space and represented in the log-polar domain. This method collected 3D data, however, the depth information is only used for the segmentation process since the modelling approach is developed in 2D only.

2.2.3 Learning-based

This category includes approaches that combine learning techniques for implicit or explicit modeling. The latter refers to learning unknown parameters in already defined models (e.g. physic or geometric). Whereas the former refers to a complete learning of the characteristics of the deformation with no previous model provided.

Cretu *et al.* [22] proposed to learn an implicit model using neural networks. The system used an MLP which receives as input a reduced representation of the object's contour as described in Section 2.1.3. Later, force and position measurements of the robotic hand are associated with the tracked contour to create a feature vector, then the MLP is trained to predict the position of the points in the contour of the next frame. The model is evaluated by using different forces while keeping fingers' position unchanged. Later in [47, 24], they extended the approach to handle volumetric objects while keeping the prediction phase almost unchanged. In this case, a series of MLPs are trained per each sampling cluster of the object mesh in order to learn the relationship between the forces and the position of the GNG shape representation. Even though original work presents the first attempt to learn an implicit deformation model using neural networks, results show predictions for the next frame only. Therefore, a long-term response of the model behavior is not explored.

Arriola-Rios and Wyatt [21] proposed to combine two models in sequence in order to predict the object's shape and reaction forces respectively. First, the proposed shape predictor uses the shape tracking information to initialize a mesh for an MSS model, based on a modified version proposed in [48] that considers elastic and plastic deformations. Nevertheless, the focus of the shape predictor is to learn the parameters that better estimate an MSS model. Thus, an evolutionary algorithm is proposed to search for the parameter space of the model. Similarly, in order to train the force predictor, the position and forces from the sensory information are used to obtain a stress-strain diagram which are approximated by implementing a regression model. As opposed to [22, 24], this model shows predictions for the object behavior in multiple steps forward, and also classifies new materials if added. However, the model is only tested for 2D deformations.

Cherubini *et al.* [49] proposed a novel framework for learning to manipulate a malleable plastic object to a target shape. In this work, the authors initially defined a set of *actions* (e.g. pushing, tapping and incising), such that each action affects only a subset of the

object's state, the latter is measured using the 2D contour, and then embedded as a feature vector. The current shape state is defined through a function that depends on the previous state and the external action applied. Thus, the problem is reduced to the minimization of the error between the actual and desired state. However, since no formal description for the state function is provided, an MLP neural network is used to implicitly learn this behavior by using image data of humans performing the task.

Hu *et al.* [50] proposed to use a GPR model to learn a deformation function that describes the relationship between gripper movements and point displacements over a sequence of manipulation tasks. In this work, the object is represented as a set of categorized 3D points associated to a particular surface region. In turn, a feature vector with the region information is defined to monitor the state of the object. Later, this approach is expanded in [51] by including a PCA formulation to reduce the dimensionality of the feature vector. Moreover, an MLP neural network is used to learn the deformation function, demonstrating better performance than the previous GPR model. In addition, an occlusion removing algorithm is also formulated to obtain more complete information about the state of the object, and thus aiming to overcome the problems of learning models in partially observable environments (e.g. when the robot arm occludes an object during manipulation).

Recently, Li *et al.* [52] proposed to learn a particle simulation engine with the purposes of describing the dynamics of complex interactive systems. The approach is based on a model proposed by Battaglia *et al.* [53], which uses a Graph Neural Network (GNN) model to make inference about the state of a physical system. Later, Mrowca *et al.* [54] updated a previous model to support 3D instead of 2D scenes, while providing more evaluations of complex environments (e.g. non-rigid objects and liquids). Although the work of Li *et al.* [52] is not the first attempt to implicitly learn the dynamics of non-rigid objects using GNNs, they extended previous works by showing results beyond simulation engines, in which a real robotic gripper is molding a non-rigid object to a target shape. Finally, these architectures differ from plain neural networks due to the added relational inductive bias of graph structures, providing a useful mechanism to represent an interactive system such as manipulation of non-rigid objects.

	Object				Model						
	Type		Interaction		Dimension		Complexity		Deformation		
	Planar	Volumetric	Single	Multiple	2D	3D	Linear	Non-Linear	Elastic	Plastic	All-Purpose[1]
Fugl *et al.* [13]		✓		✓		✓	✓		✓		
Frank *et al.* [26]		✓	✓			✓	✓		✓		
Leizea *et al.* [14]		✓	✓			✓		✓	✓		
Petit *et al.* [27]		✓	✓			✓	✓		✓		
Lin *et al.* [15]		✓		✓		✓	✓		✓		
Jia *et al.* [38]	✓			✓	✓		✓		✓		
Zaidi *et al.* [20]		✓		✓		✓		✓			
Güler *et al.* [42]		✓	✓		✓		✓		✓		
Gil *et al.* [29]	✓			✓		✓			✓		
Mateo *et al.* [44]		✓		✓		✓					✓
Hui *et al.* [10]		✓		✓	✓						✓
Cretu *et al.* [22]		✓		✓	✓						✓
Tawbe & Cretu [24]		✓	✓			✓					✓
Arriola-Rios & Wyatt [21]		✓	✓		✓			✓	✓		
Cherubini *et al.* [49]		✓		✓	✓			✓		✓	
Hu *et al.* [50]	✓			✓	✓			✓	✓		
Li *et al.* [52]		✓		✓		✓		✓			✓

[1] approaches that do not state the type of deformation.

Table 2.2: Characteristics of the surveyed strategies for modeling the deformation of non-rigid objects.

2.3 Summary

This chapter provides a comparative review of the state-of-the-art and discussions of related literature on modeling the deformation of non-rigid objects that help position our proposed work. It is observed that, traditional modeling approaches are often hand tuned by using either a physics-based (e.g. MSS, FEM) or a geometry-based (e.g. NURBS, PBD) formulation. However, a proper description of a model for each random object

found in the environment is a non-trivial process (e.g. objects with complex shapes and unknown non-homogeneous materials), which presents a clear opportunity for development of learning-based models.

Moreover, current approaches rely on shape tracking as input for the estimation of models. Even though several techniques are currently available, these are directly influenced by the specification of the modeling framework. For example, linear snake is used by Arriola-Rios *et al.* [21] to create a mesh-like representation. On the other hand, optical flow and GNG are used by Güler *et al.* [9] and Cretu *et al.* [22] respectively to create a particle-like representation. In such cases, the structure of the shape representation is particularly associated with the type of deformation model used. Similarly, Li *et al.* [52] use a graph-like representation, though shape tracking is not implemented in their approach. Instead, the object is first scanned and reconstructed using a Truncated Signed Distance Function (TSDF) fusion algorithm. Therefore, information about the object's shape is provided at the beginning, and then training is done entirely in simulation.

In this work, a framework is proposed for implicit learning of the deformation model, which combines both shape representation and deformation dynamics prediction [55]. Also, vision sensing is considered as the main source of information for the estimation of our model. This is motivated by the promising results of recent learning-based models to act as a physics engine, even though little exploration of training using sensor data has been achieved. Such an investigation is necessary due to the little support of non-rigid objects in current robotic simulators, in particular for large volumetric deformations and dexterous hands. The lack of such functionalities prevents the direct use of data-driven approaches to learn a model trained entirely in simulation. More specifically, a GNG-based model is used to produce the shape representation of non-rigid objects as a graph while tracking the shape over the sequence of frames. Then, the contact points information is added to the representation and passed to a GNN-based model in order to learn the dynamics of the deformation. Thus, a forward prediction of the state of the non-rigid object is modeled, and trained using real-world measurements.

Chapter 3

Shape Representation

As discussed in Chapter 2, current approaches based on sensor data in some way rely on shape tracking as a mechanism to estimate the deformation models. In particular, the object motion variations are closely related to the dynamics of the deformation. Therefore, a motion analysis can be used to determine whether the produced shape representation is consistent with the deformation and reflects the current state of the non-rigid object. In this way, the structure of the sensor data is first examined, and then the design consideration of the representation model are established.

Visual data produced by RGB-D sensors is commonly represented as a set of non-uniform points in 3D space, also known as point clouds. Unlike more traditional shape representations (e.g. polygonal mesh), this primitive lacks of an explicit relational mechanism to characterize the underlying structure of the data. Therefore, it is necessary to apply an additional step to construct graphs from point clouds. Moreover, due to the nature of the sensor data, some design considerations [56] of the graph construction algorithm should be defined in order to perform shape tracking and motion analysis that satisfy its practical application in robotic manipulation tasks (e.g. requirements for planning and control [57]). The latter can be formulated as follows:

- **Real-time execution:** The sensor data provides information about changes of the non-rigid object shape. The representation model should process sensor data in real-time in order to capture a correct interpretation of the object motion, thus preventing outdated information from generating incorrect interaction associations.

- **Temporal smoothing:** The signals from current sensors are inherently noisy and captured at high spatio-temporal resolutions. The representation model should operate as a smoothing filter for noise reduction in order to efficiently stabilize the displacements associated to the object motion.

- **Region correspondence:** The correspondence of point clouds to graph nodes is directly used to calculate the object motion. The representation model should generate a 3D to 3D region correspondence that continually associates a localized area of the non-rigid object shape over time.

In principle, point clouds can be easily represented as a graph by reconstructing them as a polygonal mesh. However, direct mesh conversion is not sufficient to satisfy the considerations previously established. Alternatively, capabilities for representation of dynamic distributions are observed in GNG [58]. Inspired by this behavior, several works have proposed tracking systems in 2D using images [59, 60], and more recently in 3D using point clouds [61], with reliable motion analysis. As for the existing filtering methods [62], GNG has been noted primarily for downsampling capabilities and for handling point clouds [63], which demonstrated better quality of representation compared to other methods such as voxel grids. Additionally, different algorithm adaptations have been proposed to achieve real-time execution, exploiting aspects such as computational optimization and hardware acceleration [61].

The rest of this chapter describes the formulation of the representation model that constructs dynamic graphs, G, from non-stationary distributions, P, constituting the deformation that a non-rigid object undergoes during manipulation. The methodology used to develop the model is summarized in Figure 3.1. Particularly, a GNG-based model called BC-GNG is proposed for the graph construction process, which combines and extends several of the previous works (Section 3.1). The configuration of the sensing and processing phases for collection of data on a non-rigid object is also described (Section 3.2). In addition, a comparative experimental evaluation between the different variations of GNG is also provided (Section 3.3).

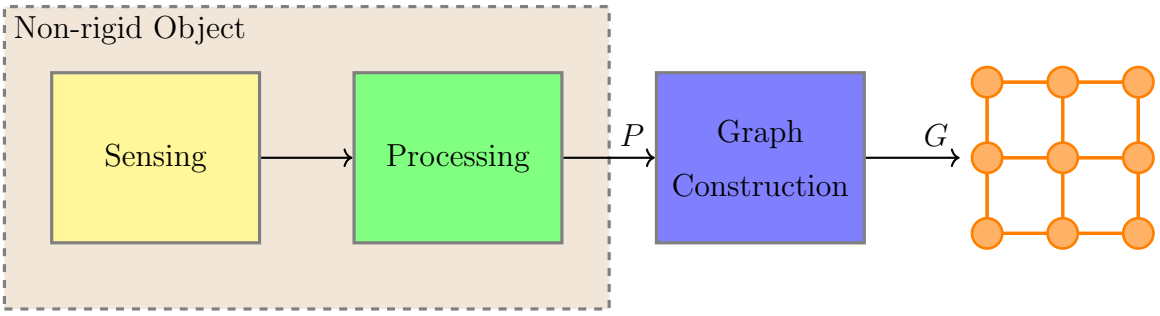


Figure 3.1: Diagram of the proposed methodology for the representation model: (Yellow) raw data taken from sensors; (Green) data processing for object detection; (Blue) GNG-based model for shape representation.

3.1 Graph Construction

First introduced by Fritzke [64], *Growing Neural Gas* (GNG) is a growing self-organizing artificial neural network that belongs to the family of unsupervised learning algorithms. This model aims to learn a structured representation of some data distribution as a graph. As opposed to a traditional Artificial Neural Network (ANN), the network structure in GNG is not fixed but rather dynamic and the training consists of competitive rather than gradient-based learning. Such differences provide GNG an interesting property of better preserving the topology (i.e. neighborhood relations) of the distribution [65]. In addition, GNG incorporates a mechanism that enables to dynamically control the growth of the structure, which unlike the original formulation of the Neural Gas network [66], provides more flexibility for the representation of unknown distributions.

3.1.1 Growing Neural Gas

The original GNG formulation receives as input the data distribution P of size N and produces as output an undirected graph $G = (O, R)$. Where, $O = \{\mathbf{o}_i\}_{i=1:N_O}$ is the set of nodes with N_O cardinality, and $R = \{\mathbf{r}_k, u_k, v_k\}_{k=1:N_R}$ is the set of edges with N_R cardinality, which connects an unordered pair of nodes u_k and v_k. Also, each node has an associated feature vector $\mathbf{o}_i = \{x_i, e_i\}$, which contains the position and spatial error, respectively. Likewise, each edge has an associated feature vector $\mathbf{r}_k = \{a_k\}$, which contains the connection age. The position is a direct measure of the spatial location of

a node with respect to the input distribution, while the spatial error and connection age serve as measures for the addition and removal processses of nodes and edges from the graph.

Similar to other learning algorithms, GNG training consists of an iterative procedure, which ends when a certain quality condition is reached. Commonly, a metric based on the number of nodes in the graph is used as stopping criterion. However, this brings an issue when size and shape of the input distribution is unknown since the user is responsible for establishing such conditions, which implies distribution-specific manual adjustments. In this work, the quantization error, QE, is used (Equation 3.1) instead of the number of nodes as stopping criterion. In competitive learning, this metric is evaluated over the data distribution P and computes the average of the distances between the winner node produced by the model, o_{s_1}, and the associated sample, ξ. The quantization error provides more control on the final representation, because nodes are dynamically created rather than being set at a maximum limit in advance. Hence, implicit sampling is performed by establishing the presence of nodes with finer or coarser density according to the characteristics of the input distribution.

$$\mathrm{QE} = \frac{1}{N} \sum_{\xi \in P} \|o_{s_1} - \xi\| \tag{3.1}$$

Therefore, the graph construction using the quantization error as stopping criterion is obtained by following the execution of Algorithm 1. First, the current data frame is received as distribution, P, and then the graph, G, is initialized. The latter process consists of creating two nodes with position set to random values (i.e. those chosen within the bounds of the distribution) and spatial error set to zero. In addition, an edge connecting these nodes is created with age set to zero. After initialization, an individual sample, ξ, is randomly drawn from the distribution, and then the ADAPTATION and GROWING phases are run iteratively until the desired maximum quantization error, $\mathrm{QE}_{\mathrm{max}}$, is reached. During the former, nodes and edges features are sequentially updated, while during the latter and after receiving a certain number of samples, λ, new nodes and edges are added incrementally to the graph. In other words, these phases control the evolution and size of the representation, respectively.

Algorithm 1 Steps of computation in GNG

Input: P

Output: G

$G \leftarrow \text{init\_graph}(P)$
while $\text{QE} > \text{QE}_{\max}$ **do**
  for all $n \in N$ **do**
    $\xi \sim P$
    ADAPTATION(G, ξ)
    if $(n \bmod \lambda) = 0$ **then**
      GROWING(G)
    end if
  end for
end while

In the ADAPTATION phase (Algorithm 2), nodes features are updated as follows:

I $\mathbf{o}_{s_1}, \mathbf{o}_{s_2}$: two closest nodes (winners), s_1, s_2, these are obtained by sorting in descending order the Euclidean distance between all node position x_i and the sample ξ.

II e_{s_1}: spatial error of the closest node, s_1, its value is accumulated with the square of the Euclidean distance computed in I.

III x_{s_1}: position of the closest node, s_1, its value is updated by applying a translation operation towards the sample ξ with learning rate ϵ_{s_1}.

IV x_n: position of neighbor node, n, its value is updated by applying a translation operation towards the sample ξ with learning rate ϵ_n. Where, $\epsilon_{s_1} > \epsilon_n$, so the influence of the closest node is always stronger than those from the neighbors. Also, $\mathcal{N}_{s_1}$ is the subset of all indices of neighbor nodes.

and edge features are updated as follows:

V $\mathbf{r}_{s_1,s_2}$: edge connecting the two closest nodes, s_1, s_2. If it does not exist, a new edge connecting the nodes is created and the feature vector is set with zero for the age a_{s_1,s_2}; otherwise, its value is only reset.

VI $a_{s_1,n}$: age of the edge connecting the closest node, s_1, and neighbor node, n, its value is incremented by one.

VII $\mathbf{r}_{s_1,n}$: edge connecting the closest node, s_1, and the neighbor node, n, when the age surpasses a maximum threshold $a_{\max}$ its connection is removed. If this process causes nodes with no edges in connection, such nodes are also removed.

Algorithm 2 Adaptation phase in GNG

Input: G, ξ

Output: G, QE

function ADAPTATION(G, ξ)
  $\mathbf{o}_{s_1}, \mathbf{o}_{s_2} \leftarrow \arg\min_{\mathbf{o}_i} \{\|x_i - \xi\|\}_{i=1:N_O}$ ▷ I. Find two closest nodes
  $e_{s_1} \leftarrow e_{s_1} + \|x_{s_1} - \xi\|^2$ ▷ II. Closest node spatial error
  $x_{s_1} \leftarrow x_{s_1} + \epsilon_{s_1}(\xi - x_{s_1})$ ▷ III. Closest node position
  for all $n \in \mathcal{N}_{s_1}$ **do**
    $x_n \leftarrow x_n + \epsilon_n(\xi - x_n)$ ▷ IV. Neighbor nodes position
  end for
  if $\mathbf{r}_{s_1,s_2} \notin R$ **then** ▷ V. Update edges
    add_edge($\mathbf{r}_{s_1,s_2}$)
  else
    $a_{s_1,s_2} \leftarrow 0$
  end if
  for all $n \in \mathcal{N}_{s_1}$ **do**
    $a_{s_1,n} \leftarrow a_{s_1,n} + 1$ ▷ VI. Neighbor edges age
    if $a_{s_1,n} > a_{\max}$ **then** ▷ VII. Remove edges
      remove_edge($\mathbf{r}_{s_1,n}$)
    end if
  end for
end function

In the GROWING phase (Algorithm 3), nodes and edges are added incrementally as follows:

I $\mathbf{o}_{f_1}, \mathbf{o}_{f_2}$: two farthest nodes, these are obtained by finding the nodes with the largest spatial error.

II $\mathbf{o}_p$: a node is created with position x_p set to the center of the nodes f_1 and f_2, and spatial error e_w set to zero.

III $\mathbf{r}_{f_1,p}, \mathbf{r}_{f_2,p}$: a pair of edges are created connecting the new node $\mathbf{o}_p$ with the two farthest nodes $\mathbf{o}_{f_1}$, $\mathbf{o}_{f_2}$. Also, the edge, $\mathbf{r}_{f_1,f_2}$, connecting the two farthest nodes is removed.

Algorithm 3 Growing phase in GNG

Input: G

Output: G

1: **function** GROWING(G)
2: $\quad \mathbf{o}_{f_1}, \mathbf{o}_{f_2} \leftarrow \arg\max_{\mathbf{o}_i} \{e_i\}_{i=1:N_O}$ ▷ I. Find two farthest nodes
3: $\quad x_p \leftarrow \frac{x_{f_1}+x_{f_2}}{2}$
4: $\quad$ add_node($\mathbf{o}_p$) ▷ II. Add new node
5: $\quad$ add_edges($\mathbf{r}_{f_1,p}, \mathbf{r}_{f_2,p}$) ▷ III. Add new edges
6: $\quad$ remove_edge($\mathbf{r}_{f_1,f_2}$)
7: **end function**

3.1.2 Continual Growing Neural Gas

Continual Growing Neural Gas (C-GNG) is a modification of the original GNG (described in Section 3.1.1) motivated by the implementation proposed in [61], which leverages the knowledge already learned during previous executions, thus adopting a continual learning approach [67]. While in the original formulation the model is retrained from scratch for each new input distribution, the continual implementation (Algorithm 4) enables to transfer the output graph and features learned from the previous data frame to initialize the execution of the model in the current data frame.

Another relevant aspect of this implementation is related to the non-stationary distribution given as input. For the case of manipulation of non-rigid objects, it is feasible to

obtain a distribution with smooth changes and preservation of samples along data frames. Hence, only the ADAPTATION phase is necessary to run iteratively during training, while the GROWING phase runs only on the initial data frame (i.e. the index origin is $t = 1$, using 1-based numbering convention) through the GNG algorithm in case the structure of the input distribution is unkwnown. This provides a significant practical improvement by only exploiting the behavior of the object dynamics. Moreover, since nodes are not added nor removed from the graph once the desired maximum quantization error is reached, a mechanism to obtain region correspondence for motion analysis is directly available. This consists of simply tracking the nodes using their indices as unique identifier, then the nodes position can be compared throughout the sequence of frames since the graph is no longer growing in number of nodes.

Algorithm 4 Steps of computation in C-GNG

Input: P, G_{t-1}

Output: G

if $t = 1$ **then**
    $G \leftarrow \text{GNG}(P)$
else
    $G \leftarrow G_{t-1}$
end if
while $\text{QE} > \text{QE}_{\max}$ **do**
    for all $n \in N$ **do**
        $\xi \sim P$
        $\text{ADAPTATION}(G, \xi)$
    end for
end while

Although in previous work [61], C-GNG demonstrated the ability to handle noisy distributions, its formulation is more sensitive. As a result, outliers affect the quality of the representation. Consider the case when a set of samples are located outside the distribution associated to the object's shape, then the closest nodes will be significantly translated towards those samples during the ADAPTATION phase, and eventually no further adaptations are made in these nodes due to the impossibility of being found as closest to other

samples that are part of the distribution. In such a case, these nodes are called dead nodes[1]. The possibility of carrying dead nodes in the graph presents a serious issue in this work. During the continual execution, each given data frame constitutes to the current state of the non-rigid object. Therefore, the nodes of the graph should be associated with some region of the shape in order to adequately reflect the dynamics of the deformation caused by the current manipulation action performed. For that reason, dead nodes significantly damage the shape representation by not adapting properly to the changes produced in such regions.

Therefore, in addition to adopting the continual learning scheme introduced in [61], this work proposes an additional procedure to the C-GNG formulation with the intention of mitigating the risk of dead nodes present in the representation. Thus, the proposal consists of including a new term, w_{s_1}, (Equation 3.2) to regularize the influence of the outliers in the process that update the closest node position during the execution of the ADAPTATION phase.

$$w_{s_1} = \begin{cases} 1, & \mu < 0 \\ K(\mu, \sigma), & \text{others} \end{cases} \tag{3.2}$$

More specifically, this term evaluates a 1D Gaussian kernel function (Equation 3.3) with mean and standard deviation parameterized by the desired maximum quantization error QE_{max}.

$$K(\mu, \sigma) = e^{-\frac{\mu^2}{2\sigma^2}} \tag{3.3}$$

Where:

- μ : mean, its value is equal to the difference between the Euclidean distance $\|x_{s_1} - \xi\|$ and maximum quantization error QE_{max}.
- σ : standard deviation or sigma, its value is proportional to the maximum quantization error QE_{max}.

[1] dead nodes are also known as dead units in competitive learning.

The regularization term, w_{s_1}, is added (Equation 3.4) into the function that updates the closest node position (step 4 of Algorithm 2). In this way, those pairs of nodes and samples which distances are large are penalized due to the possibility of being outliers, whereas those with small distances remain unchanged. As with any other parametric regularization approach, this term should be selected according to the characteristics of the input distribution.

$$x_{s_1} \leftarrow x_{s_1} + w_{s_1}\epsilon_{s_1}\left(\xi - x_{s_1}\right) \tag{3.4}$$

3.1.3 Batch Continual Growing Neural Gas

Batch Continual Growing Neural Gas (BC-GNG) is an extension developed in this work of the previous continual formulation with outlier regularization (Section 3.1.2), which implements a batch training procedure. This approach provides benefits such as computational efficiency and faster convergence of the algorithm. In GNG and C-GNG, features are updated online for each sample during an iteration of the algorithm. This sequential data feeding can significantly slow down its execution, especially for high resolution spatio-temporal distributions.

While no prior work proposes a batch training procedure of the GNG algorithm, there are implementations for other self-organizing networks, such as Neural Gas [68] and Self-Organizing Maps [69] that closely related to this formulation. Therefore, in BC-GNG (Algorithm 5), a batch training enables to update the features while avoiding an individual iteration of each sample from the distribution. This process involves the removal of a complete loop in the algorithm, and thus the entire distribution is given as input to the ADAPTATION phase.

Furthermore, the procedure that updates the features of the nodes and edges must be redefined. In particular, to produce an update in coherence with the online training, a new process is proposed to unify the contributions of a node with respect to its role with the samples. First, the node position is updated by combining the contributions when the node is found as closest and as topological neighbor. Similarly, the age of the edges connecting the closest node with its neighbors is updated by accumulating the times in which the node is found as closest.

Algorithm 5 Steps of computation in BC-GNG

Input: P, G_{t-1}

Output: G

1: **if** $t = 1$ **then**
2: $\quad G \leftarrow \text{GNG}(P)$
3: **else**
4: $\quad G \leftarrow G_{t-1}$
5: **end if**
6: **while** $\text{QE} > \text{QE}_{\text{max}}$ **do**
7: $\quad \text{ADAPTATION}(G, P)$
8: **end while**

More specifically, the Euclidean distances between all the samples in the distribution and nodes position in the graph are computed, also finding the two closest nodes at once. In this way, the contribution of each node as closest (step 4 of Algorithm 2) is reformulated as the average of the distances paired with a particular node (Equation 3.5).

$$x_{i(s_1)} = \frac{1}{N_i} \sum_{\xi \in P_i} (\xi - x_i) \tag{3.5}$$

Where:

i : closest node index.

P_i : closest node batch distribution, subset of the distribution, P, associated with each node, $\mathbf{o}_i$, found as closest.

N_i : size of the closest node batch distribution.

$x_{i(s_1)}$: contribution of node position as closest.

While, the age of the neighbor edges (step 14 of Algorithm 2) is reformulated as an increment of the size of the closest node batch distribution, N_i. Since nodes are likely to be connected with more than one edge in the graph, the contribution of each node as neighbor (step 6 of Algorithm 2) requires an additional consideration. Initially, all the distances between the node and the samples associated due to the connections with its neighbors are collected, and then reduction by mean of these collected distances is computed (Equation

3.6) similarly as in the previous step.

$$x_{i(n)} = \frac{1}{D_j} \sum_{j} \frac{1}{N_j} \sum_{\xi \in P_j} (\xi - x_i) \tag{3.6}$$

Where:

- j : neighbor node index.
- P_j : neighbor node batch distribution, subset of the distribution, P, associated with each node, $\mathbf{o}_i$, found as neighbor.
- N_j : size of the neighbor node batch distribution.
- D_j : degree (number of edges) of the neighbor node.
- $x_{i(n)}$: contribution of the node position as neighbor.

