Visual Tracking of Deformation and Classification of Object Elasticity with Robotic Hand Probing

Fei Hui

Thesis submitted in partial fulfillment of the requirements for

Master of Applied Science

in

Electrical and Computer Engineering

University of Ottawa

July 2017

Abstract

Performing tasks with a robotic hand often requires a complete knowledge of the manipulated object, including its properties (shape, rigidity, surface texture) and its location in the environment, in order to ensure safe and efficient manipulation. While well-established procedures exist for the manipulation of rigid objects, as well as several approaches for the manipulation of linear or planar deformable objects such as ropes or fabric, research addressing the characterization of deformable objects occupying a volume remains relatively limited. The fundamental objectives of this research are to track the deformation of non-rigid objects under robotic hand manipulation using RGB-D data, and to automatically classify deformable objects as either rigid, elastic, plastic, or elasto-plastic, based on the material they are made of, and to support recognition of the category of such objects through a robotic probing process in order to enhance manipulation capabilities. The goal is not to attempt to formally model the material of the object, but rather employ a data-driven approach to make decisions based on the observed properties of the object, capture implicitly its deformation behavior, and support adaptive control of a robotic hand for other research in the future. The proposed approach advantageously combines color image and point cloud processing techniques, and proposes a novel combination of the fast level set method with a log-polar mapping of the visual data to robustly detect and track the contour of a deformable object in a RGB-D data stream. Dynamic time warping is employed to characterize the object properties independently from the varying length of the detected contour as the object deforms. The research results demonstrate that a recognition rate over all categories of material of up to 98.3% is achieved based on the detected contour. When integrated in the control loop of a robotic hand, it can contribute to ensure stable grasp, and safe manipulation capability that will preserve the physical integrity of the object.

Acknowledgements

I would like to express my gratitude to my parents, 惠国庆 and 毕秋香, for their full support and encouragement during all these years that I have spent on studying in Canada. I will be grateful forever for your love.

I would like to thank Dr. Pierre Payeur and Dr. Ana-Maria Cretu for their guidance, motivation and encouragement through my study and research.

I would like to thank Jonathan Guillotte-Blouin for helping with the interface of the Barrett robotic hand.

In addition, I would like to thank Thiago Eustaquio Alves de Oliveira for his patience and company. For me, you are truly a gift from God.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1 Introduction

1.1 Motivation

Enabling robots to recognize and manipulate objects is essential not only in the research realm but also for future applications of robots in practical everyday tasks. Most robots currently found in industry are designed and tuned to perform in well-structured environments, manipulating specific objects in a predefined sequence of motions. This approach largely limits robotic manipulation to objects for which prior information about their properties is available. More recently, robots started being used in a wider range of settings, including active packaging of food, handling and folding of fabrics and papers, assembling flexible automotive parts, manipulating ropes (e.g. knotting), palpating organs and tissues, and even performing sutures. Nowadays, researchers drive their attention towards the generalization of robotic skills in order to open doors to new applications and further improve the performance of robots in traditional tasks. More specifically, the recognition and manipulation of objects should encompass non-structured environments, different viewpoints, and lack of prior knowledge about objects physical properties, e.g. whether the objects are rigid or deformable.

The manipulation of rigid objects has been extensively studied, and its procedures are well established in the literature. On the other hand, the manipulation of deformable objects is still a challenge due to the perception of changing properties inherent to such objects. The manner in which a robotic system perceives an object is an important element in the development of autonomous robotic systems. Such systems typically employ cameras and force/torque sensors in the robot end-effector (robotic hand) to gather visual information and interaction parameters that arise during manipulation tasks. The information collected enables the efficient manipulation of objects, especially in model-based setups, as well as the estimation of the objects physical properties and behavior. To ensure stable and efficient manipulation of deformable objects, such as pieces of fabric, rope or other soft or flexible objects, it is important to detect, track and classify the behavior of deformations that occur due to the interaction between the robot end-effector and the object in question. While detecting an object's static features, such as dimension and shape, can easily be accomplished by a vision system, its elastic properties after being grasped, involve tracking the dynamic changes of its contour or shape as the manipulation occurs. This thesis presents the design, development and experimental validation of a research conducted to achieve model-free characterization of the deformation behavior of objects under robotic manipulation. Given a series of manipulation actions and visual observations of an object’s contour, the proposed methodology enables the automated probing and classification of deformable objects, based on the material they are made of, as elastic, plastic, elasto-plastic, or simply rigid objects. The information determined by this automated probing and classification process can be used by robotic hand controllers in different handling strategies.

The solution developed in this thesis attempts to minimize assumptions about the scene, the end-effector, as well as the input provided by users of the system. The long-term aim of this research is to

develop a solution that will enable a robotic system to probe an unknown deformable object and then to automatically adapt a robotic hand's behavior, including the grasping strategy and the magnitude of applied forces, according to the object's physical characteristics. When integrated within a control scheme for a robotic hand, the solution can help to establish a stable grasp and to achieve precise manipulation without a priori knowledge on the shape and material of the object. In other words, this technology will contribute towards enabling autonomous robots to handle and interact efficiently with unmodeled objects.

1.2 Research Objectives

In this context, the main goal of this research is to present the design and implementation of a vision sensing and classification system that enables the categorization of 3D objects composed of various types of material by closely observing their interaction with a robotic hand while it attempts to deform them. The system detects an object in a robotic hand, extracts and tracks its contours, and after a short palpation period classifies the object based on its deformation properties. The types of material targeted by this solution exhibit an elastic, plastic, elasto-plastic, or rigid behavior. To implement the system, the following objectives are established as part of this research effort:

• Identify and implement a set of algorithms that, in combination, are able to robustly detect and extract an object of interest from RGB-D data captured by a vision and depth sensor, and under operational conditions of manipulation with a robotic hand. These algorithms should require minimal input from a human operator and be robust to different camera viewpoints;

• Design and implement an algorithm to detect and reliably track an object's contour as it dynamically deforms under the effects of manipulation. The algorithm should be able to adjust to the features that are more suitable to distinguish between the robotic hand and the object of interest, while being robust to various shapes and magnitudes of deformation;

• Develop a decision system for classifying deformable objects as elastic, plastic, elasto-plastic, or rigid, solely from a series of deformation responses collected during a probing process. The decision system should be able to establish the correspondence between contours of different lengths, and to cope with small shifts and rotations of the object;

• Evaluate and analyze the overall performance of the proposed approach from the classification rates obtained over all material types, and demonstrate that the contour tracking and classification capability resulting from a robotic probing procedure is apt to support efficient dexterous manipulation of non-rigid objects.

The following assumptions are made in the thesis for the experimental setup:

• The sensors installed on the fingertips of the robotic hand that manipulates the objects are not used to measure the magnitudes of the forces applied to the objects in order to cause the deformation, and, as a result, force sensing and force feedback control are beyond the scope of this research. Visual

information is used instead for the classification of objects' elasticity given that deformations may occur on surfaces that are not in contact with the robotic hand;

• Contact points distribution to place the robotic fingers in order to achieve the grasping task in an optimal way is beyond the scope of this research;

• The objects of interest remain relatively static in the scene, and therefore object tracking is not the essence of this work;

• The proposed tracking and classification system needs be able to deal with a diversity of objects. In other words, the system needs to extract the deformed contours and classify the objects regardless of their color, texture, shape and size.

1.3 Thesis Organization

The remainder of this thesis is structured as follows: Chapter 2 reviews the research work related to the manipulation and characterization of deformable objects. A review of methodologies used in the field, such as image segmentation, fast level set method, random sample consensus, log-polar mapping, k-dimensional tree data structure, and dynamic time warping that will be exploited in this thesis, is presented. Chapter 3 details the novel approaches for contour detection and tracking of deformable objects, and for their classification from RGB-D data; it also covers the implementation details of the proposed approaches. Chapter 4 presents extensive experimental results and evaluates the achieved performance for contour detection and tracking, and for objects' material behavior-based classification. The key parameters are analyzed and the current limitations of the approach are discussed. Chapter 5 provides concluding remarks and offers directions for future research.

Chapter 2 Literature Review

This chapter presents an overview of relevant work in the literature on the topics of object material characterization and of robotic manipulation of deformable objects (section 2.1), as well as the techniques that enable it, starting from data acquisition and interpretation (section 2.2), segmentation and tracking issues (section 2.3), and contour matching for object classification (section 2.4). Finally, section 2.5 proposes a summary of the literature review.

2.1 Manipulation and Material Classification for Deformable Objects

Object manipulation is one of the fundamental capabilities of autonomous robotic systems, yet the design and development of such systems able to reliably and safely manipulate objects, in particular deformable objects, without human intervention is still a challenging area of research. In order to achieve efficient and safe manipulation, a complete knowledge of the manipulated object is often required. In contrast to the manipulation of rigid objects which is extensively studied in the literature, and for which well-established procedures exist, the investigation of the manipulation of deformable objects represents a more recent undertaking. The most relevant related work on the topic analyzes the displacement of the object surface at and in the surroundings of the contact points where external forces are applied.

There are two main ways encountered in the literature to represent the manipulated object: (i) the model-based approach [1] that generally involves the approximate identification of elastic parameters of the object, with tuning of the parameters achieved by comparing the real and simulated object submitted to interaction, and aiming to minimize the differences between the two; and (ii) the model-free approach that does not assume a prior model or template for the object. In the category of model-based approaches, the model can be estimated using a mass-spring-damper model [2][3], a finite-element method (FEM) representation [3]–[5], or a surfel representation [6][7]. In [2], Choi et al. track the trajectory of a dropping deformable ball, which is red against a blue background, in a video stream. The elasticity parameters of a mass-spring model simulating the ball are adjusted by optimizing the differences between a virtual object and the corresponding real one. This work is restricted to elastic objects and tuning is achieved thanks to a simplified imaging scenario over a contrasting background. Without capturing visual data, Zaidi et al. [3] model a linear isotropic 3D deformable object in interaction with a three-fingered robotic hand as a mass-spring system based on a tetrahedral mesh. The estimation of contact points and object deformations is based on tracking the node positions and solving the dynamic equations of Newton's second law. Elbrechter et al. [8] model the bending of a sheet of paper under manipulation by a robotic hand. The paper geometry is represented by a 2D grid of nodes that are connected by links specifying the bending constraints, namely a resting distance between two nodes and the stiffness coefficient that are tuned manually. Marker-aided object detection and tracking is employed in order to track the folding process of the paper. In the same line of research, Mateo et al. [9] analyze deformations of 3D elastic object surfaces. The surface deformation is described

in terms of a level curves set by computing the gradients of surface points from the object geometry. The extension of this work [10] explains that the grasping process of flexible objects and the control of robot manipulation are reliable under visual surveillance to detect the deformation. No formal model is required for the object deformation or for its material. However, this work does not allow the differentiation between various categories of deformable objects.

The work of Petit et al. [4] considers RGB-D data for tracking the deformation of 3D elastic objects. The approach assumes that the object of interest is already segmented from its background, and the iterative closest point (ICP) algorithm is applied on the resulting point cloud to estimate a rigid transformation from the point cloud to a linear tetrahedral FEM model representing the object. Linear elastic forces exerted on vertices are computed from the point cloud to the mesh based on closest point correspondence and the mechanical equations are solved numerically to simulate the deformed mesh. This work is also restricted to elastic objects. Krainin et al. [6] propose a solution for in-hand modeling of 3D rigid objects using RGB-D data. A Kalman filter based on depth and visual information produces at each frame an estimate of the robot manipulator, the object and the Kinect sensor. The Kalman filter tracking uses RANSAC feature matching and an ICP algorithm variant. These estimates enable the segmentation of the object, and its model is built as a series of surfels. In [7], a registration method is proposed based on multi-resolution surfel maps providing a dense displacement field between object shapes monitored in RGB-D images. Kraft et al. [11] obtain object models that are composed of sparse sets of oriented 3D points along their contours by monitoring the manipulation process of the object with a stereo camera and then predicting the representation of object based on the motion induced by the robot. However, this approach requires prior knowledge about the experimental environment and the object itself. Schulman et al. [5] track deformable objects from a sequence of point clouds by identifying the correspondence between the point cloud and a physics-based model of the object composed of a collection of linked rigid particles, governed by dynamical equations. An expectation-minimization algorithm aims at finding the most probable node positions for the model given the measurements. Tests are performed in a controlled environment, against a green background that limits its applicability to normal conditions. An approach that does not assume a prior model or template for the object is proposed by Hur et al. [12]. They introduce a 3D deformable spatial pyramid model to find the dense 3D motion flow of deformable objects in RGB-D data. The point cloud is corrected with a depth hole-filling algorithm and treated with a Gaussian filtering prior to the computation of a series of perspectively normalized descriptors. The 3D deformable spatial pyramid finds dense correspondences between instances of a deformed object by optimizing an objective function, in form of an energy corresponding to a Markov random field, and taking into consideration factors such as: translation, rotation, warping costs and descriptors matching costs.

A model-free approach for robotic manipulation of 3D deformable objects is proposed by Navarro-Alarcon et al. [13]. The soft object is manipulated by two robots within a feedback loop, which allows to control simultaneously the object shape and final position (i.e. interest points over the object and its centroid). The shape of the deformed object is represented by the compression distance between two feature points, the folding angle estimated by three feature points on the surface of the object, and the normalized curvature measured passing through three feature points. The solution has however limited

applicability to only elastic deformations and objects with artificial control features added to it (positions of tags).

In this context, the main goal of this thesis is to design and implement a framework that enables model-free material characterization of 3D soft deformable objects exhibiting various material behaviors (mainly on elasticity) by visually observing their interaction with a robotic hand.

2.2 Data Acquisition

Visual object data acquisition can be achieved using a large variety of devices including digital cameras and RGB-D cameras. Over the past decade, digital cameras were widely and successfully used for image processing. Their extended use is supported by the existence of various software libraries, including OpenCV [14]. However, a single 2D image cannot always provide enough information about the scene, especially in the context of robot manipulation of objects, where information on the depth enables a better interaction with the environment and the manipulated object. The use of multiple digital cameras can provide more data about the scene, but stitching together data coming from multiple views is not trivial and is a time-consuming operation, not suitable for tracking a deformable object. The Microsoft Kinect for Xbox 360 sensor [15] overcomes some of the problems that are caused by the use of multiple digital cameras. Moreover, the Kinect sensor costs less than $200, which makes it an affordable solution for color and depth data acquisition.

From the objectives of the research work presented in this thesis and from the above considerations, a sensor that is able to capture both color and depth images at a relatively high frame-rate offers an interesting alternative for the tracking of a deformable object manipulated by a robotic hand. The depth image can be employed to locate and identify the object of interest in the scene (spatial perception of objects), while the color image can be used to extract the precise contour of the object (contour detection and tracking) during the manipulation of the robotic hand, while being fast enough to follow the deformation. The Microsoft Kinect for Xbox 360 is therefore selected for the work in this thesis, and is used in conjunction with the Robot Operating System (ROS) which contains drivers for the Kinect sensor [16].

2.3 Segmentation and Tracking

As the goal of this research is to classify deformable objects based on their deformation properties, the object of interest must be first identified in a given scene. Image segmentation is a fundamental procedure for dividing an image into parts that share a strong correlation [17]. Segmentation can be defined as the process of partitioning an image into constituent parts according to local image properties or features. The general approach is to allocate each pixel labelled by its characteristic properties to regions sharing similar properties. These properties depend on the image content and can include color, texture, motion, brightness, reflectivity, or depth. The result of image segmentation is a set of regions, homogeneous with respect to a chosen property, and that collectively cover the entire image, or a set of contours extracted from the image, representing the boundaries of these regions. In the present context, segmentation is used to extract objects of interest contained in the scene.

2.3.1 Segmentation Techniques

Image segmentation remains a difficult task, due to both the tremendous variability of object shapes and to the variations in image contents and quality. This justifies the interest of researchers in this topic over the years. The literature reports a great variety of segmentation methods over the past decades. According to Sonka et al. [17], these can be subdivided according to the dominant features they employ. The main classes of segmentation methods are: threshold-based, edge-based, region-based, and other advanced methods.

2.3.1.1 Threshold-Based Segmentation Methods

Threshold-based segmentation methods capitalize on knowledge of the image content in form of histograms of image features. In its simplest form, the algorithm uses a brightness constant (threshold) in a gray-scale image to segment an object from its background. The thresholding algorithm is fast and computationally inexpensive [17], but the threshold selection has a crucial impact on the segmentation results. Several methods have been proposed to adjust the value of this threshold (or of multiple thresholds) within an image in order to improve the foreground-background separation, but the results vary widely according to the content of the image.

2.3.1.2 Edge-Based Segmentation Methods

Edge-based segmentation methods are a group of methods that use information about the edges contained in an image. Edges delimitating discontinuities in gray-level, color, or texture are identified based on edge detector operators (e.g. Roberts, Sobel, Laplace, Prewitt, etc) and are combined in chains that better correspond to the boundaries of objects within the image. Prior information can be incorporated in these methods to improve the segmentation results. Edge-based methods are affected by image noise and image artifacts (i.e. unsuitable information) that lead to the presence of edges while no real border exists in the image, or to the lack of an edge where a border exists.

2.3.1.3 Region-Based Segmentation Methods

Unlike edge-based solutions that identify borders between regions, region-based methods construct the regions directly [17]. *Region growing* segmentation is an iterative region-based image segmentation method which aims at dividing an image into zones of maximum homogeneity based on properties such as gray-level, color, texture, shape, etc. Simpler versions of region growing include: region merging, region splitting, and region splitting and merging [18]–[20]. The idea behind region merging is to start with an initial segmentation of an image into small homogeneous regions (e.g. blocks of 4 by 4, or 8 by 8) around some seed pixels chosen over the image, and then merge adjacent regions that satisfy a merging criterion (e.g. pixels that do not differ by more than a threshold from the seed pixel representing the center of the region). The properties of a region are compared to those of adjacent regions, and if they match, the corresponding regions are merged into a larger region whose properties are recomputed. Otherwise, the region is classified as non-matching and the process is repeated. The selection of initial seed points plays a major role in the results of the segmentation, and several methods

have been proposed to choose these seeds, including uniform grids or user selected seeds. This selection process can also be based on prior information of the image content. *Region splitting* starts with a whole image representing a non-homogeneous region and splits it in regions based on a criterion similar to the one employed by region merging. Even if the same homogeneity criterion is used, the image segmented will not be the same for the two methods, from where stems the interest of creating *splitting and merging methods* that fuse the advantages of the previous two methods. Splitting and merging algorithms use typically a pyramid image representation.

Region growing techniques are simple, flexible and intuitive to use. They work generally better on noisy images [17], where edges are very difficult to extract. However, their main drawback is the unwanted spread or connection between pixels. In other words, two pixels may be connected due to the similarity of color, while in fact they belong to two regions in the image. Thus, a predefined threshold function is used before applying the region growing approach in practice.

The *watershed algorithm* overcomes the issue of unwanted spread or connection. The idea for the watershed algorithm [21] comes from geography: water following the topographic surface streams down to the lowest point, which is called a catchment basin. As the water continues to flow, several localized basins (local minima) eventually merge and create larger basins leaving only the highest points (maxima) or watershed lines unsubmerged. Watershed lines are the limit where the water rises in distinct catchment basins about to merge. Using this similarity and considering the gradient magnitude of an image as a topographic surface, the watershed ridge lines and the catchment basins can be extracted as being the region boundaries and the regions respectively, in the digital image, computed based on classic mathematical morphology operations. The watershed algorithm was expanded based on an immersion process by Vincent and Soille [22], with an improvement in speed and accuracy. Various other improvements have been developed. In particular, Meyer proposed the concepts of marker [23] and topographic distance [24], making the algorithm robust and flexible for segmenting objects.

Overall, the watershed algorithm demonstrates a high capability for segmenting complete objects or regions of interest because it considers edges and gradient changes in the image, unlike other thresholding algorithms that are only concerned with individual pixels, and it works well on hierarchical segmentation. Researchers are still improving the drawbacks, such as over segmentation due to its noise and local minima sensitivity.

2.3.1.4 Advanced Segmentation Methods

All the above-mentioned algorithms encounter difficulties in segmenting objects in higher dimensional spaces. A newer and more powerful generation of segmentation algorithms has been proposed to cope with three-dimensional and higher-dimensional space segmentation, including: mean-shift segmentation, fuzzy connectivity, graph cut and graph search, just to mention a few [17].

Of particular interest to the work in this thesis are *deformable contours or surface models*, because the primary objective is to track the contour of an object submitted to external forces in order to

characterize its material. There are two main groups of deformable models: parametric and geometric models. Parametric deformable models represent the borders in a parametric form. A classical algorithm belonging to this category is active contour models, or snakes. An active contour model is an energy-minimizing spline, where the energy depends on the shape and the location within an image of the desired contour. The local minima of this energy correspond to the desired location of the contour. Unlike most other contour models, snakes exhibit dynamic behavior due to their energy minimization properties. This representation allows direct interaction with the model and can lead to a compact representation for fast real-time implementation [25]. However, an approximate shape and location of the desired contour must be provided to the algorithm and their choice affects the performance. Moreover, snakes tend to be attracted by spurious edges; they sometimes degenerate the shape by shrinking and flattening; the convergence and stability of the contour deformation by minimization may be unpredictable, and they may yield intersecting boundaries in some situations [17]. Finally, they have difficulty in dealing with splitting or merging model parts, a useful property for extracting multiple objects or objects with unknown topology [26]. This difficulty is caused by the fact that a new parameterization must be constructed whenever topology change occurs.

On the other hand, geometric deformable models, such as level sets, overcome partially the problems associated with parametric models by representing surfaces using partial differential equations. The main difference with respect to parametric models is that the curves are evolved using only geometric computations, independent of any parameterization, in an implicit manner. Given their relative importance in the present research, geometric contour models will be further detailed in the following section.

2.3.2 Level Sets for Segmentation and Contour Tracking

As introduced in Section 2.3.1.4, the level set method is able to detect and track the contour of an object with topological changes. In this section, the method and its extensions, especially the fast level set method, are reviewed.

2.3.2.1 General Principles

The level set method, introduced by Osher and Sethian [27], is a computational technique based on topological changes, represented by an implicit formulation of the interface between a region of interest and the background. In [27], the authors propose the idea of Propagation of Surfaces under Curvature, a series of numerical algorithms to follow fronts propagating with curvature-dependent speed. The proposed algorithms approximate the equations of motion, resembling Hamilton-Jacobi equations. They can accurately capture sharp gradients and cusps in the moving fronts, can work in any number of dimensions, and can handle successfully topological merging and breaking in the followed surface. In the level set method, a curve *C* is represented as the zero level set of a higher dimensional implicit function $\phi(x, y, t)$ at any time t. The set of points enclosed by the curve represents the object of interest, as this curve denotes the contour of the object. Level set deformable models partially overcome the problems associated with parametric models, discussed in the previous section, by representing contours using

partial differential equations. The evolution of the curve is achieved by solving the partial differential equation (PDE) of the function $\phi(x, y, t)$ as follows [27]:

$$\frac{\partial \phi}{\partial t} = F|\nabla \phi|, \phi(x, y, 0) = \phi_0(x, y) \quad (2.1)$$

where F is the speed function in the normal direction of the curve, $\phi_0(x, y) = 0$ represents the initial level set function, and $\nabla \phi$ represents the normal vector at any point of the curve.

Based on the theory of [27], several variants and extensions [28]–[33] were proposed over the years. The review found in [34] summarizes all these developed approaches of the level set method, and indicates the remarkable results achieved with the model of Chan and Vese [31] in computer vision applications.

Chan and Vese [31] make an assumption that a curve C splits the entire image I into two regions, Ω_{in} and Ω_{out}, representing the inside and outside regions of the curve C, respectively. A fitting term [31] is considered as

$$E_1(C) + E_2(C) = \int_{\Omega_{in}} |I(x, y) - c_1|^2 dxdy + \int_{\Omega_{out}} |I(x, y) - c_2|^2 dxdy \quad (2.2)$$

where c_1 and c_2 represent the average intensities of the region Ω_{in} and Ω_{out}, respectively; and $I(x, y)$ represents the intensity at a pixel (x, y). As a conclusion of Chan and Vese [31], the object contour is identified when the fitting term is with a minimum value. This identified contour is taken to be the zero level set at any time t, $\phi(x, y, t) = 0$.

The energy function of the Chan-Vese model is composed by the fitting term (Equation 2.2) and additional regularizing terms. In other words, the Chan-Vese model is an energy-based segmentation. The energy function of the Chan-Vese model is [31]:

$$E = \mu|\partial\Omega_{in}| + \nu \int_{\Omega_{in}} dx + \lambda_1 \int_{\Omega_{in}} (I(x, y) - c_1)^2 dxdy + \lambda_2 \int_{\Omega_{out}} (I(x, y) - c_2)^2 dxdy \quad (2.3)$$

The term $|\partial\Omega_{in}|$ represents the length of the curve, and it contributes the smoothness of the curve. The second term is the area of the object region, and it is usually neglected by setting ν to zero. The last two terms represent the discrepancy between the current object of average intensity, c_1, and the background of average intensity, c_2, respectively. By minimizing the sum of these two terms, the moving curve obtained represents the contour of the object [35]. A minimization is performed on the energy function (Equation 2.3), which involves partial derivatives of the energy function in order to identify the contour of the object.

However, computing partial derivatives of the energy function (Equation 2.3) can be time consuming, from where stems the interest in simplifying the problem of curve evolution into solutions, such as the fast level set implementation described in the next section, which aims at increasing the computation speed.

2.3.2.2 Fast Level Set

Shi and Karl [36][37] proposed a fast implementation of the level set approach, which simplifies the problem of curve evolution by employing the idea of switching elements between two linked lists representing an interior contour and an exterior contour, in order to control the splitting or merging of regions in the image. Their solution handles multiple objects and topological changes, while achieving near real-time performance. Because the solution that is proposed in this thesis for contour tracking draws inspiration from the work of Shi and Karl, a detailed account of their solution is provided below.

Two neighboring lists, denoted by L_{in} and L_{out}, each representing the inside and outside neighboring pixels of the curve (Figure 2.1a) , respectively, are defined as follows [36]:

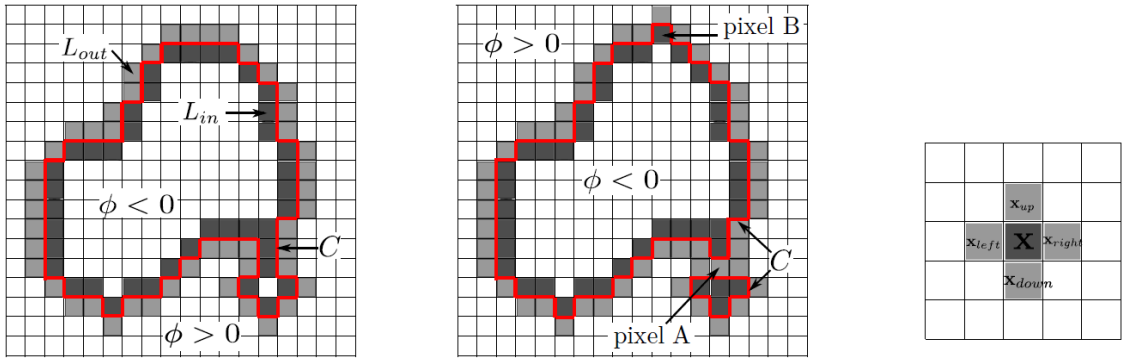
$$L_{out} = \{\mathbf{x} | \phi(\mathbf{x}) > 0 \text{ and } \exists \mathbf{y} \in N_4(\mathbf{x}) \text{ such that } \phi(\mathbf{y}) < 0\} \tag{2.4a}$$

$$L_{in} = \{\mathbf{x} | \phi(\mathbf{x}) < 0 \text{ and } \exists \mathbf{y} \in N_4(\mathbf{x}) \text{ such that } \phi(\mathbf{y}) > 0\} \tag{2.4b}$$

where $\mathbf{x}$ and $\mathbf{y}$ represent individual pixels.

In Figure 2.1a, the red line represents the initial curve, C, which splits the image into two parts: the object of interest inside the curve, and the background outside the curve. The list, L_{in}, contains the pixels located inside of the curve and in contact with the curve, C, shown in dark gray, while the list, L_{out}, contains the pixels located on the outside of the curve and in contact with the curve, C, shown in light gray. $N_4(\mathbf{x})$ represents a 4-connected discrete neighborhood of a pixel $\mathbf{x}$, $\mathbf{x} = (x, y)$ in a two-dimensional image, as illustrated in Figure 2.1c. ϕ is the level set function, and the sign of ϕ is used to distinguish the object from the background. In particular, ϕ is negative for pixels belonging to the object of interest and positive for pixels belonging to the background. It is defined as follows:

$$\phi(\mathbf{x}) = \begin{cases} 3, & \text{if } \mathbf{x} \text{ is an exterior pixel;} \\ 1, & \text{if } \mathbf{x} \text{ is in } L_{out}; \\ -1, & \text{if } \mathbf{x} \text{ is in } L_{in}; \\ -3, & \text{if } \mathbf{x} \text{ is an interior pixel.} \end{cases} \tag{2.5}$$



(a) (b) (c)

Figure 2.1: (a) Representation of the curve C and the two lists of neighboring pixels L_{in} and L_{out}; (b) the motion of switching pixels of L_{in} and L_{out}; and (c) representation of the 4-connected discrete neighborhood of a pixel $\mathbf{x}$. (Each cell denotes a pixel of the image.)

To illustrate the movement of switching pixels from L_{in} to L_{out}, and vice versa, during the tracking of the curve as the topology changes, Figure 2.1b shows an example in which the curve C moves outwards

relative to the object at pixel B and inwards at pixel A. This behavior is represented as a switch of pixel B from L_{out} to L_{in}, and a switch of pixel A from L_{in} to L_{out}. The functions for switching pixels from one list to the other are described as follows [36]:

---

switch_in(**x**):

1: Remove **x** from L_{out} and add it to L_{in}. Set ϕ (**x**) = −1.

2: For $\forall$ **y** $\in N_4$(**x**) with ϕ (**y**) = 3, add **y** to L_{out}. Set ϕ (**y**) = 1.

The function *switch_in*(**x**) is employed for a curve moving outwards at a pixel **x** $\in L_{out}$. Specifically, it switches the pixel **x** from L_{out} to L_{in} and adds all its neighboring exterior pixels to L_{out}, if they are not already part of that list. In a similar manner, *switch_out*(**x**) is employed for a curve moving inwards at a pixel **x** $\in L_{in}$.

---

switch_out(**x**):

1: Remove **x** from L_{in} and add it to L_{out}. Set ϕ (**x**) = 1.

2: For $\forall$ **y** $\in N_4$(**x**) with ϕ (**y**) = −3, add **y** to L_{in}. Set ϕ (**y**) = −1.

The curve C evolves according to the speed function F (Equation 2.1) [29] [30], which is separated into two parts: the data dependent speed function, F_{ext}, and the curve smoothness regularization speed function, F_{in} [35].

According to the theory of Shi and Karl, its extension [35] derives the speed function F_{in} from the Chan-Vese model [31]. Thida et al. [35] propose a real-time region-based contour tracking by using an adaptive speed function with the adaptive weighting parameters and demonstrate its potential for tracking non-rigid objects and weak boundary objects.

In the Chan-Vese model (Equation 2.3), the parameters c_1 and c_2 are the intensities of the regions inside and outside the curve C, respectively, with respect to the level set function ϕ, given by [35]:

$$c_1(\phi) = \frac{\int_\Omega I(x,y)H(\phi)dxdy}{\int_\Omega H(\phi)dxdy} \tag{2.6a}$$

$$c_2(\phi) = \frac{\int_\Omega I(x,y)(1-H(\phi))dxdy}{\int_\Omega (1-H(\phi))dxdy} \tag{2.6b}$$

The parameters c_1 and c_2 are updated according to the topological changes of ϕ. $H(\phi)$ denotes the Heaviside function and is defined as $H(\phi) = \begin{cases} 1, \phi < 0 \\ 0, \phi > 0 \end{cases}$, and its Dirac function $\delta(\phi)$ is defined as $\delta(\phi) = \frac{d}{d\phi}H(\phi)$ [35]. The region inside the curve C, where $\phi < 0$ as defined in Equation 2.5, is represented by $H(\phi) = 1$, and therefore Equation 2.6a is applied to compute the average intensity of this region. Equation 2.6 applies to the region outside the curve C, where $\phi > 0$.

The derivative is deduced from the Euler-Lagrange equation of ϕ, which minimizes the Chan-Vese model energy function E (Equation 2.3) [35].

$$\frac{d\phi}{dt} = \delta(\phi)[\mu\nabla \cdot (\frac{\nabla\phi}{|\nabla\phi|}) - \nu - \lambda_1(I(x,y) - c_1)^2 + \lambda_2(I(x,y) - c_2)^2] \tag{2.7}$$

For the purpose of expanding the evolution to all level sets of ϕ, the Dirac function $\delta(\phi)$ is replaced by $|\nabla\phi|$ [35]. Therefore, Equation 2.7 becomes

$$\frac{d\phi}{dt} = |\nabla\phi|[\underbrace{\mu\nabla \cdot (\frac{\nabla\phi}{|\nabla\phi|})}_{F_{in}} \underbrace{-\nu - \lambda_1(I(x,y) - c_1)^2 + \lambda_2(I(x,y) - c_2)^2}_{F_{ext}}] \tag{2.8}$$

In the two-cycle algorithm of fast level set method (Algorithm 2.1), the above equation can be formed as $\frac{d\phi}{dt} = |\nabla\phi|(F_{in} + F_{ext})$ and ν is equal to 0 so that the two speed functions are [35]:

$$F_{ext} = \text{sign}(-\lambda_1(I(x,y) - c_1)^2 + \lambda_2(I(x,y) - c_2)^2) \tag{2.9a}$$

$$F_{in} = \mu\nabla \cdot (\frac{\nabla\phi}{|\nabla\phi|}) \tag{2.9b}$$

where F_{ext} is the data dependent speed function, which leads the curve moving towards the desired contour of the object being tracked; and F_{in} is the curve smoothness regularization speed function, which makes the curve smooth.

In Equation 2.9a, the parameters λ_1 and λ_2 are used to adjust the weights of the region located inside the curve and of the region outside the curve, respectively. In [35], the adaptive data dependent speed function is proposed in order to classify each pixel accurately into its region, either the object region (the foreground) or the background. The adaptive data dependent speed function is achieved by setting $\lambda_2 = k\lambda_1$.

$$F_{ext} = \begin{cases} -\lambda_1(I(x,y) - c_1)^2 + k\lambda_1(I(x,y) - c_2)^2, & \text{if } |F_{ext}| < thd \\ -(I(x,y) - c_1)^2 + (I(x,y) - c_2)^2, & \text{otherwise} \end{cases} \tag{2.10}$$

where $I(x,y)$ is the intensity at a pixel with coordinates of (x,y), k represents a constant parameter and $k \geq 1$, and thd denotes the threshold that weighs the curve smoothness regularization speed function and data dependent speed function. The parameters c_1 and c_2 are calculated according to Equation 2.6.

The curve smoothness regularization speed function, F_{in}, is approximated by:

$$F_{in} = \mu\nabla \cdot (\frac{\nabla\phi}{|\nabla\phi|}) = \mu\kappa \tag{2.11}$$

where κ is the curvature of the evolving curve, μ is a regularization parameter, and ∇ represents the first-order derivative function. The curvature calculation is computationally expensive and therefore the Laplacian of ϕ is used instead as a simplified expression of the curvature. Furthermore, the evolution of the Laplacian of a function is equivalent to Gaussian filtering this function [36]. A Gaussian filter, G, is therefore employed as the smoothness regularization term to accelerate the fast level set

implementation. The Gaussian filter is only applied on the pixels of L_{out} and L_{in}. With the data dependent speed function, F_{ext}, the evolution of curve, C, follows Algorithm 2.1.

Algorithm 2.1 Two-cycle algorithm of fast level set method

1: Initialize the array ϕ, F_{ext}, and the two lists L_{in} and L_{out}.

2: **for** $i = 1 : N_a$ **do** ***// Cycle One***

3: Compute F_{ext}, for pixels in L_{in} and L_{out};

4: For every pixel $\mathbf{x} \in L_{out}$, apply *switch_in(***x***)* if $F_{ext}(\mathbf{x}) > 0$;

5: For every pixel $\mathbf{x} \in L_{in}$, remove $\mathbf{x}$ from L_{in} and set $\phi(\mathbf{x}) = -3$ if $\phi(\mathbf{y}) < 0$ for $\forall \mathbf{y} \in N_4(\mathbf{x})$;

6: For every pixel $\mathbf{x} \in L_{in}$, apply *switch_out(***x***)* if $F_{ext}(\mathbf{x}) < 0$;

7: For every pixel $\mathbf{x} \in L_{out}$, remove $\mathbf{x}$ from L_{out} and set $\phi(\mathbf{x}) = 3$ if $\phi(\mathbf{y}) > 0$ for $\forall \mathbf{y} \in N_4(\mathbf{x})$;

8: Check the stop condition and if satisfied, go to ***Cycle Two***; else continue this cycle.

9: **end for**

10: **for** $j = 1 : N_g$ **do** ***//Cycle Two***

11: For every pixel $\mathbf{x} \in L_{out}$, compute $G \otimes \phi(\mathbf{x})$. Apply *switch_in(***x***)* if $G \otimes \phi(\mathbf{x}) < 0$;

12: For every pixel $\mathbf{x} \in L_{in}$, remove $\mathbf{x}$ from L_{in} and set $\phi(\mathbf{x}) = -3$ if $\phi(\mathbf{y}) < 0$ for $\forall \mathbf{y} \in N_4(\mathbf{x})$;

13: For every pixel $\mathbf{x} \in L_{in}$, compute $G \otimes \phi(\mathbf{x})$. Apply *switch_out(***x***)* if $G \otimes \phi(\mathbf{x}) > 0$;

14: For every pixel $\mathbf{x} \in L_{out}$, remove $\mathbf{x}$ from L_{out} and set $\phi(\mathbf{x}) = 3$ if $\phi(\mathbf{y}) > 0$ for $\forall \mathbf{y} \in N_4(\mathbf{x})$;

15: **end for**

16: If one of *Stop Conditions* is satisfied in ***Cycle One***, terminate the algorithm; otherwise, go back to ***Cycle One***.

The two-cycle algorithm of fast level set method (Algorithm 2.1) stops whenever either of the following two conditions is satisfied [36].

Stop Conditions

1: The speed at every neighboring pixel satisfies:

$$F_{ext}(\mathbf{x}) \leq 0, \forall \mathbf{x} \in L_{out}$$
$$F_{ext}(\mathbf{x}) \geq 0, \forall \mathbf{x} \in L_{in}$$

2: A pre-specified maximum number of iterations, N_a, is reached.

In Algorithm 2.1, N_a represents the number of iterations of the functions *switch_in* and *switch_out* implemented depending on the result of the data dependent speed function F_{ext} (Algorithm 2.1 lines 4 and 6), and N_g is the size of the Gaussian filter, G, applied over the contour in order to smooth it. The parameter N_a is set higher than N_g to minimize the side effect of the smoothness regularization which weakens the sharp corners of the contour. A high value of N_a guarantees that the contour of an object can be detected accurately when the first stopping condition is fulfilled. If not, it should ensure that Algorithm 2.1 does not take a long time to achieve the second stop condition. The value of N_g cannot be large since it affects the size of the Gaussian filter, which is $N_g \times N_g$, and it should also be odd according to the properties of the Gaussian filter. As the size of the Gaussian filter gets larger, the processing time becomes longer. From the point of view of efficiency, N_g should therefore be small.

The initial level set function contour has an important impact of the quality of results and the computation time for the level set method [38]. The goal is therefore to identify a best possible initialization for the level set. This can be achieved by a proper initial identification of the object in the visual scene, supported in turns by a background removal algorithm and even further by user guidance.

In conclusion, object or contour tracking represent important tasks within the field of computer vision, which have been widely used in some typical fields of automated surveillance, traffic monitoring and so on. Object tracking is commonly used for motion analysis. It consists in detecting the object of interest, tracking it from frame to frame, and analyzing it in order to recognize its behavior [39]. The fast level set method described in this section (section 2.3.2.2) is used in the context of this thesis to segment and track the contour of an object in an image sequence [35]–[37], as it is computationally simple, and has the potential capability to detect the contours in near real-time. Furthermore, it is capable to track contours of two independent objects even when a collision occurs between them, while working on grayscale and color images as well.

2.3.3 Random Sample Consensus (RANSAC)

RANSAC, the random sample consensus (RANSAC) algorithm, was introduced by Fischler and Bolles [40]. It is a robust parameter estimation approach for generating a mathematical model from a set of data with a limited number of iterations. RANSAC is widely used to detect and segment objects with common geometric structures including lines, planes, cylinders and spheres. RANSAC consists essentially in the iteration of two steps: hypothesis generation and hypothesis evaluation. It generates a hypothesis from a random sample subset with minimal data points of the input data, and verifies the hypothesis for the entire dataset. A data element is considered as an outlier if it does not fit the hypothesis well, otherwise it is an inlier. The consensus set is composed of all the inliers. The algorithm will stop after a given number of iterations with the estimated parameters for the model extracted from the largest consensus set. The RANSAC algorithm is used in this thesis for background removal, as will be detailed in section 3.3.1.

2.3.4 Log-Polar Transform

The *log-polar transform* simulates, to some extent, the human visual model, in which a high resolution around the fovea allows the eye to fixate on an object of interest, while the rest of visual information is encoded at a lower resolution. In this work, the central blind-spot model is used because it is rotation and scale invariant [41]. The log-polar mapping of the image also has the potential to contribute to improve the image segmentation [41]–[43]. In these papers, a log-polar transformation is applied on an Cartesian image before segmenting it in order to extract the contour, edge or shape of an object of interest. Compared to the background, the object of interest fills a relatively large area in the log-polar (cortical) image. In other words, the log-polar mapping enlarges the percentage of coverage of an object of interest in the whole image, which is an advantage that this research capitalizes on.

The approach of the blind-spot model for mapping the Cartesian domain (x, y) into the log-polar (cortical) domain (ξ, η) has been proposed in [33] [37]. With the polar coordinates $(\rho, \theta) = (\sqrt{x^2 + y^2}, arctan(y / x))$ derived from the Cartesian coordinates (x, y) of a pixel, the continuous log-polar coordinates can be defined as [44]:

$$\begin{cases} \xi = log_a(\frac{\rho}{\rho_0}) \\ \eta = \theta \end{cases} \tag{2.12}$$

where a is a parameter of the non-linearity of the mapping, and ρ_0 represents the radius of the central blind spot such that all points in the range of ρ_0 are ignored.

The coordinates (u, v) of the discrete log-polar image are defined as [44]:

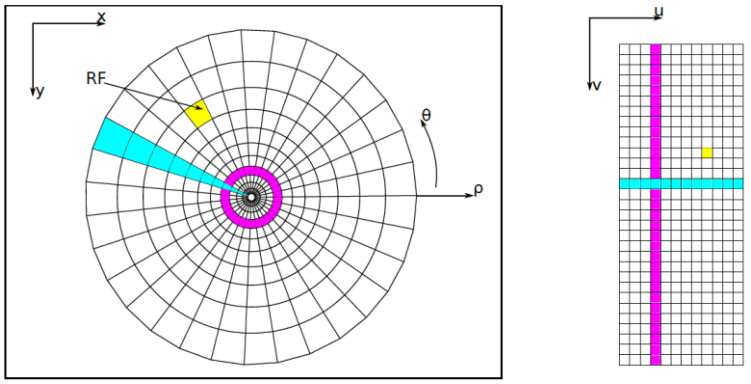
$$\begin{cases} u = \lfloor \xi \rfloor \\ v = \lfloor q\eta \rfloor \end{cases} \tag{2.13}$$

where $\lfloor . \rfloor$ denotes the integer part, and $q = S/2\pi$ is a parameter representing the angular resolution.