The contributions of each node as closest and as neighbor are included in a single expression to update the position feature (Equation 3.7), thus replacing the two-step update process (defined along steps 4 and 6 of Algorithm 2) in the online training. In this case, the parameters ϵ_{s_1} and ϵ_n correspond to the learning rates that control the influence of the adjustment of each contribution to the position feature (i.e. take the same role as in GNG and C-GNG).

$$x_i \leftarrow x_i + \epsilon_{s_1} x_{i(s_1)} + \epsilon_n x_{i(n)} \tag{3.7}$$

3.2 Dataset

This section describes the sensing and processing methodologies applied for the collection of non-rigid object data, which are used as input distribution for the evaluation of the proposed GNG-based model. The complete set of non-rigid objects utilized to construct the dataset is shown in Figure 3.2, also a brief description of each object is presented in Table 3.1.

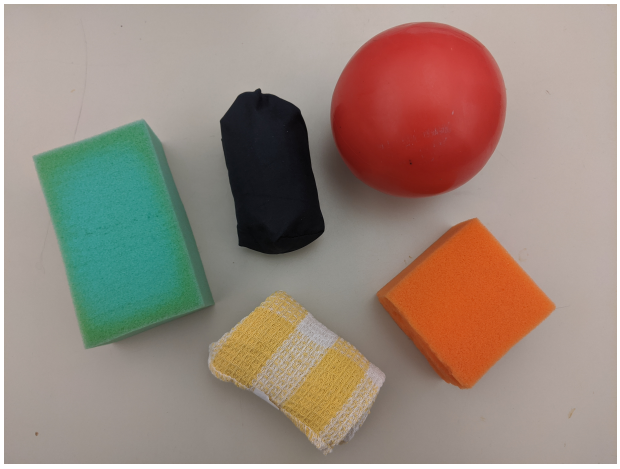


Figure 3.2: Set of non-rigid objects used to construct the dataset.

Name	**Description**	**Image**
Ball	• Radius=0.05m • Type=plastic • Shape=sphere • Use=inflatable ball are commonly found in toy stores	

Name	Description	Image
Large Sponge	• Size=0.12m x 0.07m x 0.04m • Type=foam • Shape=rectangular cuboid • Use=sponges are commonly used as cleaning utensils	
Small Sponge	• Size=0.07m x 0.07m x 0.04m • Type=foam • Shape=cuboid • Use=sponges are commonly used as cleaning utensils	
Towel	• Size=0.08m x 0.05m x 0.03m • Type=textile • Shape=cylindrical • Use=wrap towels are commonly found in warehouses	
Toy	• Size=0.09m x 0.05m x 0.03m • Type=stuffed • shape=cylindrical • Use=Stuffed animals or pillows are commonly found in toy stores	

Table 3.1: Individual description of the set of non-rigid objects.

3.2.1 Sensing

The configuration of the real-world robotic manipulation and sensing setup is shown in Figure 3.3, which consists of a Barrett BH8-280 robotic hand [70] resting on a flat table, an Intel RealSense SR305 RGB-D sensor [71] mounted overhead on a tripod, and a non-rigid object placed on the palm of the robotic hand. Also, the hardware components used in the robotic manipulation setup are operated through ROS [72]. The ROS architecture is shown in Figure 3.4. In particular, the *realsense2_camera* and *bhand_controller* ROS nodes are used to capture the sensor data from the devices and send it to the *dataset* and *trajectory* ROS nodes for further processing.

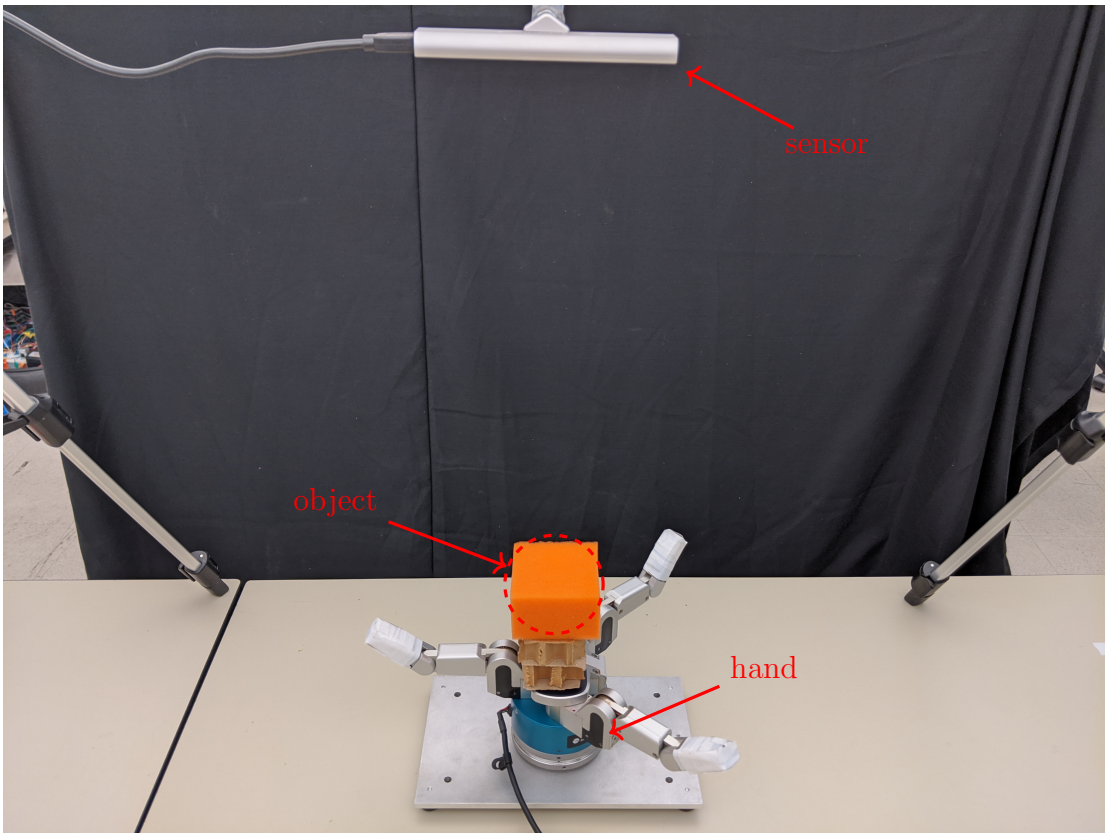


Figure 3.3: Configuration of the real-world robotic manipulation and sensing setup. The location of the sensor, non-rigid object and robotic hand are marked in red.

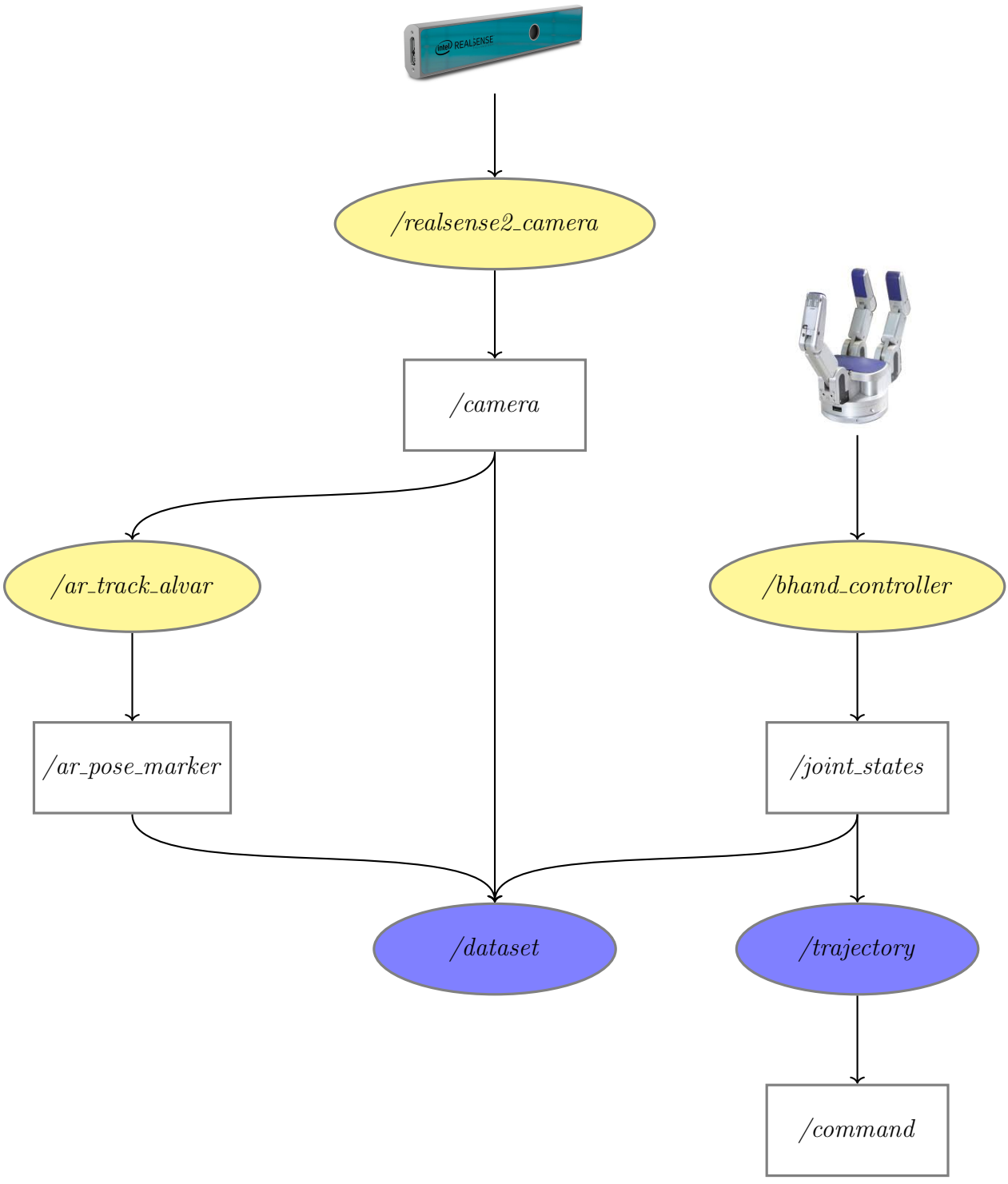


Figure 3.4: ROS architecture of the robotic manipulation setup: (Yellow) existing ROS nodes; (Blue) custom ROS nodes; (Empty) ROS topics.

Robotic Hand

The Barrett BH8-280 is a three-fingered robotic hand with 8 joints in total, but 4 DoF because only 4 of them are motorized (i.e. the rest do not have motors but are mechanically linked). These joints can be controlled to produce two types of motions, named as base and spread, for which the joints are depicted in Figure 3.5. The former refers to an opening/closing motion of each finger, while the latter refers to a spread motion around the palm. Although, this means that for spread motion, one finger is fixed to the palm and the other two share the same motor, hence move in a symmetrical way around the palm [73]. In this work, the interface with the robotic hand is managed by the ROS package of the pyHand library[2], which enables to access the joint state data (i.e. position, velocity, torque) through the *joint_states* ROS topic and to produce joint position commands through the *command* ROS topic.

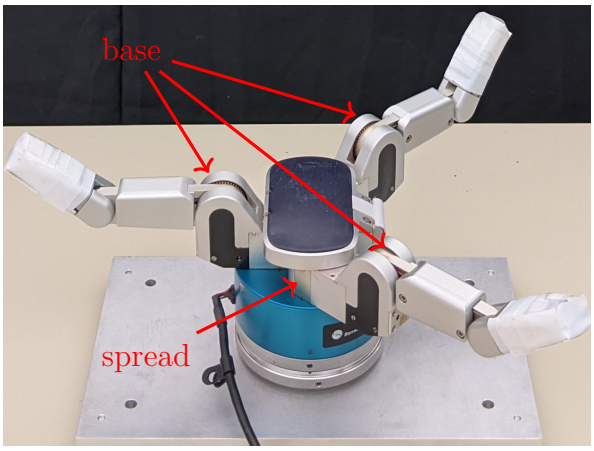


Figure 3.5: Barrett BH8-280 robotic hand joint configuration. The location of the four motorized joints are marked in red.

RGB-D Sensor

The Intel RealSense SR305 is a RGB-D sensor that integrates a depth camera along with a RGB camera. This sensor outputs a depth map at 60 fps with 0.125 mm resolution and a color texture map at 30 fps with 1080p resolution. The depth map is obtained via active stereo using coded light technology, which is a particular structure light technique that consists of emitting a sequence of different kind of patterns over time from an infrared

[2] http://wiki.ros.org/barrett_hand

projector. Then, these patterns are captured by an infrared sensor that produces an image of 640×480 pixels. The pixels in the infrared image are further analyzed by a custom 3D vision processing unit (VPU) to perform the depth reconstruction, and thus generate the final depth map. While, the image of 1920×1080 pixels captured by the RGB sensor is analyzed by an image signal processor (ISP), which performs several corrections and denoising processes to enhance the final color texture map [74]. Similarly, the interface with the RGB-D sensor is managed by the ROS package of the Intel RealSense SDK[3], which enables to access raw and aligned color and depth images, and also the camera parameters through the *camera* topic.

3.2.2 Processing

The components in the ROS architecture (Figure 3.4) are used as tools for the collection of the point cloud data of the non-rigid object. In this way, the RGB-D sensor data is processed in the *dataset* ROS node to detect the object and generate the point cloud data. Also, the robotic hand data is processed in the *trajectory* ROS node to generate the fingers trajectory. Thus, a dataset is created which consists of a file with 800 samples, using a sampling rate of 30Hz. Each file stores the data generated in synchronization with the execution of the fingers trajectory, which takes approximately 26.67 seconds to complete.

Fingers Trajectory Generation

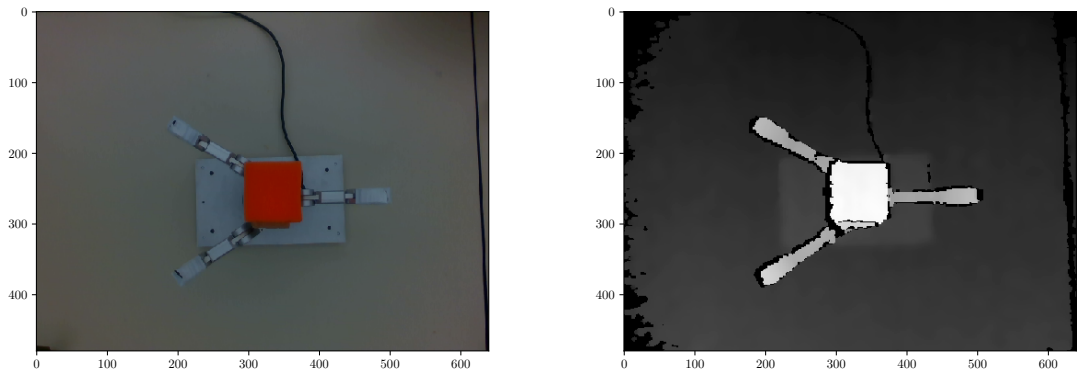
The base and spread joint positions are adjusted to generate the fingers trajectory to perform the manipulation of a non-rigid object. The manipulation action that causes deformations consists of moving the fingers of the hand towards the non-rigid object, and eventually come into contact by grasping and progressively squeezing it, thus causing a change in its shape. Considering that the selection of grasping points [57] is not part of this work, the base joints range is limited to $(-90°, 90°)$, whereas the spread joint is limited to $(-45°, 45°)$. These ranges provide a fair compromise between interaction variations and collision avoidance, given the dexterity capabilities of the robotic hand. Each trajectory is then generated taking as final configuration a random joint position within the defined ranges, and using a linear interpolation with 50 points beginning from a predefined rest

[3] http://wiki.ros.org/realsense2_camera

position of the hand (i.e. fingers are open enough that contact with the object does not occur). The trajectories are designed in this manner to produce brief rest periods at the end of each point that establishes a balance of contact forces and gravity in the system. In other words, the dynamic manipulation includes quasistatic periods [75] with the intention of preventing slippage or sliding movements of the object, and thus mainly capturing information associated with the deformation.

Non-Rigid Object Detection

The data obtained of a non-rigid object corresponds to a sequence of point cloud frames of the shape captured by the RGB-D sensor as it deforms. Each frame outputs aligned color and depth images, which are internally processed by the RGB-D sensor. In the case of the SR305 version, depth and infrared images are already pixel aligned but are rectified using a parametric distortion model, while no formal distortion model is applied to the color image. Then, the rectified depth and color images are aligned to a common viewpoint. This process consists of transforming the pixels from the depth image to match the pixels from the color image, which are also scaled using a KNN interpolation method in order to match both resolutions. In this way, Figure 3.6 shows the aligned color and depth images of 640×480 pixels captured by the RGB-D sensor.



(a) color image (b) depth image

Figure 3.6: a) Aligned color, and b) depth images captured by the RGB-D sensor. The depth image displays distance using a grayscale color map. A light color means closer objects, while a darker color means more distant objects.

In this work, classical image segmentation techniques are applied to both aligned color and depth images for the detection of non-rigid objects. The summary of these operations are depicted in Figure 3.7.

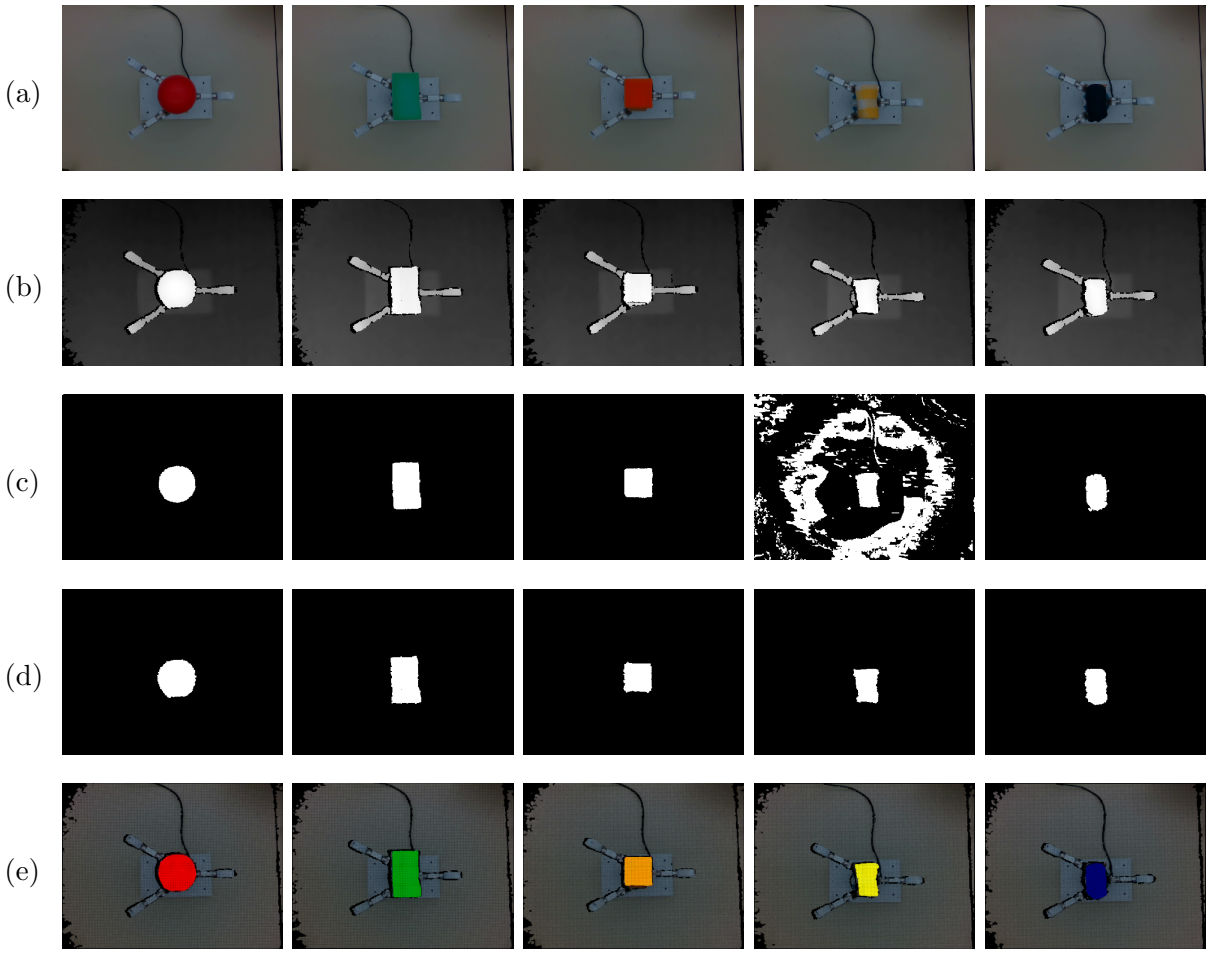


Figure 3.7: Processing operations applied to the set of non-rigid objects: a) Aligned color and b) depth images; c) resultant color and d) depth binary masks; e) point clouds of the scene with the segmented non-rigid object. The object of interest is painted with a contrasting uniform color.

Initially, the color image is transformed from the original RGB color space to the HSV color space, and then a histogram backprojection technique is applied to obtain a binary mask. This process receives as input the image and the color histogram of the object of interest, the latter is used to calculate the probability that the pixels in the image correspond to the object. Then, the mask is filtered by applying a convolution with threshold operation

to obtain a cleaner result. Moreover, the depth image is cropped by volume, truncating the spatial values based on the information of the object position relative to the camera. Thus, the resultant color and depth masks are combined and applied to the depth image to obtain the object of interest, the segmented image is then deprojected to convert the 2D pixels to 3D organized point clouds. All these operations are implemented using the OpenCV library [76].

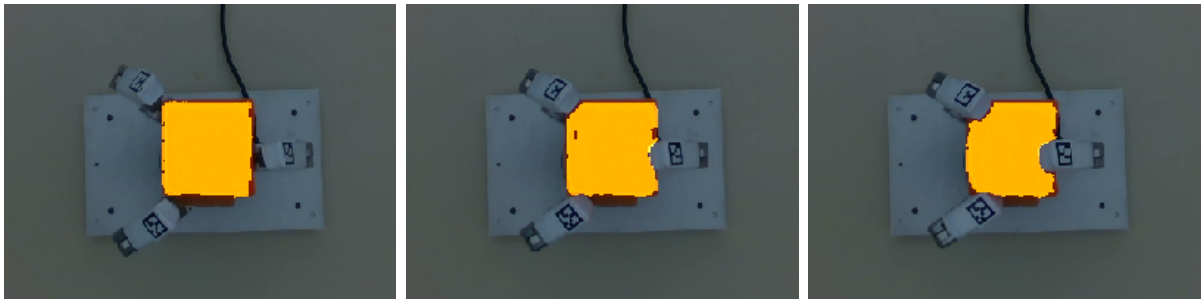
Overall, these operations produce satisfactory segmentation results for the scenarios analyzed in this work. This solution is valid given the uniform and contrasting color of the set of non-rigid objects utilized. The most complicated situation occurs with the towel (column 4 in Figure 3.7) because there exists some overlap in the color histogram with the background, thus producing a noisy color binary mask in several regions. For this reason, the mask obtained with the depth image complements the color segmentation procedure, resulting in more robustness for the final segmentation.

3.3 Experiments

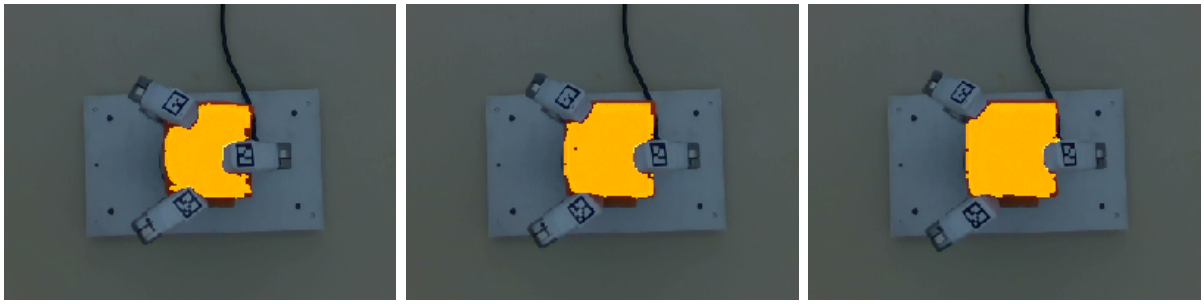
In this section, the performance of the different variations of the GNG as a representation model of non-rigid objects is investigated, thus evaluating their potential for shape tracking and motion analysis according to the considerations established at the beginning of this chapter; real-time execution (3.3.1), temporal smoothing (3.3.2) and region correspondence (3.3.3).

The experiments are run on a computer with 1× Intel Core i5-7300U @ 2.60GHz, 16 GB RAM, and GNU/Linux operating system. The GNG-based models are implemented in the Deep Graph Library [77] using PyTorch [78] as backend. In addition, the model parameters are shared as much as possible in order to consistently compare the performance of the different variations. For the GNG model, an age of 35, learning rate of 0.1 and 0.005, error decay of 0.5 and 0.9995 are used. For the C-GNG model, an age of 2,000, learning rate of 0.1 and 0.005. And for the BC-GNG model, an age of 2,000, learning rate of 0.4 and 0.01 are used. The sigma value of the regularization term used in C-GNG and BC-GNG corresponds to 0.6 for the towel and 4 for the rest of the objects. The latter is adopted since the distribution of the towel tends to include more outliers, hence it is necessary to increase the regularization.

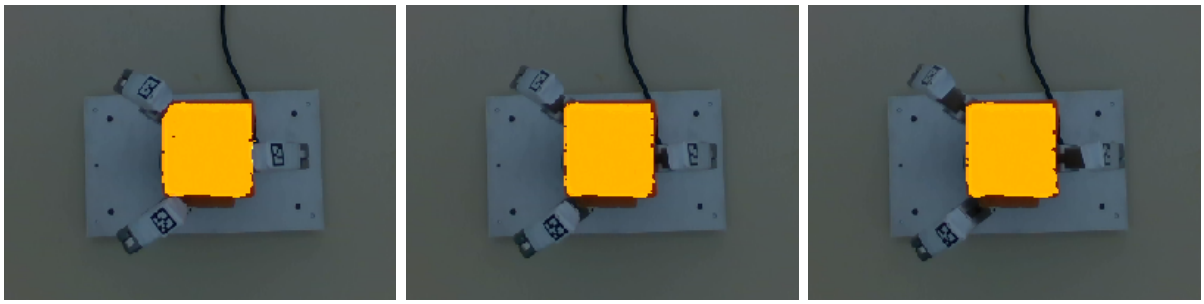
Each model is evaluated using the point cloud dataset obtained during the execution of the fingers trajectory described in Section 3.2. A subset of the data frames that progressively reflects various deformation levels using the small sponge as an example of non-rigid object is shown in Figure 3.8.



(a) $t = 1$ (b) $t = 100$ (c) $t = 200$



(d) $t = 300$ (e) $t = 400$ (f) $t = 500$

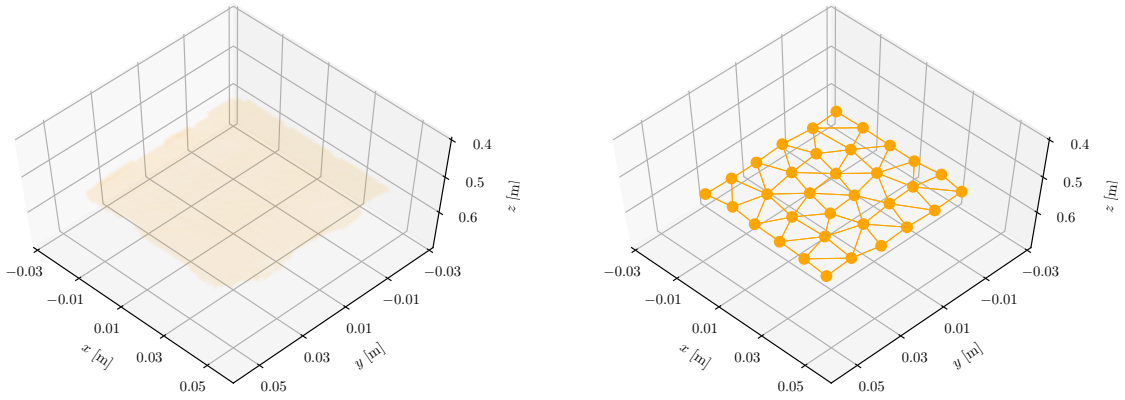


(g) $t = 600$ (h) $t = 700$ (i) $t = 800$

Figure 3.8: Color image sequence of the scene with the segmented non-rigid object at different deformation levels during the execution of a trajectory. The small sponge is depicted as non-rigid object.

An example of the input and output data of the small sponge handled by each model is shown in Figure 3.9. The input point cloud and output graph correspond to the last data

frame obtained from the dataset (taken at $t = 800$). Also, these plots correspond to a 3D visualization from a rotated perspective of the viewpoint and adjusted with pitch and yaw angles of 240° and 225° respectively. The axis scales are in meters and are measured with respect to the RGB-D sensor coordinates system.



(a) point cloud

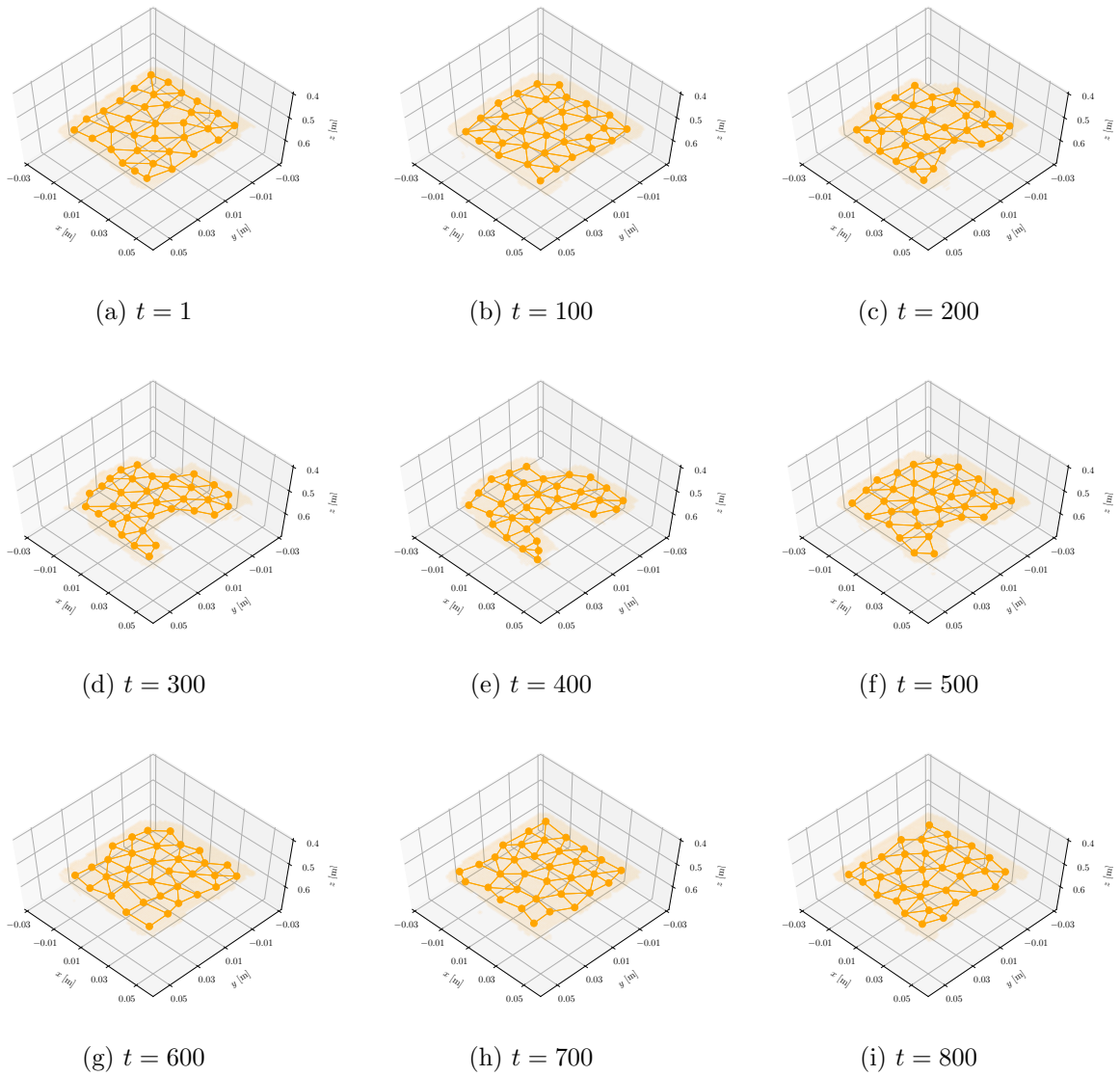
(b) graph

Figure 3.9: Shape visualization of the small sponge: a) point cloud and b) graph obtained by the GNG-based model.

The input point clouds and output graphs of the small sponge obtained by the GNG, C-GNG and BC-GNG models are shown in Figures 3.10, 3.11 and 3.12 for a subset of frames along the fingers trajectory. The small sponge is used as a recurring example of a non-rigid object. Experimental results with the rest of non-rigid objects are summarized in Appendix A.1. In this way, the plots associated to the other four non-rigid objects obtained by the C-GNG and BC-GNG models are shown in Figures A.1 to A.4. Points with solid texture and light color correspond to the input point clouds, while those in dark color correspond to the output graphs. A graph overlaps with the segmented point cloud in order to show a direct comparison of the shape representation generated by the proposed model and the actual non-rigid object. Also, these figures serve as a reference in subsequent Sections 3.3.1, 3.3.2 and 3.3.3 that aim to investigate the previously established considerations for the graph construction algorithm but oriented toward robotic manipulation tasks.

Overall, all variants of GNG produce a graph sequence that track the non-stationary distribution of a non-rigid object. However, regardless of maintaining consistency during

model training (i.e. share parameters and stopping criterion), the results obtained by the proposed BC-GNG model (Figure 3.12) produce the best visual quality of the shape representation.



(a) $t = 1$ (b) $t = 100$ (c) $t = 200$

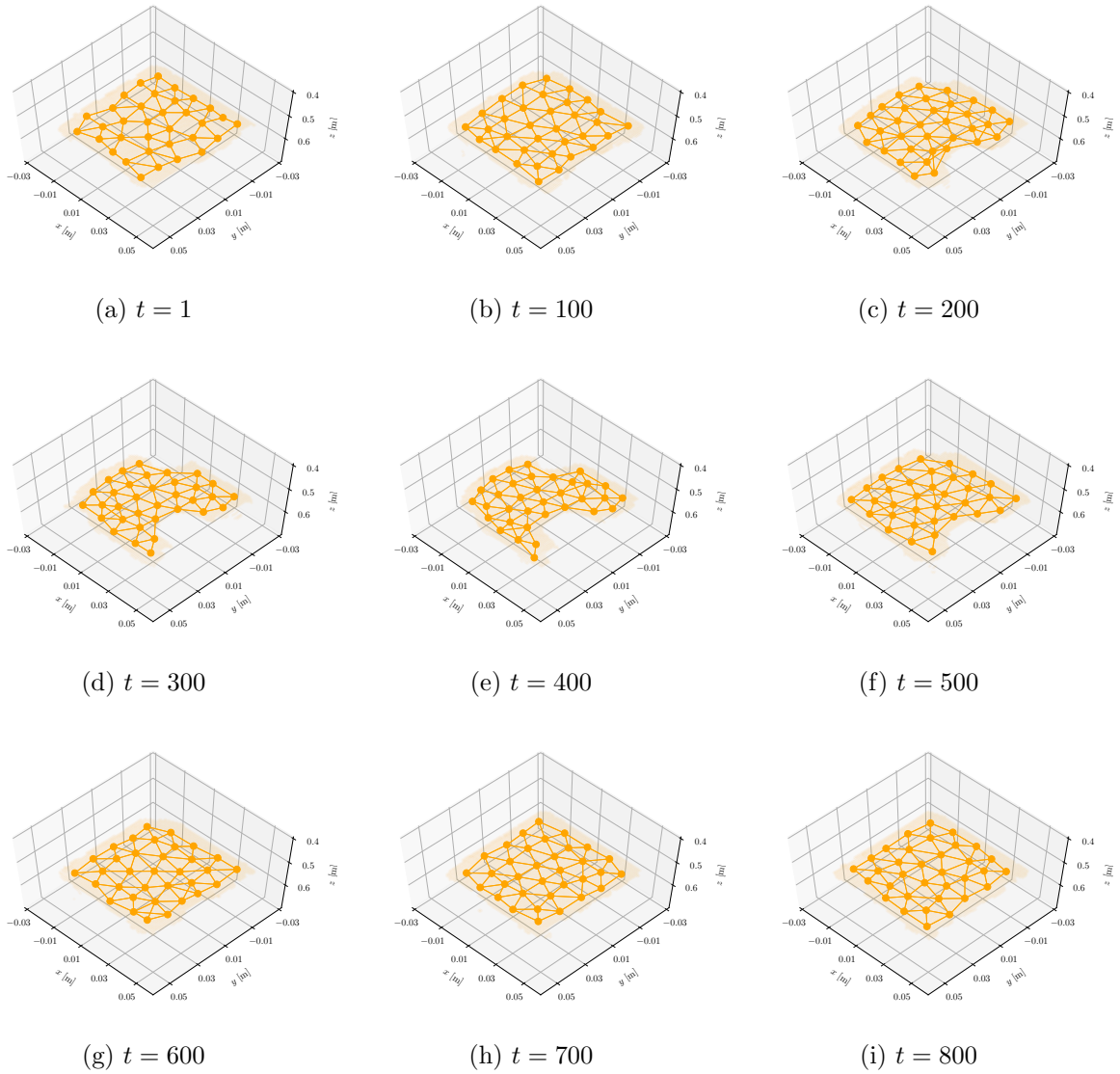
(d) $t = 300$ (e) $t = 400$ (f) $t = 500$

(g) $t = 600$ (h) $t = 700$ (i) $t = 800$

Figure 3.10: Point cloud and graph sequences of the small sponge obtained by the GNG model at different deformation levels.

Consider the data frames where the largest deformation occurs (around $t = 300$). The areas on the shape where the object is compressed more (e.g. around the center and vertices), show a higher and more natural accumulation of nodes. In contrast, the other models produce a node density with less similarity in these areas, despite the fact that

the small sponge exhibit a simple geometric shape. Also, the final representation obtained when no interaction occurs between the robotic hand fingers and the small sponge (around $t = 700$) produces a more symmetric node density that better resembles the topology of the object. These characteristics are also observed in the plots of the other non-rigid objects. In addition, the increase of the regularization for the towel model produces more appealing visual stability of the node density against outliers, this behavior is observed (Figure A.3) at various data frames during manipulation.

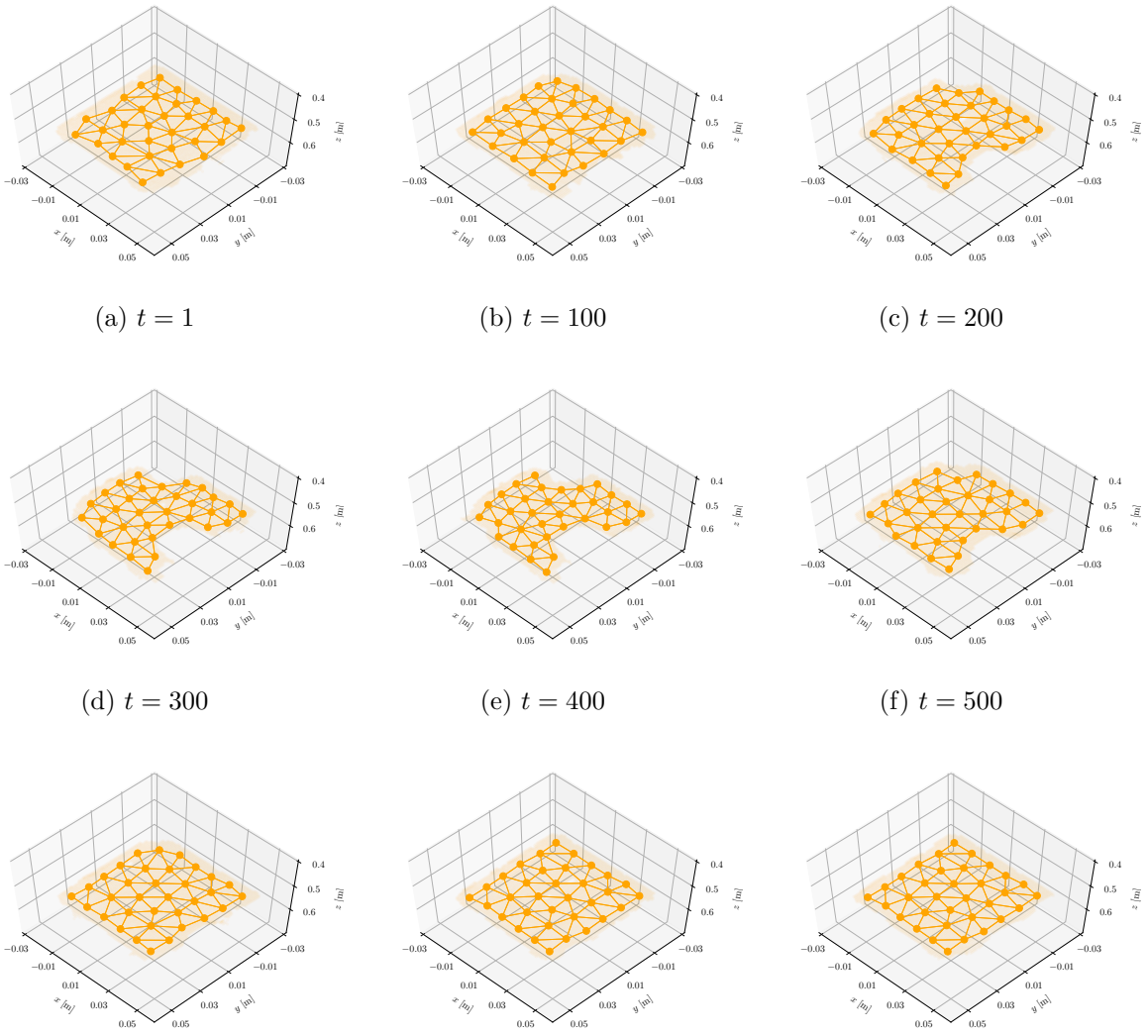


(a) $t = 1$ (b) $t = 100$ (c) $t = 200$

(d) $t = 300$ (e) $t = 400$ (f) $t = 500$

(g) $t = 600$ (h) $t = 700$ (i) $t = 800$

Figure 3.11: Point cloud and graph sequences of the small sponge obtained by the C-GNG model at different deformation levels.



(a) $t = 1$ (b) $t = 100$ (c) $t = 200$

(d) $t = 300$ (e) $t = 400$ (f) $t = 500$

(g) $t = 600$ (h) $t = 700$ (i) $t = 800$

Figure 3.12: Point cloud and graph sequences of the small sponge obtained by the BC-GNG model at different deformation levels.

3.3.1 Real-Time Execution

The computation complexity of GNG has been previously described [61] as being not suitable for direct use in the representation of non-stationary distributions with time constraints, such as object tracking. Therefore, this analysis aims to determine whether different variations of GNG can be applied for real-time tracking of non-rigid objects using point clouds as input data. In this analysis, each model is trained for three different in-

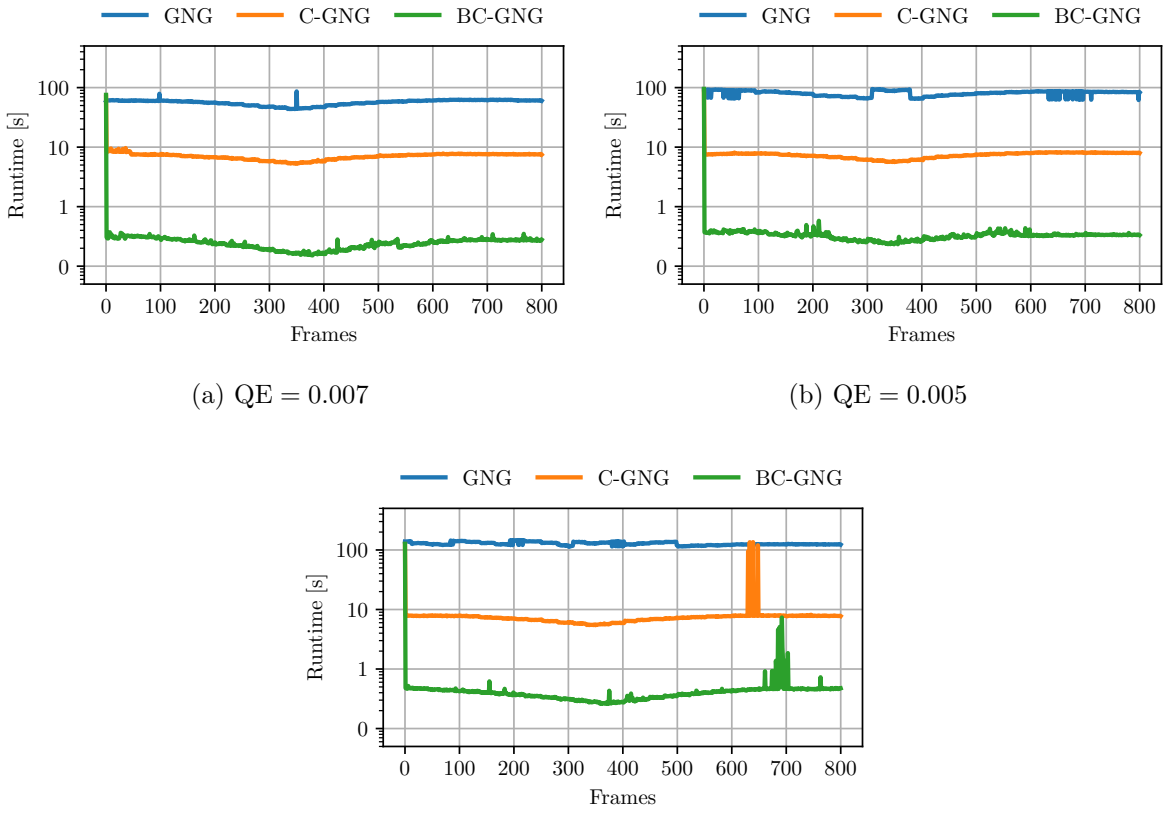
stances of quantization error. Then, the runtime sequence is recorded by each data sample over the dataset and is presented as a line plot in Figure 3.13. While, the average of the runtime sequence is depicted as a bar plot in Figure 3.14. Both figures correspond to semi-logarithmic visualizations due to the wide variability of the scales produced by the models.

In the first execution ($t = 1$), none of the models GNG, C-GNG and BC-GNG are considered real-time algorithms. In fact, Figure 3.13 shows that the three variations of GNG share a similar computational cost in the execution of the first data frame regardless of the selection of a desired quantization error. This is justified since the proposed improvements are not particularly defined for the GROWING phase, which is executed only during this first data frame.

During the sequential execution ($t > 1$), the GNG runtime remains almost constant over frames. This behavior occurs due to the retraining of the algorithm for each new input distribution. In particular, during a complete trajectory of 800 data frames, the representation of the small sponge takes an average of 80.7 seconds for the graph construction with quantization error, QE, of 0.005, also equivalent to a graph size of 34 nodes. Similarly, the average of the runtime grows linearly for cases when more precision of the representation is required (i.e. smaller quantization errors). On the other hand, C-GNG runtime is several orders of magnitude faster than GNG mainly due to the implementation of the continual learning approach by reusing previous graphs. Although, this formulation is a great improvement from the original GNG, its runtime is not yet suited for real-time applications, at least for low-power CPUs and embedded systems. It takes an average of 7.4 seconds for the graph construction at each frame with the same quantization error of 0.005. Finally, BC-GNG with the proposed batch training considerably speeds up the execution. In this case, the algorithm needs an average of 0.4 seconds to construct the same graph.

Furthermore, every formulation shows peaks during certain frames. This behavior is attributed to the execution of several iterations of the algorithm in these frames. Indeed, the runtime is closely related by the amount of iterations needed in order to reach a desired maximum quantization error. However, as described in [61], an appropriate sampling rate and faster procedure to update the nodes position relieve the number of iterations required for convergence. Even for cases of rapidly changing distributions, an exponential decay function can be included as learning rate, instead of the current constant value. In this

way, the nodes position are updated with a dynamic weight, starting from a slower initial adaptation to a faster final.



(a) QE = 0.007

(b) QE = 0.005

(c) QE = 0.003

Figure 3.13: Runtime sequence obtained by GNG, C-GNG, BC-GNG during the manipulation of the small sponge.

Among all average runtimes shown in Figure 3.14, the results obtained with quantization error of 0.005 exhibits a fair compromise in the representation for our case, in which the BC-GNG model conveniently satisfies the requirement of at least two frames per seconds for control and planning in robotic manipulation of non-rigid objects [57]. For consistency, the same temporal criterion is used to obtain the graph for the rest of non-rigid objects, the corresponding runtimes are shown for the C-GNG and BC-GNG models in Figures A.5 and A.6. Therefore, in the remaining analyses, a quantization error of 0.005 is used for the graph construction of the small sponge.

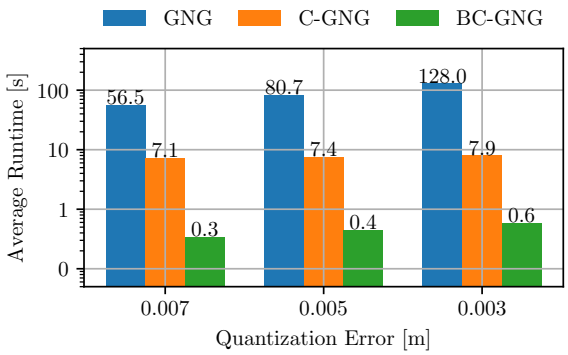


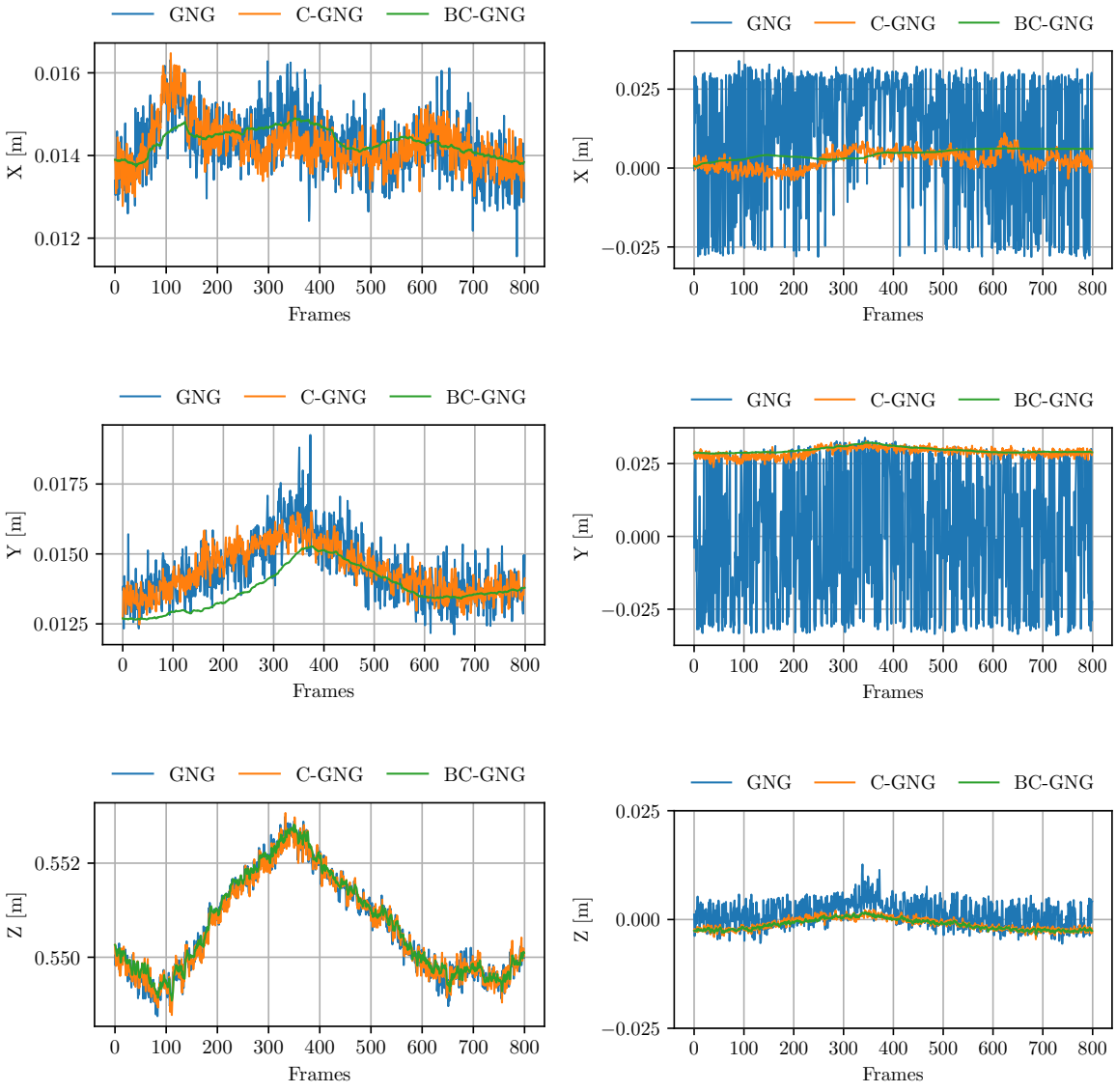
Figure 3.14: Average of runtime sequence obtained by GNG, C-GNG, BC-GNG during the manipulation of the small sponge.

3.3.2 Temporal Smoothing

It has been previously demonstrated that GNG as a downsampling algorithm [63] obtains an average distance error[4] smaller than that of voxel grid, hence maintaining a better shape representation when reducing the sampling resolution. Similarly, GNG as a re-sampling algorithm [79] provides a better density resolution of point clouds, by dynamically maintaining more details in complex regions as opposed to uniform and random sampling. Therefore, this analysis aims to determine whether different variations of the GNG can be used as a smoothing filter that generates stable displacements of the non-rigid object shape during deformation.