This discrete mapping is a non-linear transformation that maps a digital Cartesian image of $I \times J$ pixels into a discrete cortical image of R (rings) $\times$ S (sectors) pixels. The optimal S can be obtained by $S = 2\pi/(a - 1)$ for a given R [44]. In Equation 2.12, $a = e^{ln(\rho_{max}/\rho_0)/R}$ can be obtained with a defined $\rho_{max} = 0.5\text{min}(I, J)$. In Equation 2.13, $u \in [1, R]$ and $v \in [1, S]$ and the most favorable relationship between R and S is to optimize the pixel aspect ratio close enough to 1 [45]. The retinal image is obtained by the inverse log-polar mapping that follows the same principle [44].

Figure 2.2 shows a template of the mapping of an image from the Cartesian domain to the log-polar domain with the parameters: R = 12, S = 31. The fuchsia ring, the cyan sector and the yellow receptive field (RF) in the Cartesian domain correspond to the fuchsia column, the cyan row, and the yellow cell, respectively, in the log-polar domain. The RF denotes an area of intersection between a sector and a ring in the Cartesian domain, as shown in Figure 2.2. The center of the round template (Figure 2.2a) is the blind spot, with its surrounding first ring representing the maximum resolution as the first column of the cortical image (Figure 2.2b).

The shape of the RF affects the efficiency and quality of the transformation. Chessa et al. [44] demonstrate that bilinear interpolation offers the best performance and is therefore the solution chosen for this thesis. That is, the value of each pixel of the log-polar (cortical) image is acquired from the interpolation of the values of four neighboring pixels that are located nearest to the corresponding RF in the Cartesian domain.



(a) (b)

Figure 2.2: A template example maps from: (a) the Cartesian domain to (b) the log-polar domain.

2.3.5 Clustering Techniques

Clustering approaches aim at dividing a point cloud into several clusters according to spatial decomposition techniques that allow the points to be grouped together based on a given measure, such as the Manhattan distance or the Euclidean distance [46]. Clusters are obtained by searching the neighboring points within a given distance in the point cloud. Alternatively, other solutions could be employed for this purpose, for example the color-based region growing segmentation that searches clusters in the point cloud by taking both color and distance into consideration. However, such a solution takes a longer time to process the data compared to a clustering approach based on Euclidean distance alone.

In the context of this work, a *K-dimensional* (*k*-d) tree structure is exploited to support a faster neighbor search. As proposed and improved by Bentley [47], a *k*-d tree is a spatial decomposition technique for organizing a set of points in a *k*-dimensional space and is exhaustively discussed in the literature [48] [49]. The *k*-d tree is a binary tree in which each node represents a *k*-dimensional point. Constructing a *k*-d tree structure data implies splitting the data by using a hyperplane that is perpendicular to the corresponding axis. That is, at the root of the tree all the data is split based on the first dimension; each depth down in the tree divides on the next dimension, returning to the first dimension once all others have been exhausted. This structure contributes to ensure effective range search processes, and especially makes searching for nearest neighbors [50] or fixed-radius near neighbors [51] extremely fast, and accelerates distance-based clustering [48]. Several papers [52]–[55] employ the algorithm of *k*-d tree for searching nearest neighbors in order to simplify and generalize the searching process.

The algorithm for the construction of a *k*-d tree is summarized in Algorithm 2.2 [46] [56] [57]. With the input of a *k*-dimensional point cloud, P, the algorithm arranges the point cloud into a tree structure.

Algorithm 2.2 K-d tree construction algorithm

1: **function** BuildKdtree(P, d)

2: **if** P.empty() **then**

3: **return** $\emptyset$

4: **else** if P.singleton()

5: **return** a leaf storing this point

6: **else**

7: Set $axis = d \% k$

8: Set m = median(P, $axis$)

9: Set L = leftPoints(P, $axis$), R = rightPoints(P, $axis$)

10: Set $node$ = KdNode(m)

11: node.left ← BuildKdtree(L, $d + 1$)

12: node.rightChild ← BuildKdtree(R, $d + 1$)

13: **return** node

14: **end if**

15: **end function**

In the context of this work, a three-dimensional tree will be used, as the work focuses on 3D point cloud recuperated by a Kinect sensor. An example of constructed a three-dimensional tree is illustrated in Figure 2.3. For a set of data points of coordinates (x, y, z), the depths would be cycled as x, y, z, x, y, z... for successive depths of the *k*-d tree [58]. Figure 2.3 shows the structure of a 3-dimensional tree and the splitting hyperplanes. At the root, the set of data is partitioned with an x-aligned splitting plane, for its children the partition uses y-aligned splitting planes, for its grandchildren the partition is performed with z-aligned splitting planes, for its great grandchildren the partition is based on x-aligned splitting planes again, and so on. The medians of the points are selected at each depth of the tree and being used as splitting planes, with respect to their coordinates in the axis.

To better explain the working principle of k-d trees, we consider the following example in [18] that is based on the following 15 input points: (2,3,3), (5,4,2), (9,6,7), (4,7,9), (8,1,5), (7,2,6), (9,4,1), (8,4,2), (9,7,8), (6,3,1), (3,4,5), (1,6,8), (9,5,3), (2,1,3), (8,7,6) for which a *k*-d tree is built:

• Beginning with the x-coordinate, the median of values of all points over the x-coordinate is found, which is 7, so that (7,2,6) is picked as the root node from all the points (Figure 2.4). The points whose values of x-coordinate are smaller than the median are placed in the left subtree; the points whose values of x-coordinate are larger than the median are situated in the right subtree.

• At the tree depth 1, each of the subtree is partitioned based on the y-coordinate. The median of the y-coordinate of the points from the left subtree of (7,2,6) is 4, so that (5,4,2) is picked as the left node of the subtree. The points whose values of y-coordinate are smaller than the median are placed in the left subtree; the points whose values of y-coordinate are larger than the median are placed in the right subtree. The same process is used for the right subtree of (7,2,6).

• At the depth 2 of the tree, the z-coordinate is chosen to partition the points. The median of the z-coordinate of points from the left subtree of (5,4,2) is 3, so that (2,1,3) is picked as the left node of the subtree of (5,4,2). The points whose value of z-coordinate is smaller than the median are placed in the left subtree; the points whose value of z-coordinate is larger than the median are placed in the right subtree. The same principle is applied to the left and right subtrees of (9,5,3).

• At the depth 3 of the tree, the x-coordinate is chosen again to partition the points. However, each subtree of the nodes of depth 2 just contains a single point. All these points are considered as the nodes of depth 3 of the tree.

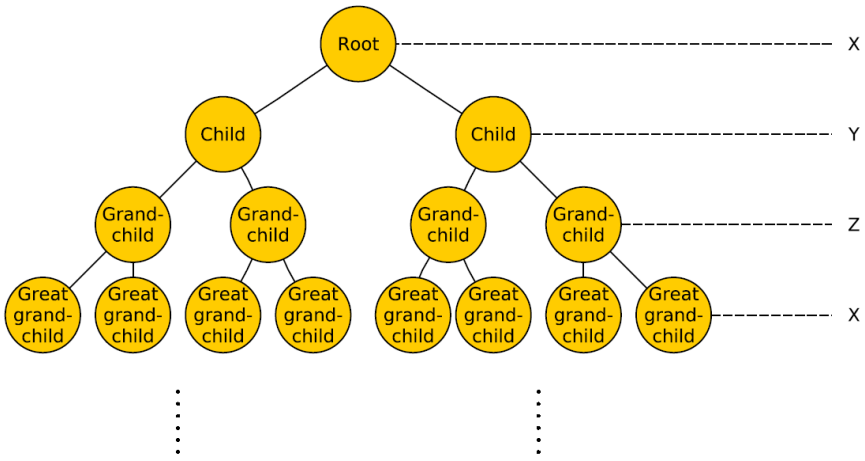


Figure 2.3: An example of a three-dimensional tree.

Figure 2.4 shows the resulting tree structure.

The *k*-d tree construction algorithm (Algorithm 2.2) is applied in the context of this thesis to the entire point cloud collected by the Kinect sensor. The established *k*-d tree structure aims to organize data to enable a faster search. For example, it enables faster identification of the nearest neighbor point of a given point by capitalizing on the tree structured data. Beginning with the root of the tree, the *k*-d tree construction algorithm saves the root as the current best point and navigates down the tree; it moves each depth down in the tree, saving the current node as the current best point if it is closer to the target point; the algorithm terminates whenever no node is closer than the current best point. At each depth, the algorithm proceeds along either left or right subtree, which speeds up the searching process while

comparing distances between all the points and the given point. This demonstrates the advantage of using the *k*-d tree algorithm to structure and search the neighboring points from a large set of data effectively and efficiently.

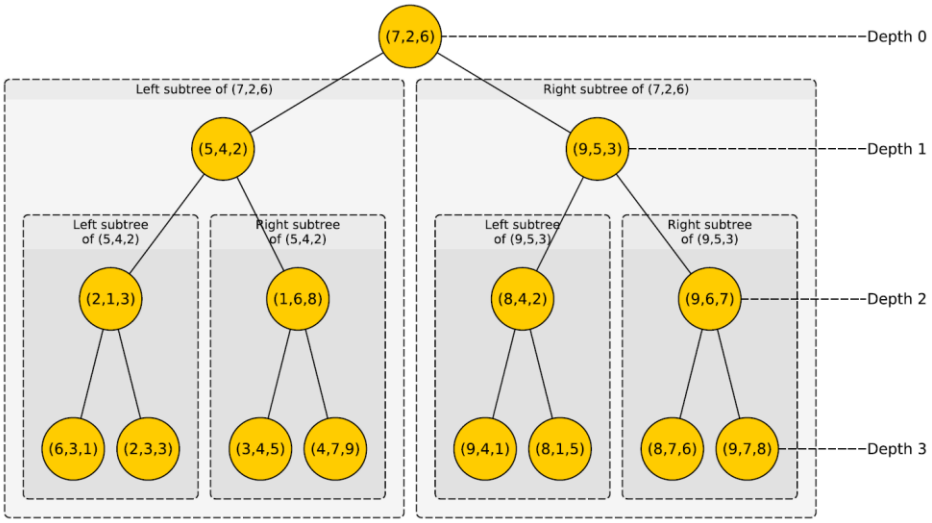


Figure 2.4: An example of 3-d tree.

2.4 Contour Matching for Object Characterization

As highlighted in section 2.1, even though great research effort was invested in modeling deformable objects in order to imitate their behavior, to study the material they are made of, or to improve the process of manipulating an object, few research initiatives have so far addressed the issue of a model-free characterization of the deformation behavior. In this thesis, the goal is to characterize and classify objects according to the material they are composed of, and therefore categorize the expected behavior of an object when submitted to manipulation, based on how the contours of the object, tracked using the fast level set method but without constructing a model, evolve.

In the work of Sun and Super [59], a Bayesian classification is performed to classify objects based on their contour. A part-based approach is favored, which uses the partial contour of the object instead of the entire one. The landmarks employed are two endpoints. Bai et al. [60] combine both contour and skeleton information for shape classification. The contour and the skeleton of a given 2D object are extracted and used together as the shape model of the object. A prior training for the classification system is required. Cretu et al. [61] investigate the problem of learning the properties of deformable objects under the manipulation of a robotic hand. Contours of a deformable object are tracked from a sequence of images as the inputs of a neural network that is used to classify and predict the object deformable behavior under future or virtual manipulation. The classification behavior and prediction rely on the choice of a fixed number of nodes (landmarks) in the neural gas network, which is also used to identify and model the tracked contour. While interesting as a concept, the method also requires a training process prior to its use. Gonzalez-Sosa et al. [62] propose an approach to compare the human body shape using MMW images (millimetre wave images). In this approach, landmarks selected on a

pair of detected body contours are matched by using dynamic time warping and the similarity of the body shapes is determined by the accumulated distance in between the pairwise landmarks.

Generally, landmarks [59]–[62] have been used to represent a typical characteristic of an object contour or shape. A contour is able to represent detailed 2D shape information, but is sensitive to non-rigid deformations [60]. Some researchers [61] [62] investigated the change of a pair of contours through comparing the displacements of contours in pairs by using contour information. The research of [61] and [62] localizes landmarks over a pair of object contours, establishes the correspondence in between landmarks of two contours and then calculates the displacements in between this pair of contours. However, the identification of landmarks, to represent the entire object, over the object contour is a difficult task.

One possible solution to deal with contours that do not have an equal length is dynamic time warping (DTW). It can be employed to establish the correspondence of pixels of two contours of an object independently from the varying length of the tracked contour as the object deforms. It is a well-known technique used on general sequences to establish correspondence between them. In [63], the author introduces the concept of DTW, provides a detailed algorithm and describes its use in fields such as data mining and information retrieval. DTW is originally applied to compare different speech patterns in automatic speech recognition [64]. It has also been successfully applied for signature recognition [65], shape matching [66]–[69], motion retrieval [63] [70] and even face features detection [71].

The algorithm obtains the optimal matching between two sequences by finding the optimal warping path, which is the one with the minimal total cost, in the DTW distance matrix of two sequences. Originating from the computationally expensive classical DTW distance matrix, several ideas have been proposed for speeding up the process, such as the global constraint conditions on the admissible warping paths [63] and its extensions [72]–[74], the step size condition for constraining the slope of the admissible warping paths [63], and local weights for calculating the local cost equation [63].

DTW copes with the problem of aligning two sequences with arbitrary lengths. It is employed to establish the correspondence between two contour sequences $C_1 = \boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_m, \ldots, \boldsymbol{p}_M$ of length M and $C_2 = \boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_n, \ldots, \boldsymbol{q}_N$ of length N. In order to compare the difference of C_1 and C_2, the local cost or distance matrix d is computed to evaluate the similarity in between each pair of pixels from the sequence of C_1 and C_2. Each element of the distance matrix $d[m, n]$ represents the Euclidean distance between the pixel, $\boldsymbol{p}_m$, from the first contour and the pixel, $\boldsymbol{q}_n$, from the second contour [73]. The value of $d[m, n]$ is low (low cost) if $\boldsymbol{p}_m$ and $\boldsymbol{q}_n$ are similar to each other in their relative location, or high otherwise [63]. Algorithm 2.3 illustrates the process of filling a warping matrix with the distance matrix d [73].

Algorithm 2.3 Standard DTW algorithm

Input:

$C_1 = \boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_m, \ldots, \boldsymbol{p}_M$ sequence of length M;

$C_2 = \boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_n, \ldots, \boldsymbol{q}_N$ sequence of length N.

Output:

DTW warping matrix.

1: $DTW = \text{matrix}\,[M + 1, N + 1]$

2: Initialize $DTW[m, 0] = \infty$

3: Initialize $DTW[0, n] = \infty$

4: $DTW[0, 0] = 0$

5: **for** $m = 1 : M$ **do**

6: **for** $n = 1 : N$ **do**

7: $DTW[m, n] = d[m, n] + \min \begin{cases} DTW[m-1, n] \\ DTW[m, n-1] \\ DTW[m-1, n-1] \end{cases}$

8: **end for**

9: **end for**

10: **return** $DTW[M, N]$

where the DTW represents the warping matrix.

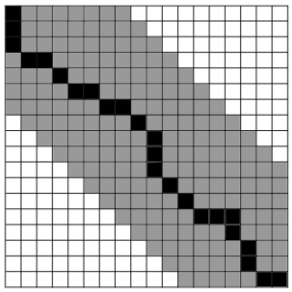


Figure 2.5: Example of a warping matrix (each cell represents a pixel of the image).

In practice, to improve the computation time, the warping matrix is calculated with an additional locality constraint r, proposed by Lemire [72]. This means that for the contour sequences C_1 and C_2, instead of calculating the DTW distance over all pairs of pixels, the DTW distance is only calculated for the part of warping matrix where the condition $|M - N| \leq r$ is satisfied. With this locality constraint, the computed DTW distance is restricted to a band (shown in gray in Figure 2.5), along the diagonal of the warping matrix, which speeds up the DTW distance calculation. As such, in Figure 2.5, the white areas

are eliminated from the calculations in order to reduce the computation time. The standard DTW algorithm with locality constraint is summarized as follows [63]:

Algorithm 2.4 Conditional DTW algorithm

Input:

$C_1 = \boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_m, \ldots, \boldsymbol{p}_M$ sequence of length M;

$C_2 = \boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_n, \ldots, \boldsymbol{q}_N$ sequence of length N.

Output:

DTW warping matrix.

1: DTW = matrix $[M + 1, N + 1]$

2: r = max$(r, |M - N|)$

3: **for** $m = 0 : M$ **do**

4: **for** $n = 0 : N$ **do**

5: $DTW[m, n] = \infty$

6: **end for**

7: **end for**

8: $DTW[0, 0] = 0$

9: **for** $m = 1 : M$ do

10: **for** n = max$(1, m - r) :$ min$(N, m + r)$

11: $$DTW[m, n] = d[m, n] + \min \begin{cases} DTW[m-1, n] \\ DTW[m, n-1] \\ DTW[m-1, n-1] \end{cases}$$

12: **end for**

13: **end for**

14: **return** $DTW[M, N]$

where r is set empirically. The optimal warping path, W, is the one with the minimal overall cost from the DTW matrix. It is represented by the series of black pixels in Figure 2.5. Each element of the optimal

warping path indicates the correspondence between a pair of pixels from the contours sequences C_1 and C_2, respectively. The optimal warping path is depicted by a sequence $W = w_1, w_2, \dots, w_l, \dots, w_L$, where $w_l = (m, n)$, $\max(M, N) \leq L < (M + N)$, and L is the length of the warping path. This warping path must satisfy the following three conditions [63]:

Warping path conditions

1: Boundary condition: $w_1 = (1, 1)$ and $w_L = (M, N)$;

2: Monotonicity condition: $m_1 \leq m_2 \leq \dots \leq m_L$ and $n_1 \leq n_2 \leq \dots \leq n_L$;

3: Step size condition: $w_{l+1} - w_l \in \{(1,0), (0,1), (1,1)\}$ for $l \in [1, L - 1]$.

This optimal path, W, is computed in reverse order of the indices starting with $w_L = (M, N)$. Suppose $w_l = (m, n)$ has been computed. Once $w_{l-1} = (m, n)$ is calculated, if $(m, n) = (1, 1)$ and $l = 1$, W is completed; otherwise, the next elements are defined using the following equation [63]:

$$w_{l-1} = \begin{cases} (1, n-1), & \text{if } m = 1 \\ (m-1, 1), & \text{if } n = 1 \\ \text{argmin}\{DTW[m-1, n-1], DTW[m-1, n], DTW[m, n-1]\}, & \text{otherwise} \end{cases} \quad (2.14)$$

With the above three *warping path conditions*, the warping path matches the two contour sequences C_1 and C_2 by assigning the pixel $\boldsymbol{p}_m$ of C_1 to the closest pixel $\boldsymbol{q}_n$ of C_2, so that an element w_l of the warping path stands for a pair of pixels $(\boldsymbol{p}_m, \boldsymbol{q}_n)$.

Due to its potential to cope with the problem of establishing the correspondence between pixels of two contour sequences with different lengths, unlike classical solutions that require a one-to-one pixel (point) correspondence of contours [61], [62], the DTW algorithm offers a viable solution for comparing tracked deformable contours that will be exploited in the context of this thesis.

2.5 Summary of Literature Review

This chapter surveyed various classical approaches from the literature for image segmentation, including region splitting, region growing and the watershed algorithm. A common issue with these algorithms is that pixels are clustered based on their intensity which leads to inefficiencies when tracking the topological changes of the contour of an object. The fast level set method provides an interesting alternative to extract complete objects and track, in near real-time, their contour during deformation, as happens when an object is submitted to external forces. The majority of level set methods require an initial curve that can be situated anywhere within the image in the Cartesian domain. With the goal to alleviate this limitation, alternative mapping methods such as the log-polar representation of the color map have been studied. These will be exploited in the current work to improve and adapt the fast level set method to the context of deformable object contour tracking.

The literature review also reveals many approaches to support the characterization of objects. Some of these approaches commonly employ contour information, often represented as features from the

contour, or landmarks distributed over the contour. However, few methods support the use of all the pixels composing the contour of an object. Many approaches build a mathematical representation of either 2D or 3D data to approximate the object. They then attempt to minimize the difference between the model and the deformation of an object to simulate, study and predict the deformation process. But given that this research focuses on the model-free material characterization of deformable objects, solutions for matching contour descriptors, especially those formed by level sets, were studied, such as dynamic time warping, which provides an interesting path for the classification of deformable objects into categories that will be discussed in the following chapter.

Chapter 3 Object Deformation Tracking and Material Behavior-Based Classification

In this chapter, an integrated approach is proposed for automated visual monitoring and material behavior-based classification of deformable objects using a robotic manipulation process. This chapter is composed of three main parts, namely: data acquisition, segmentation and tracking, and material behavior-based classification.

3.1 Proposed Approach for Object Deformation Tracking and Material Behavior-Based Classification

The proposed system designed to perform object deformation monitoring and classification according to the material the object is composed of operates from RGB-D data, as illustrated in Figure 3.1.

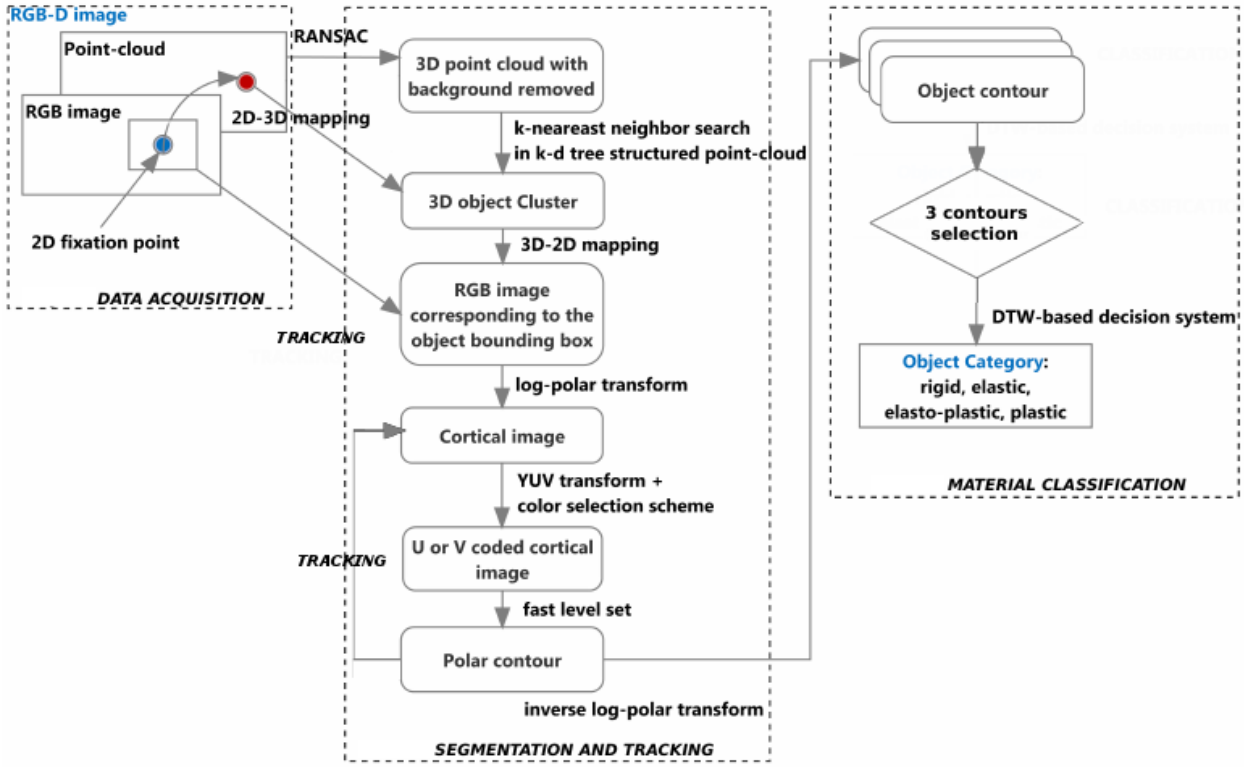


Figure 3.1: Flowchart of object detection, deformation tracking, and material behavior-based classification.

It can be briefly summarized as follows: The object shape is acquired using a Kinect for Xbox 360 sensor. The RGB Cartesian image, its corresponding point cloud (depth image) acquired by the Kinect sensor, and a user selected point in the RGB image, here called the 2D fixation point, which guides the system towards the location of the object of interest, are taken as inputs by the object segmentation and

tracking algorithm. The tracking is achieved using the fast level set method. However, a new solution is proposed for initializing the fast level set method. First, a 3D fixation point is calculated from the 2D fixation point, using a 2D-3D mapping in the RGB-D data based on the given inputs. Next, the point cloud recuperated from the Kinect sensor is processed through a background removal operation using the RANSAC algorithm. The motivation for this operation is to eliminate as many undesirable elements as possible (e.g. measurement table and other measuring equipment visible in the RGB-D data stream) for a better identification of the object shape and thus a reduction of the processing time. Then, a cluster representing the object of interest is extracted by searching the neighbor points of the computed 3D fixation point from the remaining part of the point cloud after background removal. After that, the identified 3D points are projected back to 2D and the bounding window around the identified object is computed based on the resulting projection in the RGB image. Furthermore, the region within the bounding image is encoded in the YUV color space, and one of the U or V components is automatically selected for being further processed. This choice is justified by the desire to work with higher contrast to enable more accurate segmentation. The bounding window of the selected color component (U or V) is mapped into the log-polar (or cortical) domain using the log-polar mapping. A fast level set method is applied on the cortical bounding window of the selected color component to detect and track the contour of the object of interest while it is manipulated by a robotic hand. Finally, the cortical object contour is transformed back to the Cartesian domain of the RGB image (as introduced in section 2.3.4). The resulting contour then supports an automated classification stage to characterize the physical behavior of an object according to the material is it composed of.

In the classification stage, three representative object contours are extracted from the sequence of frames captured during a predetermined manipulation process of the object that is designed to support the object characterization procedure. Each frame contains the extracted contours of the object. A dynamic time warping (DTW) approach is applied to establish pairwise correspondence between three contours during the classification process. The object is categorized using this approach as being composed of elastic, plastic, elasto-plastic, or rigid material, or for more complex objects, as exhibiting an elastic, plastic, elasto-plastic, or rigid stage of deformation. The purpose of this characterization is to enhance manipulation capabilities for a robotic hand by dealing efficiently with deformable objects composed of different materials that impact their physical behavior.

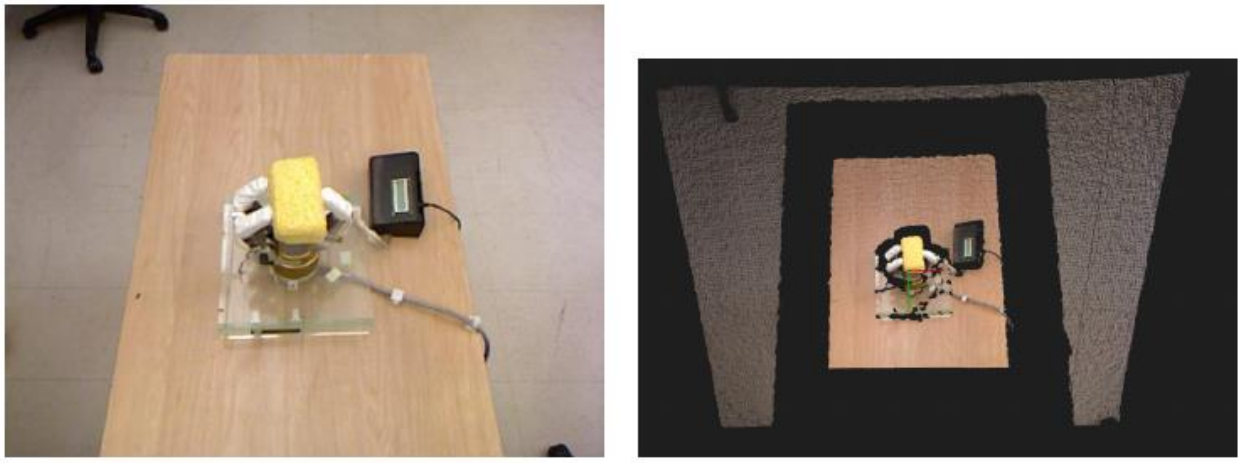
The details for every step of the proposed procedure are presented in the following sections. Section 3.2 presents the acquisition of the data on the object, including the capture of the color image, the depth image and the point cloud, the 2D fixation point selection and the algorithm for mapping this point to 3D. Section 3.3 details the implementation of the proposed solution for segmentation and contour tracking, consisting of a background removal procedure based on the RANSAC algorithm, an object cluster identification using a proposed custom 3D cluster extraction algorithm, a robust color selection scheme to enhance the object contour detection, and an original formulation of the fast level set method in the log-polar domain. Section 3.4 presents the implementation of the object material classification procedure which establishes contour correspondences based on dynamic time warping to classify the object as belonging to either the elastic, plastic, elasto-plastic, or rigid category.

3.2 Data Acquisition

This section presents the acquisition of RGB-D data using a Kinect sensor and the procedure to pick a user-selected point that helps locate and transform the object of interest from the color image to the 3D space, where it will serve to refine the object segmentation procedure.

3.2.1 RGB-D Data

The proposed system takes as inputs the RGB image and the corresponding point cloud collected by a Kinect for Xbox 360 sensor, as illustrated in Figure 3.2.



(a) (b)

Figure 3.2: (a) RGB image (640×480) from Kinect, and (b) raw point cloud from Kinect.

A detailed description of the sensor, as well as of the software used to collect the data, is provided with the experimental results in chapter 4.

3.2.2 2D Fixation Point

User guidance to help with the initialization of tracking algorithms is common in the current literature. A user-selected point can efficiently guide the system towards the location of the object of interest. For example, a prior segmentation of the object of interest is assumed to be available in [4], or the user is asked to crop the object in the initial frame in [2], [42], [43]. In the current work, a single fixation point is used that can be selected randomly by a human operator over the object of interest. This simple procedure alleviates the need for complex object recognition and localization algorithms, which is beyond the scope of this thesis.

The coordinates of the user-selected fixation point are used by the system to indicate only the general location of the object over the field of view of the Kinect sensor. The idea of using a central fixation point is inspired by the work of Mishra et al. [42] [43], where the fixation point plays an important role as the center of the transformation. The authors suggest that human retina fixates an interesting object with a high resolution captured by the fovea, while the rest of the visual information is processed at

lower resolution. In the current work, this process is replicated by the use of a log-polar mapping to perform contour detection and tracking. In the log-polar map, the 2D fixation point represents the center of the transformation, where the precision of the log-polar image is maximal.

Upon analyzing a sequence of RGB-D data, a user-selected point is picked in the first frame, as depicted by the green dot in Figure 3.3, and fed in the system to initially guide the algorithm towards the location of the object of interest. In this work, a single fixation point is needed and it can be selected randomly over the object of interest. This selection is only required in the first frame of the data stream and allows the system to rapidly locate the deformable object independently from its color, shape, location or orientation in the workspace, and also independently from the complexity of the background or from the relative configuration of the robotic hand and Kinect sensor. The coordinates of the selected point are used by the system to indicate the general location of the object. Experiments (in section 4.2.4) will demonstrate that while the algorithm works regardless of the position of this point over the surface of the object of interest, a 2D fixation point, $\boldsymbol{p}_{2D}$, chosen roughly at the center of the object leads in general to more accurate contour estimation.

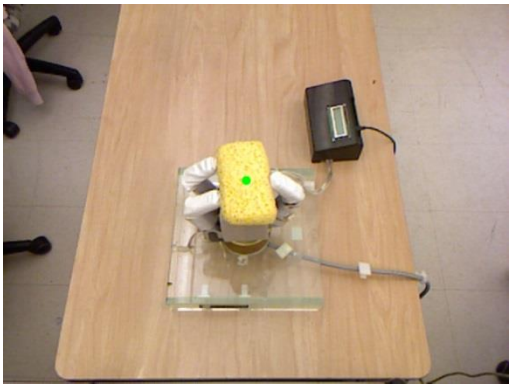


Figure 3.3: Fixation point selection over object of interest.

3.2.3 2D-3D Mapping

Due to the fact that the depth and RGB cameras are physically separated inside the Kinect sensor, before any processing can be applied, the depth image and the color image have to be registered (hardware details on the Kinect sensor and the software nodes used for registration are provided in section 4.1.1). The mapping refers in this section to a correspondence between a pixel (x, y) of the color (RGB) image and a point (X, Y, Z) of the point cloud.

A 2D-3D mapping that takes into consideration the intrinsic calibration parameters of the Kinect sensor is applied to estimate the coordinates in the point cloud of the 3D fixation point, $\boldsymbol{p}_{3D}$, corresponding to the 2D fixation point, $\boldsymbol{p}_{2D}$, in the RGB image. After the registration of the depth image and the color image collected by the Kinect sensor, the pixel $\boldsymbol{p}_{2D}$ (x, y) in the RGB image has a matching location in the depth image. If the depth value of the 2D fixation point relative to the depth image is denoted as $depth(x, y)$, the coordinates of the 3D fixation point $\boldsymbol{p}_{3D}$ (X, Y, Z) can be computed using the intrinsic parameters of the depth camera as follows [15]:

$$X = (x - c_{x\_d}) \times Z / f_{x\_d} \tag{3.1a}$$

$$Y = (y - c_{y\_d}) \times Z / f_{y\_d} \tag{3.1b}$$

$$Z = depth(x, y) \tag{3.1c}$$

where the matrix of the intrinsic parameters is [75]:

$$K = \begin{bmatrix} f_{x\_d} & 0 & c_{x\_d} \\ 0 & f_{y\_d} & c_{y\_d} \\ 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

and where

- f_{x_d} and f_{y_d} are the focal lengths of the depth camera along the x and y axis respectively in pixel units;
- c_{x_d} and c_{y_d} represent the coordinates of the principal point of the depth camera in pixel units.

The estimated 3D fixation point $\boldsymbol{p}_{3D}\ (X, Y, Z)$ is used as a guideline to separate the object of interest from the point cloud in the following section.

3.3 Segmentation and Contour Tracking

This section discusses the implementation of the object of interest's segmentation from the background and the extraction and tracking of its contour. A background removal procedure based on the RANSAC algorithm and applied over the point cloud, followed by a fixed-radius near neighbors search identifies the most probable 3D points representing the object of interest. The color image corresponding to this limited point cloud is further analyzed in the YUV color space. For this purpose, a robust color selection scheme is proposed to be employed prior to the application of a novel formulation of the fast level set method in the log-polar domain that enables near real-time contour identification, and subsequently the contour tracking in the data stream.

3.3.1 RANSAC-Based Algorithm for Background Removal

The segmentation and simplification of data have an extreme importance in point cloud processing to enable fast computation and real-time tracking of a contour. The initial goal is therefore to separate in the best possible way the points representing the object of interest from the background in the RGB-D data stream. In this section, the RANSAC algorithm, briefly introduced in section 2.3.3, is adapted for identifying the background and removing it from the point cloud.

3.3.1.1 Planar Surface Background Identification

For the experimentation phase conducted in the context of this research, the object whose material is being characterized is placed in a robotic hand installed over a table, as shown in Figure 3.2a. The assumption that the object of interest lies over a planar background located behind the object is

therefore made. This choice is justified by the fact that the goal of the research is on developing an automated characterization process for deformable objects, and not on the development of fully autonomous and comprehensive solutions for 3D object detection, localization, and manipulation over cluttered backgrounds.

Given this assumption, an efficient way to remove most of the background is to locate the planar surface representing the table and to extract that surface from the RGB-D data. This planar surface is the closest large surface to the object and its identification has an impact on better defining the area within which the object lies. The approximate localization of the planar surface corresponding to the table can be achieved using the random sample consensus (RANSAC) algorithm. As discussed in section 2.3.3, the RANSAC algorithm is especially suitable for generating mathematical models of geometric structures, including planes, which are here used as a representation of the table surface. The algorithm is applied over the 3D point cloud representing the depth information in the RGB-D data stream. Its purpose is to estimate a best plane model, defined as $ax + by + cz + d = 0$, in the point cloud. Its implementation is summarized as follows:

Algorithm 3.1 RANSAC algorithm for table identification

Input:

s	minimum size of a sample set (i.e. $s = 3$)
t	distance threshold (i.e. $t = 1$ cm)
N	number of iterations based on Algorithm 3.2
$PC = \{\boldsymbol{pc}_i\}, \boldsymbol{pc}_i = (x_i, y_i, z_i)$	3-dimensional point cloud
S, where $S \subset PC$	randomly selected sample set, $\lvert S \rvert \geq s$
$f = ax + by + cz + d$	function that computes the model parameters, $\theta = \{a, b, c, d\}$ from S
$ED(\theta, \boldsymbol{pc}_i) = \frac{\sqrt{ax_i+by_i+cz_i}}{a^2+b^2+c^2}$	distance from a point to the plane $ax + by + cz + d = 0$

Output:

θ^*	plane model parameters
S^*	largest consensus set (inliers) of the planar model

1: $n = 0, \theta^* = \emptyset$

2: Set $N \leftarrow$ Algorithm 3.2

3: **while** $N > n$ **do**

4: Select $S_n \subset PC$ such that $|S_n| \geq s$

5: Compute $\theta_n = f(S_n)$

6: **if** $\boldsymbol{pc}_i \notin S_n \wedge ED(\theta_n, \boldsymbol{pc}_i) < t$ **then**

7: $S_n \leftarrow \boldsymbol{pc}_i$

8: **end if**

9: Re-compute $\theta_n = f(S_n)$

10: **if** $|S_n| > |S^*|$ **then**

11: $S^* = S_n$

12: $\theta^* = \theta_n$

13: **end if**

14: $n = n + 1$

15: **end while**

16: **Return** S^*, θ^*

In the algorithm, the input $PC = \{\boldsymbol{pc}_i\}$ is a point cloud recuperated from the data stream of the Kinect sensor, where $\boldsymbol{pc}_i = (x_i, y_i, z_i)$ presents the vector of the 3-dimensional points. As at least three non-collinear points are required to estimate the plane model, the minimum size of the sample set, s, is set to 3. The distance threshold, t, is usually chosen empirically (section 4.2.1) and here set to $t = 1$ cm to ensure a good balance between the quality of the results for the planar surface removal and the processing time.

An important challenge with the RANSAC algorithm is to determine a proper stopping criterion for the algorithm. Generally, a randomly selected number of iterations, N, may lead to two situations: if the selected number is too low, the RANSAC algorithm cannot estimate the best planar model from the input point cloud; if the selected number is too high, the algorithm would generate the best planar model, but the increased number of iterations will waste time. Instead, the number of iterations, N, can be chosen according to Hartley and Zisserman [75] as being sufficiently high to ensure, with a certain probability, p, that at least one of the sets of random samples does not include an outlier. Usually, p is conservatively set to 0.99 [75]. Let ω be the probability that any selected data point is an inlier, while ϵ = 1 − ω is the probability of observing an outlier. Under the condition that the sample subset for the model estimation has the minimal size, *s*, at least *N* iterations need to be performed, where $(1 - \omega^s)^N = 1 - p$ [75], thus:

$$N = \frac{\log(1-p)}{\log(1-(1-\epsilon)^s)} \tag{3.3}$$

In practice, the proportion of outliers, ϵ, cannot be estimated in advance. Therefore, the algorithm to determine the suitable number of iterations starts with the worst case of ϵ and updates the estimation of N as the computation progresses. This adaptive approach for the calculation of N is summarized in Algorithm 3.2.

Algorithm 3.2 Number of iterations calculation for RANSAC

1: $N = \infty$, sample number = 0, p = 0.99

2: **while** N > sample count **do**

3: Choose a random sample and count the number of inliers

4: Set $\epsilon = \frac{\text{number of inliers}}{1 - \text{total number of points}}$

5: Set $N = \frac{\log(1-p)}{\log(1-(1-\epsilon)^s)}$

6: sample number = sample number + 1

7: **end while**

3.3.1.2 Planar Surface Segmentation and Removal

Once the best planar surface model is available, the flat surface that corresponds to the majority of the background as perceived by the Kinect sensor, now represented by the inliers of this model, is removed from the point cloud to obtain the reduced point cloud after the planar surface extraction. This process is summarized in Algorithm 3.3.

Algorithm 3.3 Planar surface removal algorithm

Input:

PC	3-dimensional point cloud
$S^*, S^* \subset PC$	largest consensus set (inliers) of the planar surface model

Output:

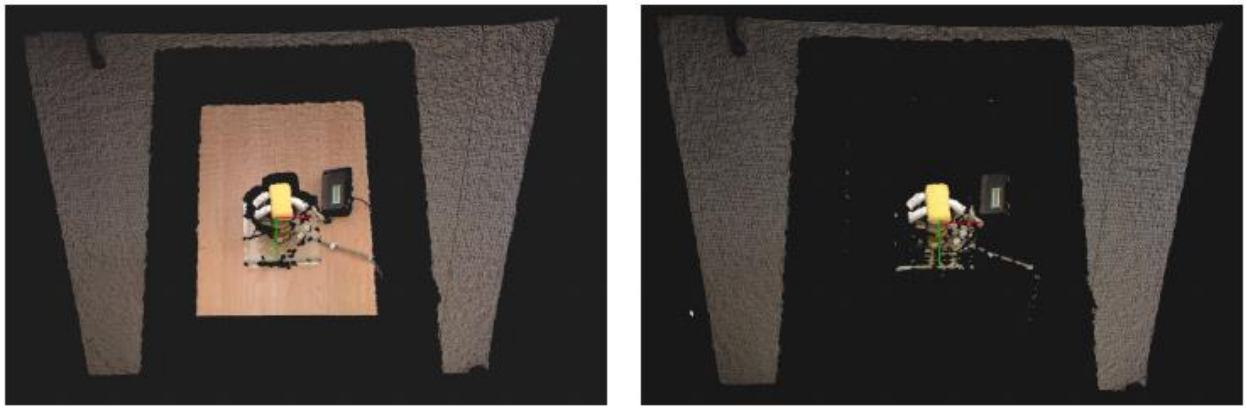
PC^*	point cloud after planar surface extraction

1: Retrieve S^* from PC based on Algorithm 3.1

2: Obtain the reduced point cloud after planar surface extraction $PC^* = PC - S^*$

3: **Return** PC^*

An example of the reduced point cloud, PC^*, obtained after removal of the table surface and the dominant background components seen in Figure 3.4a is shown in Figure 3.4b. It is noticeable that with this solution, the object of interest is not yet fully segmented from the background, as components closer (in spatial distance) to the object of interest remain as part of the reduced point cloud, PC^*. Further processing is therefore required to fully extract the cluster containing the object of interest.



(a) (b)

Figure 3.4: (a) Raw point cloud from Kinect, and (b) point cloud after table surface removal.

3.3.2 Object-of-Interest Cluster Extraction

The extraction of the cluster of the object of interest is necessary in order to initialize the fast level set method. The initialization is detailed in section 3.3.6.

An alternative to the use of this solution would be to use an edge detector to support the initialization of the level set method. However, it is complex to extract the edges of the object of interest from each frame due to the complicated setup. There are several objects lying on the table, including cables, the robotic hand and other objects that can hardly be removed from the field of view, which leads to the difficulty of extracting only the edges that belong to the object of interest among all the edges typically detected in a frame using classical edge detectors.

Once the planar background surface is extracted from the point cloud data, the next step consists of narrowing down the search area to the object of interest in the reduced point cloud, PC^*. Different from the general clustering methods found in the literature that aim at segmenting separate objects, the purpose of the procedure defined here is to extract one single cluster, which is the cluster containing the nearest elements to the given input point identified by the user (2D fixation point) in a Euclidean sense from the remaining point cloud from which the planar surface has been removed. The process is illustrated in Figure 3.5. This approach can also support the initialization process in general scenarios where the assumption of a planar surface dominating the background of the object of interest cannot be made, and as a consequence, the simplification process proposed in section 3.3.1 cannot be performed.

The solution capitalizes on a *k*-d tree structure and the use of the *k*-nearest neighbors search algorithm as well as the fixed-radius near neighbors search algorithm. Because the point cloud recuperated from the Kinect sensor is unstructured, and in order to find the neighbor of each point contained therein, a *k*-d tree data structure, as described in section 2.3.5, is created to support a faster nearest neighbors search and a more effective fixed-radius near neighbors search. Therefore, a *k*-nearest neighbors search is applied to identify the *k*-nearest points, inX, in the reduced point cloud, PC^*, to the estimated 3D fixation point, $\boldsymbol{p}_{3D}$.