First, a global displacement is defined as the path followed by the average of all nodes position of the graph, hence it estimates the rigid motion of the object and also defines the centroid. On the other hand, local displacement correspond to the path followed by each individual node, and are measured with respect to the centroid coordinate system, hence estimating the actual deformation motion of the object. The signals obtained by these displacements are analyzed in order to determine the filtering capabilities of each model. Thus, the global and local displacement in 3D obtained by different variations of GNG during the manipulation of the small sponge are shown in Figure 3.15. The global displacement is computed using the average of the 34 nodes position of the graph, while the local displacement corresponds to the node with 1 index.

[4]distance error is a similar metric to the quantization error.



(a) Global displacement

(b) Local displacement

Figure 3.15: a) Global and b) local displacement obtained by GNG, C-GNG, BC-GNG during the manipulation of the small sponge.

Also, Figure 3.15 demonstrates that the original GNG formulation is not particularly suitable for producing smooth signals as an output. In fact, a considerable amount of undesired noise is continuously present for both global and local displacements, hence requiring additional processing steps in order to filter the signal before being used in subsequent processes. On the other hand, C-GNG and BC-GNG produce more stable signals, as expected considering the adaptations described in Section 3.1. Moreover, an interesting property of

BC-GNG is the low-pass filter effect that is observed in the signal. This behavior occurs due to the characteristic of the algorithm in using the average of the nodes position during the update process. Similar smoothing effects were observed when applying the C-GNG and BC-GNG models on the other four non-rigid objects considered, as shown in Figures A.7 and A.8. Therefore, the final displacements obtained by BC-GNG are much smoother, and desirable to estimate with more confidence motion and deformation quantities directly from the graph.

3.3.3 Region Correspondence

The nodes position obtained from GNG has been previously used directly as features for more complex analysis, such as motion estimation for surveillance systems [59], and velocity fields for object tracking [60]. This analysis aims to determine whether different variations of GNG model generate node displacements that can be used as features for estimation of the object motion. In the following figures and with the intention of maintaining a plausible number of signals to show, a subset of nodes is extracted from the graph to perform the comparative analysis (i.e. the first 6 indices from a total of 34), whose indices are indicated in colored legends at the upper part of each figure.

The local displacement in 3D of a subset of nodes extracted from the graph obtained by the GNG model with quantization error of 0.005 are shown in Figure 3.16. These displacements exhibit a constant fluctuation along frames which suggest that nodes position during the ADAPTATION phase oscillate over a wide area of the shape, hence locality is not properly maintained over the sequence of frames. This behavior occurs due to the stochastic nature of the graph construction in the GNG formulation, which causes that new nodes are not created around the same location for each repetition of the GROWING phase. Therefore, the region correspondence between nodes position is nonexistent, and the direct use of the nodes as features for estimation of the object motion is accompanied by a high level of uncertainty.

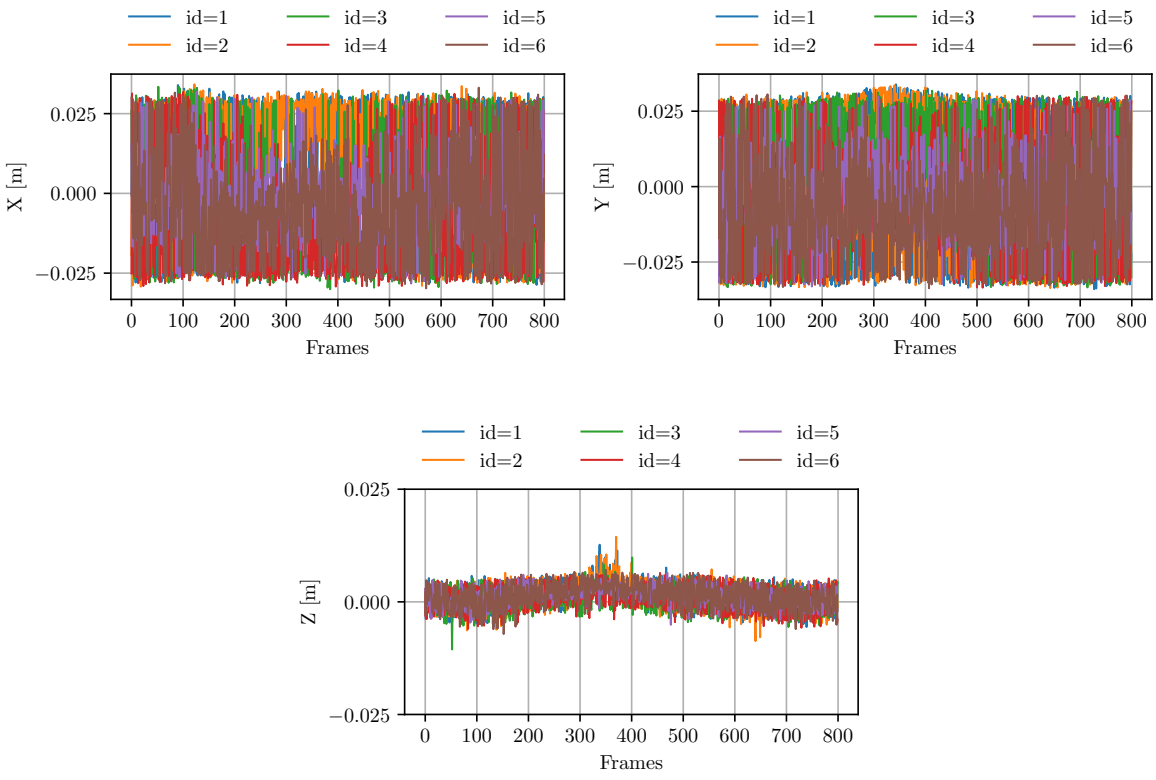


Figure 3.16: Local displacement of a subset of nodes obtained by GNG during the manipulation of the small sponge.

The local displacement in 3D of the same subset of nodes but obtained by the C-GNG model are shown in Figure 3.17. In this case, these displacements exhibit a localized motion of nodes position that preserve certain regions of the shape. Indeed, more interpretable and representative displacements are obtained in this formulation, mainly attributed to the continual learning approach by reusing the graph, hence a mechanism for region correspondence is directly available. However, there still exists some interference between nodes which causes unrecoverable positions and affects the region correspondence of the representation. For example, consider the node 4 (red) with position along the X axis. It starts at $x_{4,t=1} = (-0.0254, -0.029, 0.003)$, but then overlaps with nodes 5 (brown) and 6 (purple) along the way, which causes a motion of its original position, and ends at $x_{4,t=800} = (-0.013, -0.031, 0.004)$. Experiments suggest that such interference occurs in presence of large deformations, when signal fluctuations is more likely to cause these motions. Therefore, the applicability of the nodes as features for estimation of the object

motion obtained by the C-GNG model presents some limitations.

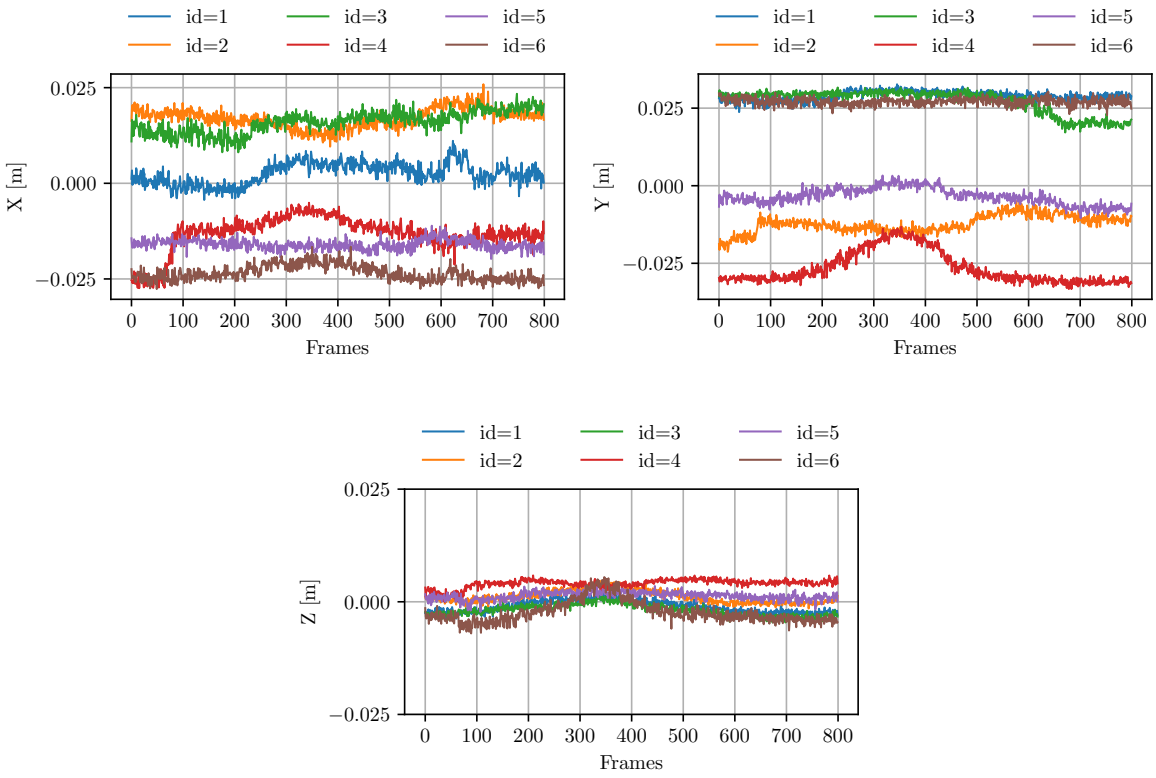


Figure 3.17: Local displacement of a subset of nodes obtained by C-GNG during the manipulation of the small sponge.

On the other hand, Figure 3.18 shows the local displacements in 3D of the same subset of nodes extracted of the graph obtained by the BC-GNG model. These displacements reflect a more localized motion of nodes position, even further to that observed in C-GNG. Interference between nodes is not causing strong deviations in their displacements, hence better preserving their correspondence throughout the manipulation task. Consider the same example as before, where the node 4 (red) with position along the X axis starts at $x_{4,t=1} = (-0.0254, -0.029, 0.003)$, and then progressively moves closer to the same nodes 5 (brown) and 6 (purple) due to deformation, but is able to recover more properly to its original position at the end at $x_{4,t=800} = (-0.0277, -0.029, 0.002)$. Other nodes exhibit a similar behavior along different axes as depicted in Figure 3.18. Also, Z values do not change significantly since deformation occurs mostly in 2D according to the displacements.

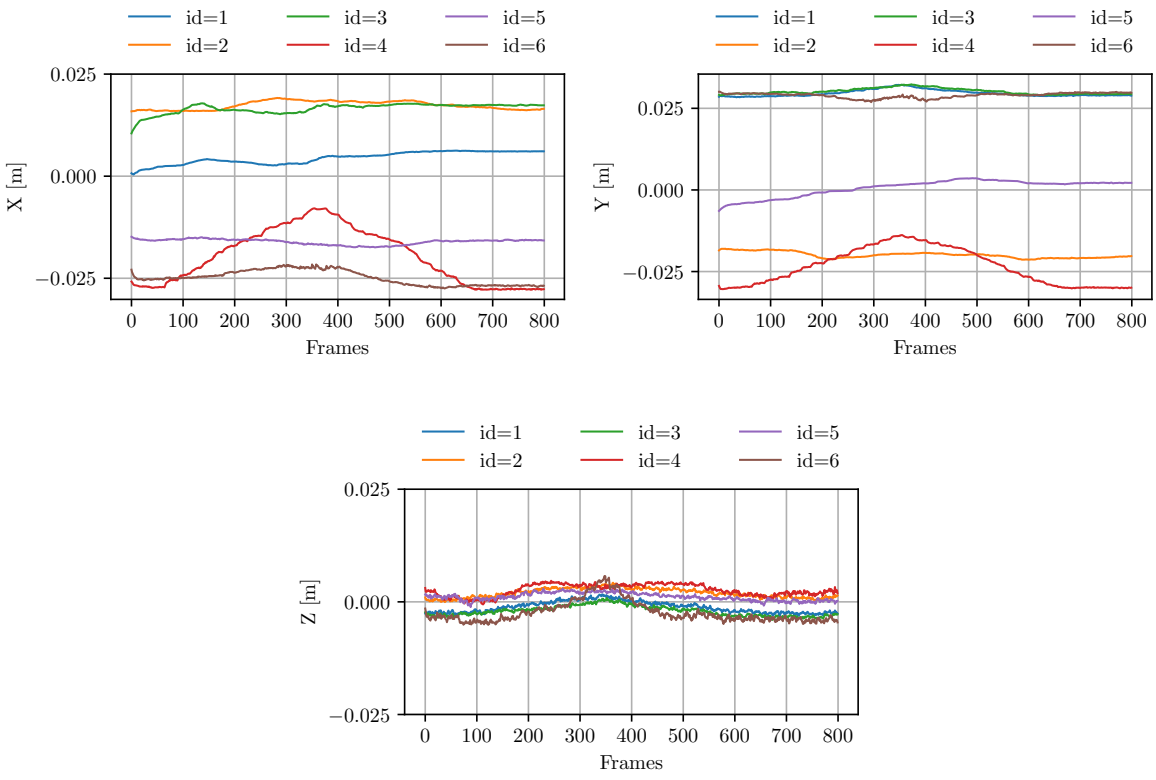


Figure 3.18: Local displacement of a subset of nodes obtained by BC-GNG during the manipulation of small sponge.

Furthermore, it is necessary to indicate that BC-GNG still exhibits some difficulty to recover the initial node position. However, unlike the previous C-GNG version, such variations do not manifest as abrupt changes in the signal due to the smoother characteristic of the displacements. This behavior is desirable since abrupt changes are directly associated with high deformations, which on the contrary do not correspond to the reality of the deformation that the object is experiencing. For example, node 3 (green) in C-GNG abruptly changes its position along Y axis somewhere after frame 600 (Figure 3.17), though, the deformation level of the object is almost reaching back to its original shape during that time. Similar stable local displacements were also observed experimentally when applying the C-GNG and BC-GNG models on the other four non-rigid objects considered, as shown in Figures A.9 and A.10. In particular, the local displacements of the ball exhibit an increase of the region correspondence error compared to other non-rigid objects. This situation may be associated to the fact that the ball contains more volume than the rest of

non-rigid objects. In such a way that occlusions cause correspondence problems by further reducing the amount of points reported by the sensor when the object is manipulated. Therefore, these experiments suggest that among other variations, the BC-GNG model obtains correspondence between regions of the shape that better reflect the deformation dynamics due to the manipulation actions. As a result, the object motion can be estimated with more confidence.

3.4 Summary

This chapter introduces a methodology for shape representation of non-rigid objects using a GNG-based model, which receives as input a 3D point clouds and produces as output a graph. The proposed formulation is called BC-GNG and extends previous work with two novel contributions. First, a regularization term is proposed to mitigate the presence of dead nodes in the graph. Second, a batch training procedure is proposed to improve the convergence of the algorithm. Both proposals are designed with the intention of producing a representation model of non-rigid objects suitable for robotic manipulation tasks.

Experimental evaluation is also provided with a set of non-rigid objects with different shapes and physical properties to support the proposed additions. Particularly, experiments evidence that BC-GNG as a graph-based representation model has potential for real-time execution which is a condition not directly accessible in other formulations. In practice, the execution time improvement is mainly attributed to the fact that the batch training can be computed efficiently using vectorized operations in current programming languages and libraries. Although, the temporal smoothing and region correspondence evaluations are not intended to be exhaustive. These suggest that the proposed batch training procedure in BC-GNG demonstrates a similar behaviour to the online training in C-GNG, while also improving the reliability of the representation by overall producing less noise and better region correspondence.

In terms of design considerations, these results suggest that, among other variations of the algorithm, BC-GNG is a more appropriate solution for applications that require graph construction of non-stationary distributions from sensor data. Also, in practical situations that require the generation of large amount of data. On the other hand, the current limitations in BC-GNG are mostly related to the GROWING phase. This phase is

not optimized in the proposed batch training procedure, hence corresponding to the most time-consuming part of the algorithm if the input distribution is unknown. At the same time, the poor graph initialization produced by the GROWING phase causes the nodes position to not recover perfectly. This directly affects the region correspondence, hence producing inaccuracies for motion analysis.

Chapter 4

Deformation Dynamics Prediction

In order to model the deformation of non-rigid objects, the shape representation obtained by the GNG-based model introduced in Chapter 3 needs to be incorporated with a new model that predicts the deformation dynamics, the latter refers to the evolution of the non-rigid object shape over time due to the manipulation actions of the robotic system. An important factor that is considered for the selection of this model is to take advantage of the structure generated in the GNG-based representation, since the structure contains important information about the physical properties of the task. In this way, learning techniques on graphs are analyzed as prediction models of the deformation dynamics of non-rigid objects.

Considering the advances in deep learning technology, several methodologies have been developed that enable its application for irregular domains such as graphs, also known as geometric deep learning [80]. Currently, *Graph Neural Network* (GNN) is considered as the main framework for learning on graph structures, being applied to a variety of scenarios with promising results [81]. In particular, among the different architecture variants formulated, the proposal of Battaglia *et al.* [82] demonstrates the ability to learn the dynamics of interactive physical systems. Within the context of this thesis, these models can be integrated to learn the complex physics behind deformation.

In the rest of this chapter, GNN-based models are studied and their ability to learn the deformation of non-rigid objects is evaluated. The methodology used to develop the model is summarized in Figure 4.1. In particular, the GNG-based representation is augmented (Section 4.1) to produce a graph that describes the state of the non-rigid object and

manipulation actions of the robotic system. Then, a GNN-based model is used to learn the deformation dynamics (Section 4.2), which uses the augmented graph representation of the current state, G_t, to predict the next state, $\widehat{G}_{t+1}$. Experimental results evaluating the performance of the GNN-based model in combination with the augmented GNG-based representation are also presented (Section 4.4).

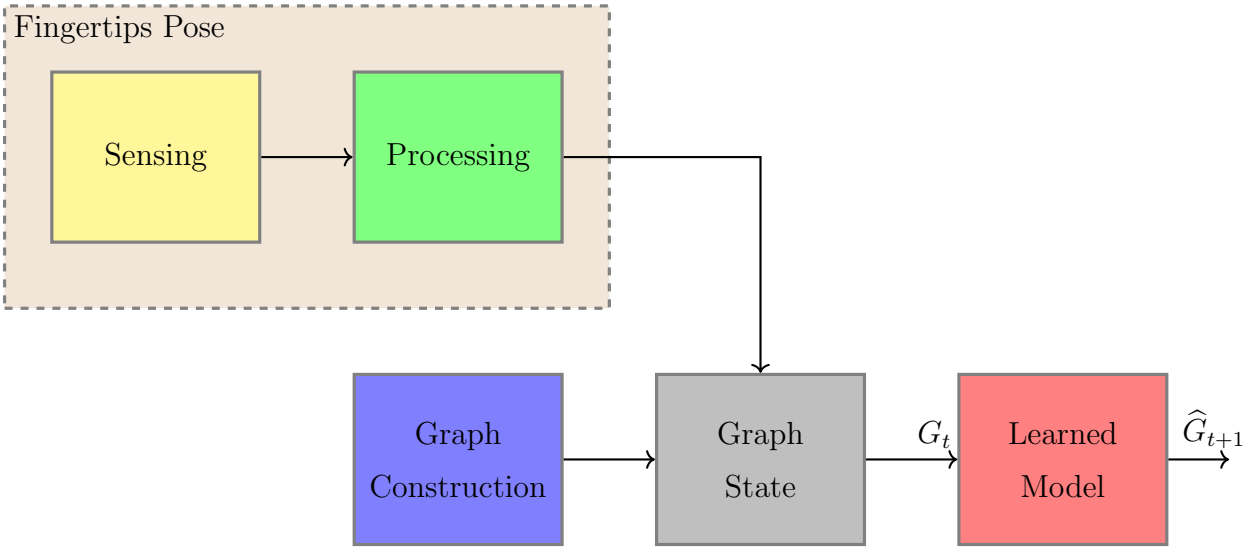


Figure 4.1: Diagram of the proposed methodology for the dynamics prediction model: (Yellow) raw data taken from sensors; (Green) data processing for fingertips pose estimation; (Blue) GNG-based shape representation; (Gray) state of the non-rigid object and contact points; (Red) GNN-based model for deformation dynamics prediction.

4.1 Graph State

As a preliminary step to the description of the GNN-based models, the GNG-based representation (detailed in Chapter 3) is augmented by including new nodes corresponding to the contact points of the fingertips in order to include the interaction information in the graph. Thus, the state representation of the non-rigid object and manipulation actions is defined as a directed graph $G = \langle O, R \rangle$. In which, $O = \{\mathbf{o}_i\}_{i=1:N_O}$ is the set of nodes with N_O cardinality, and associated feature vector $\mathbf{o}_i = \{x_i, v_i\}$, which contains the state of the system defined as position and velocity. Also, $R = \{\mathbf{r}_k, u_k, v_k\}_{k=1:N_R}$ is the set of edges with N_R cardinality, which due to the graph directionality connects an ordered pair of nodes, defined as source node u_k and destination node v_k. A visual example of the augmentation

procedure is depicted in Figure 4.2. This definition of the graph state introduces a velocity feature that is not present in GNG, as described in Section 3.1.1. For this reason, the feature vector associated to the nodes of the graph produced by GNG is also augmented by including a velocity component.

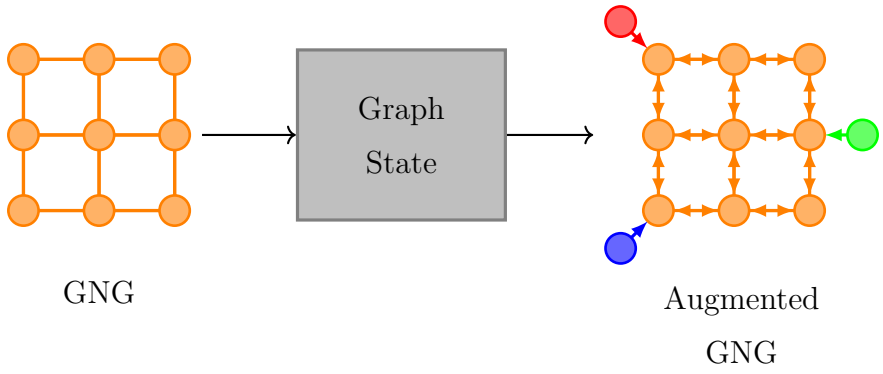


Figure 4.2: Diagram for the construction of the graph state representation of a non-rigid object and manipulation actions. Arrows ilustrate the edge direction. Contact points with the robotic hand are painted in red, green and blue colors.

The contact points (Section 4.3) provide the 6D pose of each fingertip. Therefore, the inclusion of the contact points in the graph consists of creating three new nodes, each with position feature assigned to the position component of the estimated 6D fingertip pose. In addition, an edge is created from the contact point node to the object nodes when a physical interaction occurs. The edge direction adds a causality property to the model, which establishes that the contact points nodes produce the displacement of the object nodes and not the opposite. Furthermore, the velocity feature of the nodes is computed by differentiating the signal obtained by the position feature of both, the object and the contact points nodes. In practice, the differentiation is computed by using a smooth filter, such as the Savitzky-Golay filter [83]. Thus, the velocity is estimated individually for each axis using a least-squares fit of polynomial functions.

At this point, the augmented graph contains only information of the current state of the system. However, in order to exploit the availability of historical data from previous states, a subset of the past velocities are concatenated to the current feature vector of the graph. In practice, the amount of historical data to include as new features should be selected with discretion. By concatenating too much historical data, the dimension of the feature

vector increases considerably, and thus the training of data-driven learning algorithms is affected due to the difficulty of finding an appropriate solution in such a high-dimensional space, which is a phenomenon described as the curse of dimensionality [84].

4.2 Learned Model

Initially, Battaglia *et al.* [53] introduced the *Interaction Network* (IN) as a framework for supervised learning on graph structures. Unlike standard GNNs, the IN is specifically designed to learn the dynamics of a physical interactive system represented as a graph. The learned model is characterized by being able to make predictions of the state of the system, and also to extract latent physical properties. Moreover, IN is trained following a process based on a message passing paradigm [85], which uses the evaluation of a message and an update functions to perform computations with the graph features. More specifically, the message function, ϕ_r, is responsible to perform per-edge updates. This function evaluates the collected features of the edge along with the source and destination nodes in connection, and thus compute the edge effect. Similarly, the update function, ϕ_o, is responsible to perform per-node updates. This function evaluates the collected features of the node along with those produced by the message function, and thus compute the node effect. Since the message function produces a variable number of effects associated with each node, these are reduced using an aggregation function, ρ, in order to produce a single effect. Although, the message and update functions are agnostic to the approximation method used, these are commonly approximated using ANNs.

A procedure that involves the use of message and update functions on a graph is illustrated in Figure 4.3. In this example, the objective is to produce an update of the selected node feature by determining the contributions of its four neighbors. According to our directed graph definition, the neighbors are acting as source nodes since they are sending information to the selected node, which in turn is acting as destination node. Therefore, a message function is applied to produce a computation with the features of each of the four edges in connection. In addition, an update function is also applied, which first aggregates the effects of the four edges to reduce their contributions, and then produces an update of the selected node feature. Ultimately, this procedure is repeated with all nodes to obtain a complete update of the graph features.

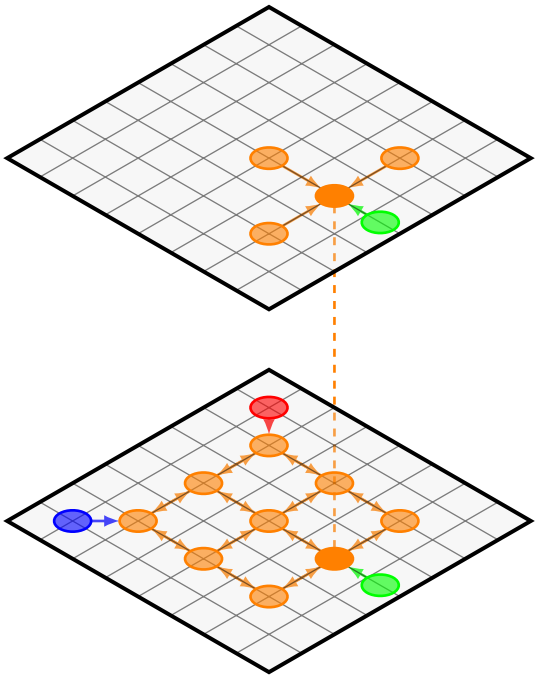


Figure 4.3: graph state given as input to the GNN-based model (bottom). ilustration of the process performed by the message and update functions on the graph (top). Selected node is painted in dark orange color, while the neighbor nodes are painted in light orange and green colors.

4.2.1 Interaction Network

The IN formulation[1] (Algorithm 6) receives at a certain time step, t, a directed graph as input, $G_t = \langle O, R \rangle$. Where, $O = \{\mathbf{o}_i\}_{i=1:N_O}$ is the set of nodes with N_O cardinality, and $R = \{\mathbf{r}_k, u_k, v_k\}_{k=1:N_R}$ is the set of edges with N_R cardinality, which connects an ordered pair of nodes u_k and v_k. Also, each node has an associated feature vector $\mathbf{o}_i = \{x_i, v_i, a_i\}$, which contains the state of the system defined as position, x_i, and velocity, v_i, and other object attributes, a_i, (e.g. mass, size). Whereas, each edge has an associated feature vector $\mathbf{r}_k = \{r_k\}$, which contains relation attributes, r_k, (e.g. elasticity, distance). In this way, IN takes the graph, G_t, and outputs a prediction of the graph features for the next time step, $\widehat{G}_{t+1}$.

[1] symbols are taken from [82], [86].

In summary, the updated graph is obtained by predicting the node features for the next time step as follows:

I $\mathbf{e}_{k,t}$: edge effect k at time t, it evaluates the message function, ϕ_r, on the collected features of the edge $\mathbf{r}_{k,t}$ and the associated source node $\mathbf{o}_{u_k,t}$ and destination node $\mathbf{o}_{v_k,t}$.

II $\mathbf{e}_{i,t}$: edge effect per node i at time t, it computes the aggregation function, ρ, it evaluates a sum operation over the edge effects $\sum_{K \in \mathcal{N}_i} \mathbf{e}_{k,t}$. Where, $\mathcal{N}_i$ is the subset of edge indices with destination node $\mathbf{o}_{i,t}$.

III $\mathbf{o}_{i,t+1}$: node feature i at time $t+1$, it evaluates the update function, ϕ_o, on the collected features of the node $\mathbf{o}_{i,t}$ and the edge effect per node $\mathbf{e}_{i,t}$.

Where, the message and update functions ϕ_r, ϕ_o are approximated using standard MLPs modules (detailed in Section 4.4), and are trained using backpropagation and gradient descent optimization.

Algorithm 6 Steps of computation in IN

Input: G_t

Output: $\widehat{G}_{t+1}$

1: $\mathbf{e}_{k,t} \leftarrow \phi_r\left(\mathbf{r}_{k,t}, \mathbf{o}_{u_k,t}, \mathbf{o}_{v_k,t}\right)_{k=1:N_R}$ ▷ I. Compute edge effect

2: $\mathbf{e}_{i,t} \leftarrow \rho\left(\mathbf{e}_{k,t}\right)_{i=1:N_O}$ ▷ II. Compute edge effect per node

3: $\mathbf{o}_{i,t+1} \leftarrow \phi_o\left(\mathbf{o}_{i,t}, \mathbf{e}_{i,t}\right)_{i=1:N_O}$ ▷ III. Update node feature

An IN is originally described [53] as a learned model for general purpose physics simulation. Indeed, results demonstrate reliable predictions for systems with several rigid bodies experiencing gravitational forces, collisions and also elastic springs. However, an IN presents some limitations when used to learn the properties of interactive systems that require long and fast propagation effects (i.e. destination nodes that should receive the effects quickly but have no direct connection to the source node that sends the message), since its formulation only considers local pairwise interactions during each time step. Therefore, several iterations of the algorithm are required in order to propagate the messages on the graph, and thus reach remote nodes.

4.2.2 Propagation Network

Li *et al.* [86] proposed the *Propagation Network* (PropNet) to mitigate the limitation of IN to learn the dynamics of various physical systems, in which long propagation of effects is necessary, such as string manipulation and box pushing. Later in [52], this model is extended to learn the dynamics of objects in environments with different material and physical properties, such as collisions of soft bodies and liquids. Moreover, these studies emphasize the benefit of using dynamic graph structures instead of fixed ones as input data to train a GNN-based model, both to mitigate memory limitation problems due to the storage of large size graphs, and also to obtain a better prediction performance by selecting physical interactions according to more realistic evolution behaviors. This also suggests that the topology of the graph is an important aspect to learn the dynamics of physical systems with evolution characteristics such as non-rigid objects using GNN-based models.

From that perspective, the PropNet formulation (Algorithm 7) proposes the inclusion of a multi-step propagation phase, which consists of computing the edge and node effects using an additional iterative process, where l corresponds to the current propagation step parameter, and is set to a value within the range of $1 \leq l \leq L$. In particular, the message and update functions, ϕ_r, ϕ_o, produce intermediate representations of the edge and node effects, which are updated per propagation step. Also, in the case of a propagation step equal to 1, PropNet is structurally equivalent to IN. Another proposal in the formulation of PropNet is the inclusion of encoder and decoder functions to better represent the edge and node features. This suggestion is based on the fact that neither the data nor the structure of the graph change during the execution of each propagation phase. In particular, the functions, ϕ_r^{enc}, ϕ_o^{enc}, are used to encode the input edge and node features, respectively. While the function, ϕ_o^{dec}, are used to decode the output node feature. In this way, these functions learn a latent representation of the graph features, which are also part of the model during training.

In summary, the updated graph is obtained by predicting the node features for the next time step as follows:

I $\mathbf{c}_{i,t}^o$: node feature encoder i at time t, it evaluates an encoder function, ϕ_o^{enc}, on the current node feature $\mathbf{o}_{i,t}$.

II $\mathbf{c}_{k,t}^{r}$: edge feature encoder k at time t, it evaluates an encoder function, ϕ_r^{enc}, on the current edge feature $\mathbf{r}_{k,t}$ and the associated source node $\mathbf{o}_{u_k,t}$ and destination node $\mathbf{o}_{v_k,t}$.

III $\mathbf{e}_{k,t}^{l}$: edge effect k at time t and propagation step l, it evaluates a message function, ϕ_r, on the collected features of the edge encoder $\mathbf{c}_{k,t}^{r}$ and the associated source node effect $\mathbf{h}_{u_k,t}^{l-1}$ and destination node effect $\mathbf{h}_{v_k,t}^{l-1}$ of the previous propagation step.

IV $\mathbf{e}_{i,t}^{l}$: edge effect per node i at time t and propagation step l, it computes an aggregation function, ρ, it evaluates a sum operation over the edge effects $\sum_{K\in\mathcal{N}_i}\mathbf{e}_{k,t}^{l}$. Where, $\mathcal{N}_i$ is the subset of edge indices with destination nodes $\mathbf{o}_{i,t}$.

V $\mathbf{h}_{i,t}^{l}$: node effect i at time t and propagation step l, it evaluates a message function, ϕ_o, on the collected features of the node encoder $\mathbf{c}_{i,t}^{o}$, the edge effect per node $\mathbf{e}_{i,t}^{l}$ and the node effect $\mathbf{h}_{i,t}^{l-1}$ of the current and previous propagation step, respectively.

VI $\mathbf{o}_{i,t}$: node feature i at time t, it evaluates a decoder function, ϕ_o^{dec}, on the node effect $\mathbf{h}_{i,t}^{L}$ in the final propagation step L.

Where, the decoder function ϕ_o^{dec}, encoder functions ϕ_o^{enc}, ϕ_r^{enc} along with the message and update functions ϕ_o, ϕ_r are approximated using standard MLPs modules (detailed in Section 4.4), and are trained using backpropagation and gradient descent optimization.

Algorithm 7 Steps of computation in PropNet

Input: G_t

Output: $\widehat{G}_{t+1}$

$\mathbf{c}_{i,t}^{o} \leftarrow \phi_o^{enc}\left(\mathbf{o}_{i,t}\right)_{i=1:N_O}$ ▷ I. Compute node encoder

$\mathbf{c}_{k,t}^{r} \leftarrow \phi_r^{enc}\left(\mathbf{r}_{k,t}, \mathbf{o}_{u_k,t}, \mathbf{o}_{v_k,t}\right)_{k=1:N_R}$ ▷ II. Compute edge encoder

$\mathbf{h}_{i,t}^{0} \leftarrow \mathbf{0}$

for all $l \in L$ **do**

  $\mathbf{e}_{k,t}^{l} \leftarrow \phi_r\left(\mathbf{c}_{k,t}^{r}, \mathbf{h}_{u_k,t}^{l-1}, \mathbf{h}_{v_k,t}^{l-1}\right)_{k=1:N_R}$ ▷ III. Compute edge effect

  $\mathbf{e}_{i,t}^{l} \leftarrow \rho\left(\mathbf{e}_{k,t}^{l}\right)_{i=1:N_O}$ ▷ IV. Compute edge effect per node

  $\mathbf{h}_{i,t}^{l} \leftarrow \phi_o\left(\mathbf{c}_{i,t}^{o}, \mathbf{e}_{i,t}^{l}, \mathbf{h}_{i,t}^{l-1}\right)_{i=1:N_O}$ ▷ V. Compute node effect

end for

$\mathbf{o}_{i,t} \leftarrow \phi_o^{dec}\left(\mathbf{h}_{i,t}^{L}\right)_{i=1:N_O}$ ▷ VI. Update node feature

4.3 Dataset

This section describes the methodologies applied for the data collection of the non-rigid object and manipulation actions used for training the GNN-based models. In that context, the configuration of the real-world robotic manipulation and sensing setup remains unchanged, as described in Section 3.2. Therefore, only the additional processing steps are further discussed.

4.3.1 Processing

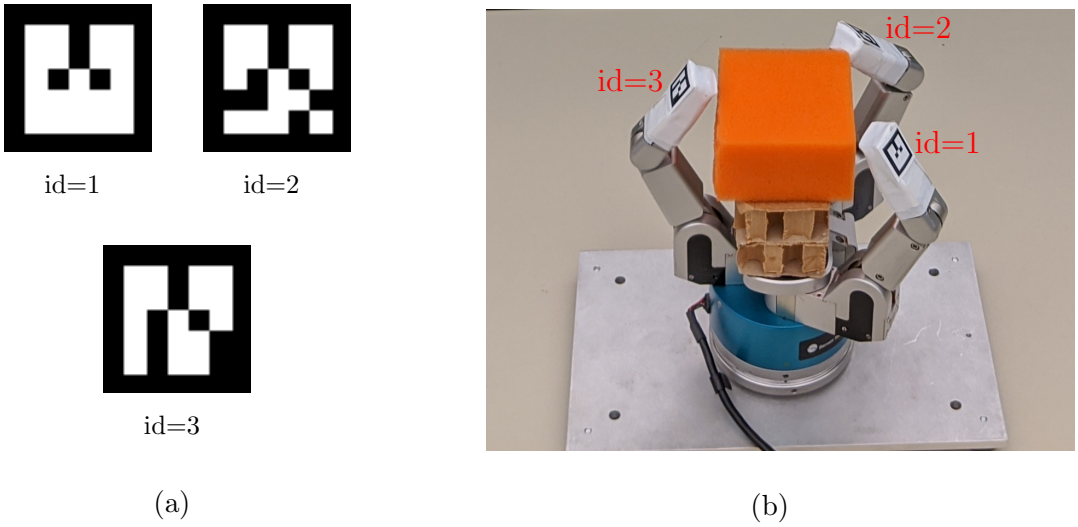
The components in the ROS architecture (Figure 3.4) are used as tools for the collection of the graph data of the non-rigid object and manipulation actions. In this way, the RGB-D sensor data is further processed in the *dataset* ROS node to estimate the fingertips pose of the hand that generates the contact points information. Then, this data is combined with the non-rigid object shape representation obtained by the proposed BC-GNG model (Section 3.1.3) to generate the final graph data. Thus, a dataset is created per object and consists of 20 files, each associated to a different trajectory. This produces an equivalent to 16,000 samples in total. This methodology provides the advantage of being easily scalable, in case more data needs to be collected.

Fingertips Pose Estimation

The data captured of the fingertips correspond to the 6D pose (i.e. 3D position and orientation) of each tip during the execution of the fingers trajectory. To facilitate the accurate estimation of the pose, a set of AR markers (Figure 4.4) are placed on each tip. Passive markers provide an affordable but reliable solution for tracking the pose of rigid objects. These markers are square and contain a unique pattern assigned to a certain tag index uniquely associated with one of the robotic hand fingers. In addition, the design is based on the ARTags fiducial marker system, and generated according to the following parameters: size of 1.8 cm, amrgin of 1-bit, and pattern of 25-bit 5×5 array. The latter controls the number of tags that can be created based on the marker dictionary. Given the physical dimensions of the robotic hand, this design enables to precisely fit each marker on the tip. In turn, the markers are visible enough to be detected in the images captured

by the RGB-D sensor.

Thus, each marker is visually identified and its 6D pose is tracked over the sequence of frames. In this way, the fingertips pose corresponds to that estimated by the markers. The pose enables to define the contact points, which are where the interactions between the fingertips and the non-rigid object take place during manipulation. In turn, the spatial relationship that establishes the occurrence of contact points is determined by a contact region with spherical shape, centered on the marker and with a radius of 2.3 cm, the latter measured considering the tip size relative to the marker location.



(a) (b)

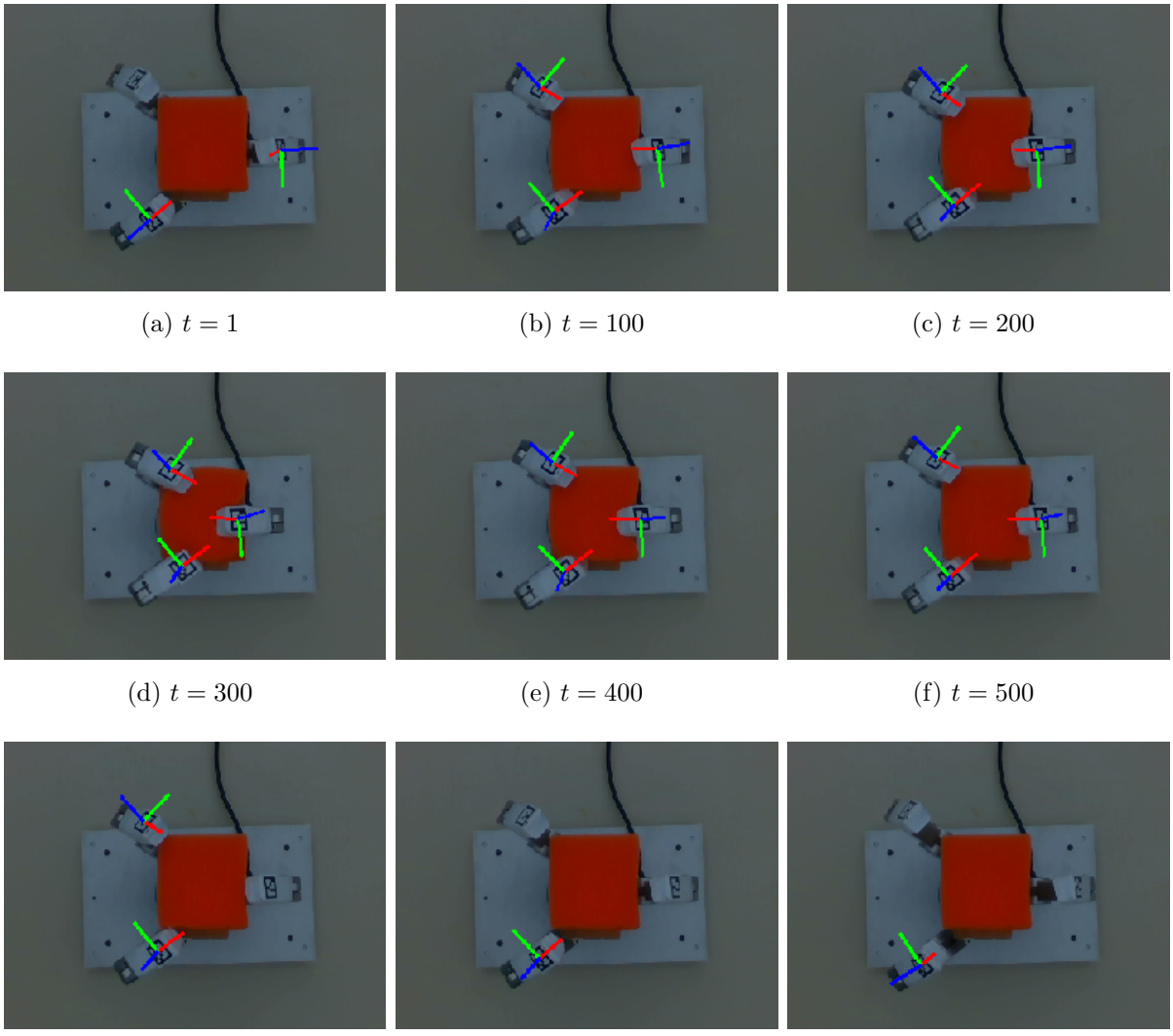
Figure 4.4: a) AR marker tags used to estimate the fingertips pose, and b) location of the markers on the robotic hand fingers.

The visual pose estimation of each marker is implemented using a combination of the Alvar Tracking SDK [87] and the OpenCV [76] libraries. This functionality is directly accessible by the ROS package of the Alvar library[2]. In this way, the *ar_track_alvar* ROS node takes as input the aligned color and depth images through the *camera* ROS topic and produces the 6D pose of each marker through the *ar_pose_marker* ROS topic.

In summary, the color image is initially processed to detect each marker. A square filtered blob detector is used to obtain a set of connected components as possible marker candidates. Then, each blob content is analyzed per pixel validating whether the pattern

[2] http://wiki.ros.org/ar_track_alvar

corresponds to one of the generated tags. In that case, the position is set to the blob center. Also, the blob is further processed to obtain its four corners, which in turn are used to estimate the orientation. The corner positions are also used in the depth image in order to fit a plane, which serves to refine the estimation, and thus obtain a more accurate pose. Ultimately, the pose is transformed relative to the camera coordinates system using the camera parameters. A subset of frames that reflects the fingertips pose estimation at various deformation levels of a non-rigid object is shown in Figure 4.5.



(a) $t = 1$ (b) $t = 100$ (c) $t = 200$

(d) $t = 300$ (e) $t = 400$ (f) $t = 500$

(g) $t = 600$ (h) $t = 700$ (i) $t = 800$

Figure 4.5: Fingertips pose estimation at different deformation levels. The fingertips pose is depicted using colored coordinates system (X=red, Y=green, Z=blue).

As a side note, for those frames in which the fingertips pose is not entirely detected due to occlusions or loss of track (e.g. first and last time steps in Figure 4.5), a linear interpolation is applied with the available data in order to fill in the missing values of the position while maintaining the last available orientation estimation. This process is performed to avoid discarding frames, and thus creating a dataset with a fixed number of samples. Also, initial tests performed with Kinect V1[3] sensor presented more complications recognizing the AR markers due to their small size in this application, hence it was necessary to move the sensor closer to the target scene. However, the quality of the depth image was affected since the range was no longer appropriate. Therefore, for this type of application, a short-range RGB-D sensor, such as the Intel RealSense SR305, is recommended.

4.4 Experiments

Despite showing interesting results to model some physics behaviors, the exploration of GNN-based models in real-world environments has been limited. Mainly due to the absence of techniques to generate training data, which in contrast are easily accessible in simulated environments. In the case of non-rigid objects, a particle simulator [88], such as NVIDIA FleX, is commonly used as an environment to provide training data. However, current robotic manipulation environments do not properly support non-rigid objects. Therefore, it becomes challenging to design a data-driven learning approach that is trained entirely in simulated environments since there is no direct access to interaction data between robots and objects.

In this section, a set of experiments are presented using the two GNN-based models (IN and PropNet) described in Section 4.2, to learn the deformation dynamics of a non-rigid object but applied for robotic manipulation tasks in a real-world environment. Unlike previous work, the construction of the graph dataset is derived from the output of the GNG-based representation, which is augmented to also provide the state of the non-rigid object and manipulation actions as described in Sections 4.1 and 4.3. In summary, GNN-based models are trained using visual observations of the physical system, which extends their usage initially restricted to simulated environments, due to the limited availability of formulations to construct dynamic graphs from sensor data.

[3]Kinect V1 refers to Kinect for XBox 360.

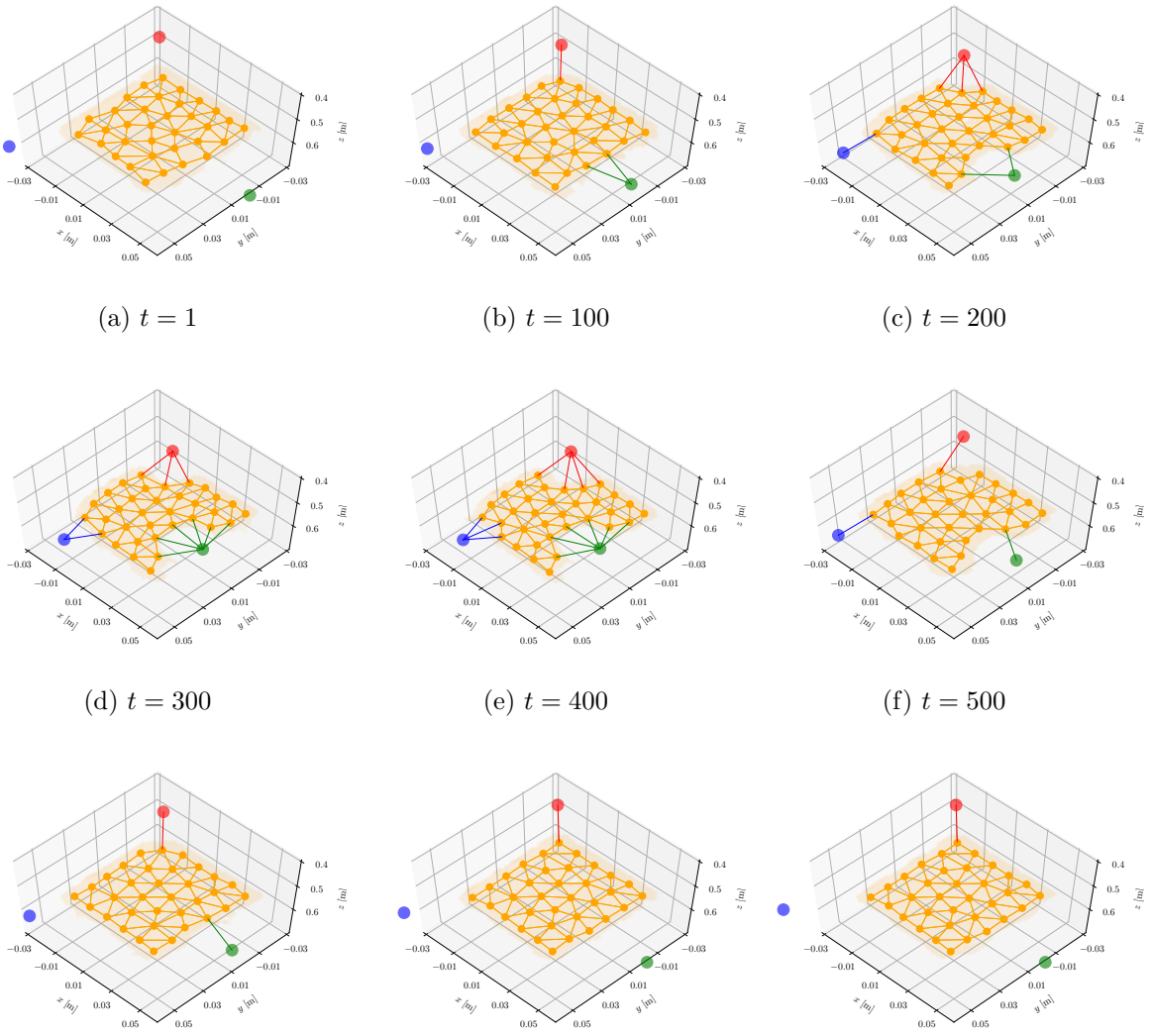
The experiments are run on a cloud instance with 1× Intel Xeon Processor @ 2.3GHz, 1× NVIDIA Tesla K80 GPU with 2,496 CUDA cores, 12 GB RAM GDDR5 VRAM, and GNU/Linux operating system. The GNN-based models are implemented in the Deep Graph Library [77] using PyTorch [78] as backend. The data preparation and signal processing steps are implemented using the SciPy library [89]. The training procedure of the models consists of 20 iterations, and using a batch size of 1. Internally, the MLP modules are trained using the Adam optimizer algorithm [90] with learning rate of 0.001 and momentums of 0.9 and 0.9999. Also, a learning rate scheduler with factor of 0.8 and patience 3 is used.

The architecture design of the models is based on the configuration presented in [52]. For experimentation purposes, the selected configuration is shared among models in order to consistently compare their performances, hence the main difference between the models is associated to the propagation step parameter, L, which corresponds to 1 for IN and 2 for PropNet. In this way, the encoder functions ϕ_r^{enc}, ϕ_o^{enc} are 2-layer MLPs with hidden sizes of 200 and 300, with output size of 200. The message and update functions ϕ_r, ϕ_o are 1-layer MLPs with hidden size of 200, and output size of 200. And, the decoder function ϕ_o^{dec} is 2-layer MLP with hidden size of 200 and output size of 3, the latter corresponds to the components of the predicted velocity in 3D. All MLP modules use the rectified linear unit (ReLU) as the activation function. The mean squared error, MSE, is used as the loss function, the prediction variables are further detailed in Section 4.4.1 For the feature design of the graph, the state feature of the current frame is a 6D vector (i.e. position and velocity in 3D) with concatenation of the velocity of the previous 5 historical frames, thus obtaining as input feature a 24D vector, while the output feature is a 3D vector (i.e. velocity in 3D). This design along with the feature dimensions create a network with 491,003 learnable parameters. In addition, each models is evaluated per object using the entire graph dataset of the non-rigid object and manipulation actions described in Section 4.3. The dataset is divided into 80% for training, 10% for validation and 10% for test, which is equivalent to 16 trajectories (i.e. 12,800 randomly shuffled samples) and 2×2 trajectories (i.e. 2×1600 samples) respectively. In addition, the dataset is normalized between 0 and 1 due to the varied scales of the position and velocity features of the graphs.

With the intention of further characterizing the properties of the non-rigid object and manipulation actions within the scope of this work, some considerations such as the object dimension and composition, the cause of the deformation and the type of manipulation are defined as follows:

- Objects are volumetric in size and only exhibit elastic deformations during interactions with themselves and with contact forces.
- Deformation in the objects are caused by the application of contact forces. Therefore, any other source of energy that may cause deformations are not considered in these experiments.
- The combination of trajectory generation with motion control processes executed by the robotic hand produces a dynamic manipulation of the non-rigid object with brief quasistatic periods.
- The manipulation of a non-rigid object is performed only by the tips of the fingers. Therefore, a maximum number of three contact points occur during the execution of the trajectory.

Furthermore, a subset of the graphs produced by the GNG-based representantion after the augmentation phase is depicted in Figure 4.6. This graph sequence shows the different states of the non-rigid object and manipulation actions corresponding to the same fingertips trajectory as in Figure 4.5. This example demonstrates the measured deformation of the non-rigid object due to the interaction with the robotic hand fingers. In addition, it visually introduces the concept of dynamic graphs since the edges that connect to the contact points and the object shape are not fixed but rather change over time according to the physical properties of the system, which in this case corresponds to the spatial location and deformation level.



(a) $t = 1$ (b) $t = 100$ (c) $t = 200$

(d) $t = 300$ (e) $t = 400$ (f) $t = 500$

(g) $t = 600$ (h) $t = 700$ (i) $t = 800$

Figure 4.6: Graph sequence of the small sponge and contact points produced by the augmented GNG-based representation at different deformation levels.