Figure 3.5: Flowchart of single cluster of interest extraction.

The proposed algorithm for cluster extraction based on a given input point consists of finding the subdivisions and boundaries to allow the data to be grouped together based on a given measure of "proximity" [46]. The measure used in the current work is the Euclidean distance. In the Cartesian domain, given two points $\boldsymbol{pc}_1 = (pc_{1\_x}, pc_{1\_y}, pc_{1\_z})$ and $\boldsymbol{pc}_2 = (pc_{2\_x}, pc_{2\_y}, pc_{2\_z})$ in the 3-dimensional space, the Euclidean distance is calculated as: $||\boldsymbol{pc}_1 - \boldsymbol{pc}_2|| = \sqrt{(pc_{1\_x} - pc_{2\_x})^2 + (pc_{1\_y} - pc_{2\_y})^2 + (pc_{1\_z} - pc_{2\_z})^2}$. The given input point for the cluster extraction algorithm is the 3D fixation point, $\boldsymbol{p}_{3D}$, computed from its corresponding 2D fixation point using the 2D-3D mapping, as described in section 3.2.3. However, because an estimation based on the intrinsic parameters of the Kinect sensor is used to approximate the location of the 3D fixation point (Equation 3.1), and also due to the errors in depth that characterize the Kinect sensor, the computed 3D coordinates for the fixation point might not correspond to an existing 3D point in the point cloud. To tackle this issue, a *k*-nearest neighbors research is applied to identify the *k*-nearest points, inX, in the point cloud, PC^*, to the estimated 3D fixation point, $\boldsymbol{p}_{3D}$. An empirically chosen value of k=100 is used to accelerate the computation of the clustering approach, also guaranteeing that the cluster of the object of interest contains at least 100 points. The *k*-nearest neighbors search algorithm [48] is used to search inX with first k points set as follows:

Algorithm 3.4 K-nearest neighbors algorithm for searching inX with first k points set

---

Input:

PC^* point cloud after planar surface removal

k number of nearest neighbors

$\boldsymbol{p}_{3D}$ input point, the estimated 3D fixation point

Output:

inX set of k-nearest-neighbors to the input point

1: **function** ClusterExtraction(PC^*, $\boldsymbol{p}_{3D}$, k)

2: **if** PC^*.empty() **then**

3: **return** $\emptyset$

4: **else**

5: **for** every point $\boldsymbol{q}_{3D}$ in PC^* **do**

6: calculate the Euclidean distance between $\boldsymbol{p}_{3D}$ and the point $\boldsymbol{q}_{3D}$

7: sort the Euclidean distances in increasing order

8: take k items with lowest distances to inX

9: **end for**

10: **end if**

11: **end function**

Beginning with these 100 points, the neighborhood of each point is further researched for extra points located within a maximum distance, d_{th}. These points can be integrated in the object-of-interest cluster. The merging condition for these points to be considered in the cluster is the Euclidean radius. The resulting object cluster is therefore composed of the initial 100 points augmented by any other points discovered within the distance threshold, d_{th}. The latter distance is chosen empirically (its impact is described in section 4.2.2) and here set to 5mm. The proposed approach to extract the object-of-interest full cluster is summarized by the following algorithm:

Algorithm 3.5 3D object-of-interest cluster extraction algorithm

---

Input:

$PC^* = \{\boldsymbol{pc}_i\}$	point cloud after planar surface removal
inX	set of k-nearest-neighbors to the estimated 3D fixation point, $\boldsymbol{p}_{3D}$ (with k=100)
d_{th}	spatial distance threshold for a point to also be considered as a neighbor (set to 5 mm)

Output:

$cluster$	object cluster

1: **function** ClusterExtraction (PC^*)

2: **if** PC^*. empty() **then**

3: **return** $\emptyset$

4: **else**

5: Build *3*-d tree representation of PC^*

6: Set up priority queue Q of the points, which need to be checked, $Q \leftarrow inX$

7: **for** every $\boldsymbol{pc}_i \subset Q$ **do**

8: **if** all the points in Q have been processed **then**

9: $cluster \leftarrow Q$

10: **else**

11: search the neighbors set P_i^k of $\boldsymbol{pc}_i$ in a sphere with radius $r < d_{th}$

12: **for** every $\boldsymbol{pc}_i^k \subset P_i^k$ **do**

13: **if** $\boldsymbol{pc}_i^k$ has not been processed **then**

14: $Q \leftarrow \boldsymbol{pc}_i^k$

15: **end if**

16: **end for**

17: **end if**

18: **end for**

19: **end if**

20: **end function**

The proposed algorithm leads to a cluster containing a single object of interest, as illustrated by the resulting point cloud shown in Figure 3.6 that is extracted from the reduced data set shown in Figure 3.4b. The red and green axes visible in Figure 3.6 represent the local reference frame, an intrinsic feature of the Point Cloud Library (PCL) [76] that is used to visualize 3D point clouds (red represent the x axis and green represents the y axis). Algorithm 3.5 is able to segment a cluster with different sizes in the point cloud, depending on how large the object of interest is with respect to the field of view of the Kinect sensor. This provides increased robustness to the relative location of the sensor with respect to the object of interest, as will be demonstrated in section 4.3.3. Furthermore, inX, one of the inputs of Algorithm 3.5, is not limited to a set of points; it can also be a single point.

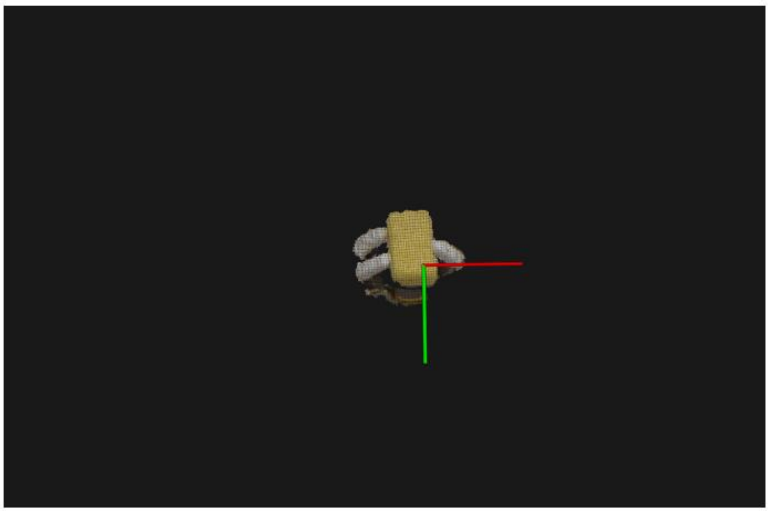


Figure 3.6: Object-of-interest cluster extracted from the point cloud shown in Figure 3.4(b).

3.3.3 3D-2D Mapping

Once the object-of-interest cluster in 3D is identified, the corresponding cluster in the 2D color image is recovered with the 3D-2D mapping in order to initialize the level set contour detection and tracking method. The 3D-2D mapping is used for projecting back the 3D coordinates to the 2D coordinates based on the intrinsic properties of the Kinect sensor. The equations for the mapping of a 3D point (X, Y, Z) to a 2D pixel (x, y) are [15]:

$$x = X \times f_{x\_rgb}/Z + c_{x\_rgb} \tag{3.4a}$$

$$y = Y \times f_{y\_rgb}/Z + c_{y\_rgb} \tag{3.4b}$$

where

- f_{x_rgb} and f_{y_rgb} are the focal lengths of the RGB camera along x and y axis respectively in pixel units;

- c_{x_rgb} and c_{y_rgb} represent the coordinates of the principal point of the RGB camera in pixel units.

The group of 2D pixels corresponding to the 3D points in the selected cluster is then employed as initialization for the level set method to be applied in the 2D color space. In particular, the bounding window with which the level set is initialized represents the bounding box of the projected 2D pixels (the farthest left, right, up and down locations) plus a border of 50 pixels around it. An example of the definition of the bounding window based on the back-projected 2D pixels is shown in Figure 3.7. The impact of the bounding window size is described in section 4.2.3. The yellow area represents the cluster of back-projected 2D pixels identified using the 3D-2D mapping of Equation 3.4; the dotted line denotes the bounding box that is a rectangle determined by the farthest left, right, up and down pixels; the solid line bounding window is the bounding box enlarged by 50 pixels in all directions. This tolerance of 50 pixels is included to increase robustness in cases where the object shifts or turns during manipulation.

This initialization process also contributes to the acceleration of the contour detection and tracking since the initialization area is close to the real object of interest.

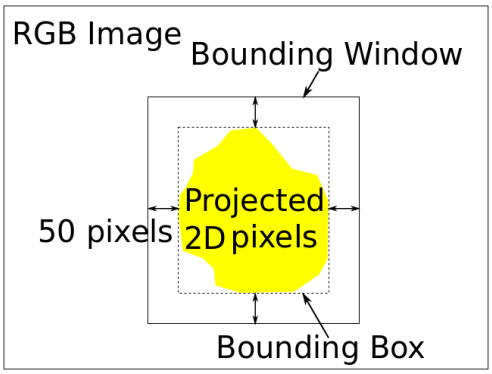


Figure 3.7: An example of the bounding window, bounding box and projected 2D pixels.

The bounding window surrounding the object of interest is extracted from the input RGB image sequence and transformed into the log-polar domain for further processing.

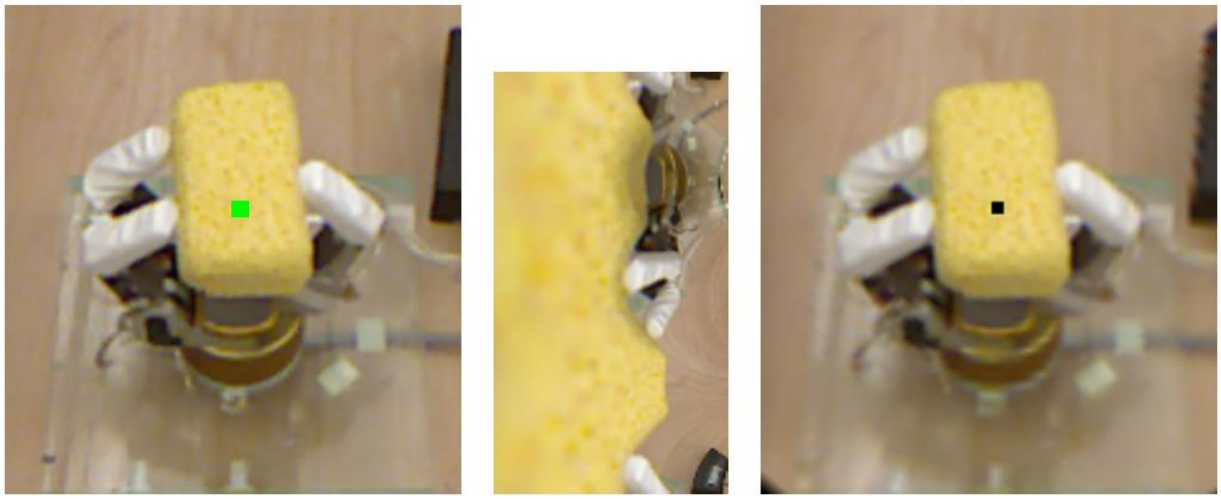
3.3.4 Log-Polar Mapping of Color Object-of-Interest Cluster

In order to segment and track the object-of-interest contour, an original formulation of the fast level set method in the log-polar domain is proposed. As presented in section 2.3.4, the log-polar transform simulates some features of the human visual model. A significant advantage of using this transformation is that the object of interest gets to fill a relatively large area of the corresponding log-polar image compared to the remaining background area. The log-polar mapping is applied over the bounding window obtained in section 3.3.3.

Figure 3.8a shows an example of the bounding window extracted from a Cartesian image, with the user-selected 2D fixation point shown in green, while Figure 3.8b represents its corresponding log-polar (cortical) map.

As it can be observed in Figure 3.8b, provided that the fixation point is selected relatively centered over the object of interest, the percentage of space occupied by the object of interest in the log-polar map is proportionally larger than in the Cartesian image. As the log-polar map is centered on the fixation point, the object of interest occupies a large part on the left side of the image, as the leftmost columns are the ones that are closer to the 2D fixation point. This results in a suitable mapping at a higher resolution over the region of interest. Moreover, the use of the log-polar mapping simplifies the search for contour points along a somewhat vertical direction for objects that are generally symmetrical, as illustrated in Figure 3.8b. Finally, the log-polar mapping typically reduces the size of the representation with respect to the original image (e.g., from 187 × 200 pixels to 93 × 167 pixels in the case shown in Figure 3.8), based on the resolution of the log-polar mapping that optimizes the pixel aspect ratio (as defined in section 2.3.4). This further contributes to accelerate the contour retrieval procedure. The influence of the selection of the fixation point and the choice of the parameters is discussed in section 4.2.4.

Later on, the fast level set method detailed in section 3.3.6 will be applied on one of the automatically selected color components (section 3.3.5) of the resulting log-polar image to extract the contour of the object of interest. As a result, the contour will also be initially retrieved in the log-polar domain. An inverse log-polar mapping, resulting in a retinal image, as shown in Figure 3.8c, is therefore required to represent the contour in the Cartesian domain, for evaluation and for supporting the classification phase that will be detailed in section 3.4.



(a) (b) (c)

Figure 3.8: (a) Original Cartesian image (187×200); (b) log-polar (cortical) image (93×167) obtained by the log-polar mapping from (a); and (c) retinal image (187 × 200) obtained by the inverse mapping from (b).

The application of the log-polar transformation on the initial Cartesian image and the inverse log-polar transformation applied on the contour segmented in the cortical domain affects the sharpness of the reprojection on the Cartesian domain to generate the retinal image, as can be seen in Figure 3.8. This is analogous to the inner workings of the human eye, where the central zone around the fixation point has high resolution and narrow field of view, while the periphery has a large field of view and lower resolution [77]. As the results of this thesis demonstrate, the sharpness of the contour is not an issue when classifying material elasticity. However, the results in this thesis will be presented by superimposing the contours onto the original Cartesian image rather than on the degraded retinal image. For the sake of performing a complete evaluation, and further justifying the use of the log-polar transformation, section 4.2.7 will report on an experimental comparison between contours detected from the original Cartesian image and contours detected from the transformed image into the cortical image.

3.3.5 Color Component Selection

The level set method segments an image based on a closed contour determined from transitions in the color information. In this subsection, the proposed approach for the color space selection is described. The goal is to create an automatic color component selection system that does not require user intervention.

Several color encoding schemes exist to represent the color components of an image. The color image acquired by the Kinect sensor is using the RGB (red, green, and blue) color coding. The RGB color space is commonly used for transmission, representation, and storage of color images on both analog and digital devices [78]. However, the RGB color space is often considered as a non-optimal choice for machine vision because any change in the lightness or brightness reflects on the three color components. Section 4.2.6 will illustrate that the RGB color space is not the optimal choice for detecting the contour of the object of interest.

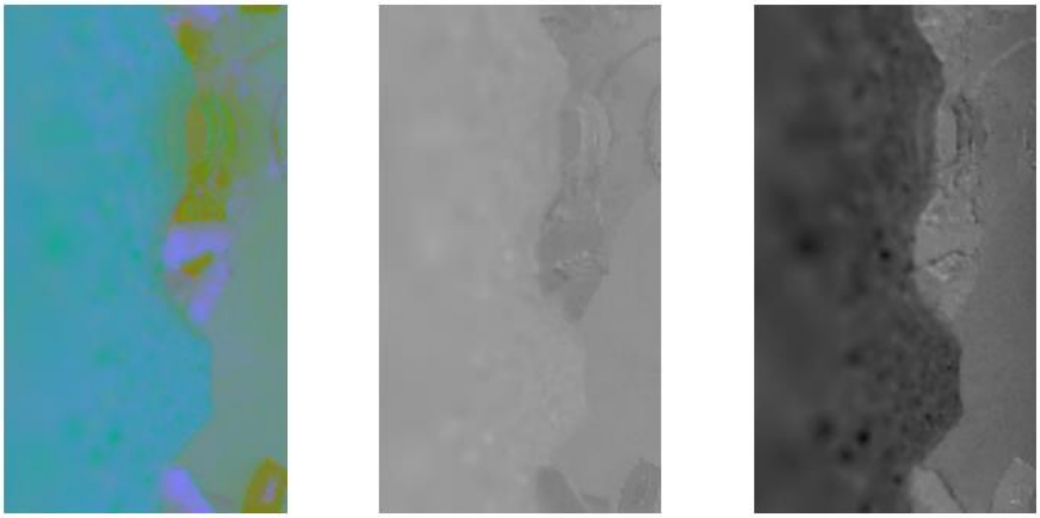
HSV and HSL are two well-known cylindrical-coordinate equivalent representations of the RGB color model [78]. HSV and HSL are both commonly used in computer graphics, especially in image editing. In the HSV color space, H, S and V denote hue, saturation, and value. In the HSL color space, H, S, and L denote hue, saturation, and lightness. Besides the V axis or the L axis, the HS plane encodes the property of color. However, the HS plane uses polar coordinates which lead to difficulties in splitting the H and S coordinates.

In this thesis, the YUV color space is preferred, where Y is the luminance of a color, and U and V represent the two chromatic components. The simplified UV plane offers a more compact and therefore faster solution to process color information. A transformation to the YUV color space is performed, and either the U or the V component is selected as the feature for contour detection.

The YUV color space can be derived directly from the RGB color space, as shown in Equation 3.5 [78].

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.14713 & -0.28886 & 0.436 \\ 0.615 & -0.51499 & -0.10001 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{3.5}$$

The following figures exhibit the YUV-coded cortical image derived from the log-polar image in Figure 3.8b, along with the corresponding U and V components respectively.



(a) (b) (c)

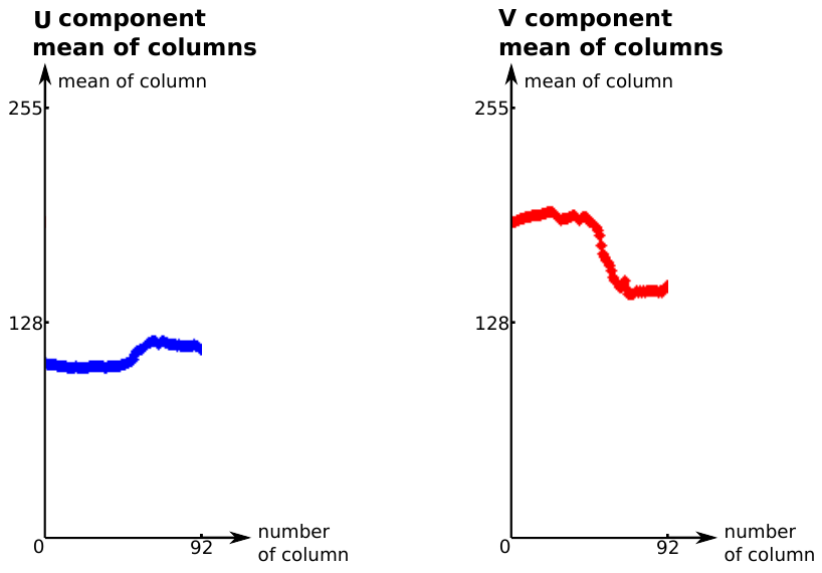
Figure 3.9: (a) YUV-coded cortical image of Figure 3.8b; (b) U component of (a); and (c) V component of (a).

The color space selection is applied to the YUV coded log-polar image obtained previously. The proposed algorithm for the color space selection computes initially the mean value of each column in the U component and the V component, respectively. Then, standard deviation values of all the mean values are calculated in the U and V components, respectively, over the entire log-polar map:

$$SD = \sqrt{\frac{1}{H}\sum_{h=1}^{H}(cl_h - f)^2}\,, \text{where } f = \frac{1}{H}\sum_{h=1}^{H} cl_h \tag{3.6}$$

where cl_h represents the mean value of the U or the V component for each column, H is the total number of columns, and SD denotes the standard deviation value [79]. The standard deviation is the most widely used measure of the spread of a set of data values. A large standard deviation value shows that the mean values of the columns are quite different, while a small standard deviation value shows that the mean values of the columns have a tendency to be similar. In order to favor more contrasting information in the log-polar map to improve segmentation accuracy, the color component (U or V) with the largest standard deviation among its columns is selected as the feature on which the fast level set method will proceed. The privileged color channel component is hence automatically selected for each testing scenario among these two possibilities. The fact that the luminance component (i.e. the Y component from the YUV color code) is not considered in the processing reduces the dependence of the solution on the local intensity of colors due to different illumination conditions and shading effects.

Figure 3.10 shows the mean value of each column of the U component image (Figure 3.9b) and the V component image (Figure 3.9c), respectively. It is obvious that the mean values of the columns of the U component tend to be similar, while the ones of the V component have a wider distribution. In this case, it is found that the standard deviation value of the U component is lower than the standard deviation value of the V component, therefore the V component is selected as the feature for the fast level set (section 3.3.6) to be applied on this image.



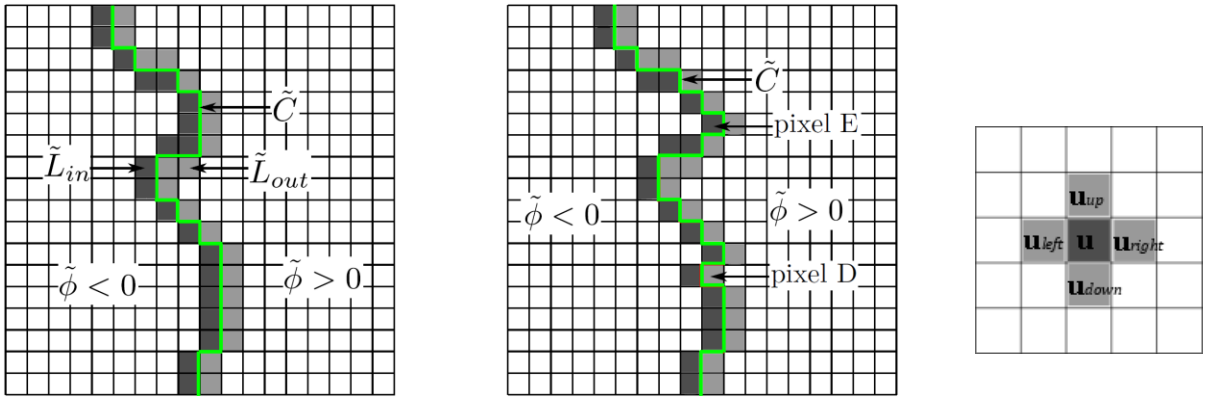
(a) (b)

Figure 3.10: Mean value of each column of: (a) Figure 3.9b; (b) Figure 3.9c.

3.3.6 Fast Level Set Method in Log-Polar Domain for Contour Detection and Tracking

The method that is proposed to detect and track an object-of-interest contour builds on the fast implementation of level sets introduced in [36][37], while originally combining it with some concepts inspired from the works of [42] and [43] that find the contours under the guidance of a fixation point that maps the image from the Cartesian domain to the log-polar domain (section 3.3.4), and being based on the YUV color coding (section 3.3.5). To the best of our knowledge, the combination of these strategies has not been experimented before, and represents an original contribution from the current thesis to tackle the challenging problem of closely monitoring the physical behavior of deformable objects made of material with considerably different characteristics. In this section, we present a novel cortical fast level set method which detects the contour of an object, achieves its segmentation, and can further track the evolution of the contour as the object deforms during manipulation by a robotic hand.

Unlike in the classical level set method implementation described in section 2.3.2, when considering images mapped in the log-polar domain, the curve $\tilde{C}$, represented as the zero level set of the level set method, is not an enclosing contour of the object, but rather an open boundary line in the vertical direction, as was demonstrated in sections 3.3.4 and 3.3.5. This situation is depicted in Figure 3.11a.



(a) (b) (c)

Figure 3.11: (a) Representation of the curve $\tilde{C}$ and the two lists of neighboring pixels $\tilde{L}_{in}$ and $\tilde{L}_{out}$ in the cortical image; (b) the motion of switching pixels of $\tilde{L}_{in}$ and $\tilde{L}_{out}$; and (c) representation of the 4-connected discrete neighborhood of a pixel $\mathbf{u}$. (Each grid cell denotes a pixel of the cortical image.)

Following the idea of the fast level set implementation, detailed in section 2.3.2.2, the fast level set method proposed here is adapted to operate in the log-polar domain, or cortical domain. It employs two neighboring lists, respectively denoted by $\tilde{L}_{in}$ and $\tilde{L}_{out}$, and representing the left and right neighboring pixels of the curve (Figure 3.11a) separating the region of interest (leftmost) from the background (rightmost). They are defined as follows [36]:

$$\tilde{L}_{out} = \left\{ \mathbf{u} \middle| \tilde{\phi}(\mathbf{u}) > 0 \text{ and } \exists \mathbf{v} \in \tilde{N}_4(\mathbf{u}) \text{ such that } \tilde{\phi}(\mathbf{v}) < 0 \right\} \quad (3.7a)$$

$$\tilde{L}_{in} = \left\{ \mathbf{u} \middle| \tilde{\phi}(\mathbf{u}) < 0 \text{ and } \exists \mathbf{v} \in \tilde{N}_4(\mathbf{u}) \text{ such that } \tilde{\phi}(\mathbf{v}) > 0 \right\} \quad (3.7b)$$

where $\tilde{N}_4(\mathbf{u})$ still represents a 4-connected discrete neighborhood of a pixel $\mathbf{u}$, $\mathbf{u} = (u, v)$ in a two-dimensional cortical image. $\tilde{N}_4(\mathbf{u})$ contains the four neighboring pixels of $\mathbf{u}$, that is $\mathbf{u}_{up}$, $\mathbf{u}_{down}$, $\mathbf{u}_{left}$, $\mathbf{u}_{right}$ as illustrated in Figure 3.11c. The use of a 4-neighbor mapping is inspired from the original work of Shi and Karl [36][37]. A fast level set function, $\tilde{\phi}$, is represented in Equation 3.8 [36][37]. This function defines that the values of pixels in the left region delimited by the curve are negative, while values of pixels in the right region are positive. In this manner, the object, initially located by the 2D fixation point, is distinguished from the background.

$$\tilde{\phi}(\mathbf{u}) = \begin{cases} 3, & \text{if } \mathbf{u} \text{ is an exterior pixel;} \\ 1, & \text{if } \mathbf{u} \text{ is in } \tilde{L}_{out}; \\ -1, & \text{if } \mathbf{u} \text{ is in } \tilde{L}_{in}; \\ -3, & \text{if } \mathbf{u} \text{ is an interior pixel.} \end{cases} \tag{3.8}$$

The interior pixels denote the pixels on the left side of $\tilde{C}$ but not in $\tilde{L}_{in}$, and the exterior pixels denote the pixels on the right side of $\tilde{C}$ but not in $\tilde{L}_{out}$.

In Figure 3.11a and Figure 3.11b, the green line represents the curve, $\tilde{C}$, which splits the cortical image into two parts: the object of interest on its left and the background to its right. The list $\tilde{L}_{in}$ contains the pixels located on the left side of the curve, shown in dark gray, while the list $\tilde{L}_{out}$ contains the pixels located on the right side of the curve, shown in light gray. An example is used to illustrate the movement of switching pixels from $\tilde{L}_{in}$ to $\tilde{L}_{out}$ and vice versa during the contour tracking of the object. Comparing Figure 3.11b to Figure 3.11a, the curve $\tilde{C}$ moves right at pixel E and moves left at pixel D. This behavior is represented as switching the pixel E from $\tilde{L}_{out}$ to $\tilde{L}_{in}$, and switching the pixel D from $\tilde{L}_{in}$ to $\tilde{L}_{out}$.

The functions for switching pixels between $\tilde{L}_{in}$ and $\tilde{L}_{out}$, *cortical_switch_in()* and *cortical_switch_out(),* follow the same rules and operate in the same way as the *switch_in()* and *switch_out()* from the classical fast level set method, respectively, presented in section 2.3.2.2. The only difference is that *cortical_switch_in()* and *cortical_switch_out()* work on the cortical (log-polar) image.

The procedure used by *cortical_switch_in()* is described as follows:

---

cortical_switch_in ($\mathbf{u}$)

1: Remove $\mathbf{u}$ from $\tilde{L}_{out}$ and add it to $\tilde{L}_{in}$. Set $\tilde{\phi}(\mathbf{u}) = -1$.

2: For $\forall\, \mathbf{v} \in \tilde{N}_4(\mathbf{u})$ with $\tilde{\phi}(\mathbf{v}) = 3$, add $\mathbf{v}$ to $\tilde{L}_{out}$. Set $\tilde{\phi}(\mathbf{v}) = 1$.

The function *cortical_switch_in()* is employed when the curve moves right at a pixel $\mathbf{u} \in \tilde{L}_{out}$. In other words, this pixel $\mathbf{u}$ is switched from $\tilde{L}_{out}$ to $\tilde{L}_{in}$ and all its neighboring exterior pixels are added to $\tilde{L}_{out}$. Similarly, the function *cortical_switch_out()* is employed when a curve moves left at a pixel $\mathbf{u} \in \tilde{L}_{in}$.

The function *cortical_switch_out()* is summarized as follows:

---

cortical_switch_out (**u**)

1: Remove **u** from $\tilde{L}_{in}$ and add it to $\tilde{L}_{out}$. Set $\tilde{\phi}(\mathbf{u}) = 1$.

2: For $\forall\ \mathbf{v} \in \tilde{N}_4(\mathbf{u})$ with $\tilde{\phi}(\mathbf{v}) = -3$, add **v** to $\tilde{L}_{in}$. Set $\tilde{\phi}(\mathbf{v}) = -1$.

The evolution of the curve $\tilde{C}$ depends on the speed function [36]. As discussed in section 2.3.2.2, the speed function is composed here as well of two parts: the data-dependent speed function, $\tilde{F}_{ext}$, and the curve smoothness regularization speed function, $\tilde{F}_{in}$. Furthermore, the data-dependent speed function is improved as in Equation 3.9, and the curve smoothness regularization speed function is simplified as the Gaussian filter for the curve on the image. Here, the data-dependent speed function, $\tilde{F}_{ext}$, is obtained from the same equation as Equation 2.10.

$$\tilde{F}_{ext} = \begin{cases} -\lambda_1(I(u,v) - c_l)^2 + k\lambda_1(I(u,v) - c_r)^2, & if \quad \left|\tilde{F}_{ext}\right| < thd \\ -(I(u,v) - c_l)^2 + (I(u,v) - c_r)^2, & \text{otherwise} \end{cases} \tag{3.9}$$

where $I(u,v)$ is the color information at pixel (u,v), coming from either the U or V component, and thd denotes the threshold that weighs the curve smoothness regularization speed function and data dependent speed function. The parameters c_l and c_r, are the mean intensities of the images on the left and right side of the curve, $\tilde{C}$, as defined by [35]:

$$c_l\left(\tilde{\phi}\right) = \frac{\int_\Omega I(u,v)H\left(\tilde{\phi}\right)dudv}{\int_\Omega H\left(\tilde{\phi}\right)dudv} \tag{3.10a}$$

$$c_r\left(\tilde{\phi}\right) = \frac{\int_\Omega I(u,v)\left(1-H\left(\tilde{\phi}\right)\right)dudv}{\int_\Omega \left(1-H\left(\tilde{\phi}\right)\right)dudv} \tag{3.10b}$$

where $H\left(\tilde{\phi}\right)$ is the Heaviside function such that $H\left(\tilde{\phi}\right) = \begin{cases} 1, & \tilde{\phi} < 0 \\ 0, & \tilde{\phi} > 0 \end{cases}$.

The curve smoothness regularization speed function, $\tilde{F}_{in}$, driven from the same idea of F_{in} in Equation 2.11, is approximated by:

$$\tilde{F}_{in} = \mu\nabla \cdot \left(\frac{\nabla\tilde{\phi}}{|\nabla\tilde{\phi}|}\right) = \mu\kappa \tag{3.11}$$

As in the classical level set method, κ is the curvature of the evolving curve, μ is a regularization parameter, and ∇ represents the derivative function. For the same reason as explained in section

2.3.2.2, the curve smoothness regularization speed function, $\tilde{F}_{in}$, is therefore simplified as a Gaussian filter, G. Furthermore, the Gaussian filter is employed only on the pixels of $\tilde{L}_{out}$ and $\tilde{L}_{in}$ in order to smooth the zero level set. Algorithm 3.6 presents the fast level set method adapted to operate in the log-polar domain.

Algorithm 3.6 Log-polar domain fast level set algorithm

Input:

N_a the number of iterations of the data dependent speed function

N_g the size of the Gaussian filter

1: Initialize the array $\tilde{\phi}$, $\tilde{F}_{ext}$, and the two lists $\tilde{L}_{in}$ and $\tilde{L}_{out}$.

2: **for** $i = 1 : N_a$ **do** ***// Cycle One***

3: Compute $\tilde{F}_{ext}$, for pixels in $\tilde{L}_{in}$ and $\tilde{L}_{out}$;

4: For every cortical pixel $\mathbf{u} \in \tilde{L}_{out}$, apply *cortical_switch_in(***u***)* if $\tilde{F}_{ext}(\mathbf{u}) > 0$;

5: For every cortical pixel $\mathbf{u} \in \tilde{L}_{in}$, remove $\mathbf{u}$ from $\tilde{L}_{in}$ and set $\tilde{\phi}(\mathbf{u}) = -3$ if $\tilde{\phi}(\mathbf{v}) < 0$ for $\forall \mathbf{v} \in \tilde{N}_4(\mathbf{u})$;

6: For every cortical pixel $\mathbf{u} \in \tilde{L}_{in}$, apply *cortical_switch_out(***u***)* if $\tilde{F}_{ext}(\mathbf{u}) < 0$;

7: For every cortical pixel $\mathbf{u} \in \tilde{L}_{out}$, remove $\mathbf{u}$ from $\tilde{L}_{out}$ and set $\tilde{\phi}(\mathbf{u}) = 3$ if $\tilde{\phi}(\mathbf{v}) > 0$ for $\forall \mathbf{v} \in \tilde{N}_4(\mathbf{u})$;

8: Check the *Stop Conditions* and if satisfied, go to ***Cycle Two***; else continue this cycle.

9: **end for**

10: **for** $j = 1 : N_g$ **do** ***//Cycle Two***

11: For every pixel $\mathbf{u} \in \tilde{L}_{out}$, compute $G \otimes \tilde{\phi}(\mathbf{u})$. Apply *cortical_switch_in(***u***)* if $G \otimes \tilde{\phi}(\mathbf{u}) < 0$;

12: For every pixel $\mathbf{u} \in \tilde{L}_{in}$, remove $\mathbf{u}$ from $\tilde{L}_{in}$ and set $\tilde{\phi}(\mathbf{u})$ = −3 if $\tilde{\phi}(\mathbf{v}) < 0$ for $\forall \mathbf{v} \in \tilde{N}_4(\mathbf{u})$;

13: For every pixel $\mathbf{u} \in \tilde{L}_{in}$, compute $G \otimes \tilde{\phi}(\mathbf{u})$. Apply *cortical_switch_out(***u***)* if $G \otimes \tilde{\phi}(\mathbf{u}) > 0$;

14: For every pixel $\mathbf{u} \in \tilde{L}_{out}$, remove $\mathbf{u}$ from $\tilde{L}_{out}$ and set $\tilde{\phi}(\mathbf{u})$ = 3 if $\tilde{\phi}(\mathbf{v}) > 0$ for $\forall \mathbf{v} \in \tilde{N}_4(\mathbf{u})$;

15: **end for**

16: If one of *Stop Conditions* is satisfied in ***Cycle One***, terminate the algorithm; otherwise, go back to ***Cycle One***.

For the initialization in line 1 of Algorithm 3.6, $\tilde{F}_{ext}$ is initialized to 0. $\tilde{\phi}$ is initialized as follows. First of all, the group of 2D pixels corresponding to the 3D points in the identified object-of-interest cluster extracted in section 3.3.2 are mapped to the cortical image and the corresponding pixels are labelled as (-3). All other pixels within the bounding window defined in section 3.3.3 are also mapped to the cortical image and the corresponding pixels are labelled as (3). An example is shown in Figure 3.12a, where the yellow area represents the projected 2D pixels and the rest of the bounding window is the background. After applying the log-polar transform on the bounding window, the projected 2D pixels represented by the pixels forming the yellow area are all mapped on the left part of the cortical image (Figure 3.12b).

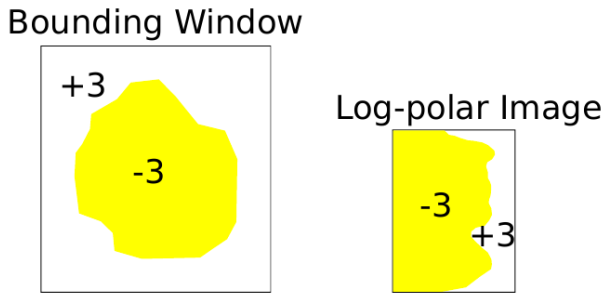


Figure 3.12: (a) Labelled bounding window in original Cartesian image; and (b) corresponding labelled pixels in the cortical image.

The lists $\tilde{L}_{out}$ and $\tilde{L}_{in}$ defined in Equation 3.7 are obtained on the log-polar transformed level set function based on a $\tilde{N}_4$ neighbour search. Conversely, for all subsequent frames, the group of pixels that are within the contour retrieved in the previous frame are labelled as (-3), and the ones outside the latest contour as (3). The lists $\tilde{L}_{out}$ and $\tilde{L}_{in}$ are updated accordingly, using Equation 3.7, as part the initialization step. This permits a continuous tracking over successive frames in the sequence with the level sets method applied in the log-polar domain. The log-polar domain fast level set algorithm (Algorithm 3.6) stops whenever either of the following two conditions is satisfied.

Stop Conditions

1: The speed at every neighboring pixel satisfies:

$$\tilde{F}_{ext}(\mathbf{u}) \leq 0, \forall \mathbf{u} \in \tilde{L}_{out}$$
$$\tilde{F}_{ext}(\mathbf{u}) \geq 0, \forall \mathbf{u} \in \tilde{L}_{in}$$

2: A pre-specified maximum number of iterations, N_a, is reached

---

As in the original level set algorithm presented in section 2.3.2.2, N_a, is the number of iterations of the data dependent speed, $\tilde{F}_{ext}$, and N_g is the size of the Gaussian filter, G, applied over the contour in order to smooth it. The parameter N_a is set higher than N_g to prevent weakening the sharp corners of the contour. In this work, the two parameters are set empirically such that N_a = 80 and N_g= 3. A

comparison between contours extracted with and without the Gaussian filter will be presented in section 4.2.8, as an evaluation of the influence of this filter.

The fast level set method adapted for the log-polar mapping of data can be used in two different ways. On one hand, it can be set such that it focuses more on robustness, in which case the level set function is initialized from the extracted cluster at each frame over the sequence acquired with the Kinect sensor. In this case, the method performs only iterative contour detection. On the other hand, it can be set such that it focuses more on speed, in which case the level set function is initialized at each iteration from the contour obtained from the previous frame. The method then achieves contour tracking on the object of interest.

3.3.6.1 Robustness-Focused Contour Detection

Robustness-focused contour detection retrieves the contour of the object of interest at each frame from a sequence of RGB-D images collected from the Kinect sensor. More specifically, it employs the point cloud from each frame for the initialization of the fast level set method. At each frame, the system is initialized with the bounding window (as shown in Figure 3.7), which is acquired from the re-projected 3D object cluster resulting from the background removal procedure. That means that all steps described in section 3.3.3 are re-executed on every frame. Only the fixation point selection (section 3.2.2) is not performed each time, and the location for the object of interest first pointed by the operator is reused at every iteration. This robustness-focus contour detection strategy supports the migration of the contour detector to an alternative object that would appear at the location of the static fixation point over the sequence. It also deals very well with temporary occlusions of the object of interest as it can recover the initial object when it reappears in the images, due to its full re-initialization at every iteration. This represents an interesting feature of the proposed solution and it makes it relatively robust to a wide range of realistic operational scenarios. However, the main drawback of this robust contour detection approach is the fact that time is spent for processing the background removal, extracting the object-of-interest cluster and computing the bounding window at each iteration. It is therefore significantly slower, as will be demonstrated in section 4.4.

3.3.6.2 Speed-Focused Contour Tracking

Speed-focused contour tracking is an approach focusing on the speed of tracking the contour of the object of interest as it evolves under the robotic manipulation. It is similar to the robustness-focused contour detection approach in that data is acquired from a sequence of frames of RGB-D images collected from the Kinect sensor. However, this approach does not reinitialize each frame based on the bounding window extracted from the object-of-interest point cloud. Only the first frame is initialized using the bounding window that is obtained from the 3D object cluster after background removal. For each following frame, the latest level set function obtained is used as the initialization level set function for the current frame to further track the contour of the object. This approach for tracking provides a continuous contour detection at each frame and is overall significantly faster. However, it tends to lose track of the contour in case of discontinuities, and is not able to quickly recover when occlusions of the object of interest appear in the scene. An extensive comparison between the robustness-focused

contour detection and the speed-focused contour tracking forms of the proposed method is performed in section 4.4.

3.4 Material Behavior-Based Classification

The classification of objects in the various categories considered depends on the deformation behavior exhibited by the object during the manipulation with a robotic hand. The contour of the object is considered here as the most intuitive expression of this behavior. The object contour is therefore employed to categorize the objects into four types of materials: elastic, plastic, elasto-plastic, or rigid. The proposed procedure can also serve to distinguish between the deformation stages for objects whose material adopts different properties according to the magnitude of force applied on them (e.g. shape memory material, or breakable objects).

In order to characterize objects and classify them into different categories based on the material they are made of, and therefore recognize their deformation behavior, an automated measurement procedure is designed that uses a three-finger robotic Barrett hand (for further details, see section 4.1.1). The hand applies a series of forces over predefined points distributed all around the object to be characterized, while the contour of the object is visually monitored with the solution described in the previous sections. The object contour, denoted as C, obtained at different deformation stages using Algorithm 3.6 applied in the log-polar domain is remapped into the Cartesian domain through an inverse log-polar mapping prior to this stage. The contour of the manipulated object is monitored during the procedure and three specific contour samples are extracted: the initial contour that is collected at the beginning of the characterization procedure before any force is applied, the deformable contour that is obtained under the largest deformation that is observed, and the final contour that is obtained after the interaction is over and any applied force is removed. The following section details the step-by-step procedure to achieve the classification.

3.4.1 Object Deformation Categories

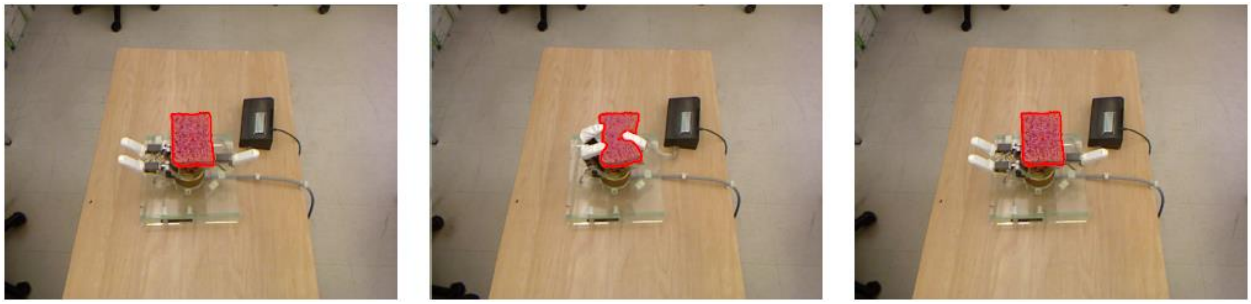
In the context of this work, deformation refers to the change of material shape or size that is caused by a sufficient loading applied on the object.

An elastic deformation is a temporary shape change, such that the object returns to its original shape once the interaction between the robotic hand and the object stops and the force is released. The typical shape changes for such an object are shown in Figure 3.13.

A plastic deformation is a permanent shape change in which the object is deformed into a new shape (Figure 3.14b) when sufficient force is applied. When the grasping force is removed, the object shape (Figure 3.14c) remains the same as when the maximum force was applied on the object (Figure 3.14b) [80].