4.4.1 Prediction

Predictions of the state of a non-rigid object using GNN-based models has been experimented recently in simulated environments [52], and using a cubic-like object as an example. Therefore, the following set of experiments aims to investigate whether the augmented GNG-based representation is capable of producing a graph state of the system suitable for training a GNN-based model to learn the deformation dynamics of a non-rigid object. More

specifically, the GNN-based models (IN and PropNet) are trained to produce as output the prediction of the nodes velocity feature for the next time step $\hat{v}_{i,t+1}$, given as input the current graph state G_t detailed in Section 4.1, which contains the position, velocity and temporal features of the current state of the non-rigid object and manipulation actions. The performance of the models is measured using the mean squared error, MSE, of the predicted and labeled nodes velocity (Equation 4.1). In our notation, this statistical metrics computes the average of the squared errors between the estimated values (predicted velocities) of the model, $\hat{v}_i$, and the observed values (labeled velocities), v_i, computed as described in Section 4.1.

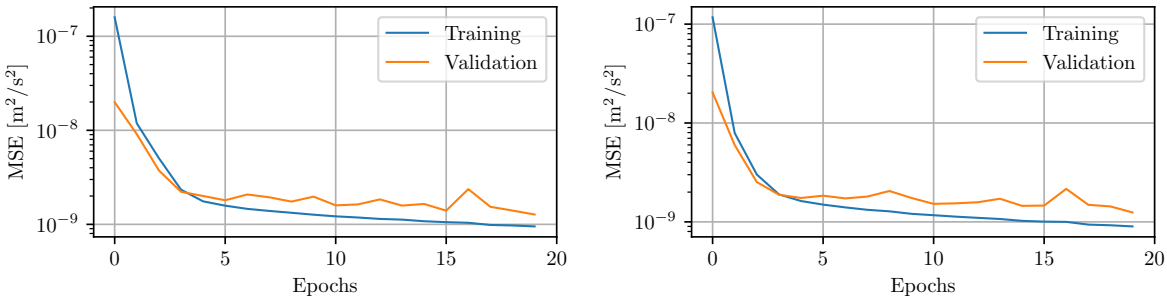
$$\text{MSE} = \frac{1}{N_O} \sum_{i=1}^{N_O} (\hat{v}_i - v_i)^2 \tag{4.1}$$

Furthermore, the models are trained in a self-supervised fashion since the labels associated at the current time step, t, correspond to the nodes velocity feature at the next time step, $t+1$. In other words, human labeling is not directly required since this approach is designed to produce all the necessary data to train the models by itself, since the temporary information of the system is available in the dataset. From this perspective, the models are analyzed primarily in two situations, first evaluating the performance of the predictions in single-step time sequences, and then evaluating the ability to generalize multi-step time sequences. Similarly to Chapter 3, the small sponge is used as a recurring example of a non-rigid object in this section. Experimental results with the rest of non-rigid objects are summarized in Appendix A.2.

Nodes Velocity

The performance of the models (IN and PropNet) is compared with respect to their training and validation errors of the predicted velocities. The results obtained for the small sponge are shown in Figure 4.7 and the other four non-rigid objects in Figure A.11. These figures correspond to semi-logarithmic visualizations due to the wide variability of the scales produced by the models throughout the epoch range. For both models, the errors of the predicted velocities of a non-rigid object exhibit a monotonic decrease as the number of iteration increases. Overall, both models produce low errors (order of magnitude 10^{-9}

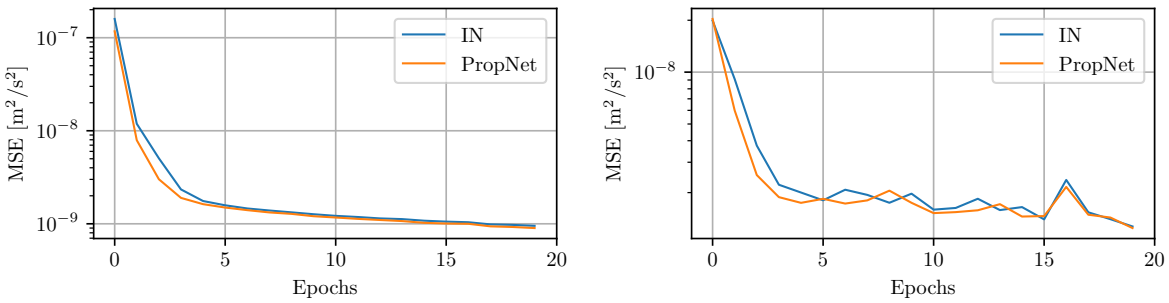
on MSE). The training error decreases significantly at the very beginning of the execution, as depicted around epoch 4 in Figure 4.7. While the validation error decreases but with constant fluctuations. The latter might be the effect of predicting small quantities from sensors, which despite being filtered during several phases, make inaccuracies contained in the representation to influence to a certain extent the stability of the model when generalizing to other scenarios in validation (i.e. others fingertips trajectories).



(a) IN (b) PropNet

Figure 4.7: Comparison of a) IN and b) PropNet models with respect to the training and validation errors of the predicted velocities obtained by using the augmented GNG-based representation of the small sponge.

Additionally, a comparison between the training and validation errors of the predicted velocities with respect to the models (IN and PropNet) is shown in Figure 4.8 for the small sponge and in Figure A.12 for the other four non-rigid objects. Even though both models start with similar MSE performances, PropNet exhibits a lower error than IN, mainly noted during the early part of iterations. Considering that the architecture design of the two models are shared, then the improvement in the performance of PropNet in this situation is attributed to the additional propagation phase, which demonstrates the property of better propagating the effects while requiring fewer iterations.



(a) Training (b) Validation

Figure 4.8: Comparison of a) training and b) validation errors of the predicted velocities obtained by IN and PropNet models using the augmented GNG-based representation of the small sponge.

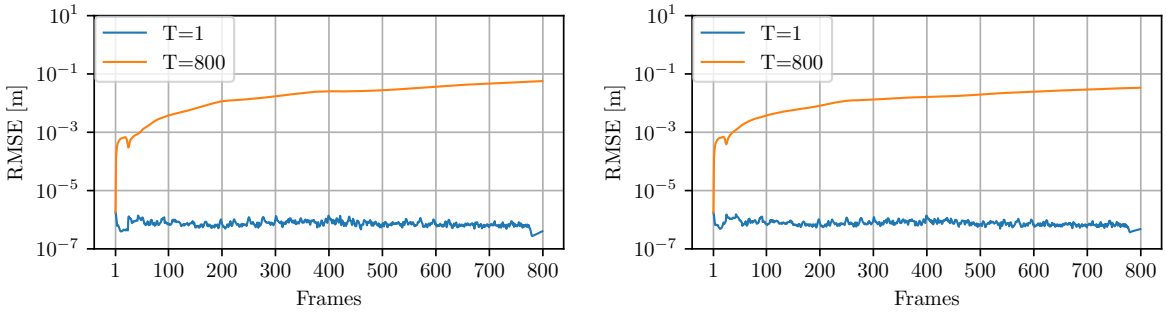
Nodes Position

The ability of the models (IN and PropNet) to produce predictions of the non-rigid object shape is investigated. More specifically, a new graph state, G_t, formed with the augmented GNG-based representation detailed in Section 4.1, is fed to the model at every T frames. Then, the model is recursively evaluated taking as input the graph state obtained in the previous frame to produce the prediction for the next frame, $\widehat{G}_{t+1}$. This process is repeated until the entire range of T frames is evaluated, $\widehat{G}_T$. In other words, for $T = 1$, a new graph is predicted from the most recent observed data at every frame; while for $T = 800$, a prediction is formed for every frame but with updates from observed data fed into the model only every 800 frames. The nodes position of the next frame, $\hat{x}_{i,t+1}$, are estimated by the direct computation of the equation of motion in classical mechanics (Equation 4.2), which uses the predicted velocities of the next frame, $\hat{v}_{i,t+1}$, and time per frame, Δt, to update the current position of each node.

$$\hat{x}_{i,t+1} = \hat{x}_{i,t} + \Delta t \cdot \hat{v}_{i,t+1} \tag{4.2}$$

A comparison of the predicted nodes position at every frame but with different T intervals is shown in Figure 4.9 for the small sponge and in Figure A.13 for the other four non-rigid objects. The hyperparameters settings are chosen based on the best performance obtained

from the validation set. In order to enable a more direct interpretation of the results, the Root Mean Square Error (RMSE) of the predicted and labeled nodes position is used as a metric. Thus, the models are evaluated in two different configurations on one of the trajectory from the test set. For the case of single-step time sequences ($T = 1$), both models obtain a relatively low and consistent error (order of magnitude of 10^{-7} on RMSE) of the nodes position throughout the entire range of frames. For the case of multi-step time sequences ($T = 800$), the errors obtained by both models remain relatively low for a prediction over a short range of frames, but degrades when the number of frames for the prediction increases. As a consequence, at some point the models are unable to predict with confidence the nodes position. In the latter case, the errors from previous executions are accumulated and the prediction diverges, causing the non-rigid object to enter an unrecoverable state. In other words, the models exhibit some robustness to produce short-term predictions of the object shape, but their performance degrades for long-term predictions.



(a) IN

(b) PropNet

Figure 4.9: Comparison of a) IN and b) PropNet models with respect to the prediction of the nodes position obtained at different time steps using the augmented GNG-based representation of the small sponge.

Additionally, a quality criterion for the predictions should be selected to better determine the extent of which the models (IN and PropNet) are capable to predict the deformation dynamics of non-rigid objects. For example, taking into account the physical dimension of the object, an average error of 1 mm (order of magnitude of 10^{-3} on RMSE) can be considered acceptable to produce visually plausible predictions of the object shape. For the evaluated trajectory, Figure 4.9 shows that both models (IN and PropNet) reaches the threshold of 1 mm in approximately 50 frames. This behavior was also observed with the

other four non-rigid objects considered, as shown in Figure A.13. Therefore, both models evidence some ability to produce short-term predictions.

4.5 Summary

This chapter presents the evaluation of GNN-based models (IN and PropNet) to learn the deformation dynamics of non-rigid objects. The main contribution in this chapter is the proposal of integration with the GNG-based representation model to enable training using real-word sensor data. The experimental evaluation presents good results by achieving low prediction errors during training, in turn showing some robustness to other manipulation trajectories during validation. As a consequence, the models effectively capture the deformation behavior of a non-rigid object when evaluated in single-step time sequences, also demonstrating potential to produce visually plausible short-term predictions of the object shape. On the other hand and despite the fact that the models are not specifically formulated to be evaluated in multi-step time sequences, the generalization results for this task are not as satisfactory as those obtained using the same GNN-based models in a similar scheme but trained in simulated environments [52]. This suggests that the representation obtained by the graph construction is not entirely capturing the long-term deformation dynamics of the non-rigid object. Such results are also most likely related to factors such as inaccuracies in sensor data, absence of additional physical quantities and a dataset that is not large and varied enough.

Nonetheless, the proposed integration has the potential to describe, to some extent, the state of a non-rigid object and can be mainly adopted in cases where a simulator is not accessible or cannot be used due to lack of support of these types of objects. Therefore, the only possible solution is to estimate a model based on sensor data. It is important to note that training data obtained by the extraction of information from sensors is a more challenging task, since more complex phenomena are inherently present in real-world environments. On the other hand, a good prediction model using real observations will significantly capture more realistic behavior of the physical system. Despite the limitations encountered, this chapter opens the possibility of exploring related techniques to improve the graph construction and better capture the long-term deformation dynamics of non-rigid objects using sensor data.

Chapter 5

Conclusions

5.1 Summary

In this work, we proposed a sensing and modeling framework to learn the deformation of non-rigid objects for scenarios in which prior knowledge about the shape or material properties of the objects is not required. This framework integrates self-organizing and graph neural network models to learn a graph representation of the shape and produce predictions of the deformation dynamics due to the manipulation actions of the robotic system.

Overall, we conclude that the proposed GNG-based shape representation model demonstrates capabilities for representing and tracking a non-stationary distribution constituting the deformation that a non-rigid object undergoes, while satisfying some design considerations for the application in robotic manipulation tasks. Hence, this framework belongs to a handful of formulations available with such properties. In turn, the GNN-based dynamics prediction model demonstrates mixed results by performing adequately for the evaluation of single-step time sequences, and exhibiting potential for the evaluation of multi-step time sequences, with the latter limited to short-term predictions. Therefore, the proposed framework is capable of modeling the deformation of non-rigid objects from sensor data to some extent including short-term predictions.

5.2 Contributions

The main original contributions from this research are:

- Development of a GNG-based approach to learn the shape representation of non-rigid objects entirely from sensor data. A new model called BC-GNG is proposed that extends previous work by including outlier regularization and batch training procedures. These additions evidence an improvement in the estimation of the object motion associated with the deformation.
- Development of a GNN-based approach to learn the deformation dynamics of non-rigid objects. The proposed approach enables to train the models entirely from sensor data, which demonstrates potential to produce short-term predictions of the deformation dynamics.
- As an extra contribution beyond the domain of non-rigid objects, a promising integration between self-organizing and graph neural networks is proposed to learn representation and prediction models of non-stationary distributions.

Until now, this research resulted in two publications: A survey paper that describes the state-of-the-art approaches in sensing, modeling and control of non-rigid object in robotic manipulation [7]. Also, a conference paper that analyzes the proposal of integration of GNG and GNN as learning models for the representation and prediction of shapes [55].

5.3 Future Works

On the shape representation model, a proposal to improve the execution of the GROWING phase in the GNG-based model is worth considering, since this phase is the most time-consuming computation when the shape of the input distribution is unknown. More work is still needed to propose a batch or similar implementation that speed up the execution of this part of the algorithm. On the deformation dynamics prediction model based on real-world data, a proposal to train longer and with more data is recommended since its performance appears to be several orders of magnitude lower than that evidenced in simulated approaches. Also, another proposal in the GNN-based model could be the inclusion

of predictions of the connection edges, since currently there is no direct mechanism that produces the topology of the graph structure as the object deforms.

In addition, a possible solution to improve the modeling capabilities of the proposed framework for long-term predictions is the exploration of other sensing measurements and modalities. Currently, the experiments presented here are limited to the data taken by a single RGB-D sensor, which captures partial observations of the environment and also suffers from occlusions with the robotic hand itself. Therefore, the combination of multiple cameras or dynamic viewpoints could be used with other techniques to complete the shape of the non-rigid object, and thus refine the shape representation model. Moreover, only visual observations are used to determine the position and velocity information of the non-rigid object, thus data extracted from tactile sensors could be used to add more features to the graph state, and thus improve on the generalization capabilities of the deformation dynamics prediction model.

Other potential areas of work in which the proposed framework can be directly applied include shape control and trajectory planning of the non-rigid object manipulation based on the predictions obtained by the models.

References

[1] O. Khatib, K. Yokoi, O. Brock, K. Chang, and A. Casal, "Robots in Human Environments: Basic Autonomous Capabilities," *The International Journal of Robotics Research*, vol. 18, pp. 684–696, July 1999.

[2] R. M. Murray, Z. Li, and S. Sastry, *A Mathematical Introduction to Robotic Manipulation*. Boca Raton: CRC Press, 1994.

[3] J. Bohg, K. Hausman, B. Sankaran, O. Brock, D. Kragic, S. Schaal, and G. S. Sukhatme, "Interactive Perception: Leveraging Action in Perception and Perception in Action," *IEEE Transactions on Robotics*, vol. 33, pp. 1273–1291, Dec. 2017.

[4] O. M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," *The International Journal of Robotics Research*, vol. 39, pp. 3–20, Jan. 2020.

[5] A. Billard and D. Kragic, "Trends and challenges in robot manipulation," *Science*, vol. 364, p. eaat8414, June 2019.

[6] J. Sanchez, J.-A. Corrales, B.-C. Bouzgarrou, and Y. Mezouar, "Robotic manipulation and sensing of deformable objects in domestic and industrial applications: A survey," *The International Journal of Robotics Research*, vol. 37, pp. 688–716, June 2018.

[7] F. Nadon, A. Valencia, and P. Payeur, "Multi-Modal Sensing and Robotic Manipulation of Non-Rigid Objects: A Survey," *Robotics*, vol. 7, p. 74, Nov. 2018.

[8] A.-M. Cretu and P. Payeur, "Harnessing Vision and Touch for Compliant Robotic Interaction with Soft or Rigid Objects," in *Advanced Interfacing Techniques for Sensors*

: Measurement Circuits and Systems for Intelligent Sensors (B. George, J. K. Roy, V. J. Kumar, and S. C. Mukhopadhyay, eds.), vol. 25 of *Smart Sensors, Measurement and Instrumentation*, pp. 269–290, Cham: Springer International Publishing, 2017.

[9] P. Güler, K. Pauwels, A. Pieropan, H. Kjellström, and D. Kragic, "Estimating the deformability of elastic materials using optical flow and position-based dynamics," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, (Seoul, South Korea), pp. 965–971, IEEE, Nov. 2015.

[10] F. Hui, P. Payeur, and A.-M. Cretu, "Visual Tracking of Deformation and Classification of Non-Rigid Objects with Robot Hand Probing," *Robotics*, vol. 6, p. 5, Mar. 2017.

[11] A. Jordt, I. Schiller, J. Bruenger, and R. Koch, "High-Resolution Object Deformation Reconstruction with Active Range Camera," in *Pattern Recognition* (M. Goesele, S. Roth, A. Kuijper, B. Schiele, and K. Schindler, eds.), vol. 6376 of *Lecture Notes in Computer Science*, (Berlin, Heidelberg), pp. 543–552, Springer Berlin Heidelberg, 2010.

[12] A. Jordt and R. Koch, "Fast Tracking of Deformable Objects in Depth and Colour Video," in *Proceedings of the British Machine Vision Conference 2011*, (Dundee), pp. 114.1–114.11, British Machine Vision Association, 2011.

[13] A. R. Fugl, A. Jordt, H. G. Petersen, M. Willatzen, and R. Koch, "Simultaneous Estimation of Material Properties and Pose for Deformable Objects from Depth and Color Images," in *Pattern Recognition* (A. Pinz, T. Pock, H. Bischof, and F. Leberl, eds.), vol. 7476 of *Lecture Notes in Computer Science*, (Berlin, Heidelberg), pp. 165–174, Springer, 2012.

[14] I. Leizea, A. Mendizabal, H. Alvarez, I. Aguinaga, D. Borro, and E. Sanchez, "Real-Time Visual Tracking of Deformable Objects in Robot-Assisted Surgery," *IEEE Computer Graphics and Applications*, vol. 37, pp. 56–68, Jan. 2017.

[15] H. Lin, F. Guo, F. Wang, and Y.-B. Jia, "Picking up a soft 3D object by "feeling" the grip," *The International Journal of Robotics Research*, vol. 34, pp. 1361–1384, Sept. 2015.

[16] Z. Kappassov, J.-A. Corrales, and V. Perdereau, "Tactile sensing in dexterous robot hands — Review," *Robotics and Autonomous Systems*, vol. 74, pp. 195–220, Dec. 2015.

[17] D. Mira, A. Delgado, C. M. Mateo, S. T. Puente, F. A. Candelas, and F. Torres, "Study of dexterous robotic grasping for deformable objects manipulation," in *2015 23rd Mediterranean Conference on Control and Automation (MED)*, (Torremolinos, Spain), pp. 262–266, IEEE, June 2015.

[18] A. Delgado, C. A. Jara, D. Mira, and F. Torres, "A tactile-based grasping strategy for deformable objects' manipulation and deformability estimation," in *Proceedings of the 12th International Conference on Informatics in Control, Automation and Robotics*, vol. 02, (Colmar, France), pp. 369–374, IEEE, 2015.

[19] A. Delgado, J. A. Corrales, Y. Mezouar, L. Lequievre, C. Jara, and F. Torres, "Tactile control based on Gaussian images and its application in bi-manual manipulation of deformable objects," *Robotics and Autonomous Systems*, vol. 94, pp. 148–161, Aug. 2017.

[20] L. Zaidi, J. A. Corrales, B. C. Bouzgarrou, Y. Mezouar, and L. Sabourin, "Model-based strategy for grasping 3D deformable objects using a multi-fingered robotic hand," *Robotics and Autonomous Systems*, vol. 95, pp. 196–206, Sept. 2017.

[21] V. E. Arriola-Rios and J. L. Wyatt, "A Multimodal Model of Object Deformation Under Robotic Pushing," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 9, pp. 153–169, June 2017.

[22] A. Cretu, P. Payeur, and E. M. Petriu, "Soft Object Deformation Monitoring and Learning for Model-Based Robotic Hand Manipulation," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 42, pp. 740–753, June 2012.

[23] F. Khalil, P. Payeur, and A.-M. Cretu, "Integrated Multisensory Robotic Hand System for Deformable Object Manipulation," in *Proceedings of the IASTED International Conference on Robotics and Applications*, (Cambridge, Massachusetts, USA), ACTAPRESS, 2010.

[24] B. Tawbe and A.-M. Cretu, "Acquisition and Neural Network Prediction of 3D Deformable Object Shape Using a Kinect and a Force-Torque Sensor," *Sensors*, vol. 17, p. 1083, May 2017.

[25] B. Frank, R. Schmedding, C. Stachniss, M. Teschner, and W. Burgard, "Learning the elasticity parameters of deformable objects with a manipulation robot," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Taipei), pp. 1877–1883, IEEE, Oct. 2010.

[26] B. Frank, C. Stachniss, R. Schmedding, M. Teschner, and W. Burgard, "Learning object deformation models for robot motion planning," *Robotics and Autonomous Systems*, vol. 62, pp. 1153–1174, Aug. 2014.

[27] A. Petit, F. Ficuciello, G. A. Fontanelli, L. Villani, and B. Siciliano, "Using Physical Modeling and RGB-D Registration for Contact Force Sensing on Deformable Objects," in *Proceedings of the 14th International Conference on Informatics in Control, Automation and Robotics*, vol. 2, (Madrid, Spain), pp. 24–33, ScitePress, 2017.

[28] A. Petit, V. Lippiello, and B. 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)*, (Hamburg, Germany), pp. 3914–3921, IEEE, Sept. 2015.

[29] P. Gil, C. M. Mateo, Á. Delgado, and F. Torres, "Visual/Tactile sensing to monitor grasps with robot-hand for planar elastic objects," in *Proceedings of ISR 2016: 47st International Symposium on Robotics*, (Munich, Germany), p. 7, IEEE, 2016.

[30] S. Caccamo, P. Güler, H. Kjellström, and D. Kragic, "Active perception and modeling of deformable surfaces using Gaussian processes and position-based dynamics," in *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, (Cancun, Mexico), pp. 530–537, IEEE, Nov. 2016.

[31] J. Montagnat, H. Delingette, and N. Ayache, "A review of deformable surfaces: Topology, geometry and deformation," *Image and Vision Computing*, vol. 19, pp. 1023–1040, Dec. 2001.

[32] D. K. Pai, K. van den Doel, D. L. James, J. Lang, J. E. Lloyd, J. L. Richmond, and S. H. Yau, "Scanning physical interaction behavior of 3D objects," in *Proceedings of*

the 28th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '01, (New York, NY, USA), pp. 87–96, Association for Computing Machinery, Aug. 2001.

[33] J. Lang, D. K. Pai, and R. J. Woodham, "Acquisition of Elastic Models for Interactive Simulation," *The International Journal of Robotics Research*, vol. 21, pp. 713–733, Aug. 2002.

[34] A. R. Fugl, H. G. Petersen, and M. Willatzen, "Simulation of Flexible Objects in Robotics," in *Simulation, Modeling, and Programming for Autonomous Robots* (I. Noda, N. Ando, D. Brugali, and J. J. Kuffner, eds.), vol. 7628 of *Lecture Notes in Computer Science*, (Berlin, Heidelberg), pp. 89–100, Springer, 2012.

[35] A. Petit, V. Lippiello, and B. Siciliano, "Tracking Fractures of Deformable Objects in Real-Time with an RGB-D Sensor," in *2015 International Conference on 3D Vision*, (Lyon, France), pp. 632–639, IEEE, Oct. 2015.

[36] 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 (IROS)*, (Madrid, Spain), pp. 4007–4013, Oct. 2018.

[37] H. Lin, F. Guo, F. Wang, and Y. Jia, "Picking up soft 3D objects with two fingers," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, (Hong Kong, China), pp. 3656–3661, IEEE, May 2014.

[38] Y.-B. Jia, F. Guo, and H. Lin, "Grasping deformable planar objects: Squeeze, stick/slip analysis, and energy-based optimalities," *The International Journal of Robotics Research*, vol. 33, pp. 866–897, May 2014.

[39] S. Duenser, J. M. Bern, R. Poranne, and S. Coros, "Interactive Robotic Manipulation of Elastic Objects," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (Madrid, Spain), pp. 3476–3481, IEEE, Oct. 2018.

[40] Z. Lazher, B. Belhassen-Chedli, L. Sabourin, and M. Youcef, "Modeling and analysis of 3D deformable object grasping," in *2014 23rd International Conference on Robotics*

in Alpe-Adria-Danube Region (RAAD), (Smolenice, Slovakia), pp. 1–8, IEEE, Sept. 2014.

[41] M. Müller, B. Heidelberger, M. Teschner, and M. Gross, "Meshless Deformations Based on Shape Matching," in *ACM SIGGRAPH 2005 Papers*, (Los Angeles, California), pp. 471–478, ACM, 2005.

[42] P. Güler, A. Pieropan, M. Ishikawa, and D. Kragic, "Estimating deformability of objects using meshless shape matching," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (Vancouver, BC), pp. 5941–5948, IEEE, Sept. 2017.

[43] C. M. Mateo, P. Gil, D. Mira, and F. Torres, "Analysis of shapes to measure surfaces: An approach for detection of deformations," in *2015 12th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, vol. 02, (Colmar, France), pp. 60–65, IEEE, 2015.

[44] C. M. Mateo, P. Gil, and F. Torres, "3D Visual Data-Driven Spatiotemporal Deformations for Non-Rigid Object Grasping Using Robot Hands," *Sensors*, vol. 16, p. 640, May 2016.

[45] C. M. Mateo, P. Gil, and F. Torres, "Computation of Curvature Skeleton to Measure Deformations in Surfaces," in *Informatics in Control, Automation and Robotics 12th International Conference, ICINCO 2015* (J. Filipe, K. Madani, O. Gusikhin, and J. Sasiadek, eds.), vol. 383 of *Lecture Notes in Electrical Engineering*, (Colmar, France), pp. 197–207, Springer International Publishing, 2016.

[46] F. Hui, P. Payeur, and A. Cretu, "In-hand object material characterization with fast level set in log-polar domain and dynamic time warping," in *2017 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, (Torino, Italy), pp. 1–6, IEEE, May 2017.

[47] B. Tawbe and A.-M. Cretu, "Data-Driven Representation of Soft Deformable Objects Based on Force-Torque Data and 3D Vision Measurements," *Proceedings*, vol. 1, no. 2, p. 22, 2016.

[48] M. Teschner, B. Heidelberger, M. Muller, and M. Gross, "A versatile and robust model for geometrically complex deformable solids," in *Proceedings Computer Graphics International, 2004*, (Crete, Greece), pp. 312–319, IEEE, June 2004.

[49] A. Cherubini, J. Leitner, V. Ortenzi, and P. Corke, "Towards vision-based manipulation of plastic materials," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (Madrid, Spain), pp. 485–490, IEEE, Oct. 2018.

[50] Z. Hu, P. Sun, and J. Pan, "Three-Dimensional Deformable Object Manipulation Using Fast Online Gaussian Process Regression," *IEEE Robotics and Automation Letters*, vol. 3, pp. 979–986, Apr. 2018.

[51] 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*, vol. 4, pp. 4255–4261, Oct. 2019.

[52] Y. Li, J. Wu, R. Tedrake, J. B. Tenenbaum, and A. Torralba, "Learning Particle Dynamics for Manipulating Rigid Bodies, Deformable Objects, and Fluids," in *International Conference on Learning Representations*, 2019.

[53] P. Battaglia, R. Pascanu, M. Lai, D. Jimenez Rezende, and K. Kavukcuoglu, "Interaction Networks for Learning about Objects, Relations and Physics," in *Advances in Neural Information Processing Systems 29* (D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, eds.), (Barcelona, Spain), pp. 4502–4510, Curran Associates, Inc., 2016.

[54] D. Mrowca, C. Zhuang, E. Wang, N. Haber, L. F. Fei-Fei, J. Tenenbaum, and D. L. Yamins, "Flexible neural representation for physics prediction," in *Advances in Neural Information Processing Systems 31* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), (Montreal, Canada), pp. 8799–8810, Curran Associates, Inc., 2018.

[55] A. J. Valencia, F. Nadon, and P. Payeur, "Toward Real-Time 3D Shape Tracking of Deformable Objects for Robotic Manipulation and Shape Control," in *2019 IEEE SENSORS*, (Montreal, QC, Canada), pp. 1–4, IEEE, Oct. 2019.

[56] S. Blumenthal, E. Prassler, J. Fischer, and W. Nowak, "Towards identification of best practice algorithms in 3D perception and modeling," in *2011 IEEE International Conference on Robotics and Automation*, (Shanghai, China), pp. 3554–3561, IEEE, May 2011.

[57] F. Nadon and P. Payeur, "Automatic Selection of Grasping Points for Shape Control of Non-Rigid Objects," in *2019 IEEE International Symposium on Robotic and Sensors Environments (ROSE)*, (Ottawa, ON, Canada), pp. 1–7, IEEE, June 2019.

[58] B. Fritzke, "A self-organizing network that can follow non-stationary distributions," in *Artificial Neural Networks — ICANN'97* (W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicoud, eds.), Lecture Notes in Computer Science, (Berlin, Heidelberg), pp. 613–618, Springer, 1997.

[59] J. García-Rodríguez and J. M. García-Chamizo, "Surveillance and human–computer interaction applications of self-growing models," *Applied Soft Computing*, vol. 11, pp. 4413–4431, Oct. 2011.

[60] H. Frezza-Buet, "Online computing of non-stationary distributions velocity fields by an accuracy controlled growing neural gas," *Neural Networks*, vol. 60, pp. 203–221, Dec. 2014.

[61] S. Orts-Escolano, J. Garcia-Rodriguez, V. Morell, M. Cazorla, M. Saval, and J. Azorin, "Processing point cloud sequences with Growing Neural Gas," in *2015 International Joint Conference on Neural Networks (IJCNN)*, (Killarney, Ireland), pp. 1–8, IEEE, July 2015.

[62] X.-F. Han, J. S. Jin, M.-J. Wang, W. Jiang, L. Gao, and L. Xiao, "A review of algorithms for filtering the 3D point cloud," *Signal Processing: Image Communication*, vol. 57, pp. 103–112, Sept. 2017.

[63] S. Orts-Escolano, V. Morell, J. García-Rodríguez, and M. Cazorla, "Point cloud data filtering and downsampling using growing neural gas," in *The 2013 International Joint Conference on Neural Networks (IJCNN)*, (Dallas, TX, USA), pp. 1–8, IEEE, Aug. 2013.

[64] B. Fritzke, "A Growing Neural Gas Network Learns Topologies," in *Advances in Neural Information Processing Systems 7*, pp. 625–632, MIT Press, 1995.

[65] T. Martinetz and K. Schulten, "Topology representing networks," *Neural Networks*, vol. 7, pp. 507–522, Jan. 1994.

[66] T. Martinetz and K. Schulten, "A "neural-gas" network learns topologies," *Artificial Neural Networks*, 1991.

[67] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural Networks*, vol. 113, pp. 54–71, May 2019.

[68] M. Cottrell, B. Hammer, A. Hasenfuß, and T. Villmann, "Batch and median neural gas," *Neural Networks*, vol. 19, pp. 762–771, July 2006.

[69] T. Kohonen, *Self-Organizing Maps*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995. OCLC: 851381250.

[70] Barrett Advanced Robotics, "Barrett Hand." https://advanced.barrett.com/barretthand, 2019.

[71] Intel Corporation, "Intel RealSense depth camera SR305." https://www.intelrealsense.com/depth-camera-sr305, 2019.

[72] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. C. Wheeler, and A. Y. Ng, "ROS: An open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[73] W. T. Townsend, "BarrettHand Grasper: Programmably Flexible Part Handling and Assembly," in *Humanoid Robotics: A Reference* (A. Goswami and P. Vadakkepat, eds.), pp. 535–551, Dordrecht: Springer Netherlands, 2019.

[74] A. Zabatani, V. Surazhsky, E. Sperling, S. B. Moshe, O. Menashe, D. H. Silver, T. Karni, A. M. Bronstein, M. M. Bronstein, and R. Kimmel, "Intel® RealSense™ SR300 Coded light depth Camera," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2019.

[75] M. T. Mason, *Mechanics of Robotic Manipulation*. The MIT Press, 2001.

[76] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, vol. 25, pp. 120–125, 2000.

[77] M. Wang, L. Yu, D. Zheng, Q. Gan, Y. Gai, Z. Ye, M. Li, J. Zhou, Q. Huang, C. Ma, Z. Huang, Q. Guo, H. Zhang, H. Lin, J. Zhao, J. Li, A. Smola, and Z. Zhang, "Deep Graph Library: Towards Efficient And Scalable Deep Learning on Graphs," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[78] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[79] A. Cretu, P. Payeur, and E. M. Petriu, "Selective Range Data Acquisition Driven by Neural-Gas Networks," *IEEE Transactions on Instrumentation and Measurement*, vol. 58, pp. 2634–2642, Aug. 2009.

[80] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric Deep Learning: Going beyond Euclidean data," *IEEE Signal Processing Magazine*, vol. 34, pp. 18–42, July 2017.

[81] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph Neural Networks: A Review of Methods and Applications," *arXiv:1812.08434 [cs, stat]*, Dec. 2018.

[82] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, "Relational inductive biases, deep learning, and graph networks," *arXiv:1806.01261 [cs, stat]*, June 2018.

[83] A. Savitzky and M. J. E. Golay, "Smoothing and Differentiation of Data by Simplified Least Squares Procedures," *Analytical Chemistry*, vol. 36, pp. 1627–1639, July 1964.

[84] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, Nov. 2016.

[85] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural Message Passing for Quantum Chemistry," in *International Conference on Machine Learning*, vol. 70, (Sydney, Australia), pp. 1263–1272, PMLR, 2017.

[86] Y. Li, J. Wu, J.-Y. Zhu, J. B. Tenenbaum, A. Torralba, and R. Tedrake, "Propagation Networks for Model-Based Control Under Partial Observation," in *2019 International Conference on Robotics and Automation (ICRA)*, (Montreal, QC, Canada), pp. 1205–1211, IEEE, May 2019.

[87] "Alvar Tracking SDK." http://virtual.vtt.fi/virtual/proj2/multimedia/alvar/index.html, 2019.

[88] M. Macklin, M. Müller, N. Chentanez, and T.-Y. Kim, "Unified Particle Physics for Real-time Applications," *ACM Transactions on Graphics*, vol. 33, pp. 1–12, July 2014.

[89] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, and P. van Mulbregt, "SciPy 1.0: Fundamental algorithms for scientific computing in Python," *Nature Methods*, pp. 1–12, Feb. 2020.

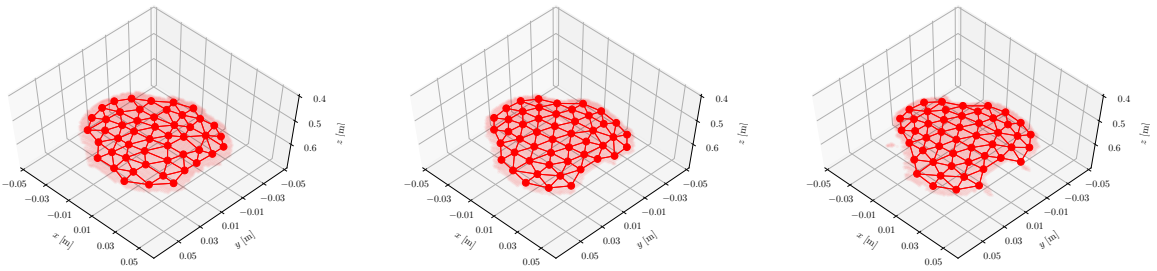
[90] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *3rd International Conference for Learning Representations*, (San Diego), 2015.

Appendix A

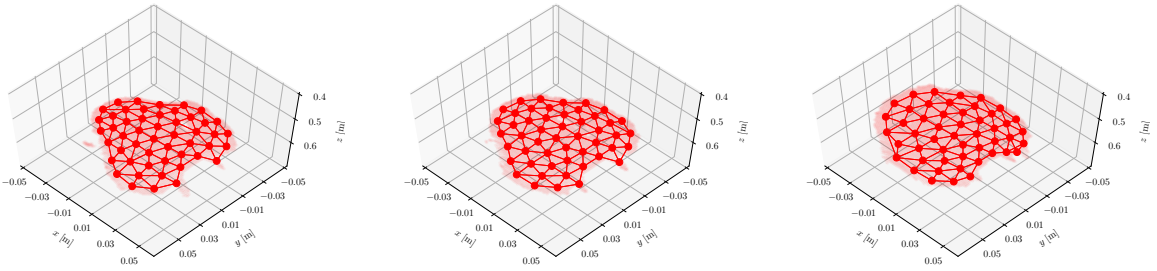
Additional Results

This appendix presents additional results obtained for the rest of non-rigid objects considered in this work. These results are used to complement the analyses performed in chapters 3 and 4, and thus provide a more comprehensive understanding of the results that support the conclusions derived in chapter 5.

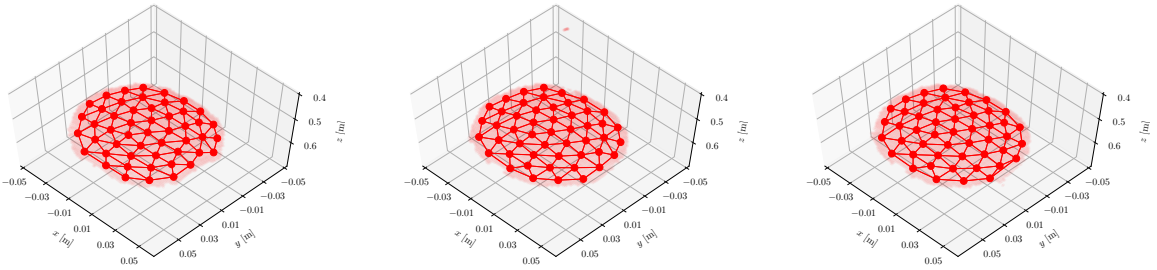
A.1 Shape Representation



(a) $t = 1$ (b) $t = 100$ (c) $t = 200$

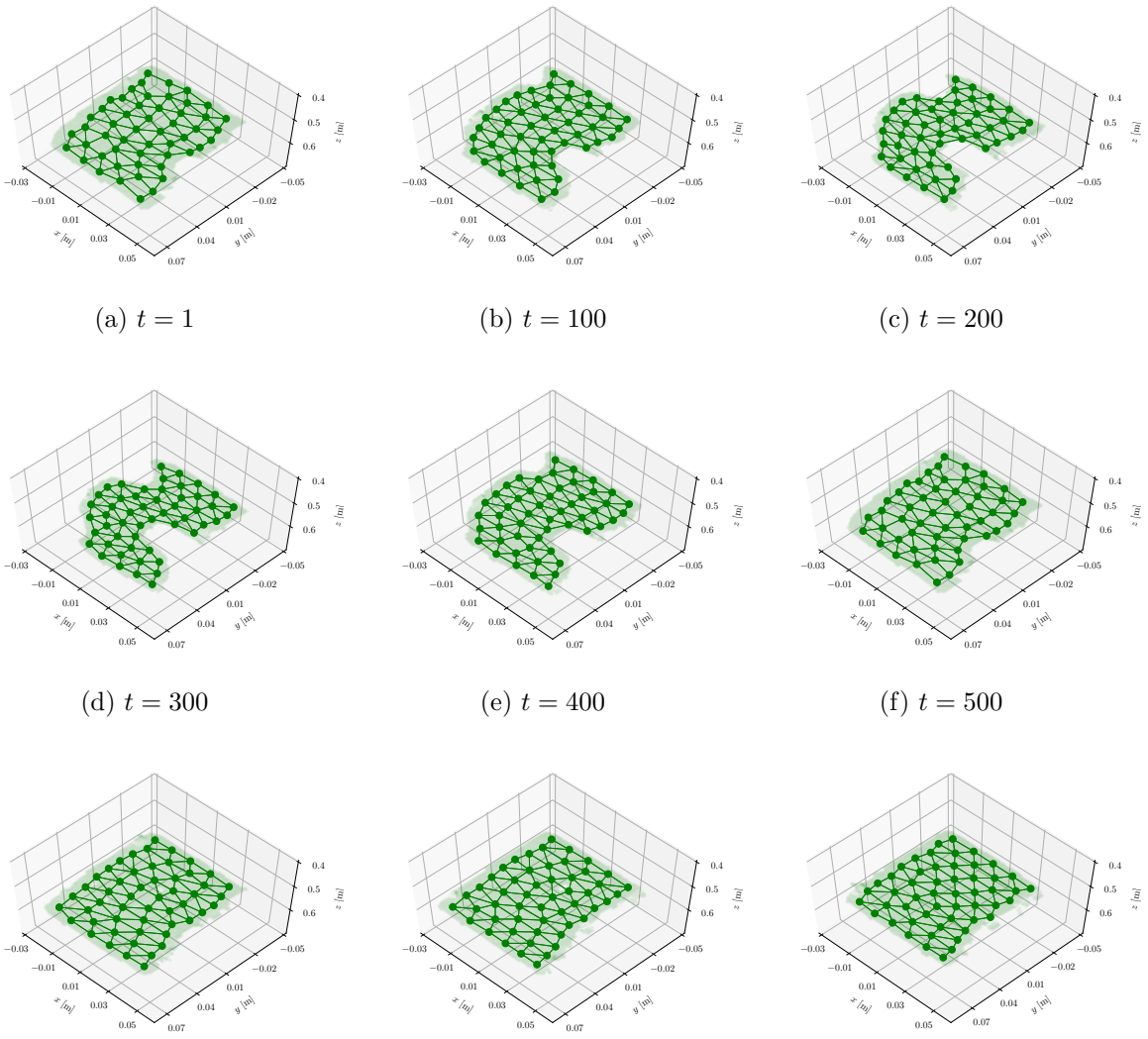


(d) $t = 300$ (e) $t = 400$ (f) $t = 500$



(g) $t = 600$ (h) $t = 700$ (i) $t = 800$

Figure A.1: Point cloud and graph sequences of the ball obtained by the BC-GNG model at different deformation levels.

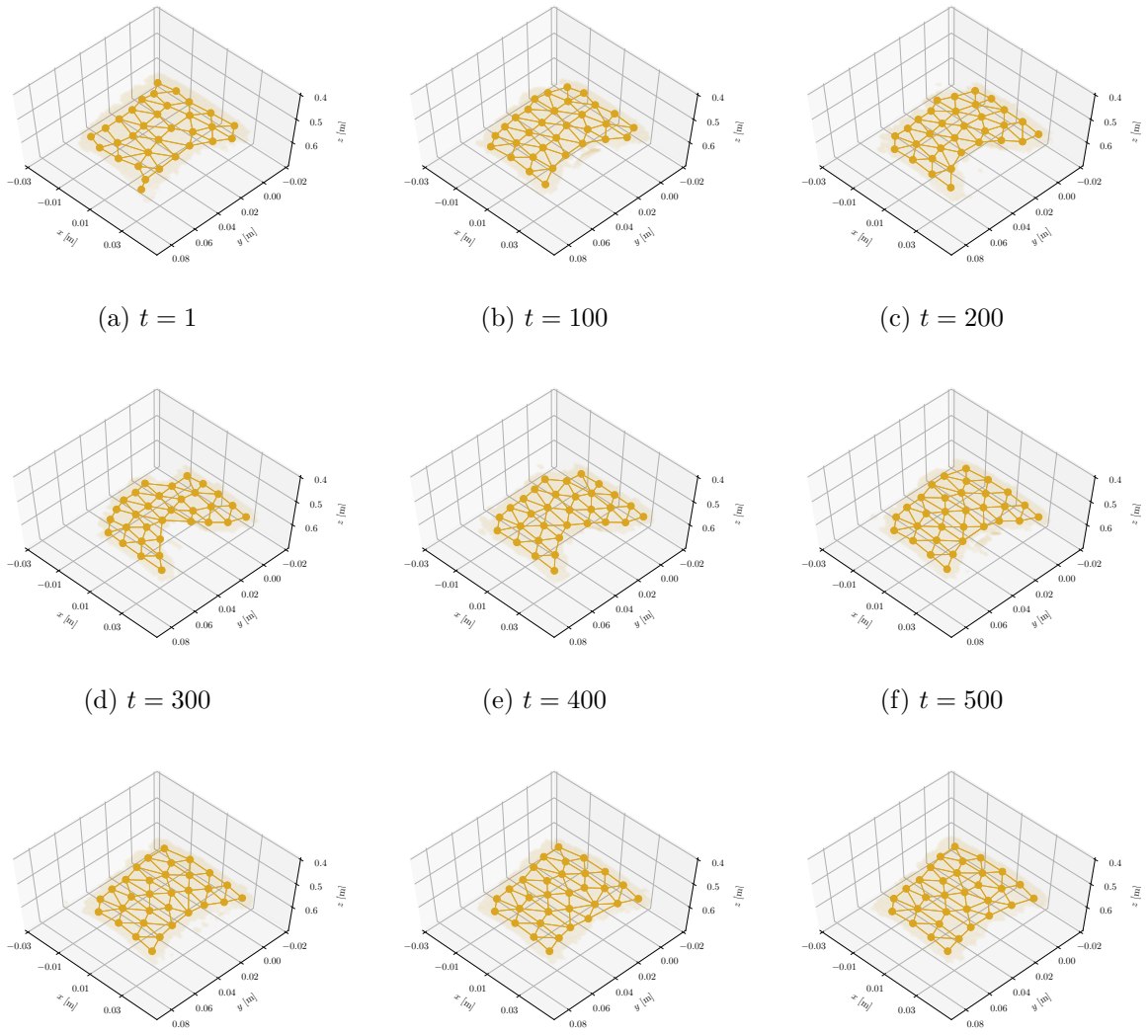


(a) $t = 1$ (b) $t = 100$ (c) $t = 200$

(d) $t = 300$ (e) $t = 400$ (f) $t = 500$

(g) $t = 600$ (h) $t = 700$ (i) $t = 800$

Figure A.2: Point cloud and graph sequences of the large sponge obtained by the BC-GNG model at different deformation levels.

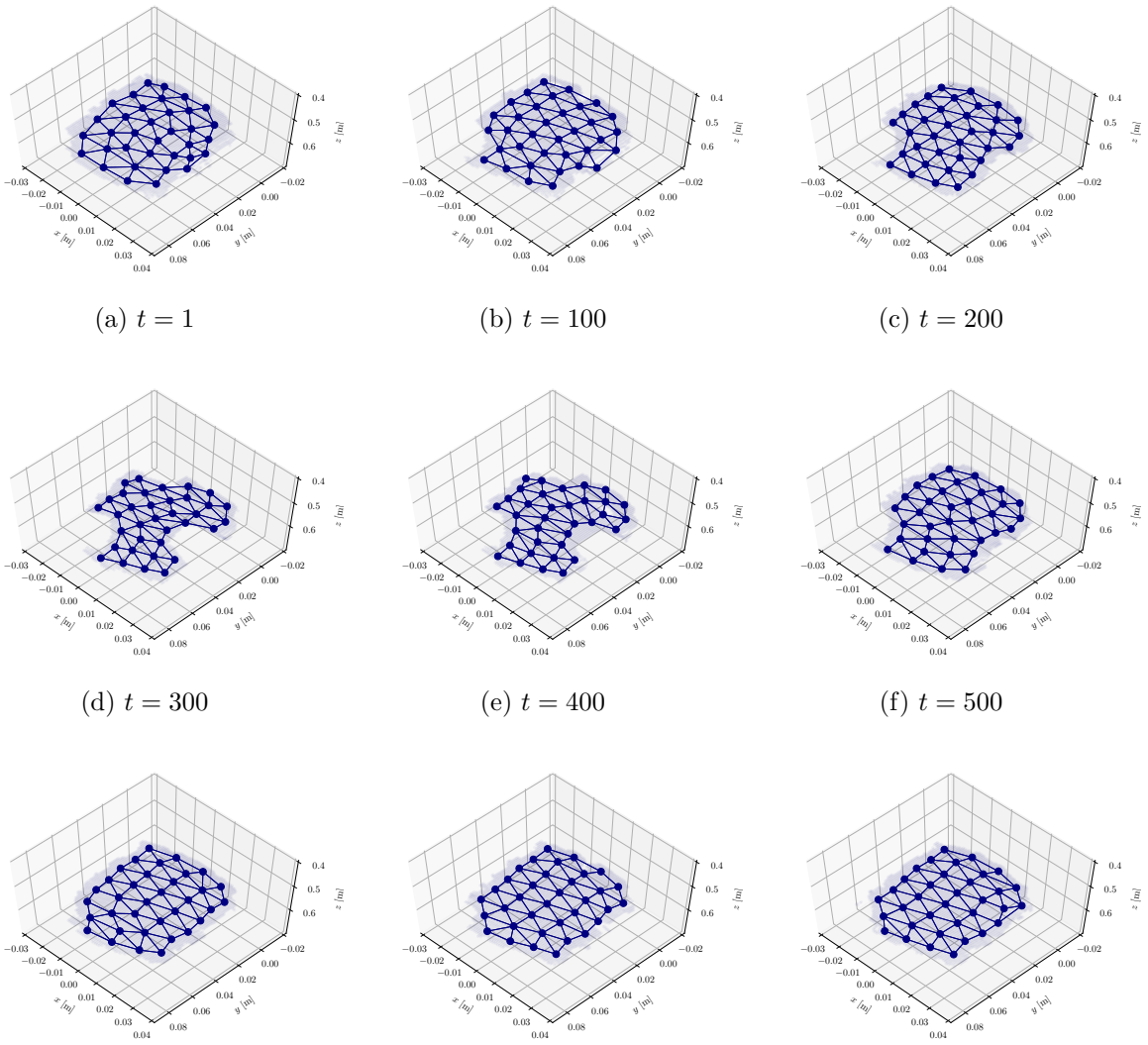


(a) $t = 1$ (b) $t = 100$ (c) $t = 200$

(d) $t = 300$ (e) $t = 400$ (f) $t = 500$

(g) $t = 600$ (h) $t = 700$ (i) $t = 800$

Figure A.3: Point cloud and graph sequences of the towel obtained by the BC-GNG model at different deformation levels.

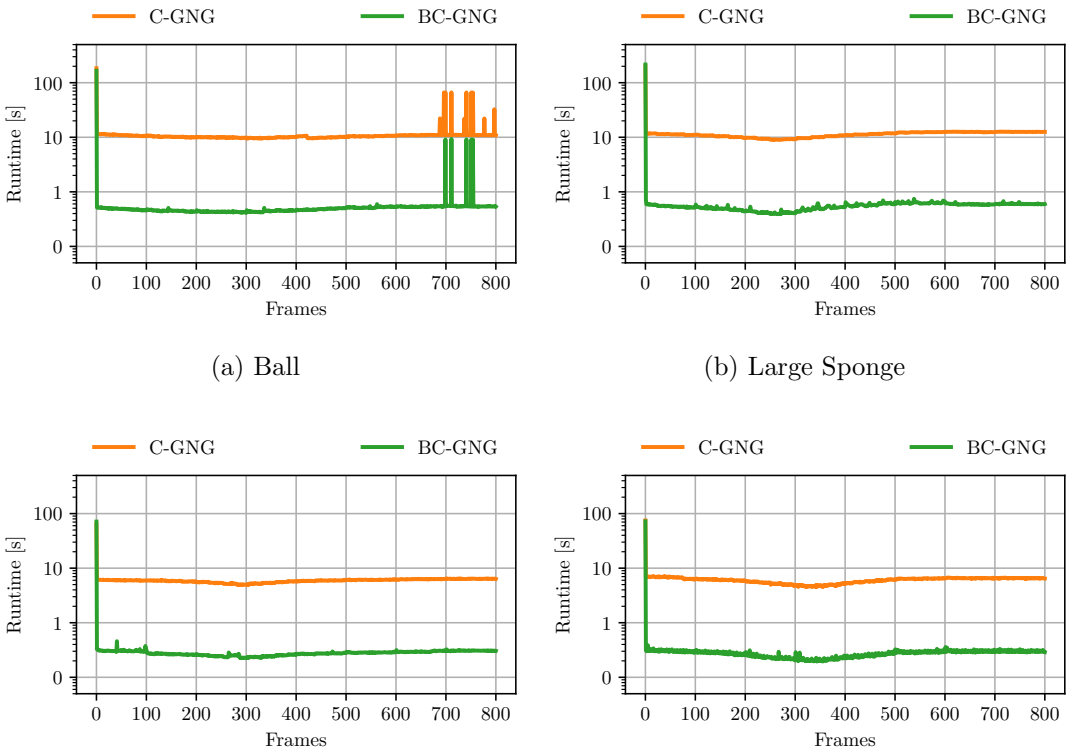


(a) $t = 1$ (b) $t = 100$ (c) $t = 200$

(d) $t = 300$ (e) $t = 400$ (f) $t = 500$

(g) $t = 600$ (h) $t = 700$ (i) $t = 800$

Figure A.4: Point cloud and graph sequences of the toy obtained by the BC-GNG model at different deformation levels.



(a) Ball

(b) Large Sponge

(c) Towel

(d) Toy

Figure A.5: Runtime sequences obtained by C-GNG, BC-GNG during the manipulation of the non-rigid objects for QE=0.005.