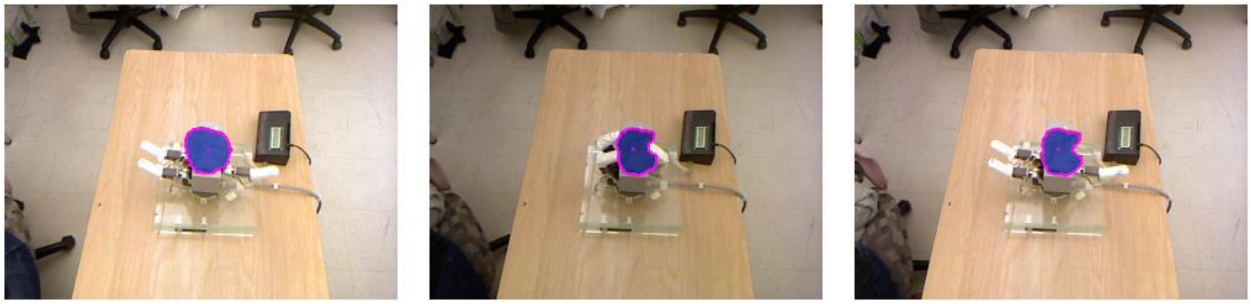
Elasto-plastic objects exhibit a combination of elastic and plastic properties [80]. The shape of objects made of such material is partially restored from the deformation after the removal of the grasping force. Figure 3.15 illustrates the property of an elasto-plastic object on which the deformation is partially, but

not totally, gone when the external force is released. As a result, three different shapes (before force is applied, under the force causing the largest deformation, and after force is removed) are observed as a result of the manipulation process.



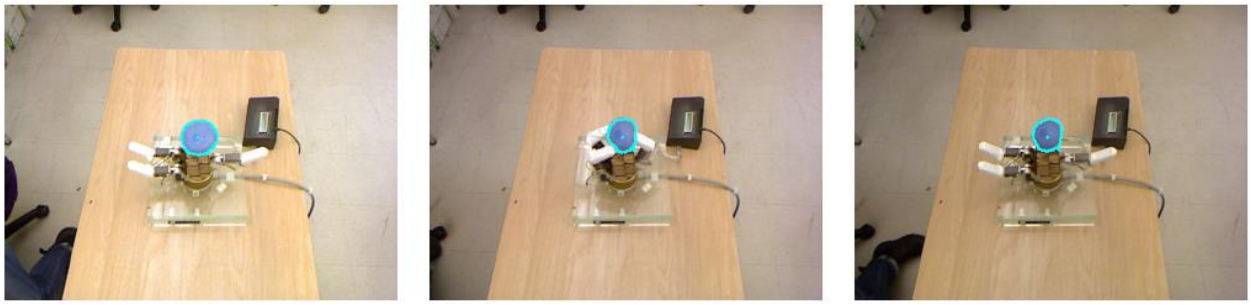
(a) (b) (c)

Figure 3.13: Elastic object: (a) before force is applied; (b) under the largest deformation; and (c) after force is removed.



(a) (b) (c)

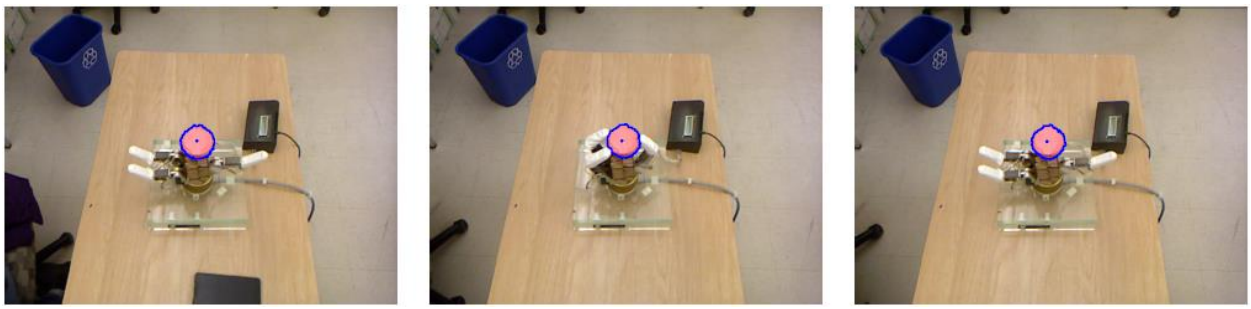
Figure 3.14: Plastic object: (a) before force is applied; (b) under the largest deformation; and (c) after force is removed.



(a) (b) (c)

Figure 3.15: Elasto-plastic object: (a) before force is applied; (b) under the largest deformation; and (c) after force is removed.

The shape of a rigid body remains in general the same regardless the external forces exerted on it. Typically, it can be considered that no deformation occurs on the rigid body. Therefore the shape remains the same in Figure 3.16. Special cases where a rigid object would break under the force applied by the fingertips of the robotic hand may occur. Such a case could be detected by the loss of contour tracking capabilities. However it is not taken into consideration in this work.



(a) (b) (c)

Figure 3.16: Rigid object: (a) before force is applied; (b) under the largest deformation; and (c) after force is removed.

The three representative contours (i.e. before any force is applied, when the largest deformation occurs, and after the interaction is over) are extracted from the tracking sequence for each of the object under study. In order to detect if the object is elastic, the final deformation contour is compared to the initial deformation contour as well as to the contour obtained when the largest deformation takes place. If the initial and the final contours are almost identical, within a certain tolerated noise margin, and different from the contour under the largest deformation, the object is classified as elastic. This situation is illustrated in the row "elastic material or elastic deformation stage" in Figure 3.17. The comparison between the initial, largest deformation and final object contours is also exploited to detect plastic and elasto-plastic deformations. If these three contours are different, beyond a predefined tolerance of a few pixels in order to cope with the noise in the measurements, it means that either a plastic or an elasto-plastic behavior occurred. The differentiation between the plastic and elasto-plastic behaviors is achieved by comparing the final deformation contour with the contour exhibiting the largest deformation. Given that an object made of plastic material memorizes the largest deformation experienced, if the final contour and the contour under largest deformation are almost identical, as shown in the row of "plastic material or plastic deformation stage" in Figure 3.17, it is considered that a plastic deformation occurred. On the other hand, an elasto-plastic material has the property to partially, but not totally, recover from the deformation when the external force is released. This property is recognized by comparing again the final contour with the contour under largest deformation. If they are significantly different, as shown in the row "elasto-plastic material or elasto-plastic deformation stage" in Figure 3.17, but not identical to the initial contour, then the material is considered as one exhibiting elasto-plastic properties. Finally, if all three contours are identical, as illustrated in the row "rigid material or rigid deformation stage" in Figure 3.17, the object is considered rigid, as no deformation is perceived by the contour tracking system under any amount of force.

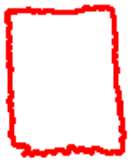
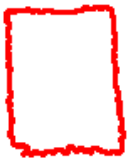
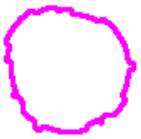
Contour / Deformation	Initial contour	Contour under largest deformation	Final contour after the force is removed
Elastic material or elastic deformation stage			
Plastic material or plastic deformation stage			
Elasto-plastic material or elasto-plastic deformation stage			
Rigid material or rigid deformation stage			

Figure 3.17: Deformation contours for elastic, plastic, elasto-plastic and rigid materials/stages.

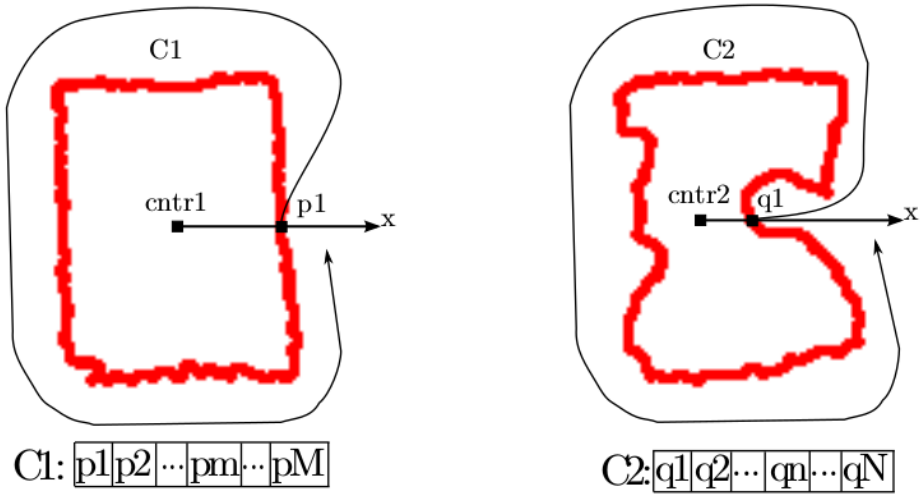
3.4.2 Dynamic Time Warping for Contour Correspondence and Material Behavior-Based Classification

In order to perform the classification of the various categories of deformable objects, the contours obtained in the previous section are compared using dynamic time warping (DTW) to detect changes in the various stages of the object's deformation. Dynamic time warping is a commonly used technique to align two sequences that may vary in length [55] [62] [63]. It provides a robust distance-based method to compare contours of variable length, which also eliminates the need for explicit point-to-point matching of contour pixels or subsampling of the contours, and therefore allows full usage of the information available. As such, DTW does not require the extraction of features from the contours. DTW is selected here because the number of pixels forming a contour can vary significantly in between successive frames for the same object, as a result of its deformation.

In order to facilitate the comparison of two contours captured respectively at time t and $(t + 1)$, they are first arranged in the same order prior to the application of the dynamic time warping method. In particular, the following equation is applied to identify the center pixel of the contour:

$$\boldsymbol{cntr} = \frac{\sum_{k=1}^{K} \boldsymbol{c}_k}{K} \tag{3.12}$$

where $\boldsymbol{cntr}\,(x_{\boldsymbol{cntr}}, y_{\boldsymbol{cntr}})$ represents the center pixel of contour, C, the latter being composed of K pixels, $\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_k, \ldots, \boldsymbol{c}_K(x_{\boldsymbol{c}_K}, y_{\boldsymbol{c}_K})$. This procedure takes place on the contour that has been remapped into the Cartesian domain through an inverse log-polar mapping.



(a) (b)

Figure 3.18: Contours at time t and $(t + 1)$: (a) before force applied at time t; and (b) under the largest deformation at time $(t + 1)$.

The first pixel, $\boldsymbol{p}_1$ in Figure 3.18a, belonging to the contour C is identified as being the pixel that satisfies the following two conditions:

First Pixel Conditions

1: $\tan\frac{y_{cntr}-y_{c_k}}{x_{cntr}-x_{c_k}} = 0$;

2: $\boldsymbol{c}_k$ is located on the right side of $\boldsymbol{cntr}$.

Once this pixel is identified and set as the first pixel of the contour sequence, the contour sequence is filled up with the remaining contour pixels in the counter-clockwise direction (along the arrow in Figure 3.18a).

Figure 3.18 shows an example of two contours that are the initial contour and the contour under the largest deformation of the object made of elastic material from Figure 3.17. At first, the respective center pixels of these two contours are identified using Equation 3.12, and denoted as $\boldsymbol{cntr}_1$ and $\boldsymbol{cntr}_2$ as shown in Figure 3.18. Next, the first pixels $\boldsymbol{p}_1$ and $\boldsymbol{q}_1$ of these two contours, respectively, are identified based on the *First Pixel Conditions* defined above. Finally, the rest of the pixels recuperated in the counter-clockwise direction are stored in the contour sequences such that two contours are in the same order. The initial contour is arranged into contour sequence, C_1, and the deformed contour is arranged into contour sequence, C_2, as follows:

$$C_1 = \boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_m, \ldots, \boldsymbol{p}_M \tag{3.13a}$$

$$C_2 = \boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_n, \ldots, \boldsymbol{q}_N \tag{3.13b}$$

where M and N are the length of sequence $C_1$and sequence C_2, respectively.

DTW is then employed to establish the correspondence between the two contour sequences from Equation 3.13. The DTW distance matrix is built by using Algorithm 2.4 (in section 2.4), where r is initially set as r = 1/5 × max(M, N) and ∞ is set as 10^7. Since the contour sequences C_1 and C_2 are pre-processed, r can be initialized to a relatively small value, which helps to speed up the standard DTW algorithm (Algorithm 2.3).

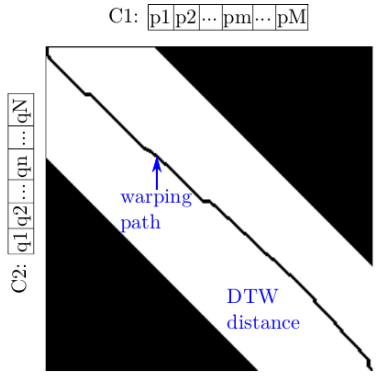


Figure 3.19: Warping matrix.

Once the optimal warping path (i.e. the one with the minimal overall cost), is identified in the DTW matrix, the two contour sequences, C_1 and C_2, are matched by assigning the pixel $\boldsymbol{p}_m$ of C_1 to the pixel

$\boldsymbol{q}_n$ of C_2 so that an element, w_l, of the warping path stands for a pair of pixels ($\boldsymbol{p}_m$, $\boldsymbol{q}_n$). The optimal path is shown as a black line over the white band in Figure 3.19.

As a result of the DTW matching, the pixels of the contour sequences, C_1 and C_2, are matched in pairs as in the following table.

Table 3.1: Correspondence of pixels from C_1 and C_2

W	w_1	w_2	w_3	w_4	w_5	w_6	w_7	$\cdots$	w_L
C_1	$\mathbf{p}_1$	$\mathbf{p}_2$	$\mathbf{p}_2$	$\mathbf{p}_3$	$\mathbf{p}_4$	$\mathbf{p}_5$	$\mathbf{p}_6$	$\cdots$	$\mathbf{p}_M$
C_2	$\mathbf{q}_1$	$\mathbf{q}_2$	$\mathbf{q}_3$	$\mathbf{q}_4$	$\mathbf{q}_5$	$\mathbf{q}_5$	$\mathbf{q}_5$	$\cdots$	$\mathbf{q}_N$

Each pair of pixels represents the pixel correspondence between two contours. Later on, the pixels are compared in pairs in order to determine the difference between two contours, which supports the classification based on the object material under the observations derived at the end of section 3.4.1 about the various categories of objects.

3.4.3 Proposed Classification of Deformable Objects

In this work, we took inspiration from the comparison mechanism proposed in [62] to evaluate the similarity, or dissimilarity, of the deformable object contours in order to achieve an accurate classification based on the material properties. Gonzalez-Sosa et al. [62] use DTW to align pixels of two contour sequences, employing Equation 3.14 to transform the displacements between contours into a similarity score, and they perform a comparison based on this score value.

$$score = e^{-\frac{DIST}{R}} \tag{3.14}$$

where $DIST$ is the obtained cumulative distance between two sequences of pairs of pixels that is known to be minimal. The parameter R represents a normalization factor that takes into account the number of aligned pixels between the two sequences.

In the present case, the contours from consecutive images do not have the same amount of pixels and the contour at time $(t + 1)$, beyond being deformed, may be also slightly rotated or shifted when compared to the contour at time t. This is an expected behavior resulting from the interaction of the robotic hand on the object. To cope with the above two problems, we use the DTW to establish the correspondence of pixels of two contours as discussed in section 3.4.2. Our experimentation demonstrated that the DTW method is able to match the contour at time t with the contour at time $(t + 1)$ even with shifting or slight rotation of the entire contour. However, using the score proposed in [62], it is impossible to distinguish whether the object is deformed or shifted according to its value. In other words, the shift of the object leads to a high $DIST$ value and as a result, the value of the score is low. But the value of the score is also low when deformations happen. Therefore, the decision system would erroneously consider that deformations occurred on the object even if the low score value is in fact caused by a shift of the object.

It is therefore proposed to make the score more robust, and still reliable for proper material category classification while taking into account shifting and slight rotation of the object that may happen during manipulation. The proposed score is calculated as:

$$score(C_1, C_2) = e^{-\frac{\sum_{l=1}^{L} DIST_l}{L}} \tag{3.15}$$

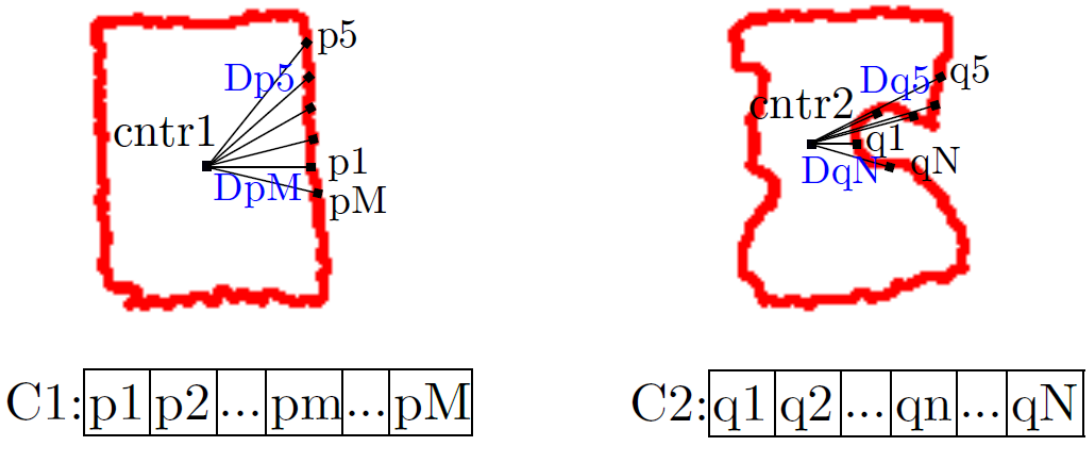
where L is the length of the warping path. $DIST_l$ is the difference between Dp_l and Dq_l, such as $DIST_l = |Dp_l - Dq_l|$, where $Dp_l$is the Euclidean distance between the pixel $\boldsymbol{p}_m$ from the first contour considered and its contour center $\boldsymbol{cntr}_1$, $Dp_l = \big||\boldsymbol{p}_m - \boldsymbol{cntr}_1|\big|$; and Dq_l is the Euclidean distance between the pixel $\boldsymbol{q}_n$ from the second contour considered and its contour center $\boldsymbol{cntr}_2$, $Dq_l = \big||\boldsymbol{q}_n - \boldsymbol{cntr}_2|\big|$. The proposed $DIST_l$ is therefore calculated as illustrated in Table 3.2, where the correspondence of pixels from C_1 and C_2 is established as shown in Table 3.1.

Table 3.2: $DIST_l$ of each pair of pixels from C_1 and C_2

W	w_1	w_2	w_3	w_4	w_5
Dp	$\|\mathbf{p}_1 - \mathbf{cntr}_1\|$	$\|\mathbf{p}_2 - \mathbf{cntr}_1\|$	$\|\mathbf{p}_2 - \mathbf{cntr}_1\|$	$\|\mathbf{p}_3 - \mathbf{cntr}_1\|$	$\|\mathbf{p}_4 - \mathbf{cntr}_1\|$
Dq	$\|\mathbf{q}_1 - \mathbf{cntr}_2\|$	$\|\mathbf{q}_2 - \mathbf{cntr}_2\|$	$\|\mathbf{q}_3 - \mathbf{cntr}_2\|$	$\|\mathbf{q}_4 - \mathbf{cntr}_2\|$	$\|\mathbf{q}_5 - \mathbf{cntr}_2\|$
$DIST$	$\|Dp_1 - Dq_1\|$	$\|Dp_2 - Dq_2\|$	$\|Dp_3 - Dq_3\|$	$\|Dp_4 - Dq_4\|$	$\|Dp_5 - Dq_5\|$

w_6	w_7	...	w_L
$\|\mathbf{p}_5 - \mathbf{cntr}_1\|$	$\|\mathbf{p}_6 - \mathbf{cntr}_1\|$	...	$\|\mathbf{p}_M - \mathbf{cntr}_1\|$
$\|\mathbf{q}_5 - \mathbf{cntr}_2\|$	$\|\mathbf{q}_5 - \mathbf{cntr}_2\|$	...	$\|\mathbf{q}_N - \mathbf{cntr}_2\|$
$\|Dp_6 - Dq_6\|$	$\|Dp_7 - Dq_7\|$	...	$\|Dp_L - Dq_L\|$

Figure 3.20 illustrates the Euclidean distances between a pixel and the center of the contour for the two contours considered previously.



(a) (b)

Figure 3.20: Relative distance of points to contour center for contours at time t and $(t + 1)$: (a) before force applied at time t, and (b) under the largest deformation at time $(t + 1)$.

The score proposed in Equation 3.15 makes the system more robust to shift and rotation than the score in Equation 3.14 due to the fact that Equation 3.15 takes a sum of individual local pairwise differences of distances, $DIST_l$, into consideration rather than only a cumulative distance, $DIST$, as in Equation 3.14.

However, because it is desired that the method is robust to noise and fast at the same time, two supplementary conditions are imposed in order to consider that a contour has been deformed. In particular, we verify if a displacement of the contour occurred, and whether the displacement affects more than three contiguous pixels over the contour. Considering the sequence of $DIST_l$ values, if the value of an element in this vector is larger than a threshold (e.g. 4 pixels in our work to alleviate sensitivity to noise), it is considered that a movement occurred at that location on the contour; otherwise, the difference is attributed to noise and it is considered that no movement occurred. In case the contour has moved, we also verify the number of contiguous pixels affected by the movement. In particular, if the number of contiguous pixels with movement is larger than 3 pixels, the contour is considered being deformed. An example is shown Figure 3.21. The blue contour represents the contour at time t, and the red one is the contour at time $(t + 1)$. The pairs of pixels $(\boldsymbol{p}_1, \boldsymbol{q}_1), (\boldsymbol{p}_2, \boldsymbol{q}_2), (\boldsymbol{p}_3, \boldsymbol{q}_3), \dots, (\boldsymbol{p}_m, \boldsymbol{q}_n)$ and $(\boldsymbol{p}_M, \boldsymbol{q}_N)$ from these two contours, respectively, are matched by the DTW algorithm. $DIST_l$, located down the pair of contours, is the distance between two pixels, $\boldsymbol{p}_m$ and $\boldsymbol{q}_n$, and its value is smaller than 4 in this case. Therefore, there is no movement at pixel $\boldsymbol{p}_m$. On the other hand, $DIST_1$ is the distance between two pixels $\boldsymbol{p}_1$ and $\boldsymbol{q}_1$ and its value is larger than 4, so that pixel $\boldsymbol{p}_1$ is considered in movement. The same can be noticed for $DIST_2, DIST_3$ and $DIST_L$, meaning that the pixels $\boldsymbol{p}_2$, $\boldsymbol{p}_3$ and $\boldsymbol{p}_L$ are in movement as well. As four contiguous pixels exhibit movement, it is considered that the contour at the pixels $\boldsymbol{p}_L, \boldsymbol{p}_1, \boldsymbol{p}_2$ and $\boldsymbol{p}_3$ is deformed. Only the deformed contours are evaluated using Equation 3.15.

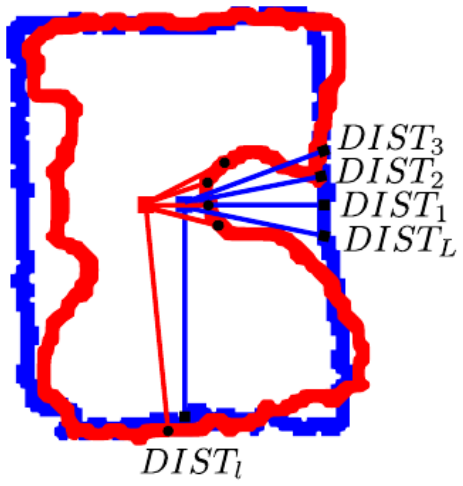


Figure 3.21: Supplementary conditions validated on contour deformation detection.

If the contour is deformed, based on the scores computed pairwise between the initial contour, bw_0, the contour under largest deformation, bw_1, and the final contour after the force is removed, bw_2, using Equation 3.15, the decision process for classifying the object as rigid, elastic, plastic or elasto-

plastic is illustrated in Figure 3.22. The value of the threshold thr applied on the score was empirically set to 0.75. The impact of this threshold is illustrated in the experimental results part of the thesis, in section 4.2.6.

Figure 3.22: Flowchart for material behavior-based object classification.

In order to evaluate the system performance, the classification results for the four material categories will be displayed using confusion matrices [81] in section 4.5.

Chapter 4 Experimental Results and Evaluation

This chapter presents the implementation of the proposed contour detection and tracking framework that uses RGB-D data to track the contour deformations and classify an object under interaction with a robotic hand into the four categories considered (elastic, plastic, elasto-plastic, or rigid), based on the response of the material the object is composed of. The performance of the proposed framework is evaluated using ten objects belonging to the four different categories of material properties.

4.1 Experimental Setup and Software Architecture

4.1.1 Hardware Components

As the goal of this thesis is to track the contour of the object of interest and classify this object according to how it responds to interactions with a robotic hand, the hardware components used in the experiments are the Microsoft Kinect for Xbox 360 and the Barrett BH8-262 robotic hand, which are respectively used to collect the RGB-D data based on which the object contour is tracked, and to manipulate the object. During the experimentation, the Kinect sensor and the robotic hand are placed as shown in Figure 4.1. This configuration leads to the assumption on the presence of a dominant planar surface in the background, considered in section 3.3.1.1. This setup is appropriate in the present context given that the goal of the thesis is not to develop a fully autonomous solution for object manipulation with a robotic hand, but rather to concentrate on the classification of object elasticity.

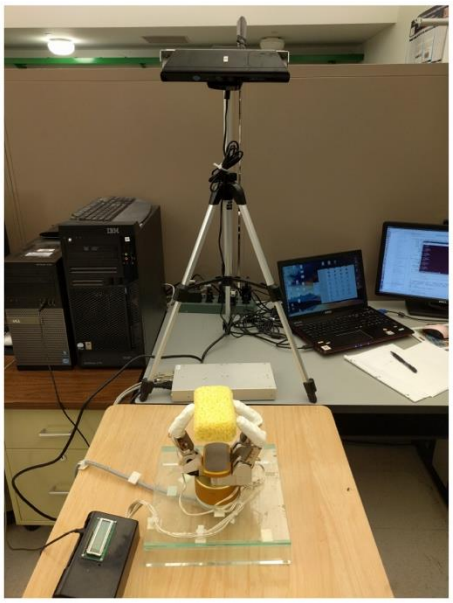


Figure 4.1: Setup of Barrett robotic hand and Kinect sensor.

4.1.1.1 Microsoft Kinect for Xbox 360

The Microsoft Kinect for Xbox 360 [15] is an inexpensive device that comes with RGB and depth sensors which can be used for visual identification and spatial perception of objects. Moreover, it is fully supported and compatible with ROS (Robot Operating System) [82], which is exploited in the current thesis to perform basic operations on the collected data. The combination of these factors makes Microsoft Kinect for Xbox 360 a suitable device for the experiments. Figure 4.2 identifies the major components of the Kinect hardware. The RGB camera is a standard camera that captures the visible light at 640×480 pixels with 8-bit per channel. The infrared laser (IR) emitter and the infrared camera work together to generate the depth image [15].

Figure 4.2: Kinect for Xbox 360 sensor.

4.1.1.2 Robotic Hand and Probing Process

The robotic hand used for manipulating the object is from the Barrett Hand BH8-262 series [83]. The robotic hand shown in Figure 4.3 has three fingers that can pivot around the palm and open/close individually. Their angular positions around the palm are fixed during the interaction with an object. Since some of the objects used in the experimentation are small in size, a cardboard box placed on the palm is employed to lift them up. This ensures that the interactions of the object with the robotic hand

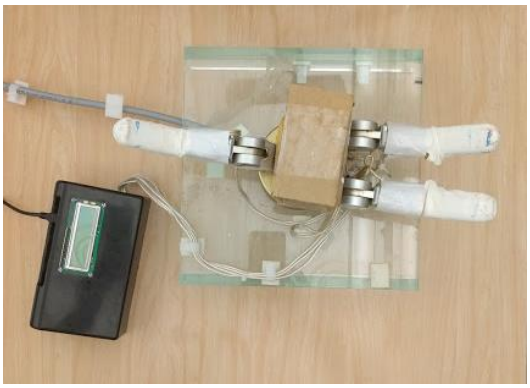


Figure 4.3: Barrett robotic hand.

fingertips are occurring along the sides of the object, therefore avoiding the occlusion of the object shape by the fingers during manipulation and applying proper forces to deform the contour of non-rigid objects. This enables the extraction of the contour representing the deformations of the object resulting from the forces applied on it. In Figure 4.3, it can be noticed that the robotic hand fingers are covered

with white rubber gloves. This proved useful for several reasons, including the protection of the sensible touch sensors mounted (but not used in this work) on the robotic hand, to ensure the uniformity of color over the entire surface of the fingers, and making them sufficiently contrasting to the background surface and objects.

The robotic hand moves relatively slowly while performing the object probing. The closing and reopening of the fingers, while in touch with the object, takes on average 2 s. As shown in Figure 4.4, the robotic hand applies balanced grasping forces at its fingertips in order to maintain the object centered over the palm and without significant slippage (Figure 4.4a,d,g); the hand then contracts the fingers to compress the object and pauses briefly at the largest deformation of the object under the applied forces (Figure 4.4b,e,h); lastly, the hand relaxes its fingers and moves them back to the starting positions (Figure 4.4c,f,i). In the experiments, three different magnitudes of forces are applied to probe the object, denoted as the “light”, the “medium”, and the “strong” forces, respectively corresponding to about 1.3 N, 3.9 N and 17.6 N on each finger.

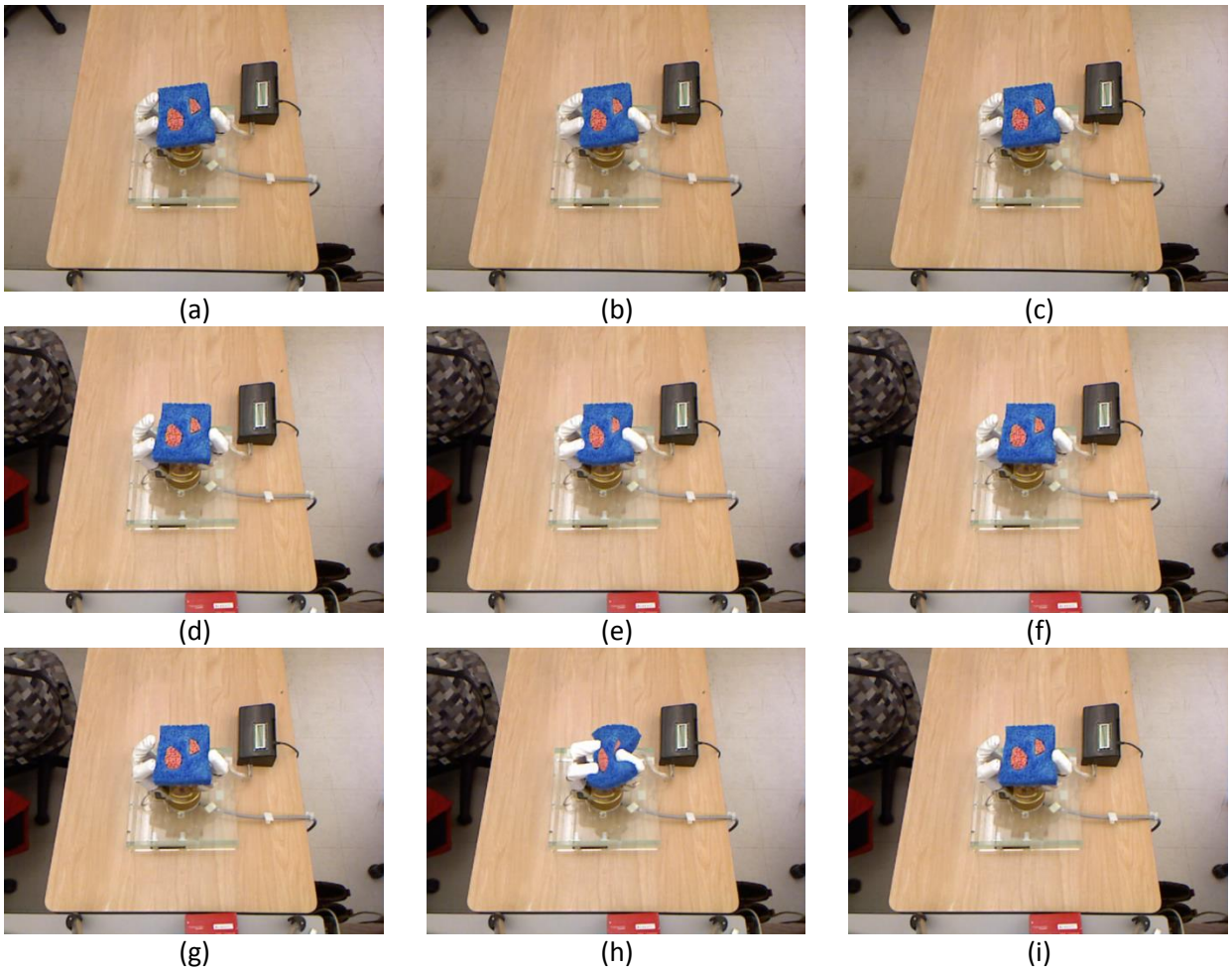


Figure 4.4: Process of probing an object with the robotic hand: (a)-(c) with light force (~1.3N); (d)-(f) medium force (~3.9N); and (g)-(i) strong force (~17.6N).

Each row of Figure 4.4 illustrates the deformation produced on a same object, respectively under a light, medium and strong force. In practice, the cardboard that holds the object at the proper height over the palm may sometimes hinder the movement of closing fingers. This can lead to situations in which no significant difference on the deformation of the object can be observed between the cases when a medium force and a strong force are applied by the fingers.

4.1.2 Software Architecture

The contour tracking and classification system developed in this research is implemented in C++, within the Robot Operating System (ROS) framework [82], and utilizes some functions of the Point Cloud Library (PCL) [84] and OpenCV library [85], to manipulate the RGB-D data acquired using the Microsoft Kinect for Xbox 360. ROS provides libraries, tools and a message passing system that facilitates the integration and communication between various nodes. All these features of ROS make the PCL and OpenCV libraries fully compatible with one another. A ROS system is comprised of many communicating nodes. Each of these nodes is an executable file. Figure 4.5 depicts the contour tracking and classification system, as designed and implemented in this thesis. It exhibits the processing of the RGB image (2D data), the point cloud (3D data), and the information exchange between them. Every node displayed as a rectangular box in Figure 4.5 has been designed, implemented, and tested as part of this research in order to meet the objectives.

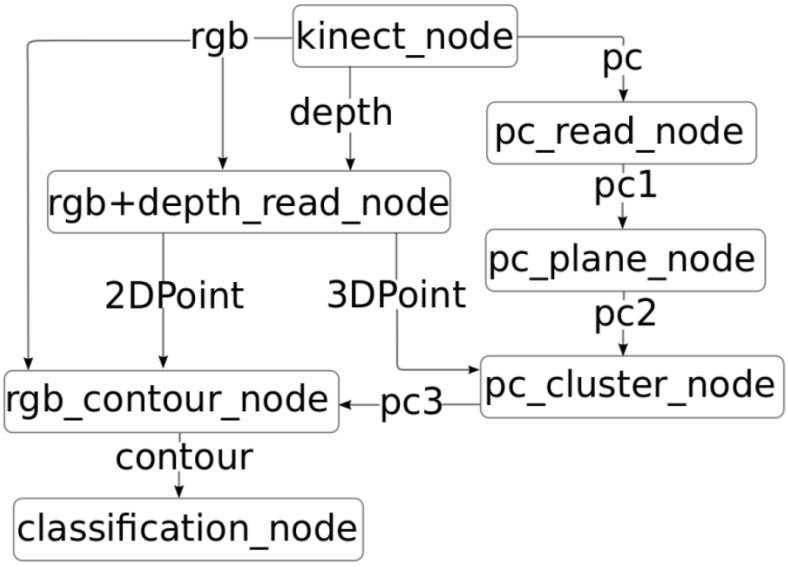


Figure 4.5: Top-level software architecture for the proposed contour tracking and classification system.

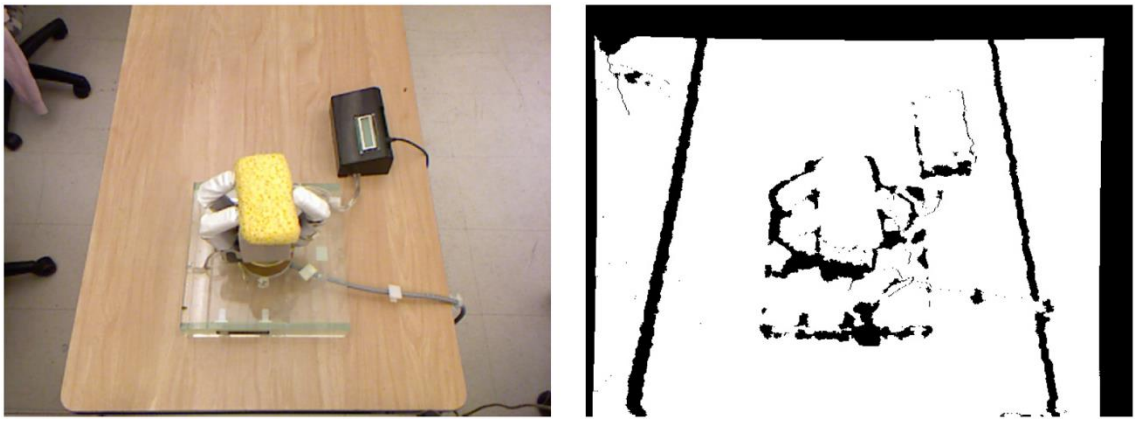
A brief description of each node appears in the following sections.

4.1.2.1 Kinect Node

This node captures the RGB and depth images, as well as the point cloud from the Kinect sensor. The RGB channel is used for receiving the user selected point, called 2D fixation point, and for detecting the object contour; the depth channel is used for identifying the 3D fixation point converted from the 2D

fixation point; whereas the point cloud channel is used for detecting the object of interest, mapping the 3D information onto the corresponding 2D RGB pixels, and initializing the process of contour detection.

The color and depth images obtained from RGB and depth (IR) cameras of the Kinect sensor have to be registered before taking measurements since the two cameras on the device are physically separated. In ROS, a package called freenect_camera [86] provides the depth registration functionality which aligns every pixel in the depth image (shown in Figure 4.6b) with its corresponding pixel in the RGB image (shown in Figure 4.6a). In Figure 4.6b, white regions show where the pixels are registered, and black regions correspond to areas where the depth value of pixels is not valid.



(a) (b)

Figure 4.6: Data from Kinect sensor: (a) RGB image; and (b) registered depth image.

4.1.2.2 RGB+Depth Read Node

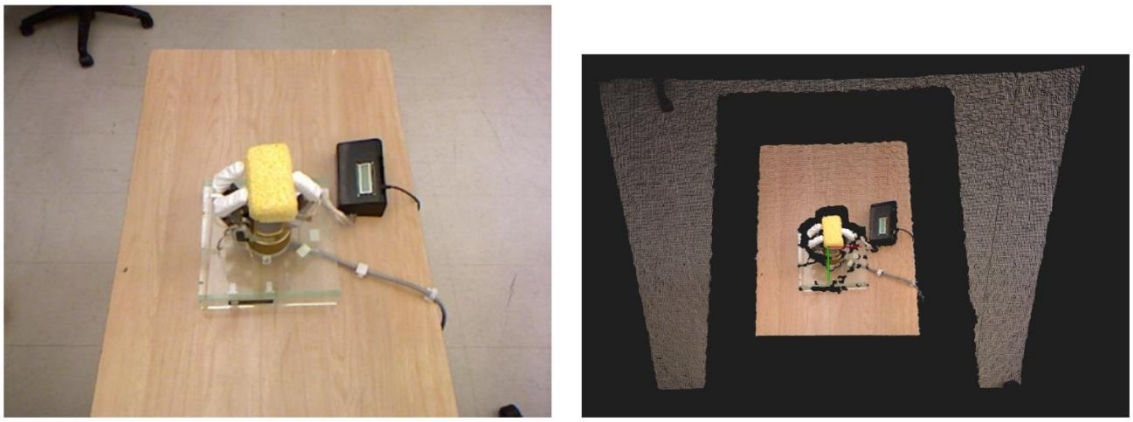
This node receives the user input point, the 2D fixation point, and identifies its corresponding 3D fixation point, using the 2D-3D mapping described in section 3.2.3. The 2D fixation point is a pixel identified by the user on the RGB image (Figure 4.6a). In order to convert the 2D fixation point into the 3D fixation point, the registered depth image (Figure 4.6b) is used to specify the distance of the 2D fixation point with respect to the Kinect sensor.

4.1.2.3 PC Read Node

This node handles the raw point cloud collected by the Kinect sensor. The unorganized points in this point cloud are filtered using the "removeNaNFromPointCloud" function [87] provided by the Point Cloud Library (PCL) to remove points with NaN (Not a Number) values from the point cloud. This function removes the points with x, y or z coordinate equaling to a NaN value, which indicates measurement errors and/or inaccuracies.

Generally, erroneous points correspond to points located out of the operational distance limitation of the sensor or missed because of the reflective surface of an object. The coordinates of such points are characterized by the presence of NaN values. These points are removed from the point cloud as shown in Figure 4.7, while the remaining points have valid, finite values. The NaN removed point cloud

(henceforth, we refer to it as point cloud, which is denoted as pc1 in Figure 4.5) is sent to the pc_plane_node.



(a) (b)

Figure 4.7: (a) RGB image from Kinect; (b) NaN removed point cloud.

4.1.2.4 PC Plane Node

The pc_plane_node is responsible for extracting the points that make up the planar surface (i.e. measurement table) in the point cloud, as discussed in section 3.3.1. PCL provides various algorithms for point cloud processing, one of which is random sample consensus (RANSAC). The latter is used for segmenting the planar surface [88] by identifying all the points that support the planar model and to remove them from the point cloud. This node efficiently removes most of the background, while the remaining points (pc2 in Figure 4.5) are sent to the pc_cluster_node.

4.1.2.5 PC Cluster Node

The pc_cluster_node executes the cluster extraction algorithm described in section 3.3.2 on the point cloud transmitted by the pc_plane_node in order to obtain the cluster of the object of interest in the point cloud. Before clustering the point cloud after plane extraction, a *k*-d tree data structure [89], provided by PCL, is employed on these points to organize the 3D points structure in order to speed up the object clustering process. All the points constituting the cluster of the object of interest (pc3 in Figure 4.5) are sent to the RGB_contour_node.

4.1.2.6 RGB Contour Node

This node mainly manages the identification of the contour of the object of interest on the RGB image and obtains the 2D fixation point (2DPoint in Figure 4.5), which has the same coordinates as the input of the rgb+depth_read_node. The latter is also read here for the log-polar mapping. The node first receives the selected cluster (pc3) and maps the corresponding 3D points in the cluster to the RGB image, as detailed in section 3.3.3, which is then used as the initialization for the level set method. Next, the bounding window is obtained from the RGB image by expanding a border outwardly with respect to the initialization area. Thereafter, the log-polar mapping approach [90], which is a function provided by the

OpenCV Library, is employed on the bounding window by using the 2D fixation point as the transformation center to obtain the cortical image, as presented in section 3.3.4. This is followed by application of the fast level set method on this cortical image in order to detect the cortical contour of the object of interest, as detailed in section 3.3.6. Finally, the contour of the object of interest is mapped back to the Cartesian domain and sent to the classification node.

4.1.2.7 Classification Node

This node handles the characterization of objects with different material properties as rigid, elastic, plastic, or elasto-plastic, based on the tracked contours of the manipulated object, as detailed in section 3.4. The contours of an object at three different stages are provided as inputs to this node. By comparing the similarities of pairwise contours in order, the changes on the shape of the object under the manipulation with the robotic hand are detected so that the characteristics of the object can be evaluated.

4.2 Data Acquisition and Parameter Settings

In this section, the impact of various parameters involved in the proposed approach is studied. Several experiments are performed using possible values of the parameters in order to determine the optimal tuning of the system.

The devices involved in the proposed contour tracking and classification system are placed as shown in Figure 4.1. The robotic hand with its base, used for interacting with the deformable object, is situated on a table and the object is placed in the robotic hand during experimentation. A Kinect sensor is mounted on a tripod located on the table. The distance of the Kinect sensor with respect to the object surface is typically around 55 cm. In order to obtain the optimal RGB and depth images, and the corresponding point cloud, from the Kinect sensor, as well as a reliable classification, the distance of the Kinect sensor with respect to the object surface should be situated in the range of 50 cm to 80 cm.

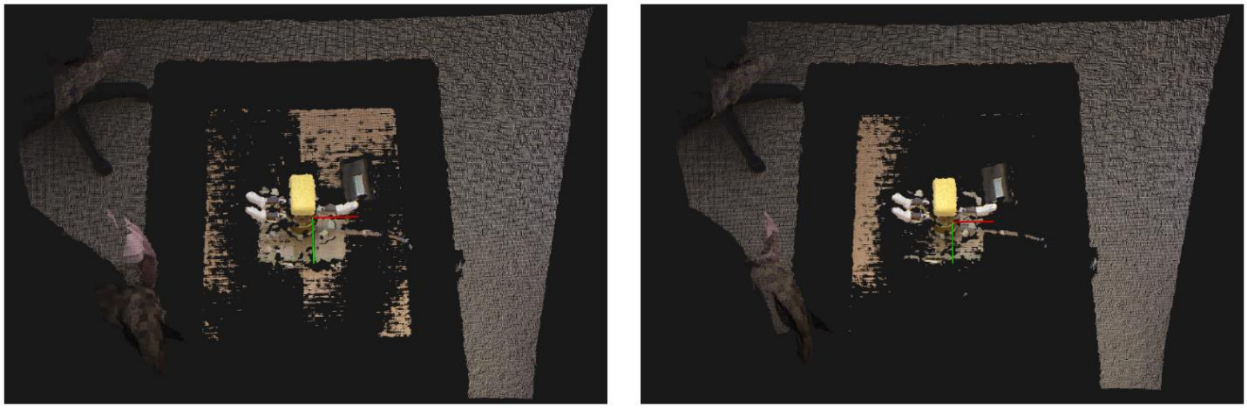
4.2.1 RANSAC-Based Algorithm for Background Removal

In section 3.3.1, the background removal procedure based on RANSAC algorithm was detailed. In the current context, the background mainly refers to the planar surface of the table top. The goal being to track the object's contour changes when grasped by the robotic hand, the object sitting in the robotic hand is mainly the region of interest. In order to increase performance, the points that surround the object of interest, such as the table surface, are discarded at an early stage.

The parameter, t, used in the RANSAC algorithm, represents the distance threshold that constrains points fitting the planar surface model in Algorithm 3.1, in section 3.3.1.1. Figures 4.8, 4.9 and 4.10 illustrate the impact of the distance threshold, t, on the performance of the RANSAC algorithm.

Figures 4.8a and 4.8b show the results of the table removal procedure when the threshold is set to $t = 0.5$cm and $t = 0.75$cm, respectively. Under such stringent fitting conditions, the table model is not properly extracted and therefore the table surface is not totally removed, as shown in both cases in

Figure 4.8. The extracted planar model of Figure 4.8a is $0.00843111x + 0.51796y + 0.855358z - 0.878826 = 0$, and the planar model of Figure 4.8b is $0.0895423x + 0.51796y + 0.855358z - 0.878826 = 0$, which we see are relatively similar.



(a) (b)

Figure 4.8: (a) Planar background removal with RANSAC parameter *t*=0.5 cm; and (b) with *t*=0.75 cm.

Figure 4.9 exhibits the results of the point cloud after plane extraction with the threshold set to t = 1cm. It is obvious that the points fitting the planar model are totally removed from the point cloud while the objects located over the table top (i.e. closer to the Kinect sensor) are preserved. The planar model of Figure 4.9 is $0.00437805x + 0.517733y + 0.855531z - 0.879282 = 0$, which exhibits more significant variation with respect to the models estimated with tighter *t* settings.

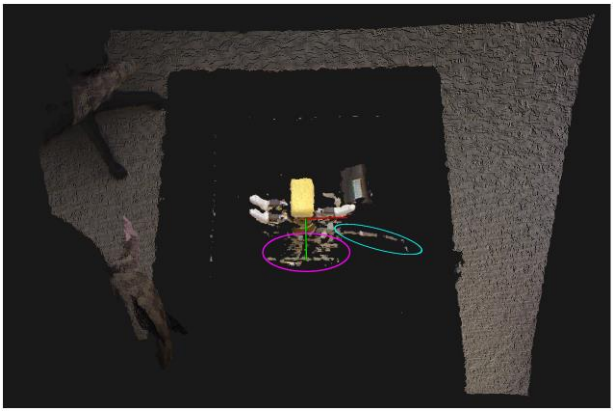
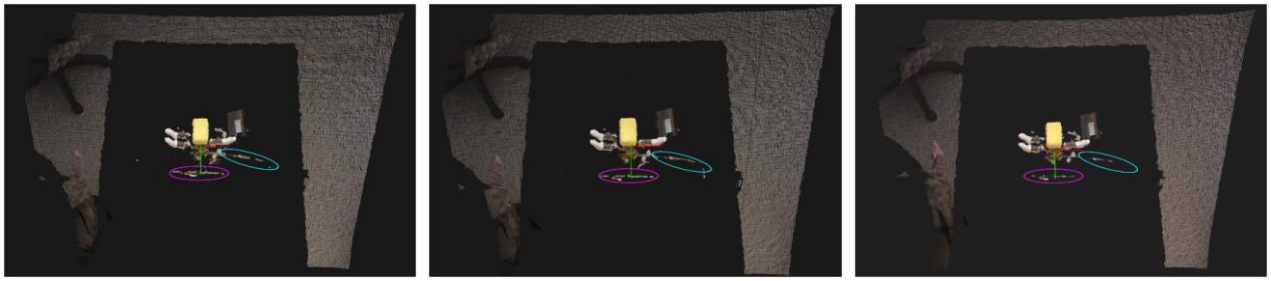


Figure 4.9: Planar background removal with RANSAC parameter *t*=1 cm.

Furthermore, Figure 4.10 shows the results of the table removal procedure with threshold values set to $t = 1.25$ cm, $t = 1.5$ cm, and $t = 2$ cm, respectively. The larger distance thresholds increase the robustness of the RANSAC algorithm. Compared with Figure 4.9 where the base of the robotic hand and the cables are circled in magenta and cyan, it can be observed that the robotic hand and the cables highlighted in the same colors in Figures 4.10b and 4.10c, are partially merged in the planar model and removed from the point cloud. This may be perceived as a more appropriate tuning for the experimental setup considered. Figure 4.10a, where the parameter $t = 1.25$ cm, shows a comparatively good result that detects the whole planar surface model and rarely misclassifies the points from the base and

cables. However, the speed of tracking and classification must also be taken into account. Under the premise of a proper planar surface fitting model, a smaller distance threshold merges fewer points in the planar surface, which speeds up the process of background removal.



(a) (b) (c)

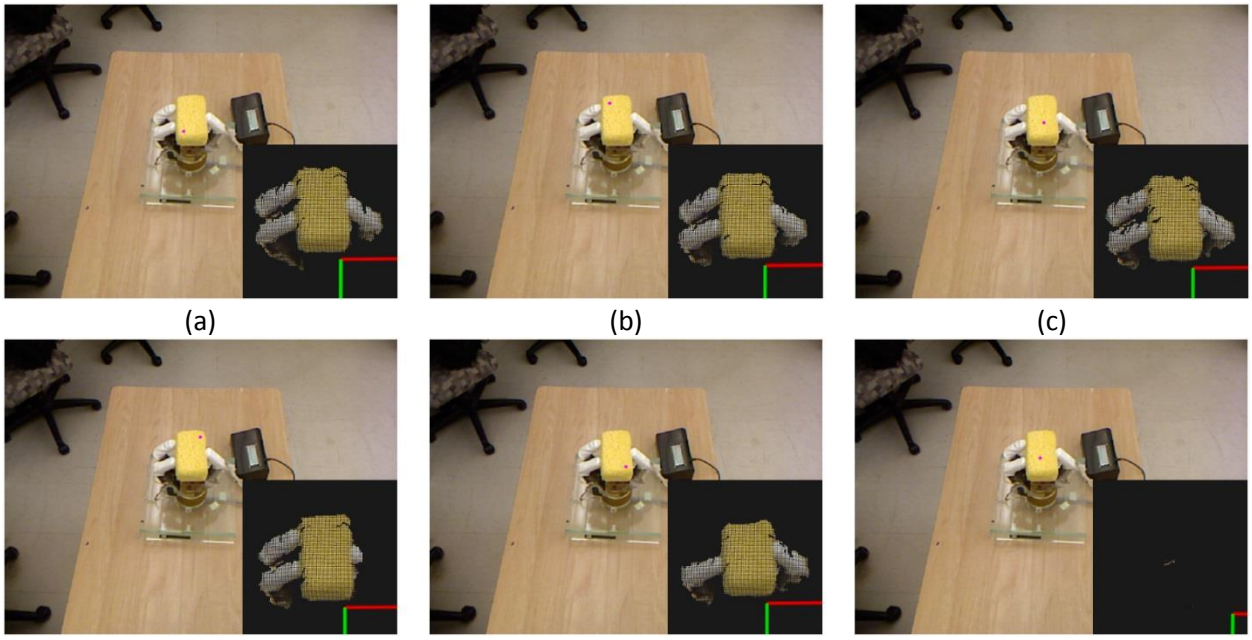
Figure 4.10: (a) Planar background removal with RANSAC parameter *t*=1.25 cm; (b) with *t*=1.5 cm; and (c) with *t*=2 cm.

In conclusion, the RANSAC algorithm (Algorithm 3.1) produces slightly better results as the value of threshold t increases. However, a smaller distance threshold preserves fewer points composing the planar model, which speeds up the background removal process. For these reasons, the intermediate value of $t = 1$ cm is selected in our experiments.

4.2.2 Point Cloud Cluster Extraction

In section 3.3.2, the process of recovering the point cloud cluster around the object of interest from the remaining point cloud after plane extraction is introduced. It is based on a 3D fixation point, computed from the user-provided 2D fixation point through the 2D-3D mapping process. The possible impact of the location of the 3D fixation point on the clustering result is examined here. In the cluster extraction algorithm (Algorithm 3.5 in section 3.3.2), for a given 3D selected point, the other 3D points situated inside a sphere with a radius, d_{th}, and centered on the selected point, are considered to belong to the same cluster. The distance threshold, d_{th}, is therefore an important parameter that constrains the extent of the distance between two 3D points considered as neighbors. In this work, it is chosen empirically. The impact of d_{th} on the performance of the cluster extraction algorithm is studied in order to determine a proper value. The following results are reported for a value of the parameter $t = 1$ cm, as identified in the previous section.

The following figures show the clustering results, with parameter d_{th} = 2.5 mm, for five different locations (Figures 4.11a through 4.11e) of the selected 2D fixation point, denoted by a red dot over the surface of the object, and with parameter d_{th} = 2 mm for a 2D fixation point located close to the center of the object (Figure 4.11f). Figures 4.11a through 4.11e show that the obtained 3D clusters (bottom right corner of each image) have some holes. A region of the upper part of the object is missing in Figure 4.11e, beyond the missing robotic hand finger on the left side. Moreover, when decreasing further the distance, d_{th}, the cluster cannot be recovered by the extraction algorithm (Algorithm 3.5), as shown with parameter d_{th}= 2mm in Figure 4.11f. The few points visible in Figure 4.11f (bottom right image) are the 100 nearest neighboring points of the 3D fixation point from Algorithm 3.5 that are preserved by default.



(a) (b) (c)

(d) (e) (f)

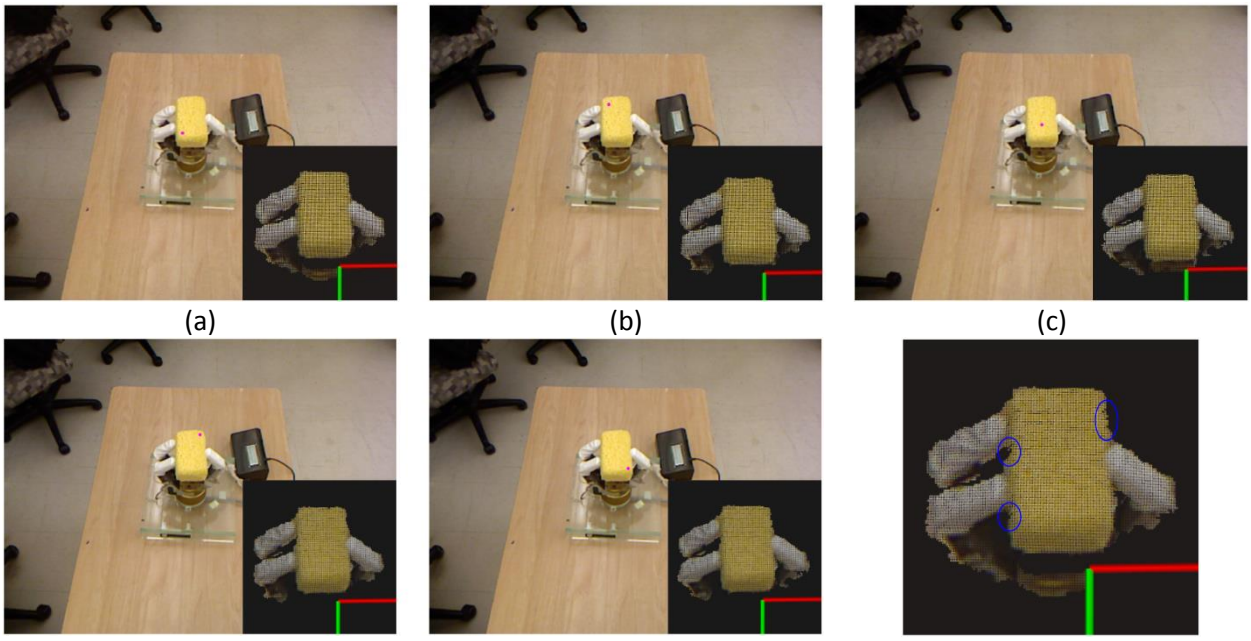
Figure 4.11: Cluster extraction with parameter d_{th} = 2.5 mm and 2D fixation point of coordinates: (a) (292, 207); (b) (293, 161); (c) (307, 193); (d) (319, 159); (e) (320, 207); and (f) cluster extraction with parameter d_{th} = 2mm and 2D fixation point of coordinates (310, 192).

Figures 4.12a-e show the clusters obtained as result of the application of Algorithm 3.5 with the parameter value d_{th} = 3mm, for five different fixation points. These clustering results are improved, as there are no obvious holes on the clusters of the objects when compared with the ones in Figure 4.11. However some fine details are still missing around the edges, which can be clearly seen inside the blue circles highlighted over the enlarged clustering result in Figure 4.12f. In other words, the clustering results with the parameter value d_{th} = 3mm are not very satisfactory, neither.

The clustering results with different values of the distance threshold shown in Figure 4.11 and Figure 4.12 suggest that the results improve as the threshold imposed on the distance, d_{th}, increases, which is expected. For the results in Figure 4.13, the distance threshold is further increased to d_{th} = 5mm. The clustering results of the object are, in this case, enhanced around the object edges, which are smoother, as one can observe in Figure 4.13f. Algorithm 3.5 with d_{th} = 5mm performs more optimally in that the clustering results present full details of the object, especially around the contour which is important here, when compared to the previous two clustering results with smaller distance thresholds. Furthermore, Algorithm 3.5 with d_{th} = 5mm obtains the same high quality regardless of the position of the fixation points, as can be observed for the five significantly different locations of the fixation point in Figure 4.13a-e.

To pursue the study with even larger values on the neighborhood search distance parameter, the following clusters are recovered from Algorithm 3.5 with the parameter d_{th} = 10mm. The object clustering results shown in Figure 4.14 are similar to the ones in Figure 4.13. However, according to Algorithm 3.5, the two points belong to one cluster if their distance is less than the distance threshold, d_{th}. Therefore, setting d_{th} = 10mm becomes risky because a larger value to be considered as the

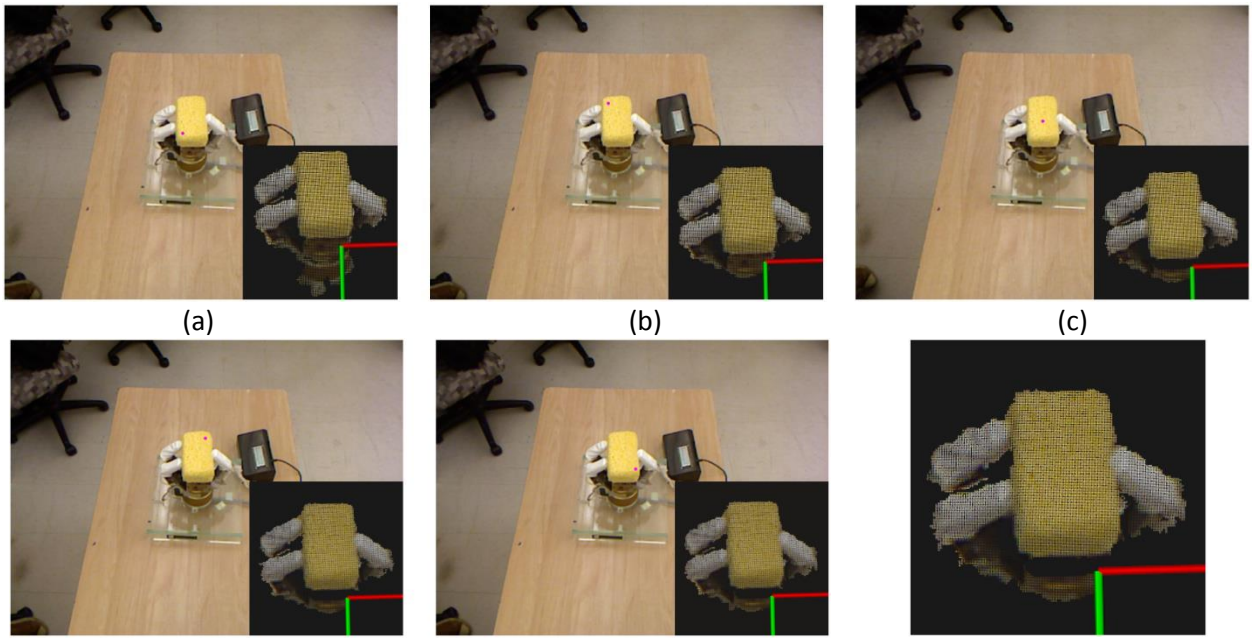
maximum distance in between two 3D points that should belong to a single object may lead to group in a same cluster parts of the scene that are actually belonging to different objects.



(a) (b) (c)

(d) (e) (f)

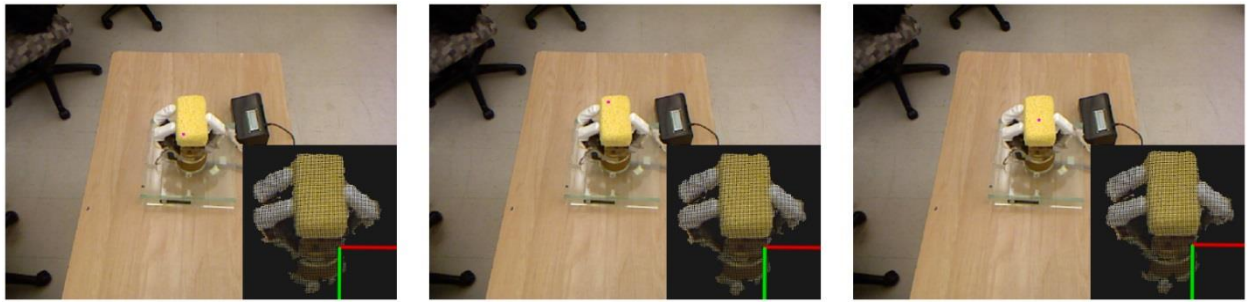
Figure 4.12: Cluster extraction with parameter d_{th} = 3mm and 2D fixation point of coordinates: (a) (290, 208); (b) (292,161); (c) (305,194); (d) (319,155); (e) (324,210); and (f) enlarged clustering result of (e).



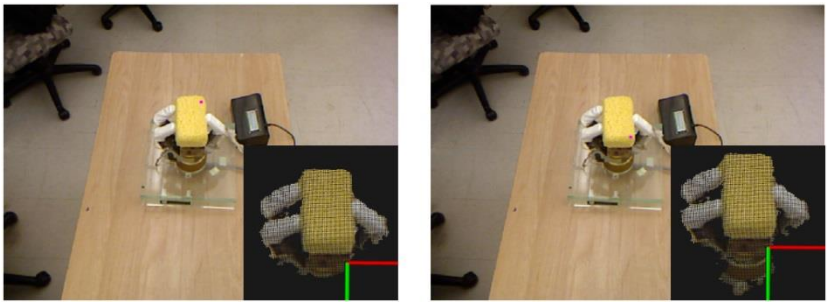
(a) (b) (c)

(d) (e) (f)

Figure 4.13: Cluster extraction with parameter d_{th} = 5mm and 2D fixation point of coordinates: (a) (290,209); (b) (290,160); (c) (305,189); (d) (317,159); (e) (325,209); and (f) enlarged clustering result of (c).



(a) (b) (c)



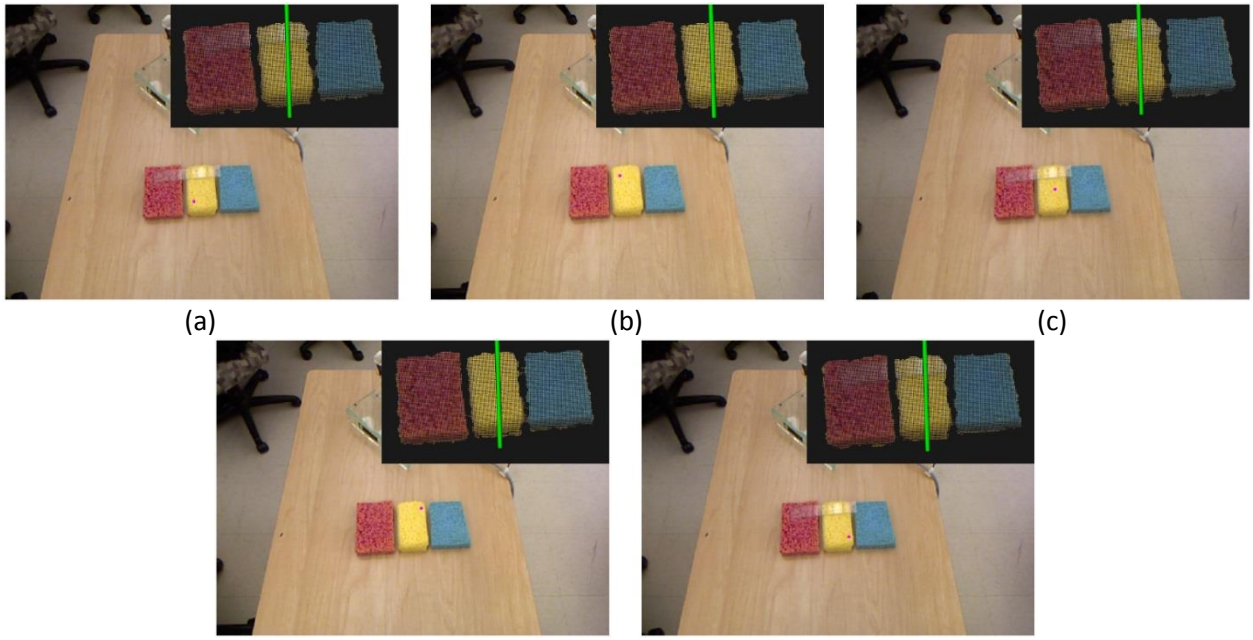
(d) (e)

Figure 4.14: Cluster extraction with parameter d_{th} = 10mm and 2D fixation point of coordinates: (a) (291,211); (b) (293,159); (c) (304,188); (d) (321,158); and (e) (324,214).

Figure 4.15 illustrates such an erroneous situation where multiple objects are grouped as part of one 3D cluster (upper right sub-image). A transparent ruler (seen over the objects in Figures 4.15a, 4.15c and 4.15e) is used to measure the actual width of the gap between the pink sponge and the yellow sponge, which is here 1cm. In this scenario, d_{th} set to 10mm is too large with respect to the scale of the actual objects and their separation for reliably extracting the cluster of a single object with Algorithm 3.5. The green line overlapping the yellow object in Figure 4.15 is only an artifact from the Point Cloud Library (PCL) [76] used to display the resulting 3D cluster. It has no particular meaning in this figure.

In conclusion, Algorithm 3.5 is found to perform well with the distance threshold set to d_{th} = 5mm when the testing setup shown in Figure 4.1 is used. This setting allows the object cluster recovery from the point cloud after background planar surface extraction with enough details about the object of interest to support the next processing stages. The tests performed and reported in Figures 4.11 through 4.15 revealed that if d_{th} is small, an object tends to be split into multiple clusters and therefore the cluster containing the object of interest tends to exhibit missing parts. On the other hand, if d_{th} is larger, the object itself can reliably be recovered and the base of the robotic hand can also be recuperated in the object cluster. However, for a too high value of d_{th}, multiple objects can be grouped in a single cluster, which is not desirable in this research, as the objective is to categorize the deformation behavior of one object at a time. Obviously, tuning the parameter d_{th} is dependent on the setup used, the resolution of the sensor, and the scaling of the scene that depends on the relative distance between the objects and the sensor. A tuning procedure such as the one described in this section should be performed whenever the acquisition stage is modified. Furthermore, as shown in Figures 4.11 through 4.15, the location of the 3D fixation point, that is the correspondent in the point cloud to the 2D fixation point in the RGB

image, does not significantly affect the clustering results. Given the concerns for efficiency and accuracy of the object clustering procedure from the point cloud after background planar surface extraction, the parameter, d_{th}, is set to d_{th} = 5mm for all experiments in this thesis.



(a) (b) (c)

(d) (e)

Figure 4.15: Cluster extraction with parameter d_{th} = 10mm with multiple objects in the scene and 2D fixation point of coordinates: (a) (307, 320); (b) (308, 278); (c) (323, 300); (d) (334, 272); and (e) (338, 319).

4.2.3 Bounding Window Size

As detailed in section 3.3.3, the extracted 3D object cluster is projected back onto the RGB image and the resulting projection is used to compute a bounding window around the identified object, and that also serves for the initialization of the level set method. In particular, the bounding window with which the level set is initialized represents the bounding box around the 2D points (the farthest left, right, up and down locations) corresponding to the extracted 2D cluster, plus a border of several pixels around it, which is a parameter determined by the user. Here, we study the contour detection results obtained with five different sizes of the bounding window in two cases: *i)* when the object is held by the fingers of the robotic hand, and *ii*) when the object sits on the palm of the robotic hand. These results are obtained using the following values for the parameters: the distance threshold for the planar surface model extracted is set to t = 1cm, and the distance threshold to generate the 3D point cloud cluster is d_{th} = 5mm, as established earlier. Also the blind spot size of the log-polar mapping is set to ρ_0 = 3. The impact of this parameter will be examined in section 4.2.4.

The contour detection results on an object sitting directly on the palm of the robotic hand and when no interaction occurs with the robotic hand fingertips, are displayed in Figure 4.16. This figure shows the Cartesian bounding window images (first row) and their corresponding cortical images (second row)

when an additional border of 1 pixel, 20 pixels, 30 pixels, 50 pixels and 100 pixels, respectively, is used all around the initially extracted cluster, as discussed in section 4.2.2.

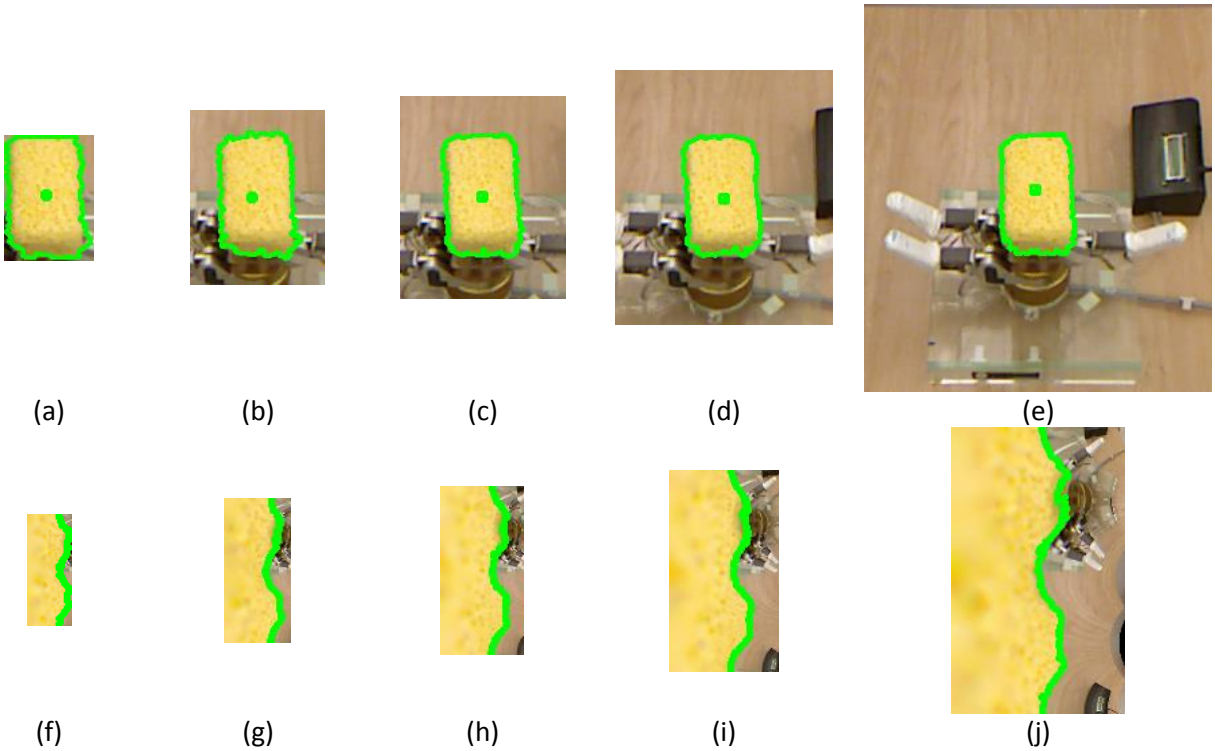


Figure 4.16: Contour detection on an object sitting on the palm of the robotic hand: (a) Cartesian bounding window with additional border of 1 pixel, (b) 20 pixels, (c) 30 pixels, (d) 50 pixels, (e) 100 pixels; and (f) cortical image of (a), (g) cortical image of (b), (h) cortical image of (c), (i) cortical image of (d), (j) cortical image of (e).

Comparing Figure 4.16a to 4.16e, we notice that the contour becomes smoother when the number of pixels of the additional border increases. Examining the respective columns of Figure 4.16, we can observe that the size of the cortical image also varies proportionally to the size of its corresponding Cartesian image, a relationship that originates from Equation 2.12 and Equation 2.13 in section 2.3.4.

Similarly, Figure 4.17 shows the Cartesian contour detection results when the object is held by the fingers of the robotic hand for different sizes of the bounding window images and their corresponding contour detection results overlapped on the cortical images. The same five cases are considered here, with an additional border of 1 pixel, 20 pixels, 30 pixels, 50 pixels and 100 pixels, respectively. Comparing Figure 4.17a to 4.17e, we again observe that the contour is better defined when the bounding window image is larger, and the same trend on the size changes between the cortical images in Figure 4.17f to 4.17j occurs.

The reasoning behind these observations is that a larger bounding window in the Cartesian image provides more information for its cortical image counterpart on which the fast level set is applied, resulting in the cortical contour being detected with more precision, which in turns leads to a smoother Cartesian image display of the contour at the output. The relative size of the bounding window depends

on the size of the object in the RGB image. However, because near real-time performance is desired for contour detection or tracking, the size of the bounding window should not be too large in practice, as it makes for supplementary pixels, or cortical image pixels, to be processed.

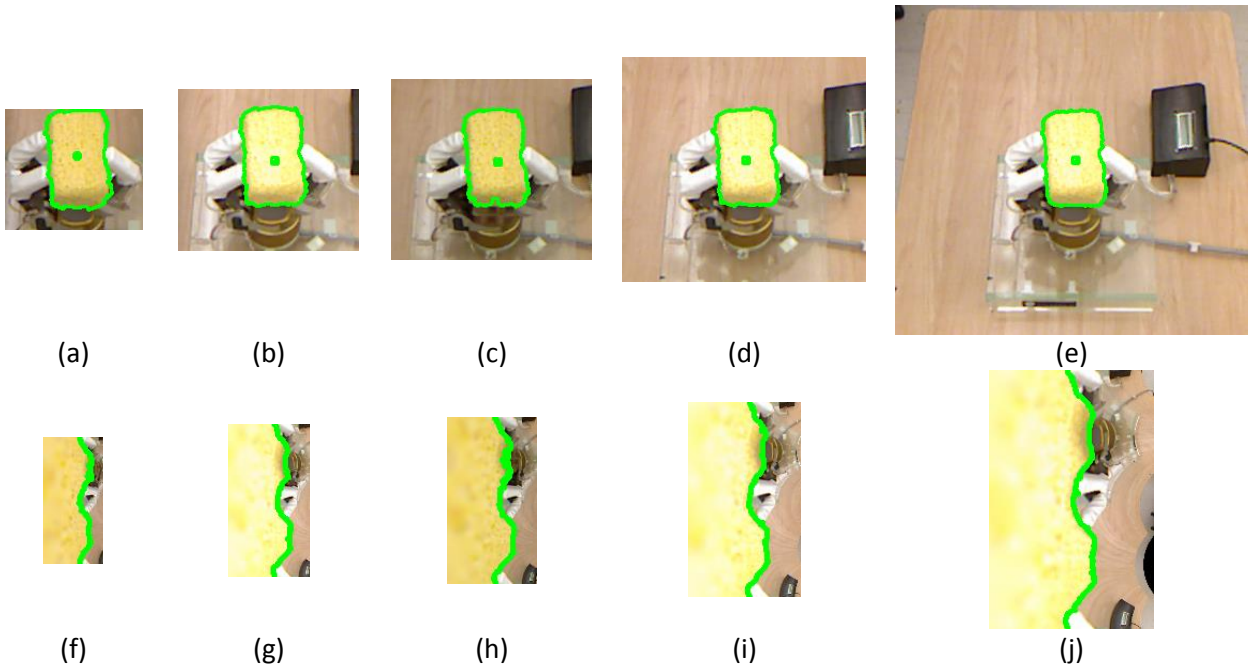


Figure 4.17: Contour detection on an object held by the fingers of the robotic hand: (a) Cartesian bounding window with additional border of 1 pixel, (b) 20 pixels, (c) 30 pixels, (d) 50 pixels, (e) 100 pixels; and (f) cortical image of (a), (g) cortical image of (b), (h) cortical image of (c), (i) cortical image of (d), (j) cortical image of (e).

When we compare each figure from Figure 4.16 to its corresponding figure from Figure 4.17, it can be seen that the tracking contour results are more accurate in Figure 4.17. This is due to the fact that the size of the bounding window image in Figure 4.17 is larger than the size of its corresponding image in Figure 4.16, especially on the width. This is caused by the initialization stage of the fast level set method. In Figure 4.16, the initialization area is projected from the 3D cluster points that correspond to the rough estimation of the object surface only as the robotic hand fingers are not touching the object; while the initialization area in Figure 4.17 is projected for the 3D cluster points that contains the object and parts of the robot fingers, making it wider, and resulting in overall larger images for the area of interest, even with similar number of extra pixels added all around.

Based on this experimentation, and in order to achieve as effective and as precise as possible contour detection, we selected the parameter for the additional border as 20 pixels for the case when the distance of the Kinect sensor with respect to the object surface is about 75cm. As the surface of the image that is covered by the object of interest increases when the distance separating the Kinect sensor and the object decreases, a proportionally larger surface needs to be assigned for the bounding window when the sensor is set closer to the object. As a result, a larger value for the border size, of 50 pixels, is used when the Kinect sensor is situated at about 55 cm above the surface of the object of interest. This provides a mechanism to take into account the effect of the scaling factor.

4.2.4 Log-Polar Mapping: ρ_0 and 2D Fixation Point

The cortical image obtained through log-polar mapping from the Cartesian image depends on the position of the 2D fixation point on the object, as well as on the radius of the blind spot (in pixels), ρ_0, as it will be shown in the experiments presented in this section. As introduced in section 2.3.4, in the log-polar mapping approach, the 2D fixation point plays the role of the center of the transformation, around which point the precision of the cortical image is maximal (the area with the highest resolution mapping from the Cartesian image is located on the left part of the cortical image, as the u axis is depicted horizontally from left to right). The 2D fixation point also represents the blind spot, ρ_0, that is the area in which the Cartesian image is not mapped to the cortical image. This results in a black area on the object surface in the retinal image. Particularly, the first column of the cortical image is composed by the nearest neighboring pixels to the blind spot in the Cartesian image.

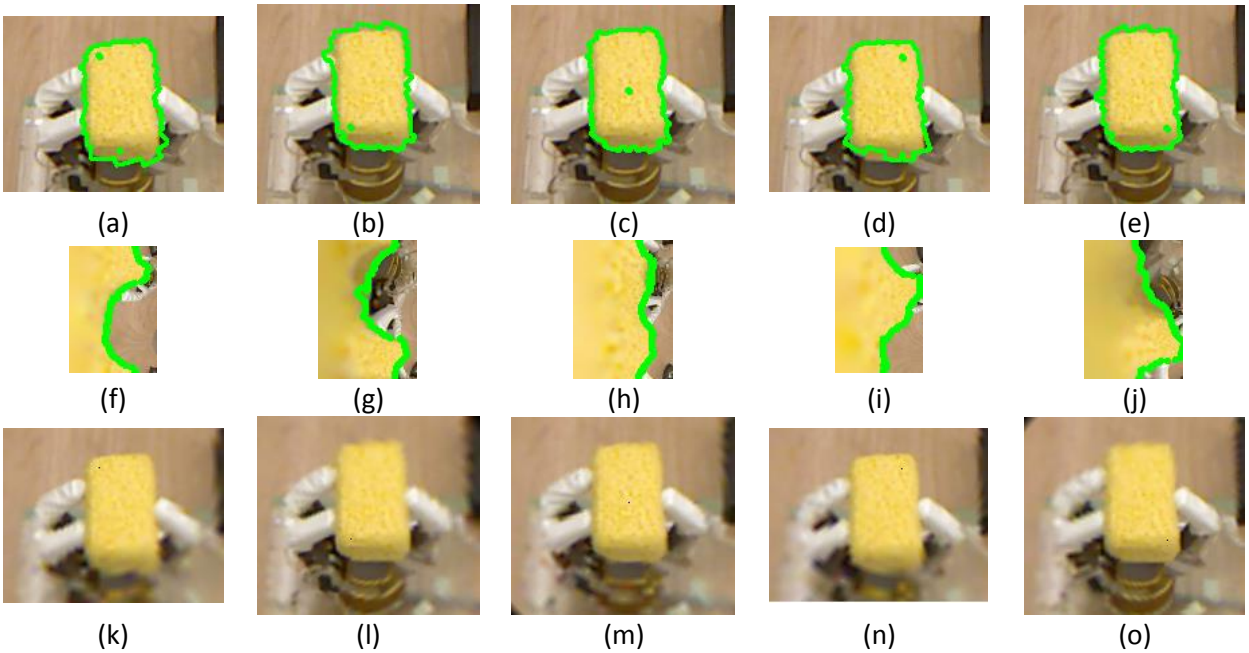


Figure 4.18: Contour detected with a blind spot size of ρ_0=1 pixel: 2D fixation point at (a) the top-left corner of the object; (b) the bottom-left corner of the object; (c) the center of the object; (d) the top-right of the object; (e) the bottom-right of the object; (f-j) corresponding cortical images; and (k-o) corresponding retinal images.

The experimental results show the impact of the blind spot size and of the position of the fixation point on the performance of the proposed solution, and are achieved for the other parameters set as follows: the distance threshold for the planar surface removal procedure is t = 1cm, the distance threshold for point cloud cluster extraction is d_{th} = 5mm, and a border of 20 pixels around the bounding window is used given that samples considered here were acquired with the Kinect sensor located at about 75 cm from the object. The results in Figures 4.18 to 4.21 show the contours of the object held by the fingers of the robotic hand for different sizes of the blind spot accompanied by various positions of the 2D fixation point, depicted as a green dot over the object in the input Cartesian image. The corresponding area

where the object surface is not mapped to the cortical image is depicted as a black dot in the retinal image, which enlarges with the size of the selected blind spot.

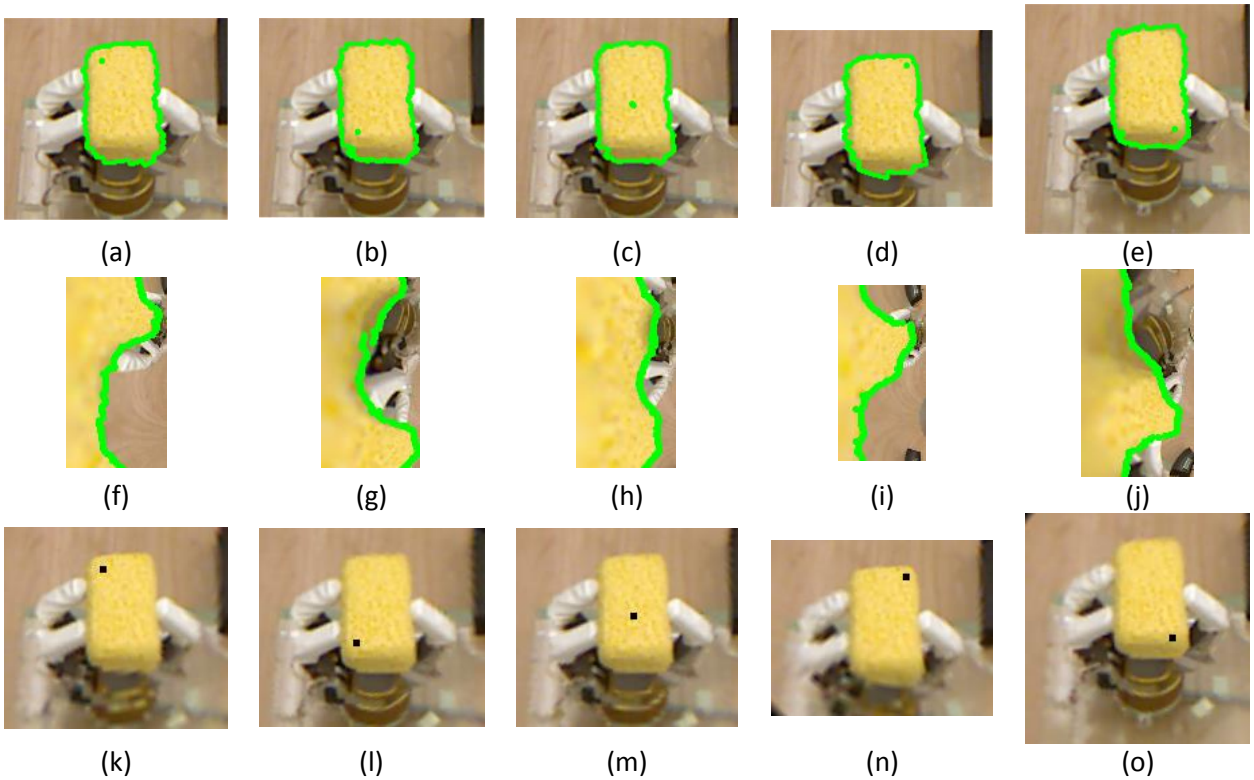


Figure 4.19: Contour detected with blind spot size of ρ_0=3 pixels: 2D fixation point at (a) the top-left corner of the object; (b) the bottom-left corner of the object; (c) the center of the object; (d) the top-right of the object; (e) the bottom-right of the object; (f-j) corresponding cortical images; and (k-o) corresponding retinal images.