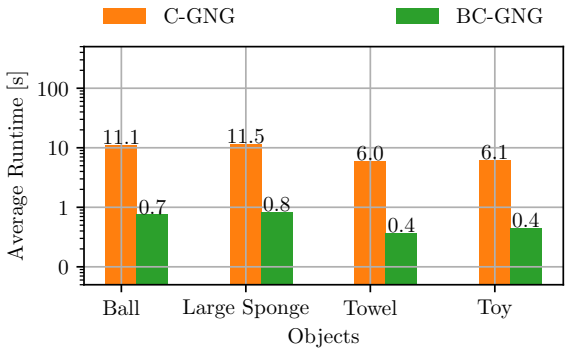
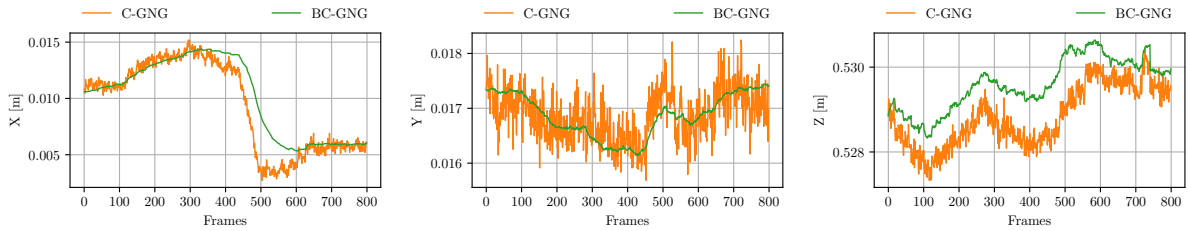
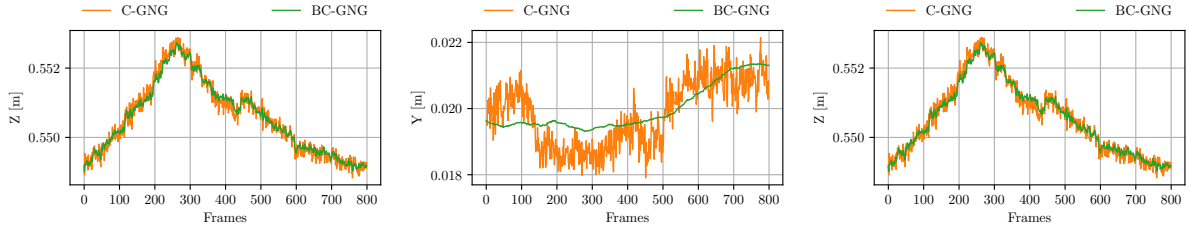


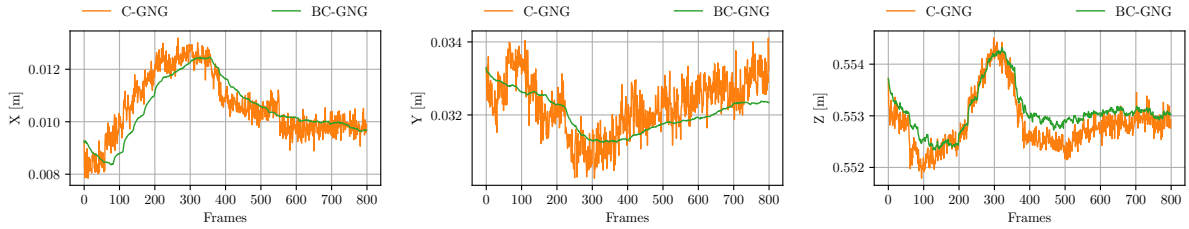
Figure A.6: Average of runtime sequences obtained by C-GNG, BC-GNG during the manipulation of the non-rigid objects for QE=0.005.



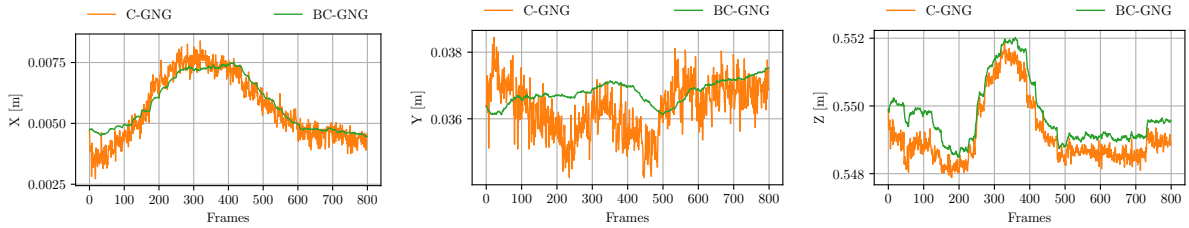
(a) Ball



(b) Large Sponge

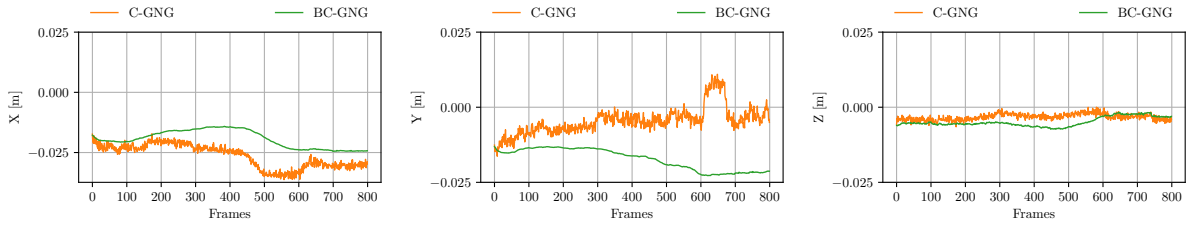


(c) Towel

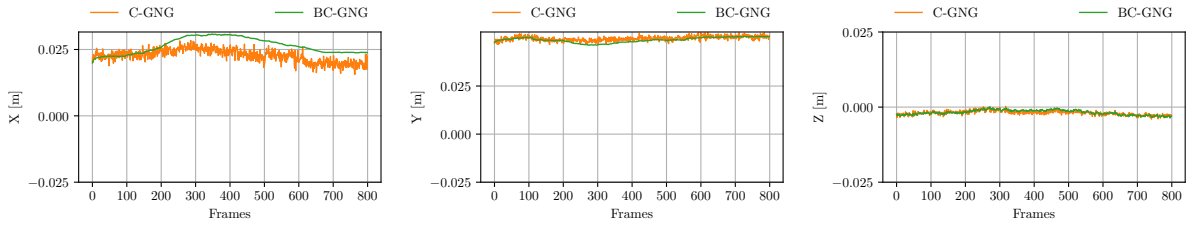


(d) Toy

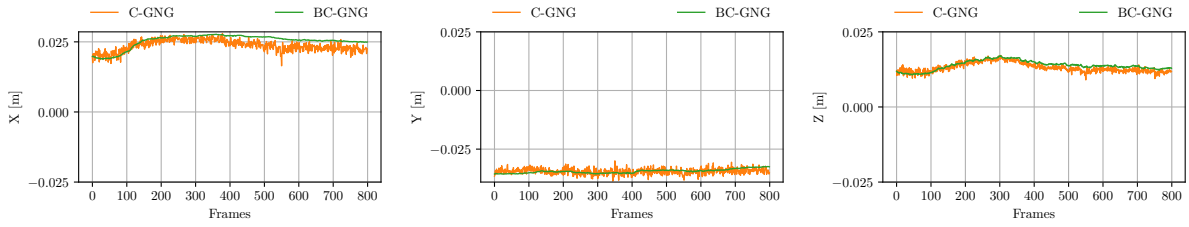
Figure A.7: Global displacement obtained by C-GNG, BC-GNG during the manipulation of the non-rigid objects.



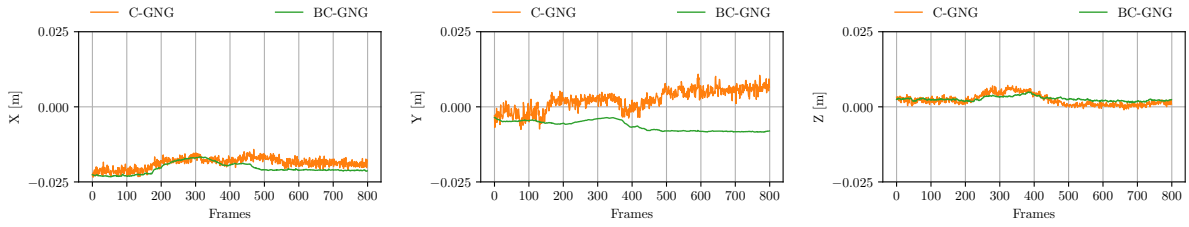
(a) Ball



(b) Large Sponge

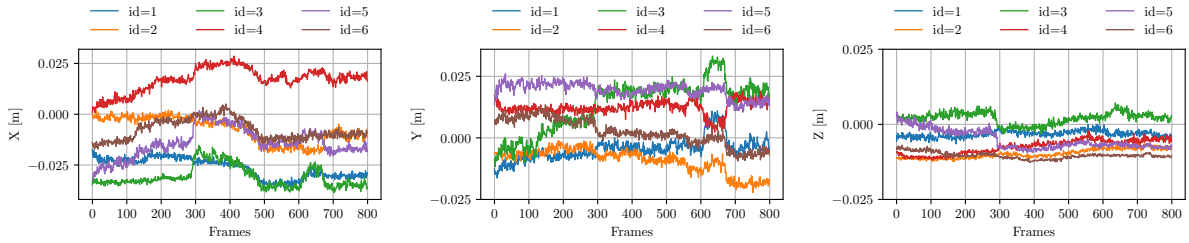


(c) Towel

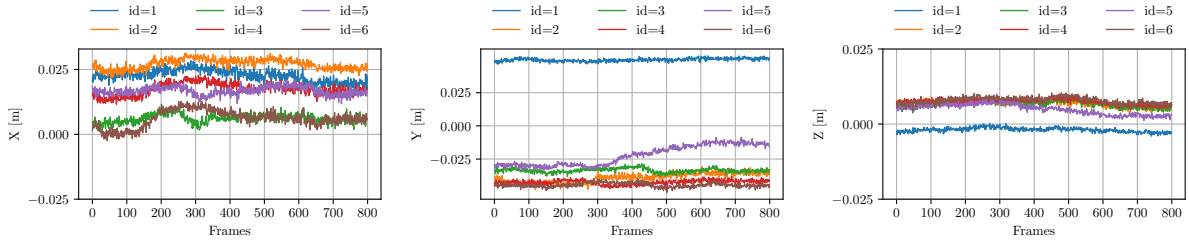


(d) Toy

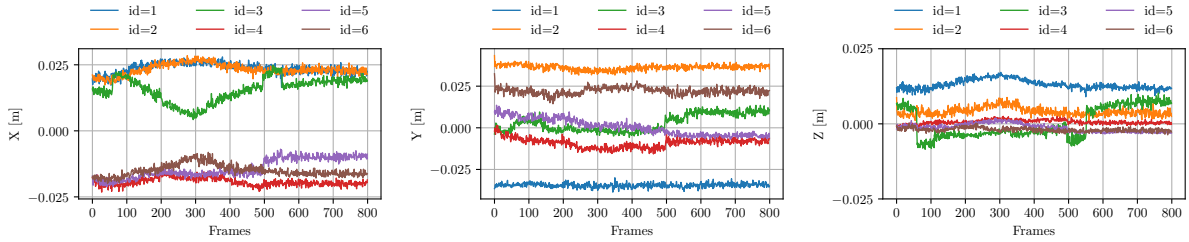
Figure A.8: Local displacement obtained by C-GNG, BC-GNG during the manipulation of the non-rigid objects.



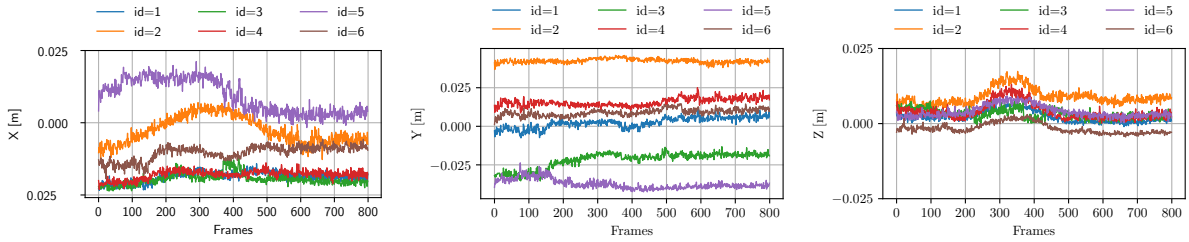
(a) Ball



(b) Large Sponge

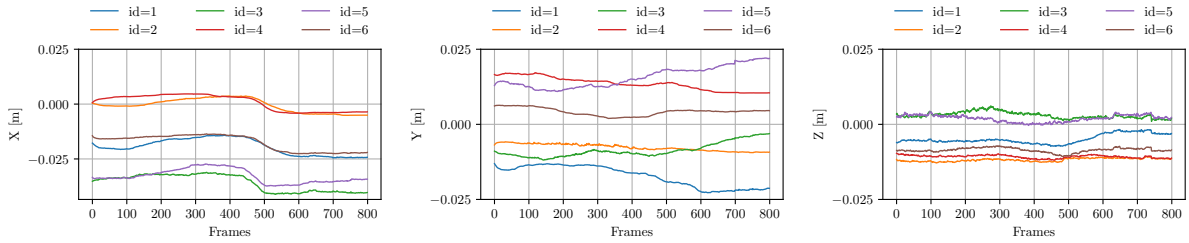


(c) Towel

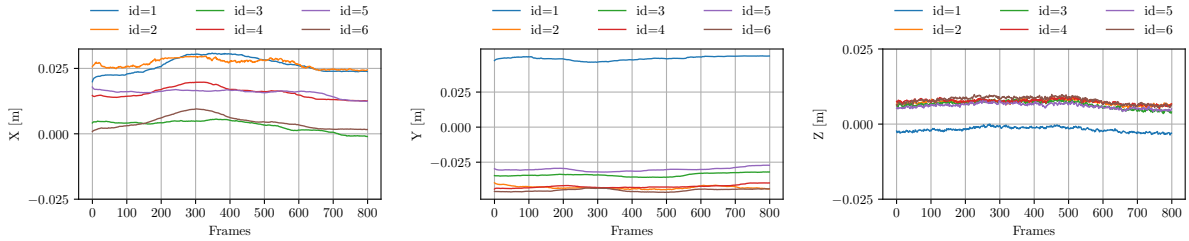


(d) Toy

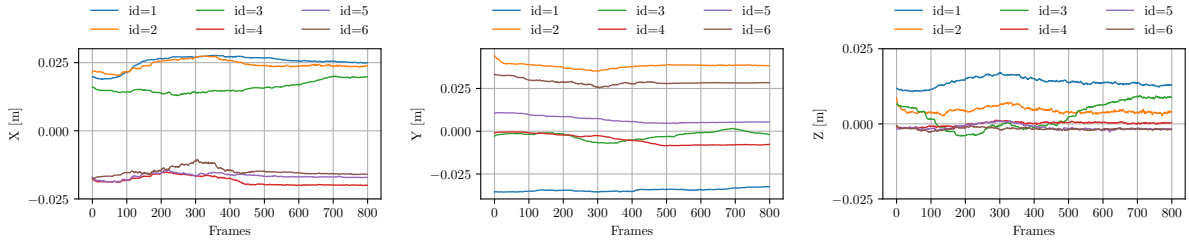
Figure A.9: Feature correspondence obtained by C-GNG during the manipulation of the non-rigid objects.



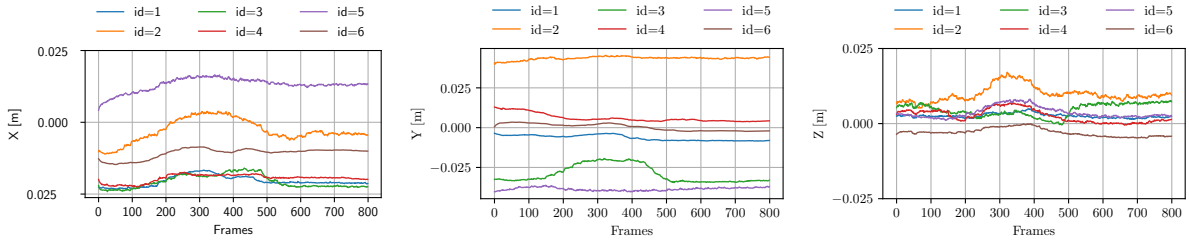
(a) Ball



(b) Large Sponge



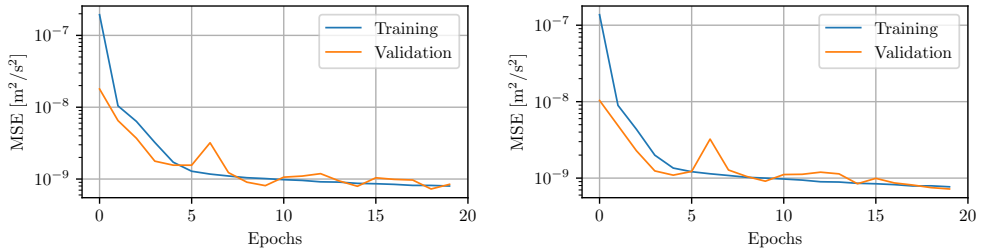
(c) Towel



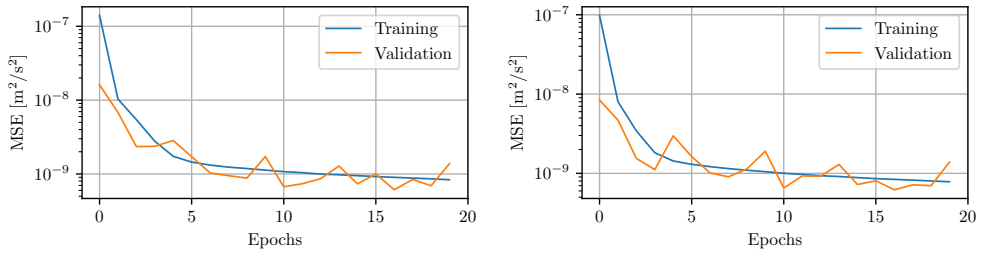
(d) Toy

Figure A.10: Feature correspondence obtained by BC-GNG during the manipulation of the non-rigid objects.

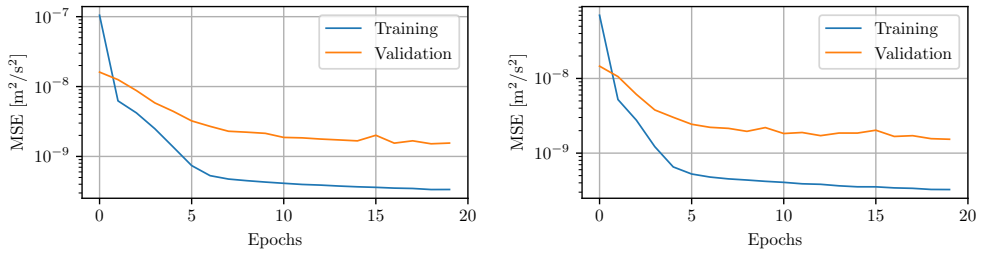
A.2 Deformation Dynamics Prediction



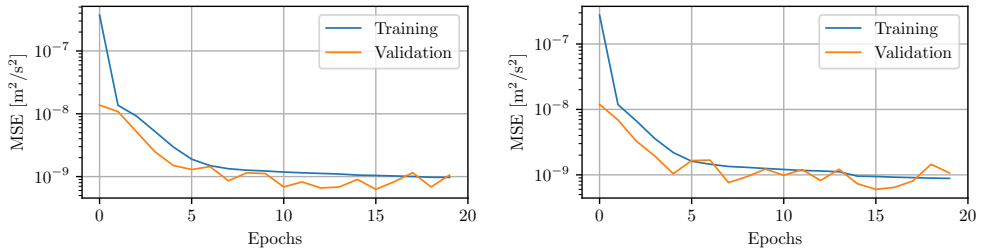
(a) Ball



(b) Large Sponge

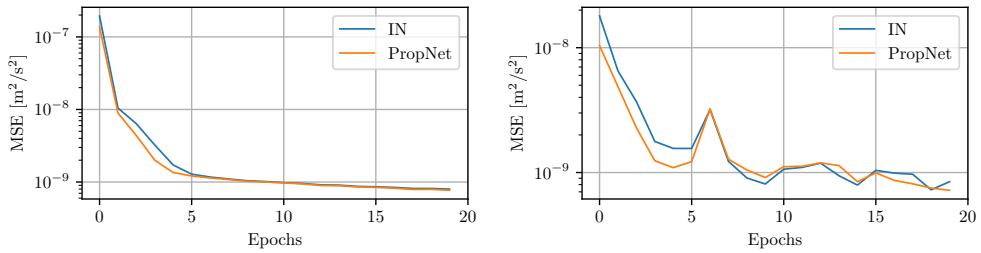


(c) Towel

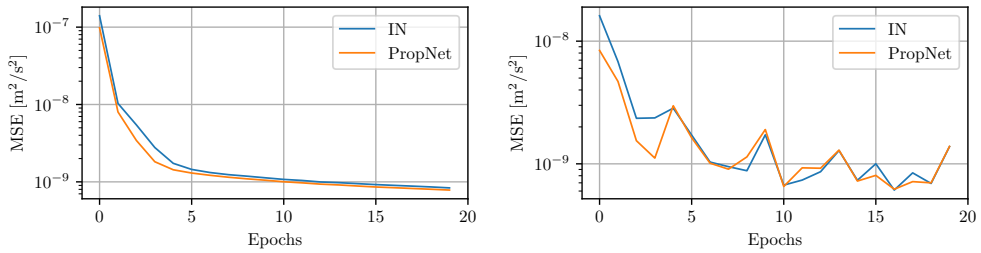


(d) Toy

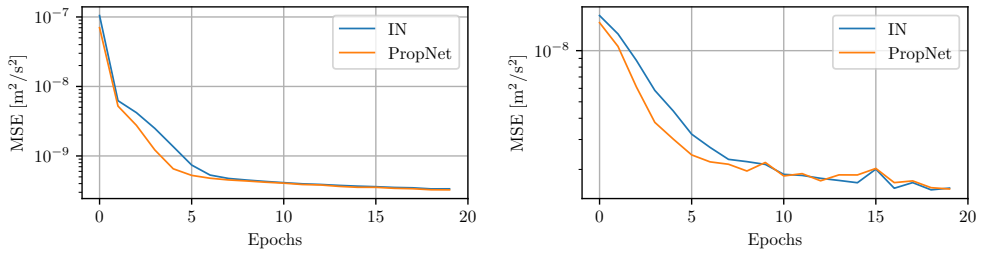
Figure A.11: Comparison of IN (left) and PropNet (right) models with respect to the training and validation errors of the predicted velocities obtained by using the augmented GNG-based representation of the non-rigid objects.



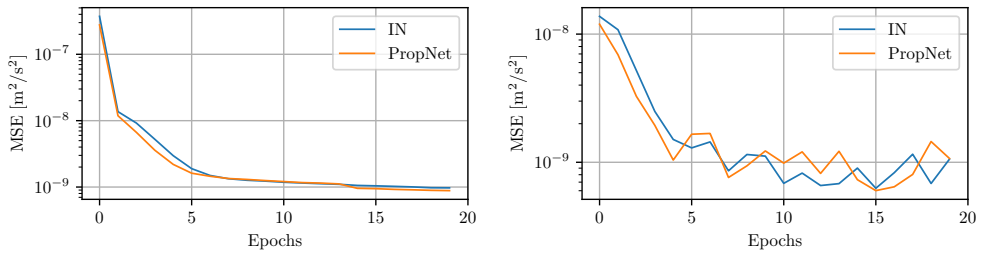
(a) Ball



(b) Large Sponge

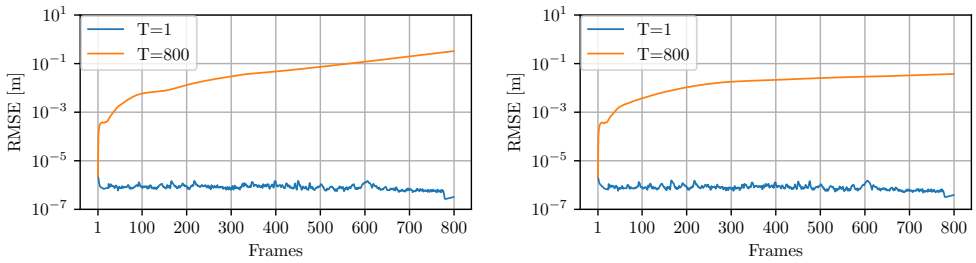


(c) Towel

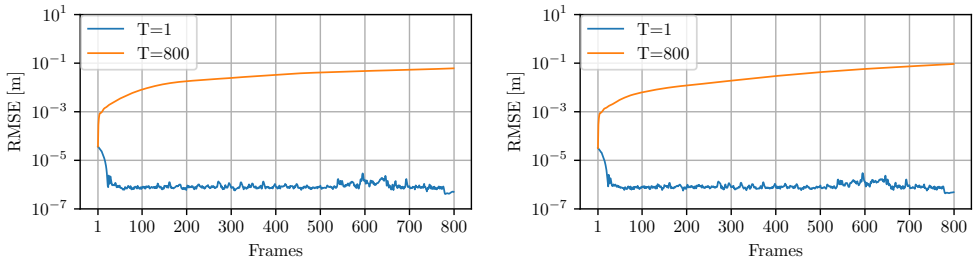


(d) Toy

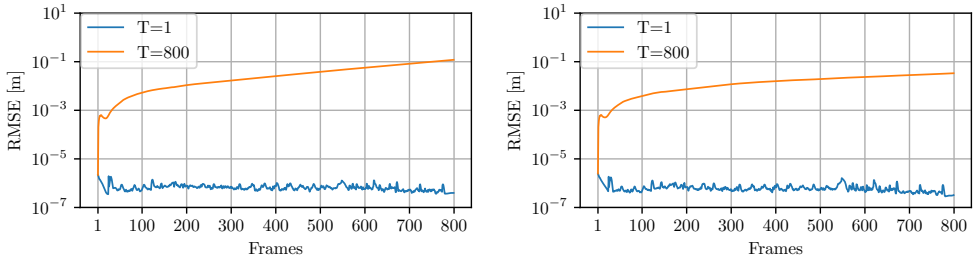
Figure A.12: Comparison of training (left) and validation (right) errors of the predicted velocities obtained by IN and PropNet models using the augmented GNG-based representation of the non-rigid objects.



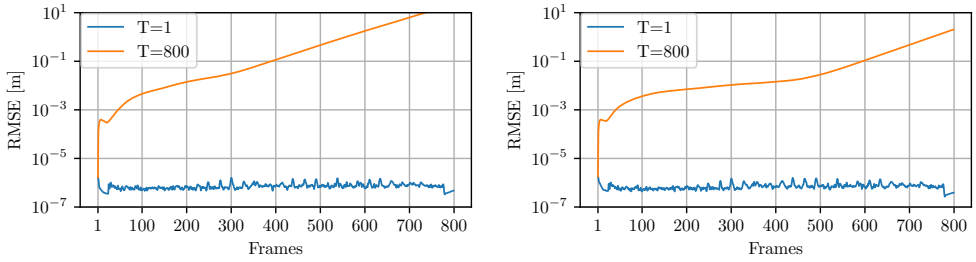
(a) Ball



(b) Large Sponge



(c) Towel



(d) Toy

Figure A.13: Comparison of IN (left) and PropNet (right) models with respect to the prediction of the nodes position obtained at different time steps using the augmented GNG-based representation of the non-rigid objects.