Comparing the contour detection results for different positions of the 2D fixation point in the first row of Figure 4.18 to 4.21, we can see that, when the 2D fixation point is located in the central area of the object of interest, the contour results are smoother compared to the cases where the 2D fixation point is located close to a corner of the object in general. Although not all of the detected contours fit perfectly with the object of interest, the fast level set method still works well on detecting those contours. In order to minimize the noise on the detected contours, we recommend the selection of the 2D fixation point to be as close to the center of the object of interest as possible.

Each column of Figures 4.18 to 4.21 shows the Cartesian image, the corresponding cortical image transformed with the 2D fixation point for the one of the five different positions on the object, and the retinal image resulting from the inverse log-polar mapping. By comparing the sizes of the Cartesian images to their corresponding cortical images, a reduction of size can be observed for the cortical images. All the cortical images and the retinal images of Figures 4.18 to 4.21 are obtained with different values for the blind spots, where the pixels within the radius of the blind spot are not mapped to the cortical images in the second row of each figure, and the corresponding blank area can be observed as a black point on the object in the last row of each figure.

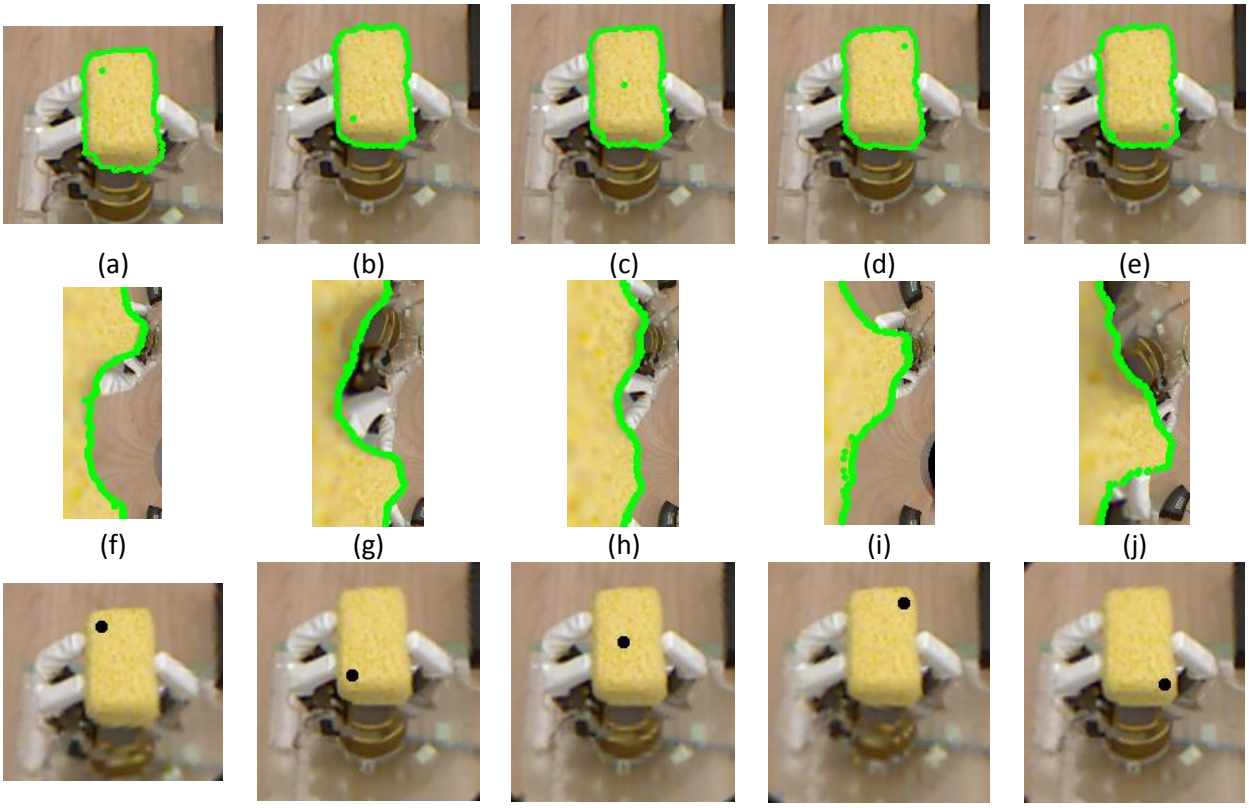


Figure 4.20: Contour detected with blind spot size of ρ_0=5 pixels: 2D fixation point at (a) the top-left corner of the object; (b) the bottom-left corner of the object; (c) the center of the object; (d) the top-right of the object; (e) the bottom-right of the object; (f-j) corresponding cortical images; and (k-o) corresponding retinal images.

The size of the blind spot affects the size of the sector, and therefore the height of the cortical image resulting from the log-polar mapping. Equations in section 2.3.4 detail that the size of blind spot affects the size of the cortical domain. Comparing Figure 4.18 to 4.21, one can notice that the height of the cortical image increases with an increase in the size of the blind spot. In general, a larger cortical image can provide more accurate contour information, so that the detected contour appears smoother when remapped in the Cartesian domain. However, this is not always the case. One can observe in Figure 4.21b that one corner of the object contour, surrounding the selected fixation point, is missing. This is explained by the fact the blind spot area is so large in that case that the section of the object contour in the Cartesian image is not transformed to the cortical image, as it can be seen in the center part of Figure 4.21g, where the green line is missing as that important section of the object is occluded by the blind spot.

In summary, to make the selection of the 2D fixation point flexible and robust, it is recommended setting the parameter of the blind spot size (ρ_0) to a value that will not affect the relevant pixels corresponding to the contour of an object. On the other hand, the detected contour is preferably smooth when remapped in the Cartesian domain to enable a reliable comparison for material characterization. For that reason, the value of ρ_0 cannot be too low neither. In this work, a value of ρ_0 = 3 pixels is empirically selected for all test cases as it provides a proper balance between these two

constraints. For this value, Figures 4.19a to 4.19e demonstrate that the results are relatively robust with respect to the location of the 2D fixation point, marked by a green dot in the input Cartesian images. It can be noted that a more central position of the 2D fixation point also tends to lead to a smoother contour, while the contour is more jagged if the 2D fixation point is selected along the sides of the object.

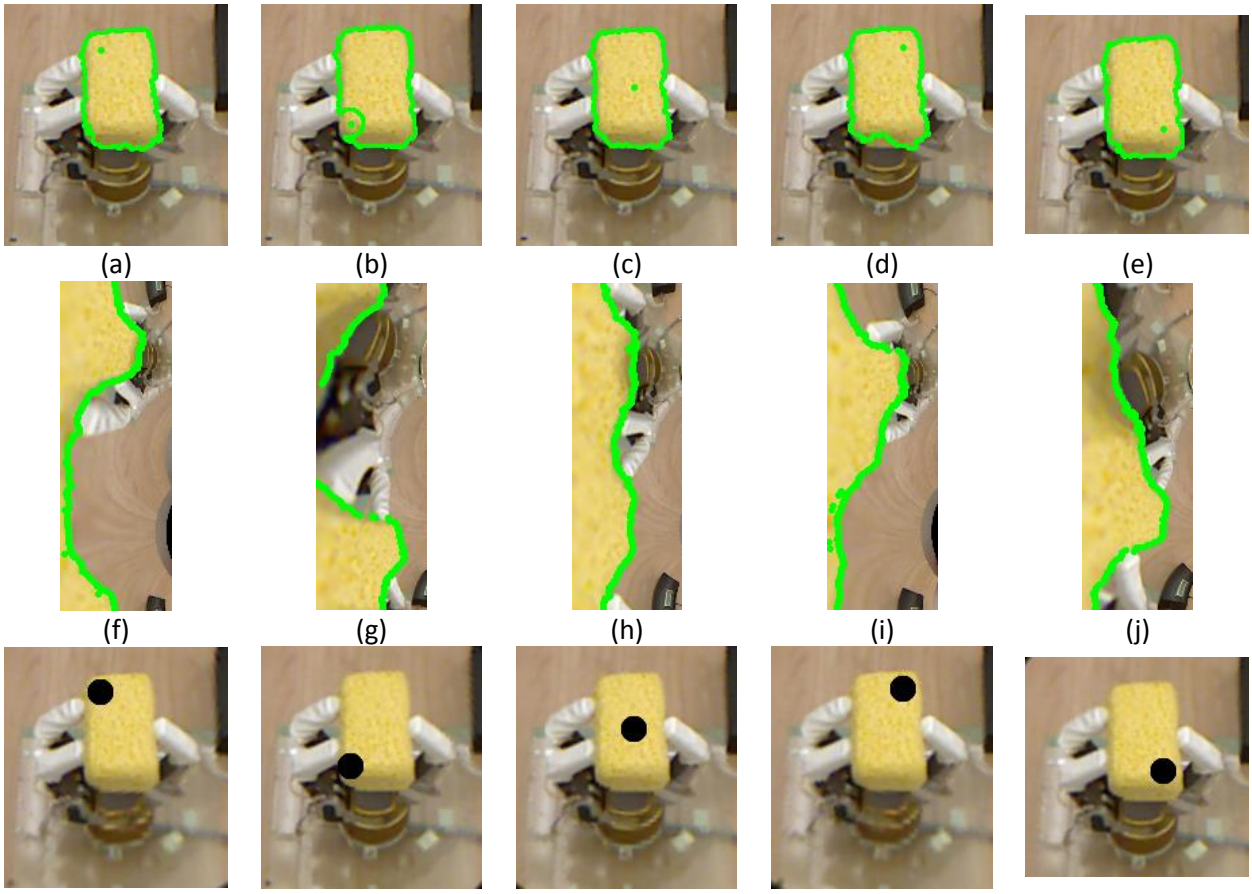


Figure 4.21: Contour detected for blind spot size of ρ_0=10 pixels: 2D fixation point at (a) the top-left corner of the object; (b) the bottom-left corner of the object; (c) the center of the object; (d) the top-right of the object; (e) the bottom-right of the object; (f-j) corresponding cortical images; and (k-o) corresponding retinal images.

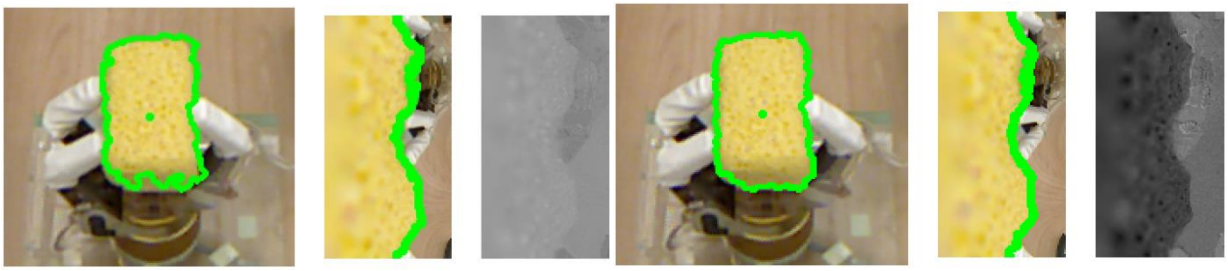
4.2.5 Color Selection Scheme

The proposed algorithm for color space selection computes, in the cortical domain, the mean intensity values and their corresponding standard deviations for each column in the U and V components, respectively, with the purpose of selecting one of them as a feature for the fast level set method. In particular, the component having the larger standard deviation is chosen. As discussed in section 3.3.5, a large standard deviation value indicates that the mean values of the columns are more widely distributed, while a small standard deviation value shows a tendency for all the values considered to be similar to the mean. As a stronger contrast of color is preferable to enable a simpler and faster contour

detection process, the component with a larger standard deviation among the U or the V channels is selected.

The following figures where five different locations of the 2D fixation point are considered show the contour detection results in the Cartesian domain and in the cortical domain obtained by using the U or the V component as features for the fast level set method, respectively. All the experimental results are achieved for the same tuning of the parameters as defined in the previous sections.

Figure 4.22 shows the contour tracking results in the Cartesian domain and in the cortical domain, as well as the U and V components of the initial image encoded in the YUV color space. Figure 4.23 presents the mean values computed over the columns for the U and V components respectively. It can be noted that the Cartesian contour (Figure 4.22d) obtained from Figure 4.22f is more precise and sharper compared to the one obtained from Figure 4.22c and shown in Figure 4.22a. Comparing Figure 4.22c and Figure 4.22f, it can be noted as well that the difference between the object and the background is clearer for the V component than from the U component in this case. The same conclusion can be reached by studying the mean values in Figure 4.23. We can observe that the mean values of the columns of the U component over the image tend to be relatively constant (blue line), while the mean values of the V component are quite variable (red line). These aspects justify the choice of the V component as the feature for the level set in this case.



(a) (b) (c) (d) (e) (f)

Figure 4.22: Contour detection with 2D fixation point at the center of the object: (a) Cartesian contour obtained from (c), (b) cortical contour obtained from (c), (c) U component of YUV color coded cortical image; (d) Cartesian contour obtained from (f), (d) cortical contour obtained from (f), and (f) V component of YUV color code cortical image.

The location of the 2D fixation point on the Cartesian image affects the distribution of its cortical image. For example, when the 2D fixation point is roughly selected in the center of the object, it leads a log-polar image with two distinct regions separated along the vertical direction (Figures 4.22b and 4.22e), corresponding to a clear change in the mean values in between the columns of the cortical image (Figure 4.23). Even when the 2D fixation point is not in the center of the yellow sponge, as shown in Figures 4.24, 4.26, 4.28 and 4.30, the mean values in the columns of the V component have significant changes compared with the U component, for this test scenario, which results in a more accurate and smoother contour being detected using the V component rather than using the U component, independently from the location of the fixation point.

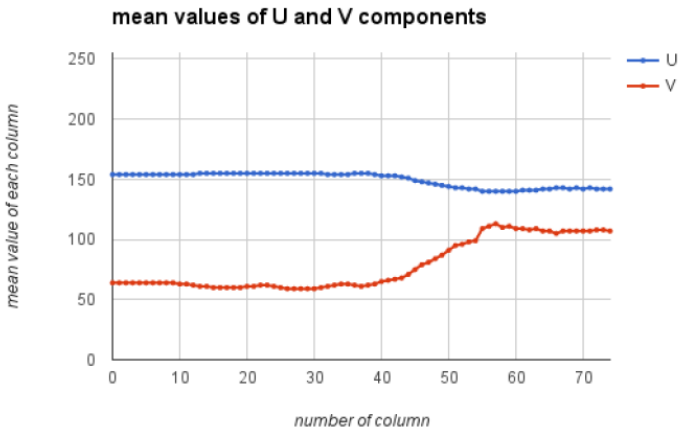
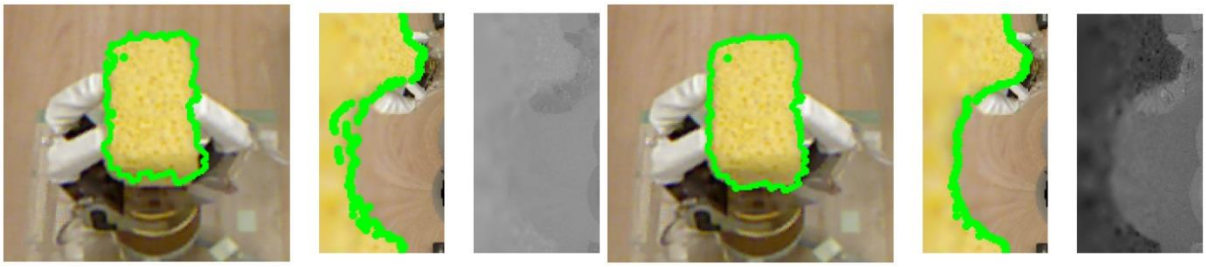


Figure 4.23: Mean values of U and V components in each column of the cortical images.

As expected, the experimental results reveal that the choice of the fixation point has no impact on the color component selection, which only depends on the trend of the intensity value of each column of the cortical image on a particular color component.



(a) (b) (c) (d) (e) (f)

Figure 4.24: Contour detection with the 2D fixation point at the top-left corner of the object: (a) Cartesian contour obtained from (c), (b) cortical contour obtained from (c), (c) U component of YUV color coded cortical image; (d) Cartesian contour obtained from (f), (d) cortical contour obtained from (f), and (f) V component of YUV color coded cortical image.

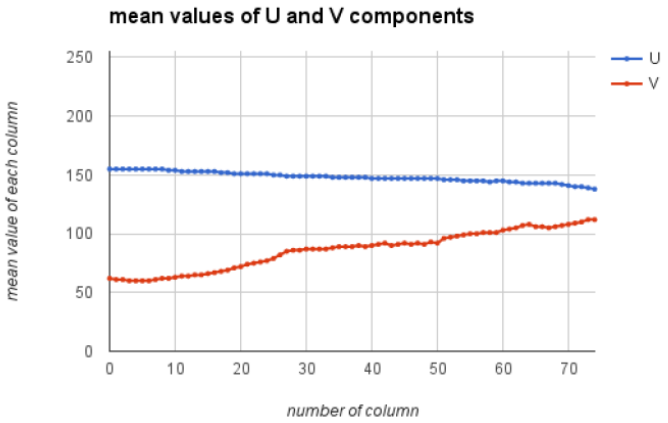
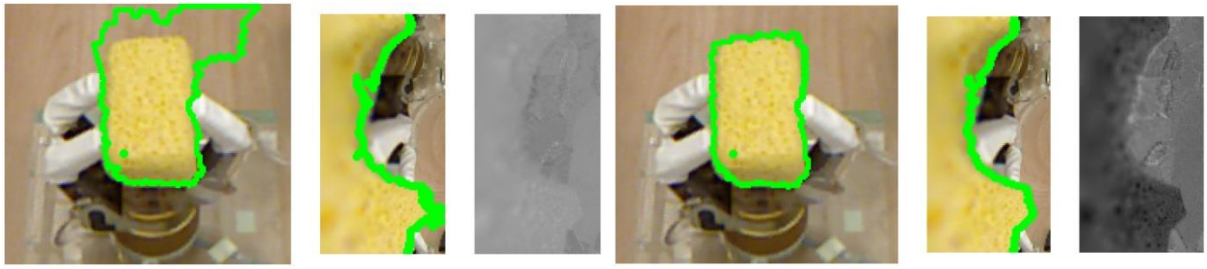


Figure 4.25: Mean values of U and V components in each column of the cortical images in Figure 4.24.



(a) (b) (c) (d) (e) (f)

Figure 4.26: Contour detection with the 2D fixation point at the bottom-left corner of the object: (a) Cartesian contour obtained from (c), (b) cortical contour obtained from (c), (c) U component of YUV color coded cortical image; (d) Cartesian contour obtained from (f), (d) cortical contour obtained from (f), and (f) V component of YUV color coded cortical image.

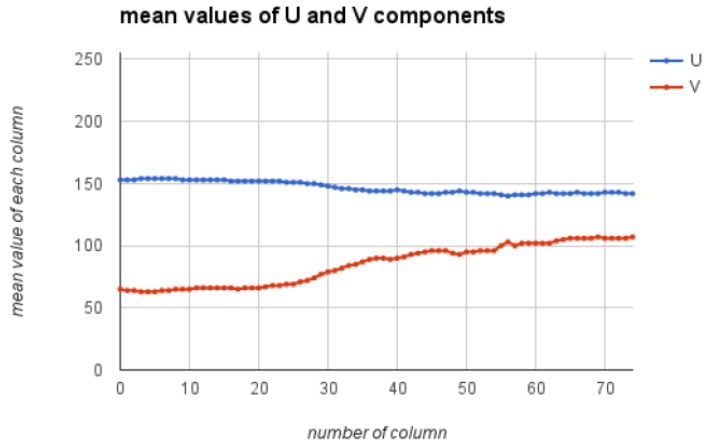
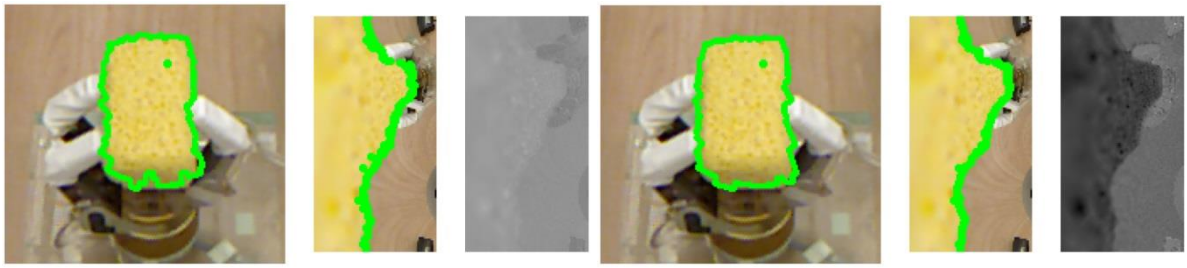


Figure 4.27: Mean values of U and V components in each column of the cortical images in Figure 4.26.



(a) (b) (c) (d) (e) (f)

Figure 4.28: Contour detection with the 2D fixation point at the top-right corner of the object: (a) Cartesian contour obtained from (c), (b) cortical contour obtained from (c), (c) U component of YUV color coded cortical image; (d) Cartesian contour obtained from (f), (d) cortical contour obtained from (f), and (f) V component of YUV color coded cortical image.

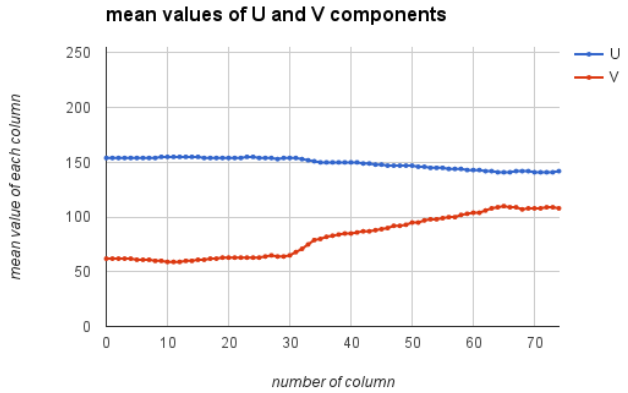
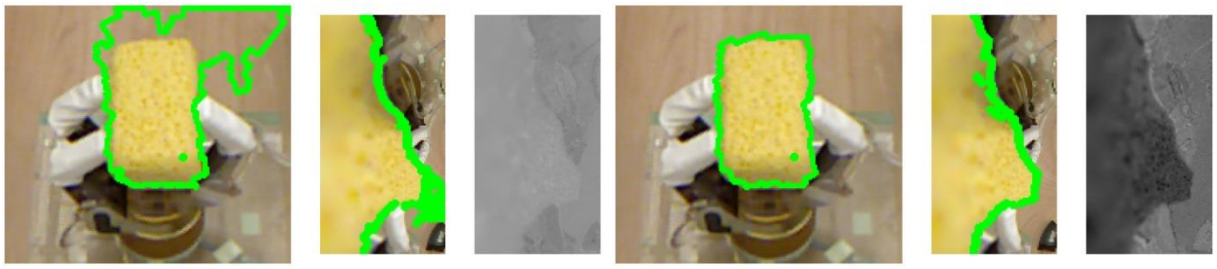


Figure 4.29: Mean values of U and V components in each column of the cortical images in Figure 4.28.



(a) (b) (c) (d) (e) (f)

Figure 4.30: Contour detection with the 2D fixation point at the bottom-right corner of the object: (a) Cartesian contour obtained from (c), (b) cortical contour obtained from (c), (c) U component of YUV color coded cortical image; (d) Cartesian contour obtained from (f), (d) cortical contour obtained from (f), (f) V component of YUV color coded cortical image.

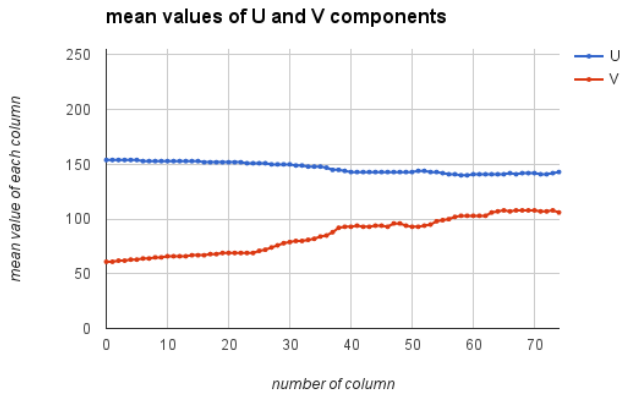
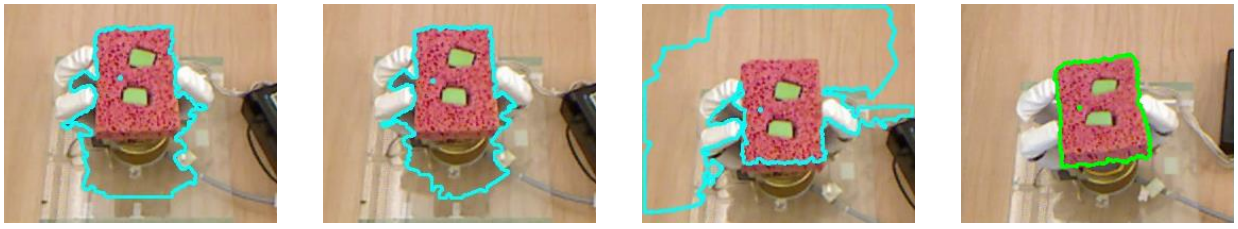


Figure 4.31: Mean values of U and V components in each column of the cortical images in Figure 4.30.

4.2.6 Contours Extraction from RGB Color Space

Since the color frame captured from the Kinect sensor is a 3-channel RGB frame, the system could alternatively operate directly in the RGB space, therefore avoiding the transformation from RGB to YUV color space that is proposed. Experiments were conducted to evaluate how the contours detected directly from the RGB color space compare, in relevance and accuracy, to contours obtained when the proposed approach to automate color selection from the U or V component (detailed in section 3.3.5) is used.



(a) (b) (c) (d)

Figure 4.32: Contours of object of interest are detected: (a) from B color component, (b) from G color component, (c) from R color component, and (d) from U color component, selected using the automatic color selection scheme enabled by the log-polar transformation.

Figure 4.32 presents a representative sample case from these experiments. Figure 4.32a-c shows the contour detected from the B, G, and R color components, respectively. Figure 4.32d shows the contour extracted on the same view, but from the U color component, which was selected with the proposed automatic color component selection scheme. The close contours extracted by applying fast level set in the RGB color space (Figure 4.32a-c) are not entirely matching with the actual contour of the object of interest. This is mainly due to the R, G and B color components' sensitivity to brightness. In contrast, the object's contour detected from the U selected color component presents a much more accurate object contour detection. This phenomenon was observed on several samples tested. Therefore, it was concluded that the RGB color space is not the optimal choice for detecting the contour of the object of interest and the framework for automated color component selection in the YUV color space was developed.

4.2.7 Contour Extraction in the Cartesian Domain

Since the color frame captured from the Kinect sensor is a Cartesian image, it was also considered that the system could operate directly in this domain, therefore avoiding the transformation to the log-polar domain. This could theoretically save some computation time. However it may increase the necessary human intervention on the system to select the proper area of interest. This is due to the fact that such a solution removes the possibility to rely on the proposed automatic color selection scheme that operates from an on-line analysis of the contrast over a neat separation line in between the object and the background. The loss of the concentration of pixels of interest, surrounding the user selected fixation point, for a more arbitrary distribution of pixels corresponding to the object of interest over the Cartesian map, makes the cluster extraction more difficult, and therefore impedes on the final contour extraction. This section reports on experiments conducted to compare contours detected from an

original Cartesian image to contours extracted with the proposed method over a transformed cortical image. Figure 4.33 presents a representative sample case from these experiments.

(a) (b)

Figure 4.33: Contours of object of interest: (a) detected over the original Cartesian image, (b) detected over the transformed cortical image.

Figure 4.33a illustrates how the detected contour encloses the object of interest as well as the fingers of the robotic hand, which are not part of the desired contour. In contrast, the contour of the object of interest is detected with more accuracy when the proposed scheme is applied, involving the use of the log-polar mapping and the automated color selection. Timewise, it takes about 307 ms for extracting the object's contour from the original Cartesian image in Figure 4.33a, while it takes about 253 ms to obtain a more accurate object's contour from the log-polar domain. The expected reduction in processing time is not observed because of the way information is organized in the log-polar domain (as presented in section 3.3.4) that favors the contour convergence.

4.2.8 Influence of the Gaussian Filter

In the proposed fast level set method, the initial curve evolves according to the speed function, which has two components: the data-dependent speed function and the curve smoothness regularization speed function. The latter is simplified as a Gaussian filter, as it was detailed in section 3.3.6. Experiments were also performed to study the impact of the Gaussian filter on the fast level set.

(a) (b)

Figure 4.34: Contours of object of interest are detected: (a) without Gaussian filter, (b) with Gaussian filter.

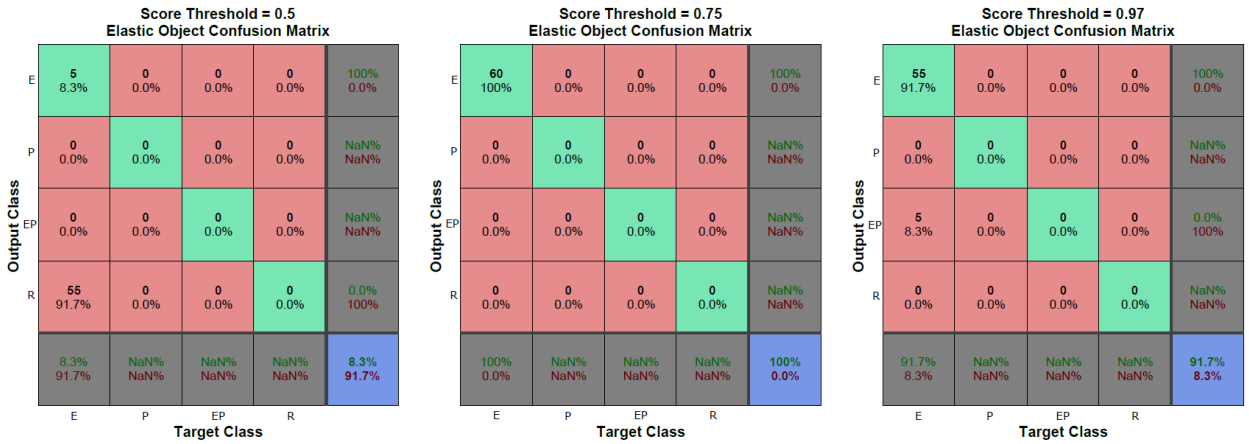
As can be seen in Figure 4.34a, the detected contour of the object of interest without a Gaussian filter is relatively rough given the low resolution of the camera, and especially in some regions such as the partial contour that is circled in yellow. In contrast, the contour extracted with a Gaussian filter, in Figure 4.43b, is smoother and it more accurately represents the contour of the object of interest.

In general, our experimental results revealed similar behaviors, which are expected, as discussed from a theoretical point of view in sections 2.3.2. and 3.3.6. Therefore, it was concluded that the Gaussian filter should advantageously be used as part of the fast level set method.

4.2.9 Influence of Score Threshold

Once three representative contours of the object are obtained from the sequence of RGB-D data (as detailed in section 3.4.2), the conditional DTW algorithm (Algorithm 2.4) is applied to establish a pairwise correspondence between pixels from these three contours. The classification system reaches a decision according to the process shown in Figure 3.22, where the calculated score value (defined in section 3.4.3) is compared to a score threshold.

A series of experiments is performed to study the impact of the value of the score threshold, thr, on the proposed classification approach. The scores are computed pairwise between the initial contour, the contour under largest deformation, and the final contour after the force is removed. Only if the value of the score is lower than its set threshold, does the classification system consider that a deformation occurs. To evaluate the impact of the threshold value on the object classification, tests are repeated 60 times for three different threshold values and the results are shown in Figure 4.35. The figure presents the results under the form of confusion matrices [81], obtained on an elastic object for a threshold value of 0.5, 0.75 and 0.97 respectively.



(a) (b) (c)

Figure 4.35: Classification results presented as confusion matrices for an elastic object and for various values of the score threshold: (a) thr = 0.5; (b) thr = 0.75; (c) thr = 0.97. The material type is denoted as E (elastic), P (plastic), EP (elasto-plastic), and R (rigid).

In the confusion matrix, E, P, EP and R denote the material type, which are elastic, plastic, elasto-plastic and rigid, respectively. Each row represents the predicted class (Output Class), while each column shows the true class (Target Class), which is considered as the ground truth, determined from the prior knowledge that we have about each object. Correct classifications (diagonal green cells) and misclassifications (off-diagonal red cells) are highlighted, and their corresponding average success and failure rates are calculated on the confusion matrix. The gray column on the far right of the matrix represents the overall accuracy for each predicted class, and the gray row at the bottom of the matrix

represents the overall accuracy for each true class. The bottom right blue cell shows the overall accuracy across all categories of material considered.

Given that an object made of elastic material is considered in this case, which is the yellow sponge shown in previous examples, the confusion matrix matches the predicted class of an object (output of the classifier) against the true class that is expected (an elastic object) in order to determine the correct classification rate (highlighted in green characters) and the misclassification rate (highlighted in red characters). The value of the score threshold, thr, under examination here, is chosen as the lowest value that achieves the highest correct classification rate. One can observe that, in this case, the object is classified with an accuracy of 100% as the elastic object in Figure 4.35b, for a threshold value of 0.75. In this thesis, thr is therefore empirically set to 0.75.

Experiments also revealed that for low threshold values (e.g. thr = 0.5 as in Figure 4.35a), the approach is insufficiently responsive to slightly different contours under the lighter grasping force. Because of the low threshold value, the object is misclassified as rigid in almost all the test cases (up to the 91.7% of the trials). On the other hand, for larger threshold values (e.g. thr = 0.97 as in Figure 4.35c), the proposed classification approach is overly sensitive and therefore is easily impacted by noise which eventually degrades the classification performance. In this case, the impact of a large threshold value is that the object is categorized as either elastic (91.7%), which is correct, or elasto-plastic object (8.3%), which is wrong.

4.3 Influence of Kinect Position and Orientation

In this section, the impact of various configurations of the Kinect sensor with respect to the object is studied. Several experiments are performed, including tests for various distances of the Kinect sensor with respect to the object, various positions of the Kinect sensor around the object, and various initial orientations of the object within the robotic hand.

4.3.1 Various Positions of Kinect Sensor

The aim of this section is to demonstrate that the proposed tracking approach is able to correctly identify the object contour regardless of the position of the Kinect sensor with respect to the object. The following figures exhibit the tracking results for eight different positions (upper, lower, left, right, upper-left, upper-right, lower-left, and lower-right with respect to the robotic hand) of the Kinect sensor over the object as illustrated in Figure 4.36. For these experiments, the Kinect sensor, with a fixed distance of 55cm from the surface of the object, is mounted on a tripod as shown in Figure 4.37a.

The parameters used for these tests are defined as follows: t = 1cm, d_{th} = 5mm, bounding window enlargement= 20 pixels, and ρ_0 = 3.

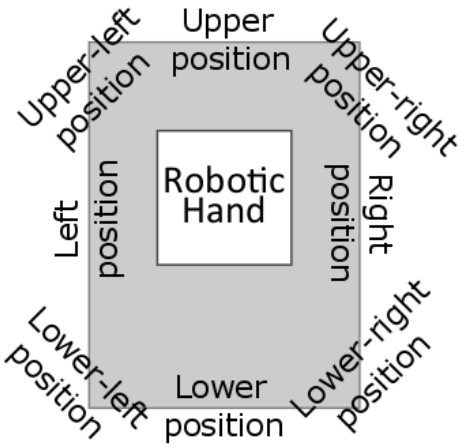
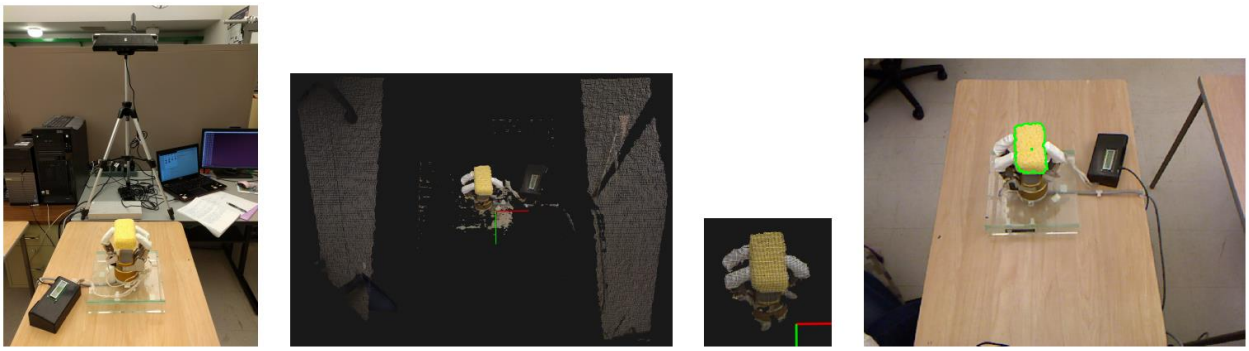


Figure 4.36: Positions of the Kinect sensor with respect to the object and robotic hand.

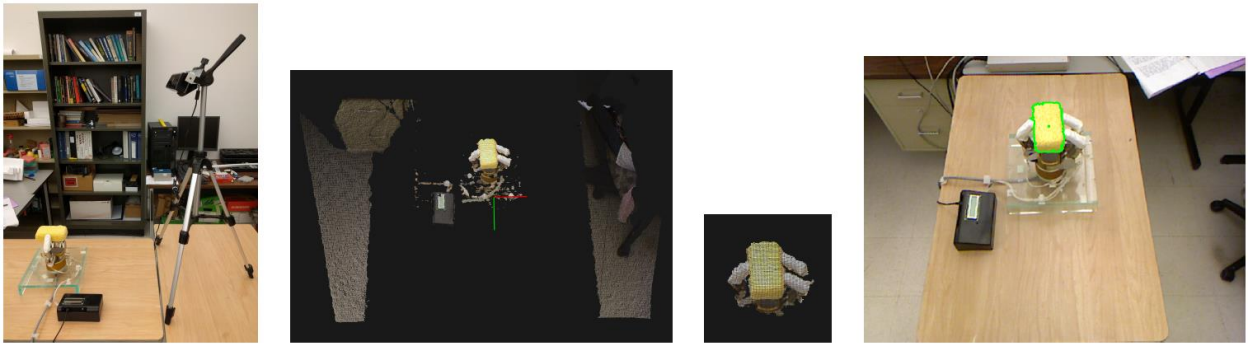
Figure 4.37a shows the Kinect sensor placed on the upper position, which is the common position for sensor in the thesis. In Figure 4.37b, the points of the planar surface (with the model $(-0.00352003)x + (0.464335)y + (0.885653)z + (-0.871453) = 0$) are removed from the point cloud. Figure 4.37c shows the extracted cluster surrounding for 3D fixation point, computed through 2D - 3D mapping of the user selected input point. The user input point (282, 150) is denoted by the green dot in Figure 4.37d. The latter also exhibits the tracked contour, in green.



(a) (b) (c) (d)

Figure 4.37: Upper position: (a) setup, (b) point cloud after background plane extraction, (c) cluster extracted, and (d) contour detection result.

Figure 4.38a shows the Kinect sensor located on the position denoted as “lower position” in Figure 4.36. Using the same parameters, Figure 4.38b shows the point cloud after the background table removal procedure (with the planar surface model $(0.00743257)x + (0.477439)y + (0.878633)z + (-0.866308) = 0$). The clustering works well as shown in Figure 4.38c. An example of the detection contour around the 2D fixation point (311, 118) can be seen in Figure 4.38d.



(a) (b) (c) (d)

Figure 4.38: Lower position: (a) setup, (b) point cloud after plane extraction, (c) cluster extracted, and (d) contour detection result.

Figure 4.39 shows the Kinect sensor positioned on the left side of the robotic hand, the point cloud with the background plane removed (with model $(0.0047889)x + (0.481047)y + (0.876682)z + (-0.867559) = 0$), the cluster representing the object in 3D, and the contour around the 2D fixation point (285, 191), shown in Figure 4.39d.

(a) (b) (c) (d)

Figure 4.39: Left position: (a) setup, (b) point cloud after plane extraction, (c) cluster extracted, and (d) contour detection result.

Figures 4.40 exhibits the result of each step when the Kinect sensor is placed on the position denoted as "right position" with respect to the robotic hand. The points composing the background table (with the planar surface model $(0.00929031)x + (0.480185)y + (0.877118)z + (-0.86441) = 0$) are well removed from the point cloud. A set of remaining points are clustered to represent the object of interest. The contour tracking based on the point cloud cluster is shown in Figures 4.40d for a user input point of coordinates (300, 213).

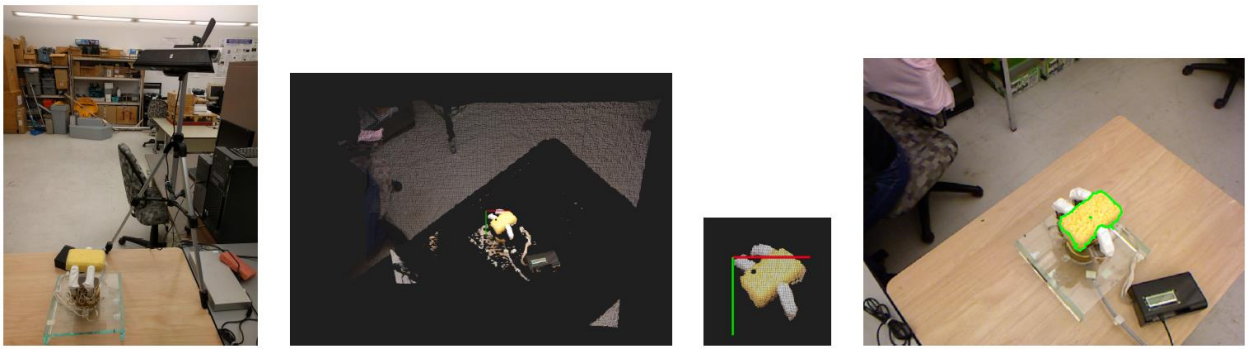
Figures 4.37 through 4.40 show the result of each step from the contour tracking system when the Kinect is placed in upper, lower, left and right positions, respectively. According to the obtained point cloud after background plane extraction, the cluster of the object of interest, and the detected contour, it is observed that the different positions of the Kinect sensor do not affect the performance of the proposed contour extraction system. In order to further demonstrate that the location of the Kinect has

no effect on the proposed contour tracking system, the following figures show the contour detection results when the sensor is located at an angle with respect to the object and robotic hand.

(a) (b) (c) (d)

Figure 4.40: Right position: (a) setup, (b) point cloud after plane extraction, (c) cluster extracted, and (d) contour detection result.

Figure 4.41 shows the Kinect sensor placed in the upper-left position with respect to the robotic hand and the result of each significant step. By processing the point cloud read from the Kinect from this position, the background plane (with the model $(0.00524693)x + (0.490637)y + (0.871348)z + (-0.870766) = 0$) is successfully extracted as illustrated in Figure 4.41b. The cluster recovered from the point cloud after plane extraction is shown in Figure 4.41c. The extracted contour of the object of interest detected based on the 2D fixation point of coordinates (381,265) is shown in Figure 4.41d.



(a) (b) (c) (d)

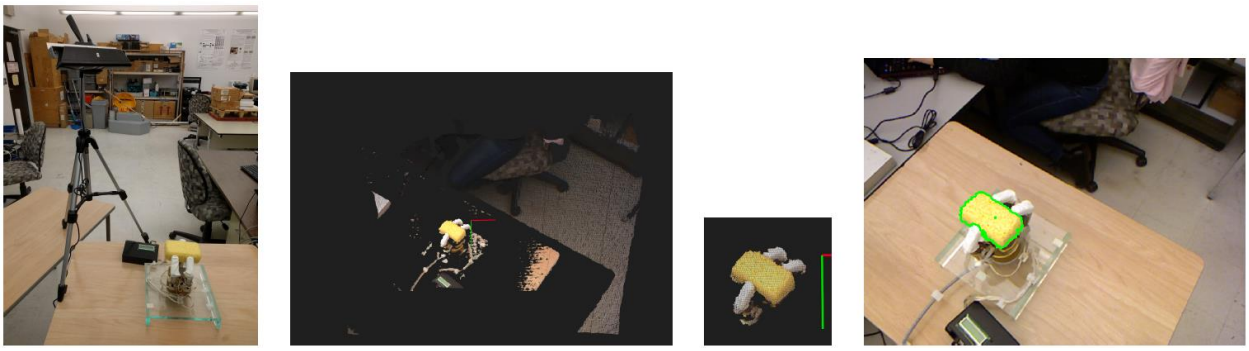
Figure 4.41: Upper-left position: (a) setup, (b) point cloud after plane extraction, (c) cluster extracted, and (d) contour detection result.

In Figure 4.42a, the Kinect is positioned in the upper-right position with respect to the robotic hand. From the point cloud and the RGB image read from Kinect sensor and the 2D fixation point (216,274) specified by the user, all the points composing the background plane model $((0.00895816)x + (0.472622)y + (0.88122)z + (-0.872909) = 0$) are discarded when we cluster the points of the object from the point cloud after plane extraction. The extracted contour is correctly placed around the object of interest (Figure 4.42d).

(a) (b) (c) (d)

Figure 4.42: Upper-right position: (a) setup, (b) point cloud after plane extraction, (c) cluster extracted, and (d) contour detection result.

Figure 4.43 shows the Kinect in the lower-left position of the robotic hand. The points belonging to the background plane model $((0.0134665)x + (0.482376)y + (0.875861)z + (-0.873357) = 0)$ are extracted from the point cloud read from the Kinect. The points of the object are grouped together as shown in Figure 4.43c. The contour detected is shown in Figure 4.43d for a user input point of coordinates (221,267).

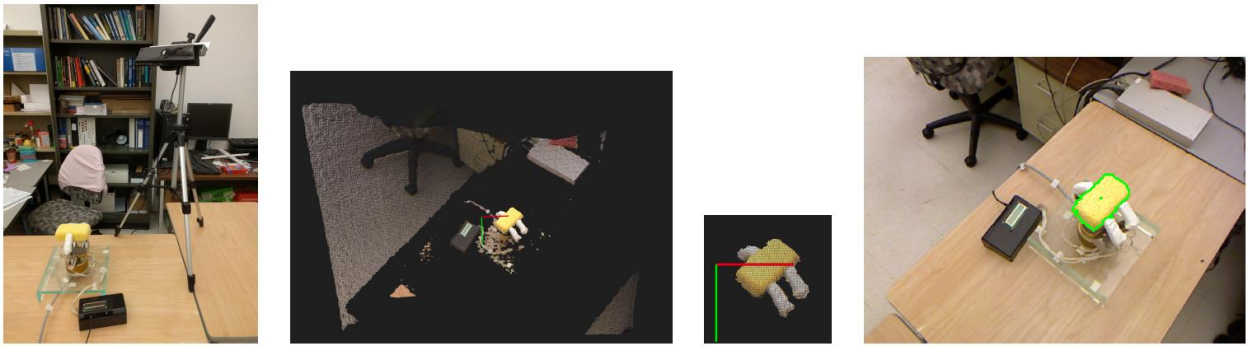


(a) (b) (c) (d)

Figure 4.43: Lower-left position: (a) setup, (b) point cloud after plane extraction, (c) cluster extracted, and (d) contour detection result.

Finally, the point cloud and the RGB image are captured from the Kinect sensor placed in the lower-right position with respect to the robotic hand (Figure 4.44a). Figure 4.44b shows the point cloud after background plane extraction (with the planar surface model $(-0.00967231)x + (0.484017)y + (0.875005)z + (-0.875494) = 0$), and Figure 4.44c shows the cluster recovered from it. Figure 4.44d shows the detected contour over the RGB image with a user input point of coordinates (397,237).

In conclusion, tests were performed for 8 positions of the Kinect sensor surrounding the robotic hand (Figure 4.36). Figures 4.37-4.44 demonstrate the robustness of the approach for different locations of the Kinect sensor, which indicates that the proposed contour detection system correctly captures the object contour independently of the placement of the Kinect sensor.

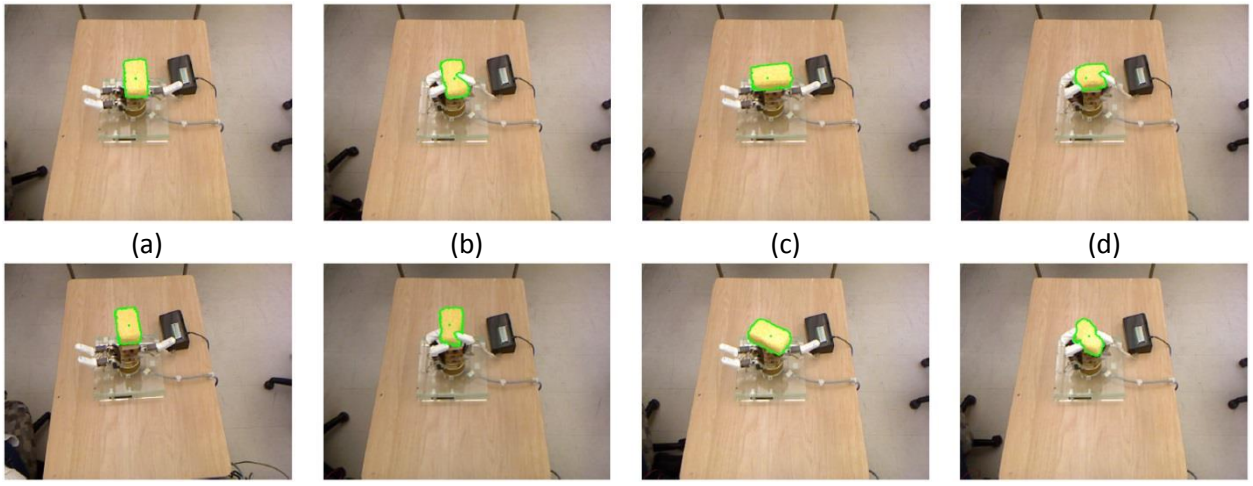


(a) (b) (c) (d)

Figure 4.44: Lower-right position: (a) setup, (b) point cloud after plane extraction, (c) cluster extracted, and (d) contour detection result.

4.3.2 Various Orientations of Object

The proposed contour detection approach is also capable of extracting the object contour regardless of the location at which the object is placed in the robotic hand, as it is demonstrated by the experimentation in this section. Various objects with different properties, under various starting conditions (i.e. placed with different orientations within the robot fingers), and under the manipulation of various grasping forces are tested. The experiments are conducted in order to evaluate the robustness of the proposed object contour detection system with respect to various positions and orientations of the object, and the results are shown in Figure 4.45 through 4.47.



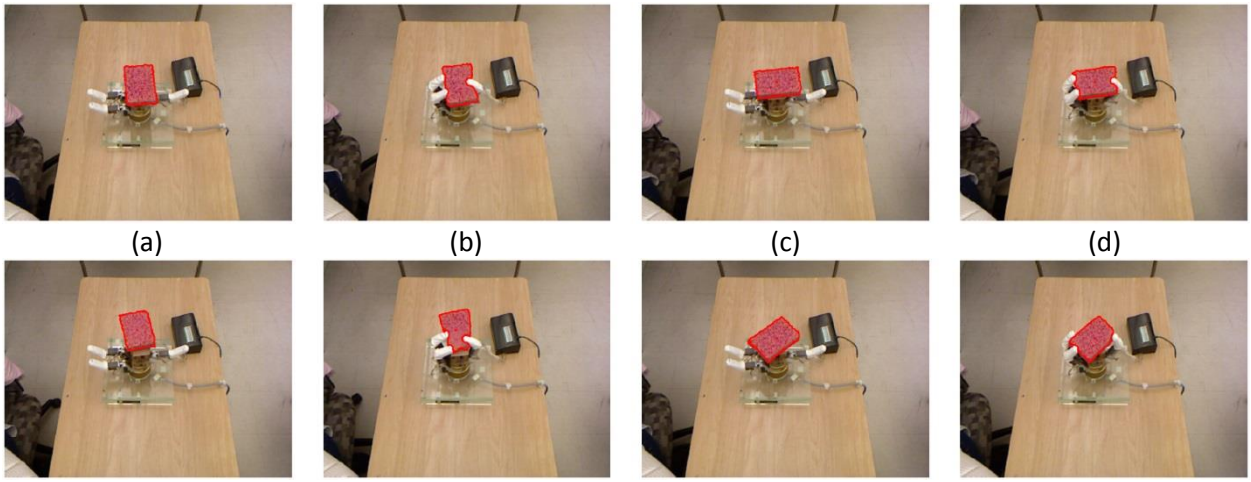
(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 4.45: Robustness of detected contours on an elastic yellow sponge at various stages of deformation: (a-b) object positioned horizontally along its short side; (c-d) object positioned horizontally along its long side; (e-f) object shifted slightly along its long side; and (g-h) object rotated by 45°.

Figures 4.45, 4.46, and 4.47 demonstrate that the contours around each object are correctly identified throughout the manipulation sequence. These results indicate that the proposed contour tracking system can successfully detect and track the contour of the object of interest regardless of the initial

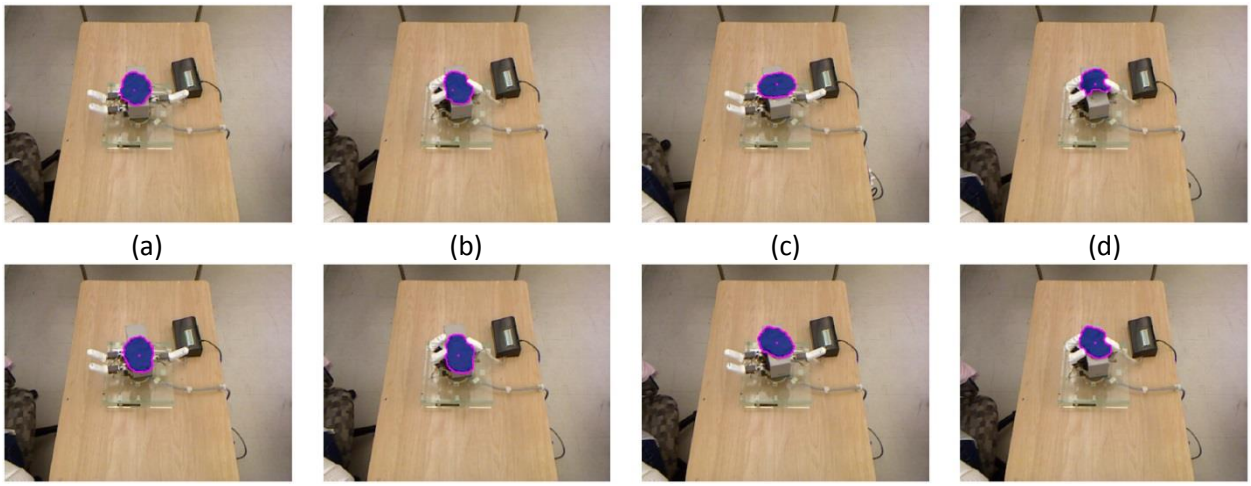
position of the object in the robotic hand, and under several magnitudes of applied force and levels of deformation.



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 4.46: Robustness of detected contours on a red elastic sponge at various stages of deformation: (a-b) object positioned horizontally along its short side; (c-d) object positioned horizontally along its long side; (e-f) object shifted along its long side; and (g-h) object rotated by 45°.



(a) (b) (c) (d)

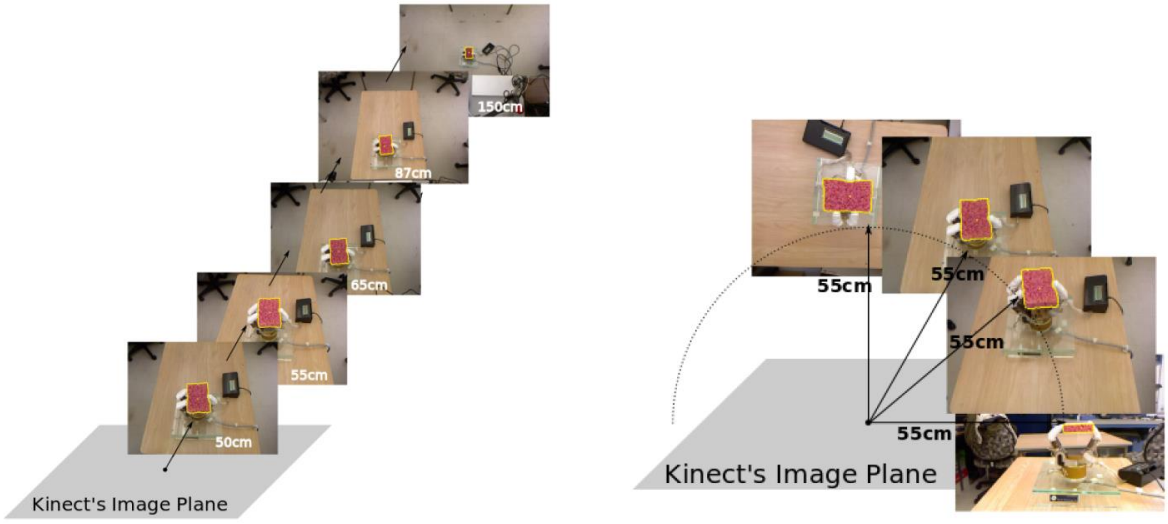
(e) (f) (g) (h)

Figure 4.47: Robustness of contours detection on blue plasticine dough (with plastic characteristics) at various stages of deformation: (a-b) object positioned horizontally along its short side; (c-d) object positioned horizontally along its long side; (e-f) object shifted along its long side; and (g-h) object rotated by 45°.

4.3.3 Sensor-to-Object Distance

The tests shown in Figure 4.48a are conducted to determine the limits on the distance at which the Kinect sensor can be located with respect to the surface of the object for its contour to be reliably extracted and tracked. The contour can be extracted over a range from 50cm up to 150cm separating the Kinect sensor from the object. However, the relative sensor-to-object distance has an impact on the performance of the object classification procedure since the object of interest appears smaller in the

image with a larger sensing distance, resulting in fewer details over the object contour to properly support the classification in the four categories of material considered. As a result, when only light forces are applied on the object, the magnitude of deformation is limited and may be insufficient to be perceived during contour tracking from a low resolution image. Therefore, the best range of distances to be used is between 55 and 75cm. In the example shown in Figure 4.48b, the Kinect sensor is positioned along the circumference of a semicircle of radius 55cm, centered at the object of interest. The tests indicate that the Kinect sensor can reliably be placed at any distance within the specified range from the object, and at various positions around the object of interest, while achieving reliable classification results, as will be discussed in section 4.5.



(a) (b)

Figure 4.48: (a) Effect of the distance between the surface of the object and the Kinect sensor; (b) various positions of the Kinect sensor with respect to the object.

4.4 Object Contour Detection and Tracking

In this section, the two proposed approaches for the detection and tracking of an object contour, detailed in section 3.3.6 are experimentally evaluated: The first one focuses on robustness, the second one on speed. In these two approaches, the RGB image and the point cloud are collected directly from the Kinect sensor with a frame rate of 30 Hz. The experiments are carried out on a 2.6 GHz Intel Core i7 with 8 GB of RAM laptop.

4.4.1 Robustness-Focused Contour Detection

This approach initializes the fast level set for each frame independently. In other words, the process of extracting the 3D object cluster from the background removal point cloud, projecting it to the color space and computing a bounding window is implemented as an initialization step for every frame.

As observed in Figure 4.49a-f, this object detection approach offers the capability to detect the object contour of the blue sponge with red inserts even under an occlusion caused by another object (Figure

4.49b); when the occluding object is moved over the 2D fixation point, the contour migrates to the occluding object (Figure 4.49c,d); the object contour of the blue sponge with red inserts is redetected after the occluding object is moved away (Figure 4.49e,f). This approach also exhibits the capability of correctly extracting the desired object contour (blue sponge with red inserts) even when another red sponge is introduced in the field of view, which has the same color as the internal details of the desired object (Figure 4.49g), that do not get confused as an inner contour. The fact that a re-initialization from the fixation point (except that the original user selection is preserved and the user does not need to intervene after the first frame) is performed at each iteration provides an interesting degree of robustness for extracting the object contour in scenarios where the scene is not well structured or occlusions happen. In the latter case, our experiments demonstrated that the recovery capabilities of the system are high.

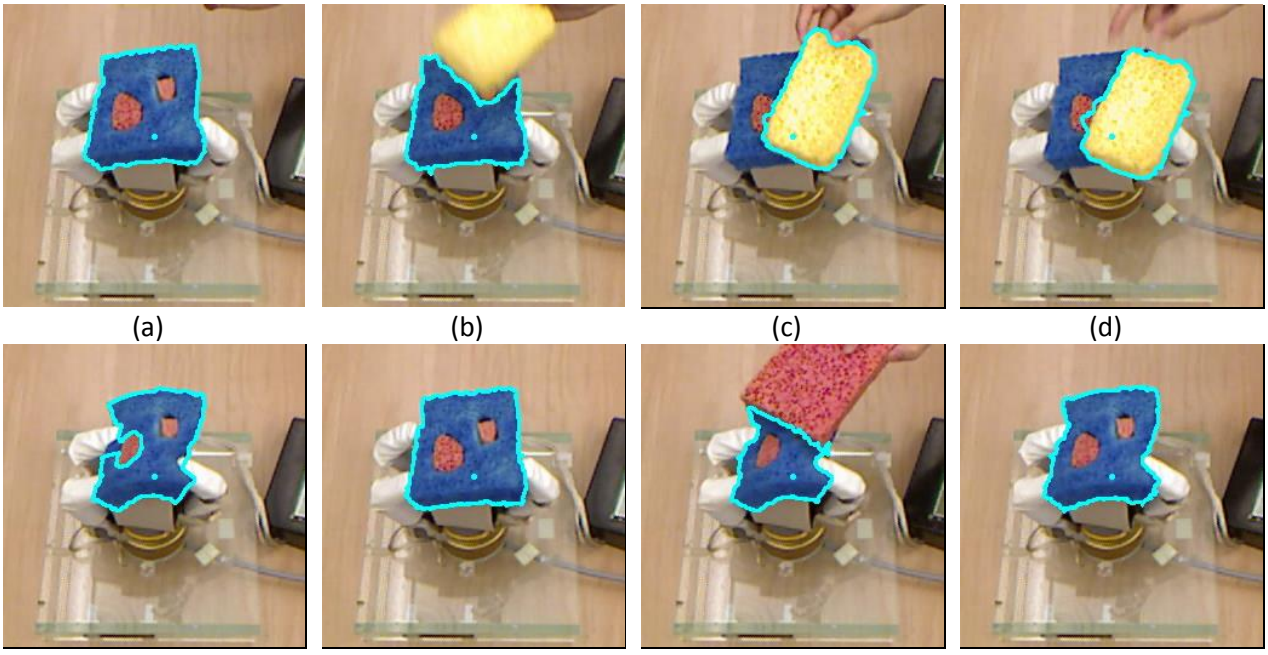


Figure 4.49: Robustness-focused contour detection: (a-e) a series of detected object contours with occlusions by a yellow sponge reaching over the 2D fixation point; and (f-h) a series of detected object contours with occlusions by a color conflicting sponge.

However, the penalty for robustness is that this approach requires a longer processing time to detect the contour on each frame due to a complete re-initialization. For each frame, the average detection time is 255.35 ms. Detailed statistics on the detection time per frame required for a variety of objects made of different materials, and under the three levels of force considered, are listed in Table 4.1.

The shape of object, its color, the type of material it is made of, and the amount of force applied do not influence significantly the required detection time per frame. This approach achieves an update rate of the contour of about 4 Hz due to the computationally expensive method for planar surface removal and clustering required to initialize the contour for every iteration. This is well under the 30 Hz frame rate supported by the Kinect sensor to acquire in parallel the RGB image and the point cloud.

Table 4.1: Contour detection time per frame (ms)

	Object	Force	Detection time (ms)
Elastic Objects	Blue+red sponge 	Light	262.05
		Medium	262.74
		Strong	266.81
	Red+green sponge 	Light	254.19
		Medium	258.73
		Strong	257.73
	Yellow sponge 	Light	256.81
		Medium	258.20
		Strong	257.40
Plastic Objects	Blue plasticine dough 	Light	253.29
		Medium	253.42
		Strong	253.16
	Yellow plasticine dough 	Light	253.04
		Medium	252.72
		Strong	252.33
	Orange plasticine dough 	Light	253.55
		Medium	253.10
		Strong	252.40
Elasto-plastic Object	Stress ball 	Light	256.61
		Medium	254.58
		Strong	253.81
Rigid Objects	Red package 	Light	252.21
		Medium	252.14
		Strong	252.27
	Pink case 	Light	253.55
		Medium	253.81
		Strong	254.13

4.4.2 Speed-Focused Contour Tracking

The second approach proposed attempts to address the main limitation of the previous one, and is focusing on accelerating the process of contour detection, by integrating contour tracking. It tracks the contour of the object of interest over a sequence of frames of RGB images collected from the Kinect sensor by employing the point cloud only once, for the initialization of the fast level set method. The processes of extracting the 3D object cluster with the background removal procedure, projecting it to the color image, and computing a bounding window are executed only once as an initialization for the first frame only. For the following frames, the output contour obtained from the previous application of the level set function is used as the initialization for the next iteration of the level set function to be applied on the current frame, producing an active tracking of the contour of the object. However, the resulting dependency on the contour from the previous frame implies that the tracking system needs to be restarted each time a new object of interest or tracking target is used or enters into the scene. This contour tracking process achieves near real-time speed, but is not as robust when the object of interest is changed or occluded during the tracking process, as shown in Figure 4.50.

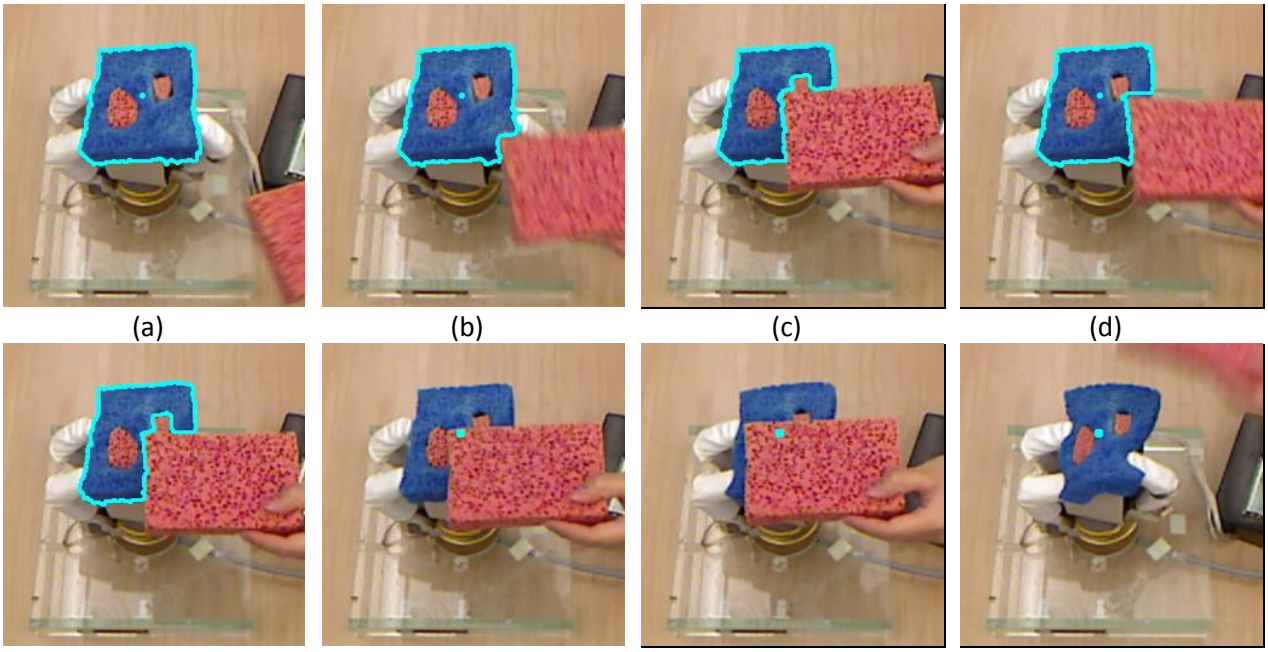


Figure 4.50: Speed-focused approach: (a-e) a series of tracked object contours with partial occlusions by a color conflicting object; and (f-h) losing track of object contour due to occlusion of the fixation point by another object.

In terms of computation speed, for each frame, the contour tracking time is 44.4 ms on average, which is 5 times faster than the performance achieved using the approach focused on robustness in section 4.4.1. The contour tracking time per frame for a variety of objects and applied forces is listed in Table 4.2. In conclusion, the contour tracking process achieves near real-time tracking, supporting an update rate of up to 23 Hz, which is closer to the available acquisition frame rate of the Kinect sensor at 30 Hz.

As more information can be processed, a smoother evolution of the contour is achieved, which can be beneficial for the control of the robotic hand that is to be designed out of this sensing stage.

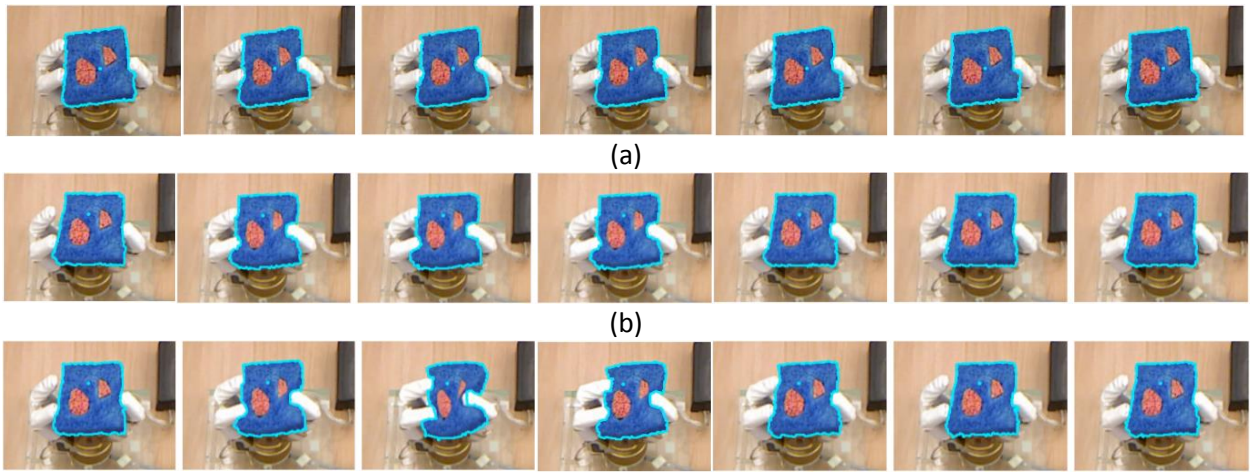
Table 4.2: Contour tracking time per frame (ms)

	Object	Force	Tracking time (ms)
Elastic Objects	Blue+red sponge 	Light	48.33
		Medium	49.23
		Strong	49.70
	Red+green sponge 	Light	55.16
		Medium	61.70
		Strong	61.68
	Yellow sponge 	Light	47.00
		Medium	44.05
		Strong	45.00
Plastic Objects	Blue plasticine dough 	Light	36.77
		Medium	36.56
		Strong	36.36
	Yellow plasticine dough 	Light	41.07
		Medium	39.68
		Strong	44.33
	Orange plasticine dough 	Light	36.85
		Medium	38.28
		Strong	38.04
Elasto-plastic Object	Stress ball 	Light	36.26
		Medium	35.81
		Strong	37.43
Rigid Objects	Red package 	Light	86.96
		Medium	93.68
		Strong	88.88
	Pink case 	Light	36.40
		Medium	37.71
		Strong	37.01

However, this approach tends to be more subject to losing track of the contour and is not able to quickly respond when there are occlusions in the scene. It is able to keep tracking the object contour when an occlusion happens as shown in Figure 4.50a-e, but once the occluding object is moved over the 2D fixation point, the approach can neither track the contour of the desired object, nor the contour of the occluding one (Figure 4.50f-g). It cannot recover the contour on the initial object of interest even after the occluding object is completely removed from the scene (Figure 4.50h). In other words, this approach has the capability to keep tracking the contour of the object, under partial occlusion, but with the condition that the occluding object does not come over the 2D fixation point.

4.4.3 Contour Detection and Tracking Performance for Various Object Materials and Force Magnitudes

Figures 4.51 through 4.59 demonstrate testing scenarios where contours, marked in cyan, are successfully detected over a series of frames during the probing process of objects made of materials with different properties. The objective of these tests is to validate the capability of the proposed approach to detect the contour around an object and to follow it as it evolves in spite of the objects' various colors, materials and textures (the user-selected 2D fixation point is marked by a cyan dot). These results are obtained using the robustness-focused contour detection approach detailed in section 4.4.1. It can be noticed that the approach only extracts the external contour and ignores the internal details of the object. For example, the external contours are captured in Figure 4.51 and Figure 4.52 while the inserts are ignored, and the contour extraction is not disturbed by the printed graphical patterns over the packaging in Figure 4.58.



(a)

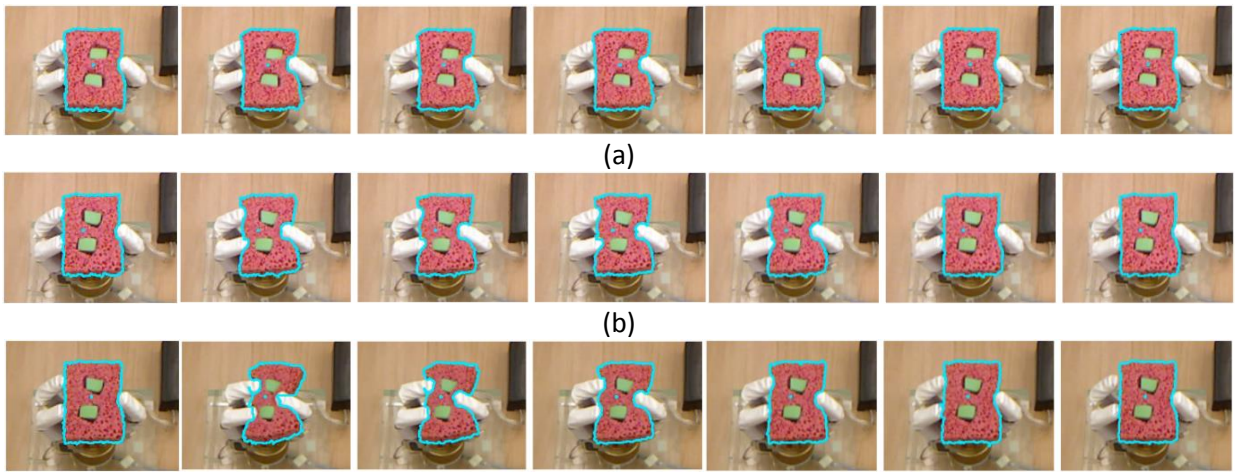
(b)

(c)

Figure 4.51: Series of contours around elastic blue sponge with red inserts under manipulation with: (a) light force, (b) medium force, and (c) strong force.

Figure 4.51 shows a series of deformations on an elastic blue sponge with red inserts, and its extracted contour under the manipulation with the robotic hand when light, medium, and strong forces are applied, respectively. As expected, the object deformation is more visible when the applied force gets stronger.

Figure 4.52 shows the extracted contour on an elastic red sponge with green inserts that varies along with the object's deformation under various grasping forces.



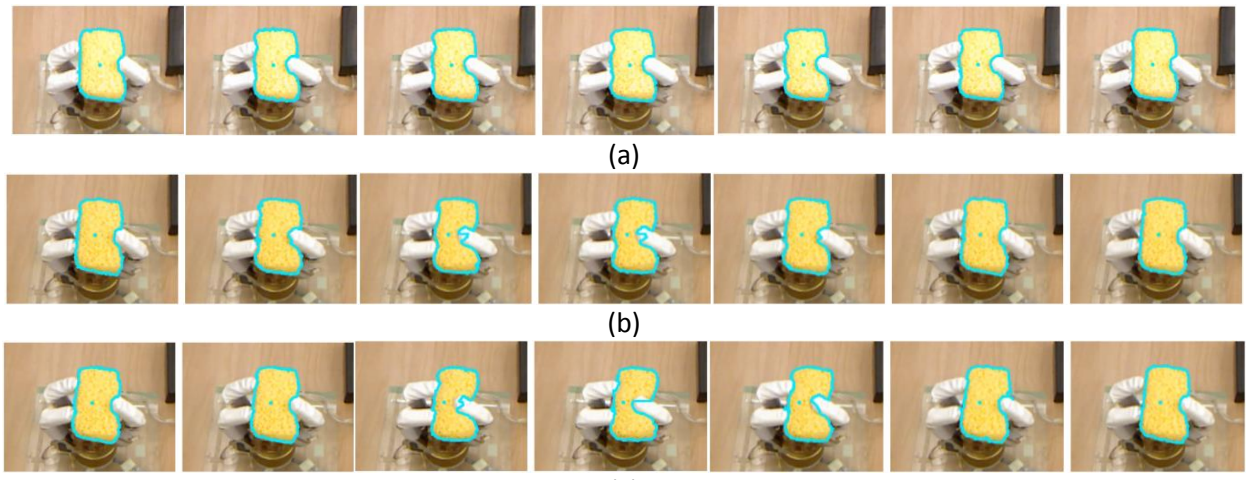
(a)

(b)

(c)

Figure 4.52: Series of contours on an elastic red sponge with green inserts under manipulation with: (a) light force, (b) medium force, and (c) strong force.

The contour of an elastic yellow sponge along with its deformation is correctly captured as shown in Figure 4.53. The contour of the object and its deformation are apparent at three contact points with the robotic hand.



(a)

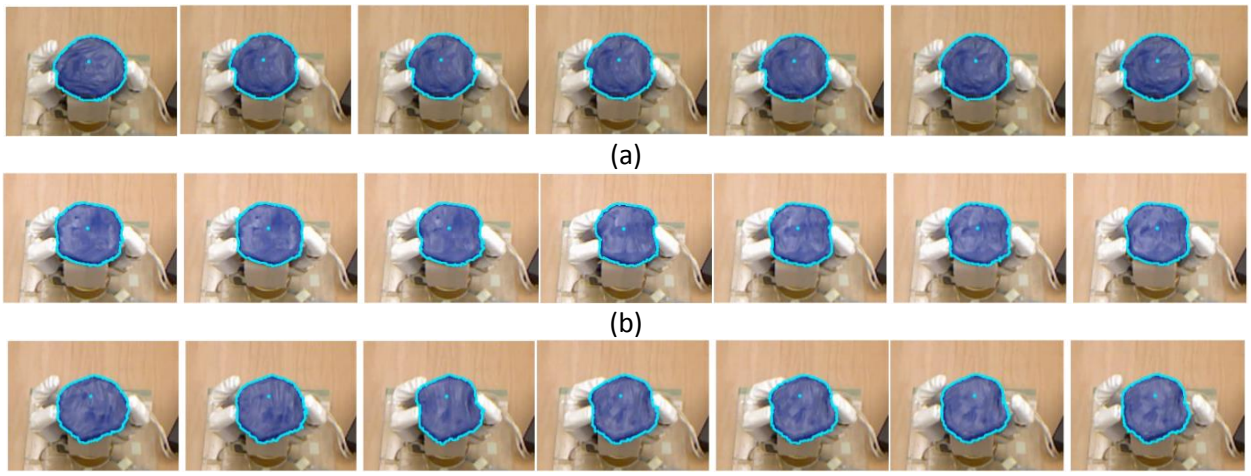
(b)

(c)

Figure 4.53: Series of contours on an elastic yellow sponge under manipulation with: (a) light force, (b) medium force, and (c) strong force.

Blue, orange and yellow plasticine doughs used in our experimentation to represent the case of plastic objects are each packed in a transparent plastic wrap in order to protect their surfaces from drying out and cracking when they are exposed to the air. The plastic wrap also prevents the material from sticking to the robotic hand fingertips. Figure 4.54 shows the blue plasticine dough deformation and the evolution of its contour under the light, medium and strong forces, respectively. The deformations and

corresponding contours obtained on the orange and yellow plasticine doughs can be observed in Figure 4.55 and Figure 4.56, respectively.

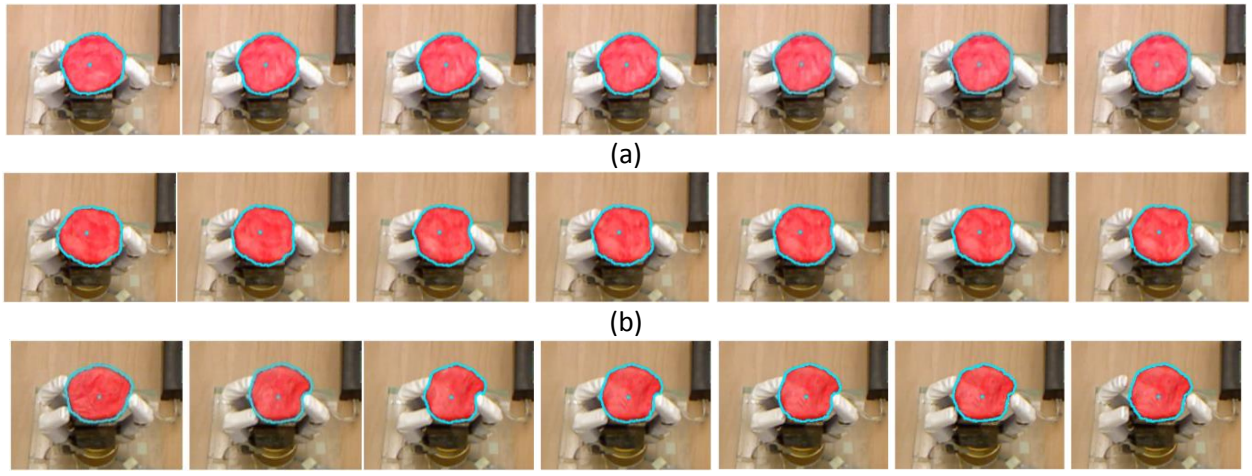


(a)

(b)

(c)

Figure 4.54: Series of contours on plastic blue plasticine dough under manipulation with: (a) light force, (b) medium force, and (c) strong force.



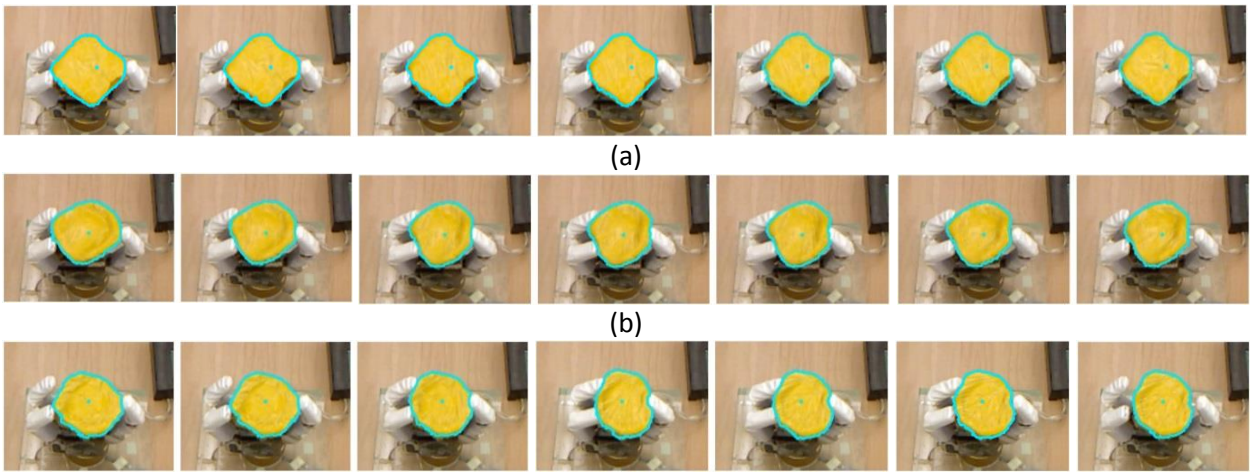
(a)

(b)

(c)

Figure 4.55: Series of contours on plastic orange plasticine dough under manipulation with: (a) light force, (b) medium force, and (c) strong force.

The stress ball shown in Figure 4.57 is the only good example of an elasto-plastic object that we could identify for testing. It shows the particular property of this type of material that recovers only partially its shape after the grasping force is removed. The object contours are correctly detected under various grasping forces as shown in Figure 4.57.

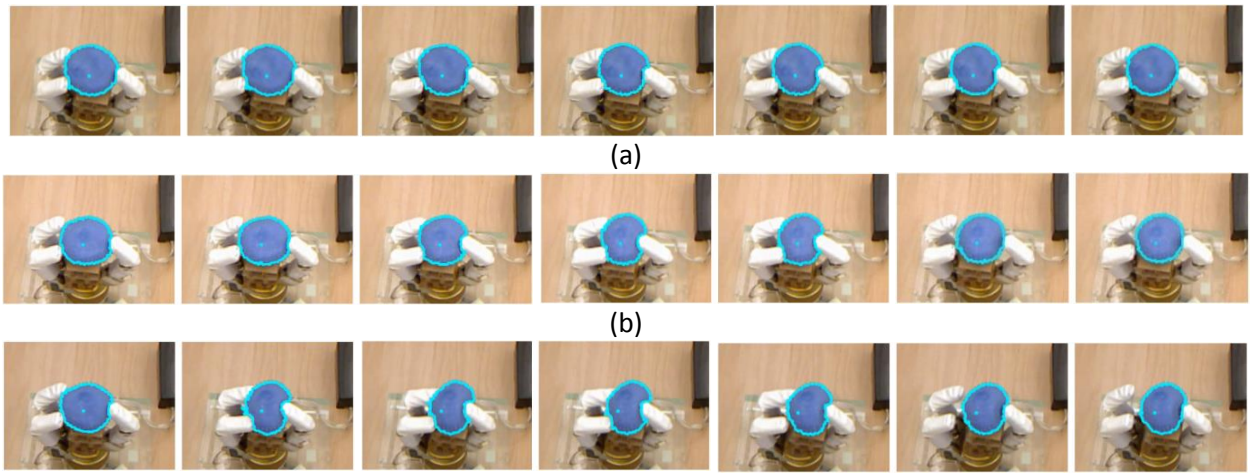


(a)

(b)

(c)

Figure 4.56: Series of contours on plastic yellow plasticine dough under manipulation with: (a) light force, (b) medium force, and (c) strong force.



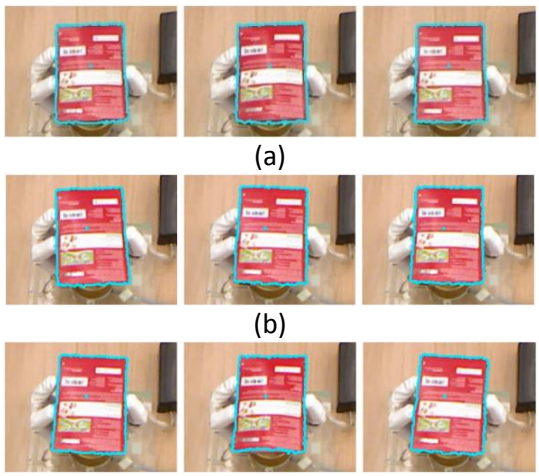
(a)

(b)

(c)

Figure 4.57: Series of contours on an elasto-plastic stress ball under manipulation with: (a) light force, (b) medium force, and (c) strong force.

A red package with printed labels on its surface (Figure 4.58) and a pink case (Figure 4.59) are used as examples of rigid objects. It can be observed that the details on the red package are not detected since surface patterns as not involved in the proposed contour detection approach. As observed in Figure 4.58, the red package is not deformed, but only slightly rotated, under the various grasping forces. Compared to the red package, the pink case is a smaller object. In this case also there is no visible deformation of the pink case at the three points of contact in Figure 4.59. Furthermore, the pink case has no obvious rotation caused by the force exercised on the object.

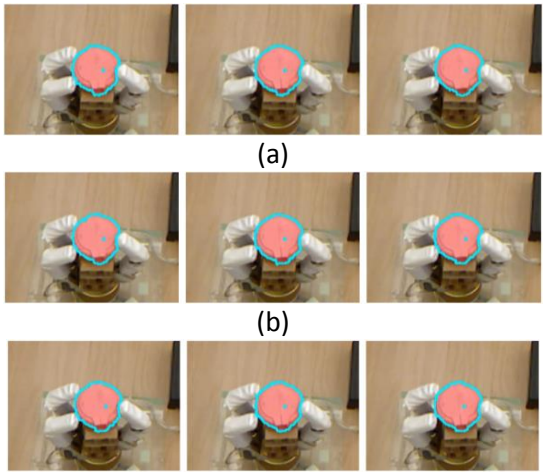


(a)

(b)

(c)

Figure 4.58: Series of contours on a rigid red package under manipulation with: (a) light force, (b) medium force, and (c) strong force.



(a)

(b)

(c)

Figure 4.59: Series of contours on a rigid pink case under manipulation with: (a) light force, (b) medium force, and (c) strong force.

Figure 4.60 presents a direct time comparison from the numbers reported in Table 4.1 and Table 4.2 for all test cases considered. In this figure, the blue and red columns represent the robustness-focused and speed-focused approaches detection or tracking time per frame, respectively.

Regarding robustness-focused contour detection time consumption, it can be observed from Figure 4.60 (blue bars) that there is no significant difference on time consumption as a function of the objects and the material they are made of, and neither with respect to the magnitude of the force applied. This is explained by the fact that the approach runs the RANSAC-based background removal and the point cloud cluster extraction on the object of interest at every frame; the two processes occupy a major part of the processing time which dominates over the actual contour extraction.

Regarding speed-focused contour tracking time consumption, it is seen that tracking the contour of the red sponge with green inserts (Red+green sponge in Figure 4.60) under the category of elastic objects,

and of the red package with surface printing under the category of rigid objects, takes a longer time than the others, especially tracking the contour on the red package. This results from the fact that the red package is larger than the other objects and therefore occupies more space in the image, making for a larger window of interest and more pixels to process. It also has more internal details, resulting in the region-based fast level set to take longer time to compute the mean intensity of the region for each iteration. Similarly, processing the red sponge with green inserts takes slightly longer to track the contour at each frame, due to the fact that its size is almost as large as the rigid red package. The contour detection time on the stress ball under the category of elasto-plastic objects and the pink case under the category of rigid objects are both slightly shorter than the average because these two objects are relatively simple in texture and smaller than the average with respect to their coverage of the image.

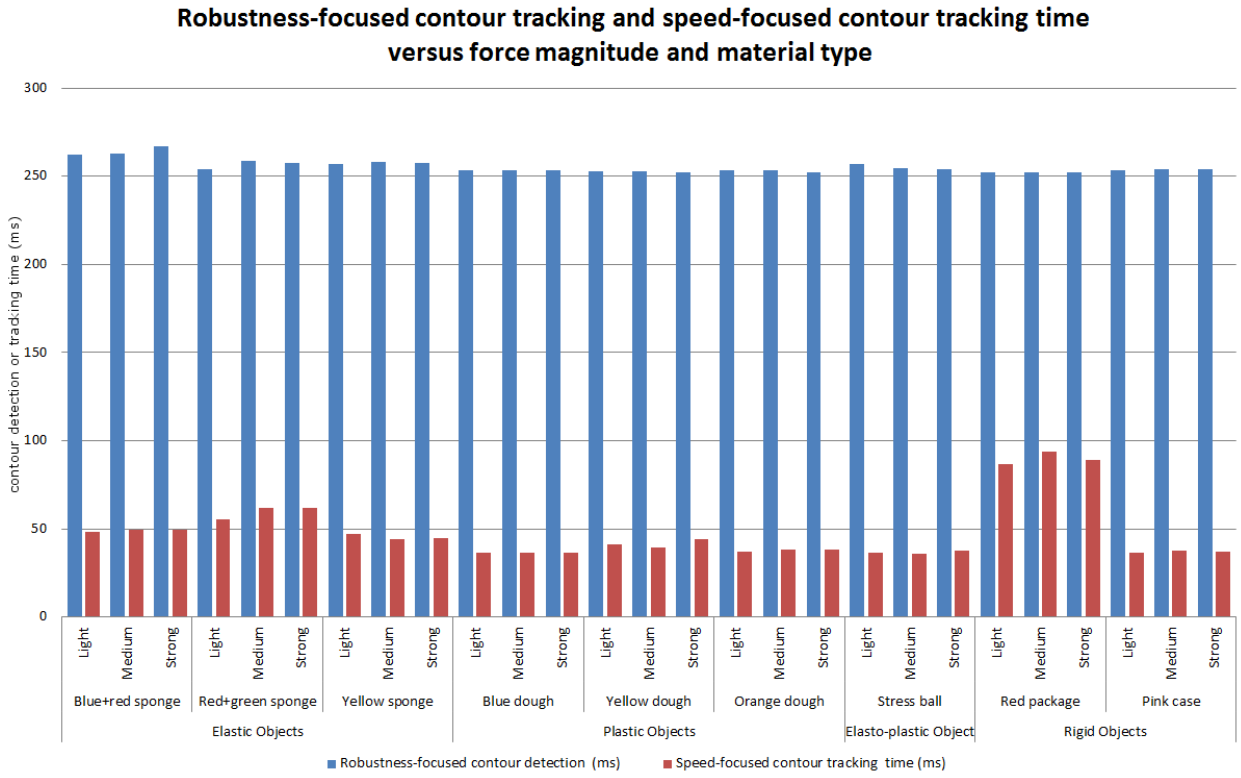


Figure 4.60: Robustness-focused contour detection and speed-focused contour tracking time versus force magnitude and material type.

Comparing the computation time for the robustness-focused contour detection (blue columns in Figure 4.60) against the speed-focused contour tracking (red columns in Figure 4.60), the robustness-focused contour detection approach takes almost five times longer per frame than the speed-focused contour tracking to produce contours of similar accuracy, assuming that the object of interest is not occluded and no visual perturbations are brought in the field of view. This fact confirms that the two processes of RANSAC-based background removal and point cloud cluster extraction consume overall a longer time than the level set method.

Figure 4.61a and Figure 4.61c show the average robustness-focused contour detection and speed-focused contour tracking time versus the force magnitude for individual frames. One can notice slight

differences between the robustness-focused contour detection and the speed-focused contour tracking time under different magnitudes of forces, with maximal time differences being less than 2 ms for both robustness-focused contour detection and speed-focused contour tracking. The robustness-focused contour detection and speed-focused contour tracking under a light force achieve faster results because of tiny movements on the contours. On the other hand, the robustness-focused contour detection and speed-focused contour tracking time remains similar under medium and strong forces. The time consumption in Figures 4.61a and Figure 4.61c reveals that various magnitudes of applied forces have low impact on the performance of contour extraction.

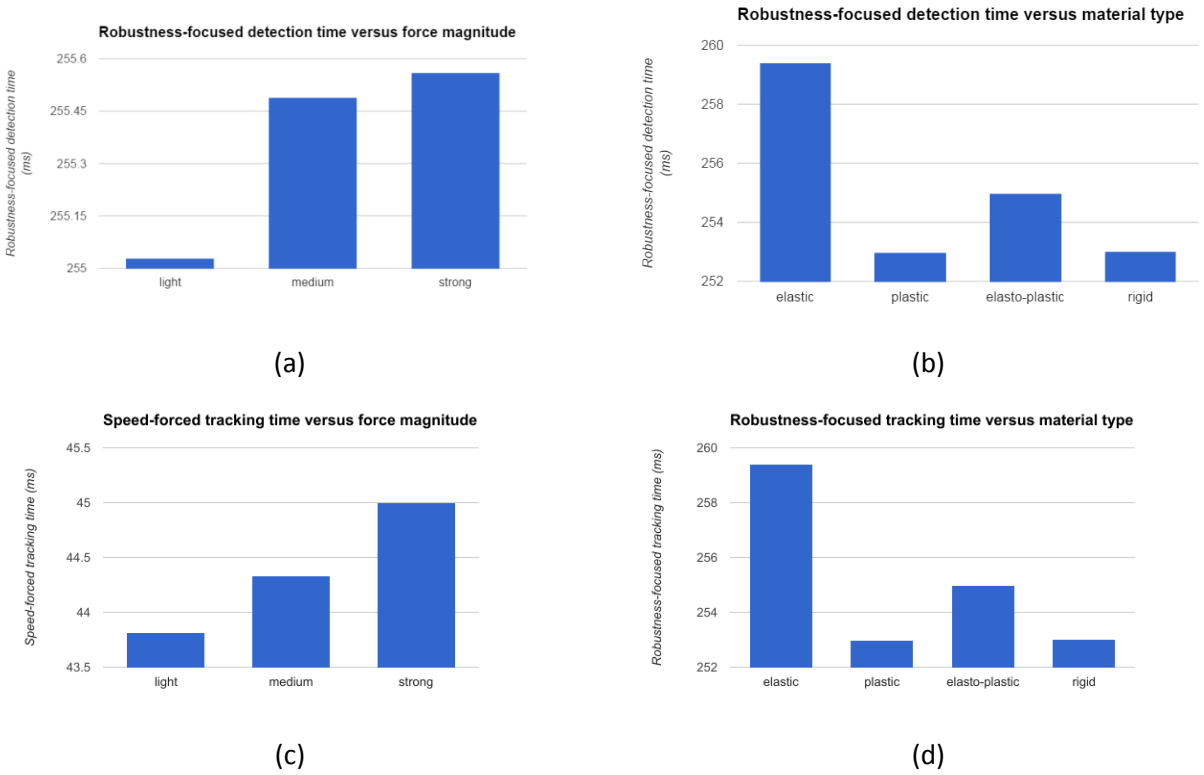


Figure 4.61: Average computation time per individual frame for robustness-focused contour detection: (a) according to magnitudes of applied forces, and (b) according to the types of tested materials; and average computation time per individual frame for speed-focused contour tracking: (c) according to magnitudes of applied forces, and (d) according to types of tested materials.

Figure 4.61b and Figure 4.61d show the average robustness-focused contour detection and speed-focused contour tracking time versus the material type for each individual frame. The averages are computed while considering all test objects belonging to any of the four material categories. Some differences on the computation time are observed in between materials. Elastic material objects require slightly longer time for their contour to be extracted independently of the approach (robustness-focused or speed-focused). For elastic materials, this difference is caused by the more significant deformation of the elastic object during the manipulation process where the fingertips reach further inside the object. During the probing, elastic objects are more distorted from their original shape and reach higher levels of deformation, and then recover back to their original shape. Following the contour over its entire path

is slightly more challenging and therefore takes slightly longer, though the variation remains on the order of 10 to 15 ms. Also, the average size of elastic objects is larger than the one of objects of other categories considered here, which influences the contour detection time, as demonstrated previously. In addition, one can observe that rigid material can also require longer computation time as shown Figure 4.61d. In this example, the longer tracking time is mainly impacted by the heavily textured and larger image surface coverage of the red package considered in the category of rigid objects.

4.5 Object Material Behavior-Based Classification Performance

As detailed in section 3.4.3, the classification of the type of material of an object is performed by comparing pairwise among three characteristic contours respectively, that are the initial contour, the contour under the largest deformation, and the final contour once the force is removed. Furthermore, the object classification generates the same results whether the contours are extracted in one of the two possible ways introduced in section 3.3.6.1 and section 3.3.6.2, namely the robustness-focused contour detection or the speed-focused contour tracking. The proposed classification approach is tested on nine objects with four different material properties under three grasping force strengths. Each of the nine objects (i.e., 3 elastic objects, 3 plastic objects, 1 elasto-plastic object, and 2 rigid objects) is repeatedly tested 60 times using the probing process described previously. During each experiment, the object is compressed by a light, medium, or strong force and then released.

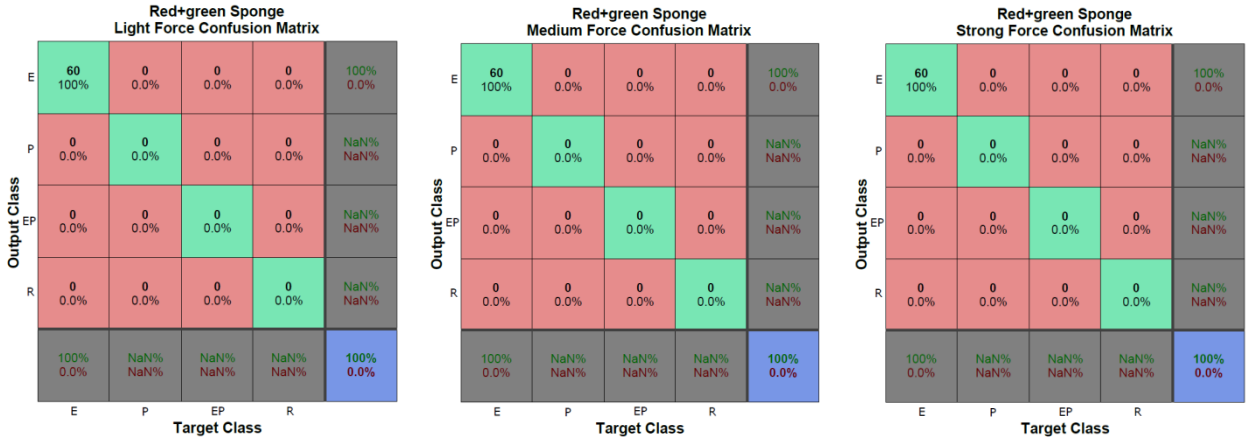
4.5.1 Classification Accuracy of Object Material Characterization

As detailed in section 4.2.6, confusion matrices [81] are used to represent the results from material classification for each tested object. Figures 4.62 to 4.70 respectively present the classification results for each of the objects considered individually, each tested 60 times, while Figure 4.71 regroups the classification results for all objects together, therefore supporting general conclusions. In these confusion matrices, E, P, EP and R denote the material type as elastic, plastic, elasto-plastic or rigid, respectively.

(a) (b) (c)

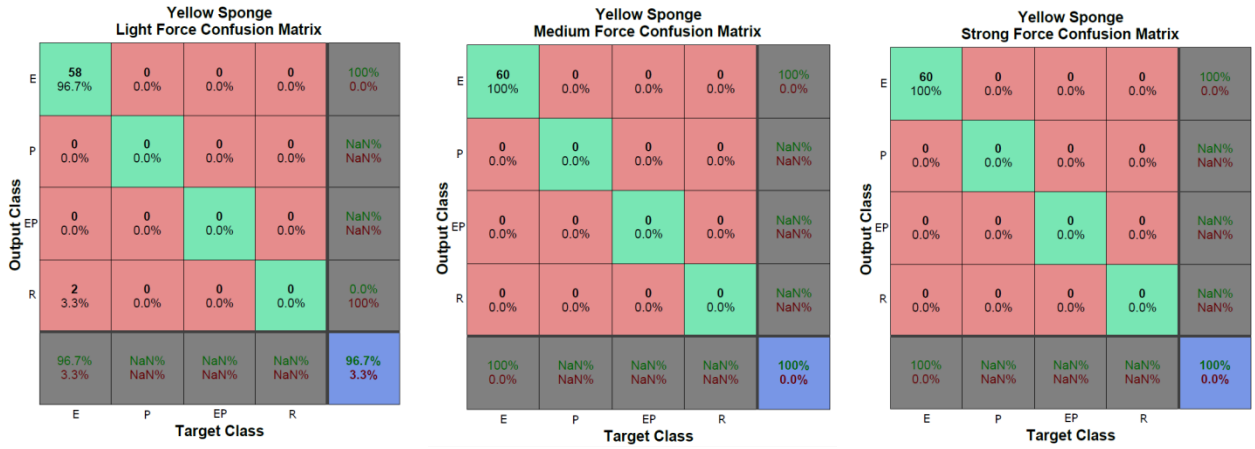
Figure 4.62: Confusion matrices for classification results on elastic blue sponge with red inserts under manipulation with: (a) light force, (b) medium force, and (c) strong force.

As shown in Figure 4.62, the elastic blue sponge with red inserts (Figure 4.51) is misclassified as a rigid object in one case under light force, while in the other cases, with medium and strong forces, it is correctly categorized as an elastic object. The recognition rate for the elastic red object with green inserts (Figure 4.52) under all magnitudes of force reaches 100% as shown Figure 4.63. That is, all the tests correctly classify that object as an elastic one independently from the magnitude of force applied. A recognition rate of 100% is also achieved for the elastic yellow sponge (figure 4.53) under either a medium or a strong force, and 96.7% classification rate is obtained for that object under a light force as shown in Figure 4.64. It can be observed that the elastic object under the light force is misclassified as a rigid object in 2 from 60 cases. This happens because the tiny movements of the deformed contour are identified as noise in the proposed classification approach. On the other hand, as the magnitude of the applied force increases on each finger, under the medium and strong forces, the recognition rate regularly achieves 100% for these objects made of elastic material.



(a) (b) (c)

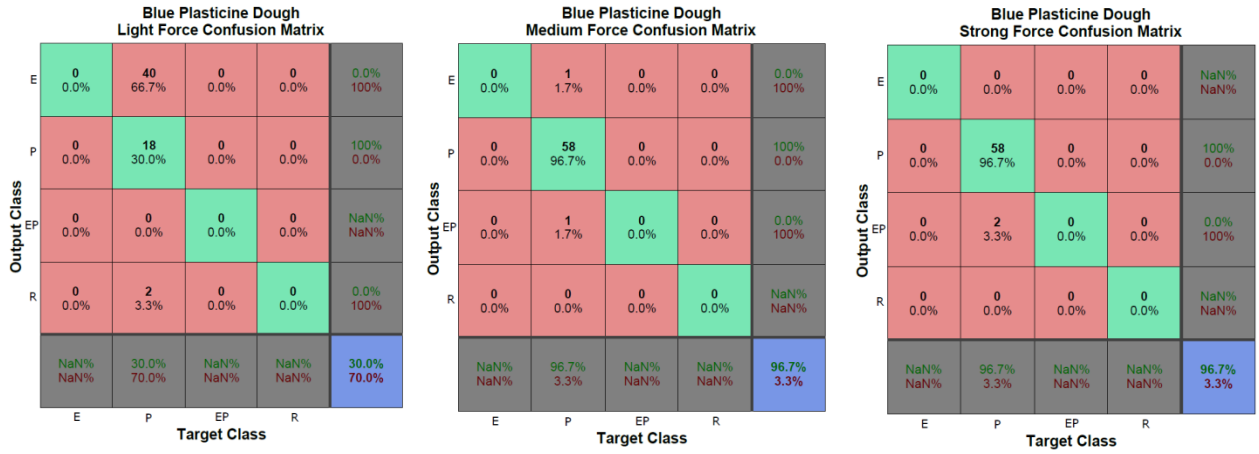
Figure 4.63: Confusion matrices for classification results on elastic red sponge with green inserts under manipulation with: (a) light force, (b) medium force, and (c) strong force.



(a) (b) (c)

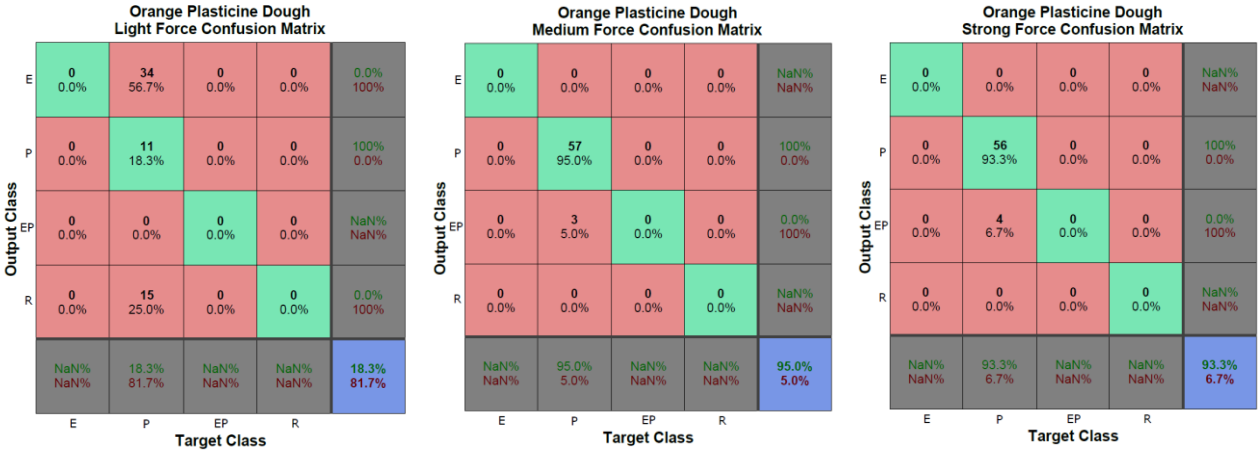
Figure 4.64: Confusion matrices for classification results on elastic yellow sponge under manipulation with: (a) light force, (b) medium force, and (c) strong force.

Under the interactions exercised with the robotic hand, the plastic material represented by the blue plasticine dough (Figure 4.54) is not properly identified as a plastic object in 42 of the 60 trials when a light force is applied, as the P column shows in Figure 4.65a. Instead, it is more often classified as an elastic object (in 66.7% of cases). However, the recognition rate increases significantly when medium or strong forces are applied, reaching a correct identification rate of 96.7% in both cases, as shown in Figure 4.65b and 4.65c.



(a) (b) (c)

Figure 4.65: Confusion matrices for classification results on plastic blue plasticine dough under manipulation with: (a) light force, (b) medium force, and (c) strong force.

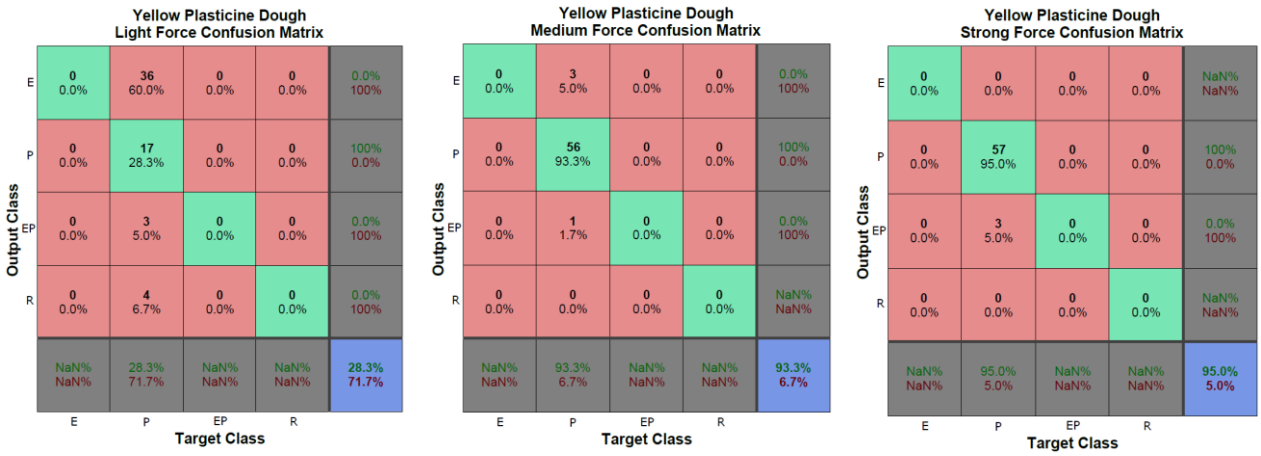


(a) (b) (c)

Figure 4.66: Confusion matrices for classification results on plastic orange plasticine dough under manipulation with: (a) light force, (b) medium force, and (c) strong force.

Similarly, it can be observed in Figure 4.66 and Figure 4.67 that the recognition rate of the orange and yellow plasticine doughs (Figures 4.55 and 4.56) increases significantly when a stronger grasping force is applied. The recognition rate of the orange and of the yellow plasticine dough achieves 95% under medium force. The fact that plasticine dough properties cause the dough to take time, or need a strong applied force, to become malleable leads to these results. The light force applied is not strong enough to shape these plasticine doughs. As a result, these objects are mainly categorized as elastic objects under

a light grasping force. However, when a large enough force is applied to deform the plasticine dough, these objects are correctly classified as plastic objects.



(a) (b) (c)

Figure 4.67: Confusion matrices for classification results on plastic yellow plasticine dough under manipulation with: (a) light force, (b) medium force, and (c) strong force.

Figure 4.68 shows the confusion matrices for the classification rates achieved on the elasto-plastic stress ball (Figure 4.57) under manipulation with light, medium, and strong forces, respectively. When medium or strong grasping forces are used, recognition rates of 100 % are achieved, as shown in Figure 4.68b,c. For the case when a light force is applied (in Figure 4.68a), more than half of the tests correctly categorize the stress ball as an elasto-plastic object (56.7%) and the rest of the tests misclassify the object as an elastic object (43.3%). This occurs because a light force is insufficient to substantially change the shape of the object, resulting in relatively small contour deformations that are hard to capture with the Kinect sensor due to its limited resolution. For the cases where a medium or strong force is employed, the contour deformations are sufficient in magnitude for the stress ball to be correctly identified as an elasto-plastic object.

(a) (b) (c)

Figure 4.68: Confusion matrices for classification results on elasto-plastic stress ball under manipulation with: (a) light force, (b) medium force, and (c) strong force.

Finally, the red package (Figure 4.58) and the pink case (Figure 4.59) are used as examples of rigid objects. For these objects, a recognition rate of 100% is obtained under the three different magnitudes of grasping force, as shown in Figure 4.69 and Figure 4.70. It was observed in Figure 4.58 that the interaction force applied at each fingertip slightly rotates the red package. This example is interesting as it demonstrates that the dynamic time warping used for aligning the contour sequences in the classification approach is tolerant to these rotations, ensuring that the red package is correctly identified as a rigid object even when rotation and shift occur.

(a) (b) (c)

Figure 4.69: Confusion matrices for classification results on rigid red package under manipulation with: (a) light force, (b) medium force, and (c) strong force.

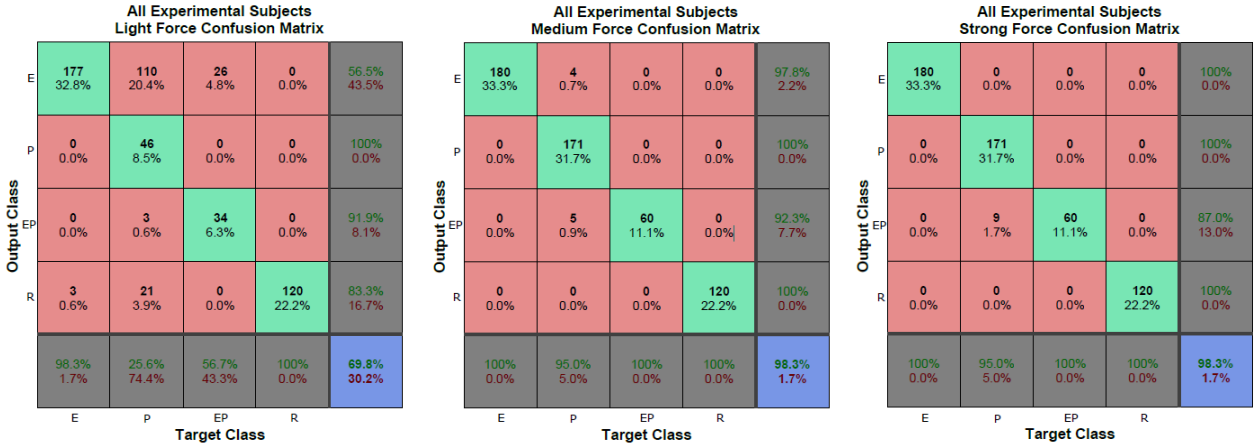
(a) (b) (c)

Figure 4.70: Confusion matrices for classification results on rigid pink case under manipulation with: (a) light force, (b) medium force, and (c) strong force.

To summarize, general confusion matrices for all nine objects considered, and under the three possible strengths of grasping force, are shown in Figure 4.71.

In terms of the magnitudes of applied forces, for all material types, an overall recognition rate of 69.8% (Figure 4.71a) is obtained when a light grasping force is applied, of 98.3% (Figure 4.71b) when a medium force is applied, and of 98.3% (Figure 4.71c) when a strong force is applied. The low performance of the

classification for objects under a light force interaction originates from the light force producing only relatively minor contour deformations. In addition, for some materials, the light force is not strong enough for the material to become malleable. Moreover, the low resolution of the Kinect sensor provides images with low accuracies, which leads to small deformations being identified as noise. On the contrary, objects are correctly categorized under medium and strong forces in 98.3% of the test cases because these two magnitudes of force are strong enough to deform objects up to a stage that is visually perceptible by the Kinect sensor. That is, changes in the object structure can be captured by the Kinect sensor and satisfy the proposed conditions of identifying the contour as being deformed, as detailed in section 3.4.3. As the magnitude of forces applied on objects increases, the performance of the classification significantly improves. This leads to the conclusion that a sufficiently large force should be applied during the probing process, while attempting to characterize a deformable object. The minimum force magnitude corresponding to a medium force in these experiments was about 3.9N, as defined in section 4.1.1.2.



(a) (b) (c)

Figure 4.71: Confusion matrices for classification results on all objects under manipulation with: (a) light force, (b) medium force, and (c) strong force.

In terms of material types, for all magnitudes of grasping forces, an overall recognition rate of 99.4% is reached for elastic material (average of E columns in Figure 4.71), 71.9% for plastic material (average of P columns in Figure 4.71), 85.6% for the elasto-plastic material (average of EP columns in Figure 4.71) and 100% for rigid material (average of R columns in Figure 4.71). These recognition rates prove that the approach works effectively in classifying most types of objects, but has more limitations for plastic objects.

Elastic objects (E columns in Figure 4.71) are identified correctly in most cases, while few cases are confused with rigid objects, but only when a light grasping force is applied. The lowest performance among these four cases is for plastic material, represented by the three plasticine doughs with different colors. These doughs are wrapped in a transparent plastic wrap for two reasons: *i*) the robot fingers tend to stick to the plasticine dough, especially when medium and strong forces are applied, impacting the contour detection or tracking, and the related category recognition; *ii*) the plasticine dough dries out when exposed to air, resulting in changes in the properties of the plastic material. As observed in

column P of the confusion matrix for a light force, Figure 4.71a, just about a quarter of the samples are correctly classified as a plastic material (25.6%) when a light force is applied. As the magnitude of force applied by the robot fingers increases, as high as 95% of the plastic samples are properly identified under medium or strong forces. The significant improvement on the classification, under medium and strong forces, when compared with light force, demonstrates that a sufficient force needs to be applied in order to create a clear deformation on a plastic object such that the object does not tend to be confused with an elastic, elasto-plastic or even rigid object in the classification process. For the elasto-plastic objects (EP columns in Figure 4.71), the recognition rate varies from 56.7% under a light force, to 100% under medium or strong forces. The elasto-plastic properties under a light force are confused with those of an elastic object in most cases, which results from the partial elastic properties of an elasto-plastic object and an insufficient strength of the applied force. Lastly, a high performance on rigid objects classification (R columns in Figure 4.71) is obtained regardless of the magnitude of the applied force. This is essentially due to the simplicity of rigid objects, no significant deformation occurring under any force, therefore resulting in no significant contour changes.

In conclusion, for non-rigid objects, a sufficient magnitude of force (i.e., of the order of 3.9 N for the objects considered in this study), needs to be applied to significantly deform them and properly characterize their material, in accordance with the ground-truth categories of elastic, plastic, and elasto-plastic objects.

4.5.2 Classification Computation Time

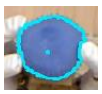
The material behavior-based object classification process is implemented as an offline process since it relies on data extracted during the probing process conducted with the robotic hand and the contour detection process. For each object, the average object classification time is 35.2 ms after the three characteristic contours, which is the initial contour, the contour under largest deformation and the final contour, are received by the classification stage. The experiments are carried out on a 2.6 GHz Intel Core i7 with 8 GB of RAM laptop. Table 4.3 presents the respective classification time per object.

Figure 4.72 shows a more explicit distribution of the classification time reported in Table 4.3, according to material properties and magnitude of force applied.

Regarding the computation time required for material behavior-based classification from the extracted contours (Figure 4.72), it can be observed that some differences occur between objects. A close analysis reveals that the classification time is mainly impacted by the relative size that the object of interest occupies in the processed RGB images. Classifying objects that are larger, like the rigid red package (Figure 4.58), takes a longer time. The contours of these larger objects are composed of a larger number of pixels. Consequently, in the DTW process that supports the classification, a longer warping path is identified to establish the correspondence between the pixels of the contours that are compared. For example, classifying the red package under the category of rigid object is longer than processing the other ones, independent of their material characteristics, due to the fact that the size of the red package is larger than the one of the other objects. Respectively, the warping paths for the rigid red package, the

plastic orange dough, the pink case, and the elastic yellow sponge are of length 623, 478, 356 and 514 pairs of pixels.

Table 4.3: Material behavior-based object classification time per object (ms)

	Object	Force	Classification time (ms)
Elastic Objects	Blue+red sponge 	Light	36.17
		Medium	49.57
		Strong	42.33
	Red+green sponge 	Light	44.59
		Medium	45.50
		Strong	44.87
	Yellow sponge 	Light	34.24
		Medium	39.43
		Strong	42.00
Plastic Objects	Blue plasticine dough 	Light	30.38
		Medium	32.73
		Strong	27.61
	Yellow plasticine dough 	Light	28.72
		Medium	31.56
		Strong	30.25
	Orange plasticine dough 	Light	21.07
		Medium	24.73
		Strong	24.42
Elasto-plastic Object	Stress ball 	Light	24.41
		Medium	24.49
		Strong	39.15
Rigid Objects	Red package 	Light	53.08
		Medium	54.36
		Strong	50.76
	Pink case 	Light	20.72
		Medium	25.70
		Strong	26.37

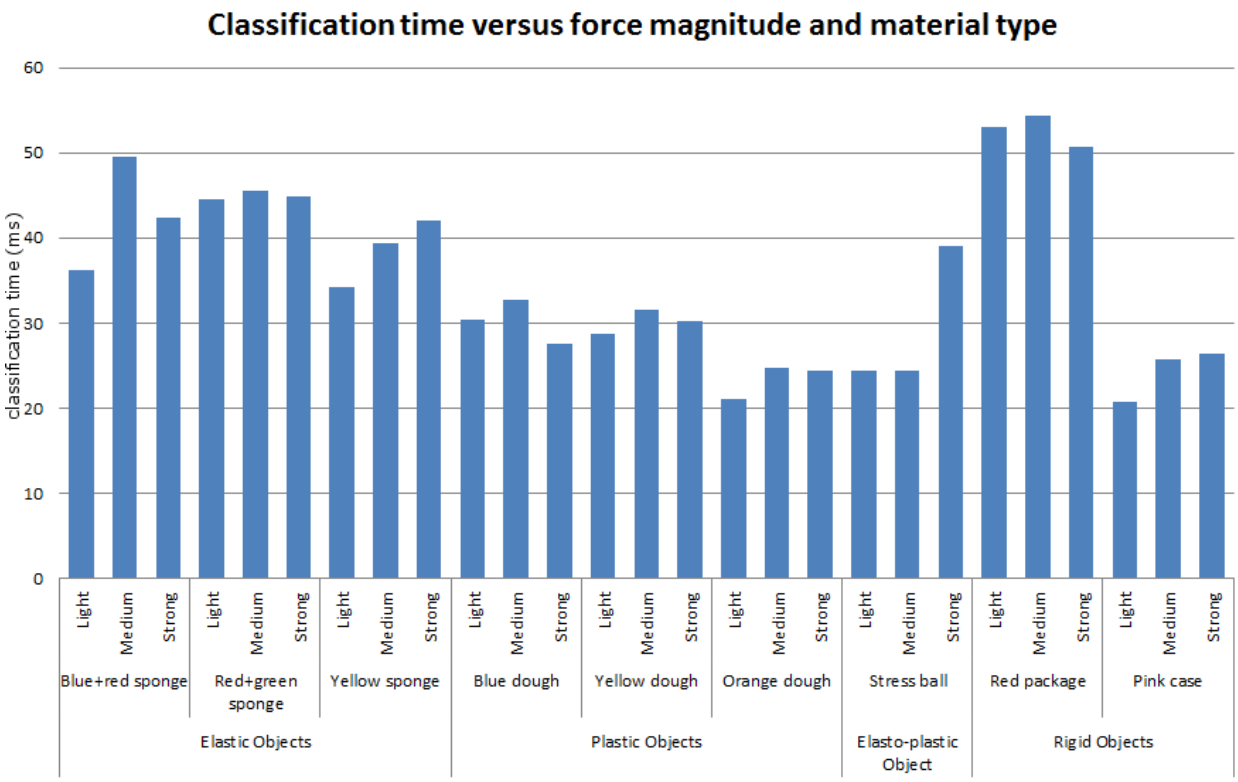
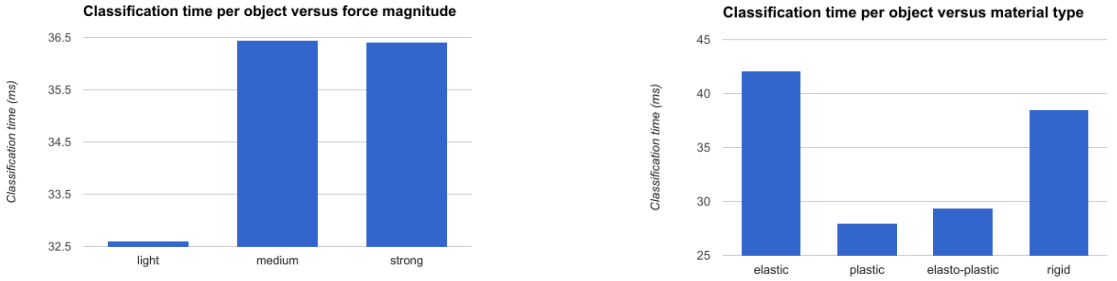


Figure 4.72: Classification time per individual object versus force magnitude and material type.

Figure 4.73a shows the average classification time per individual object based on various magnitudes of force applied to the objects. The averages are computed over all nine objects when submitted to a similar magnitude of applied force. One can notice that slight differences on classification time occur under different magnitudes of forces. The maximum time differences are however less than 4 ms. Classification under a light force achieves faster results because of the tiny movements produced on contours. On the other hand, the classification time remains similar for both medium and strong forces. Figure 4.73a reveals that various magnitudes of applied forces have overall low impact on the computation time required for classifying objects.

Figure 4.73b shows the average classification time per individual object according to the four types of material considered. The averages are computed over all objects belonging to a same category of material. Some differences in the computation time are observed, with elastic and rigid materials taking longer to be categorized than plastic and elasto-plastic materials. For elastic materials, the longer average classification time is due to the more pronounced deformation behavior of elastic materials that exhibit the most significant deformation magnitude over the manipulation process, in which the elastic object is distorted further from its original shape, and then recovers back its original shape. During this process, three representative contours of various lengths establish the pairwise correspondence in the DTW process, which takes a longer time. In addition, the average region covered in the image for elastic objects tends to be larger than with objects of other categories (e.g. plastic objects), which also influences the average duration of tracking and classification. For rigid materials, the longer average

classification time is mainly a consequence of the heavily textured and larger red package object considered under the category of rigid objects.

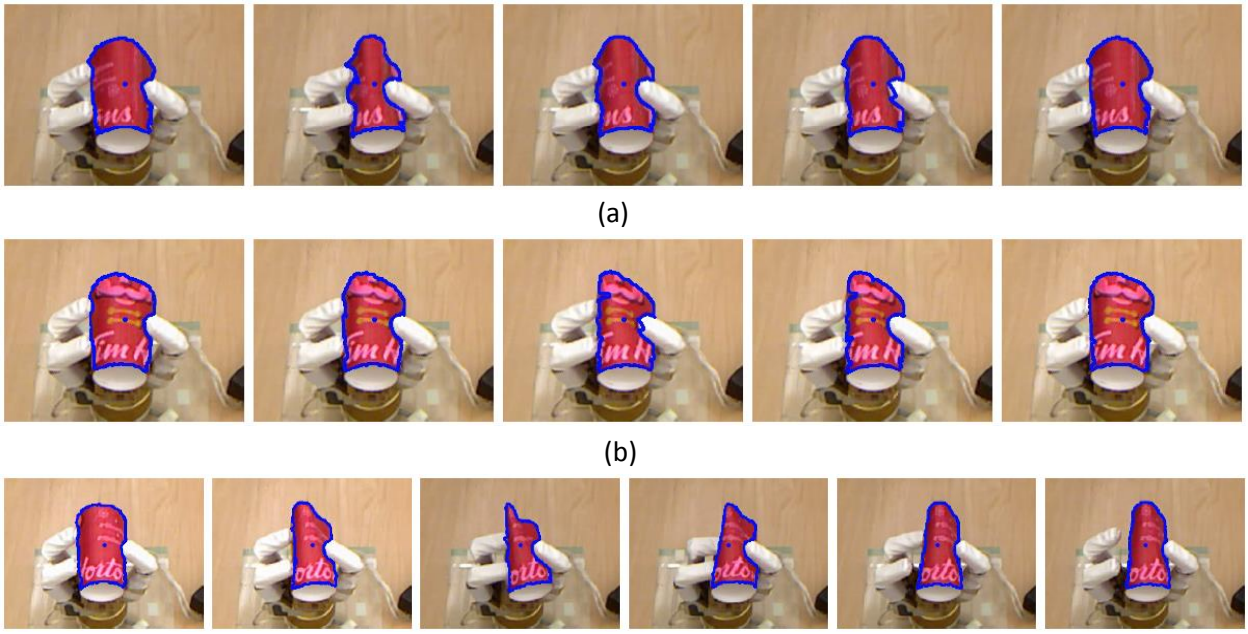


(a) (b)

Figure 4.73: Average classification time per individual object: (a) according to magnitude of applied forces, and (b) according to the type of tested material.

4.5.3 Special Case: Object with Multiple Material Deformation Stages

Some objects exhibit complex behaviors that characterize materials with multiple deformation stages. Figure 4.74 presents the case of an object exhibiting such a complex behavior under various magnitudes of grasping force. The object considered here is a cardboard cup without a lid. It is a good example of an object composed of a material that can exhibit multiple deformable stages when it is in interaction with the robotic hand. The deformation stage of the cup varies along with the strength of the applied force.



(a)

(b)

(c)

Figure 4.74: Cardboard cup in various deformation stages: (a) elastic deformation stage under light force; (b) elastic and elasto-plastic deformation stages under medium force; and (c) plastic deformation stage under strong force.

The cup performs an elastic behavior, then a transitional elasto-plastic behavior, to finally reach closer to a plastic behavior when the magnitude of the force increases from light to strong. This is coherent with the expected behavior of such an object than can relatively easily be destroyed under the application of sufficient forces.

Table 4.4: Cup classification under three grasping force strengths

Object Classification / Force	Elastic	Plastic	Elasto-plastic	Rigid
Light	60 (100%)	0	0	0
Medium	35 (58.3%)	0	25 (41.7%)	0
Strong	0	1 (1.7%)	59 (98.3%)	0

Table 4.4 presents the classification results obtained on this object at various deformation stages. The tests are repeated 60 times for each grasping force. The cup initially exhibits an elastic behavior under a light force in Figures 4.74a, where the cup recovers back to its original shape after the applied force is released. It can be observed in the first row (for light force) of Table 4.4 that the classifier categorizes the object as elastic (that is into its elastic stage). Under a medium force and up to a certain limit, the cup remains in its elastic stage, and then transfers to the elasto-plastic stage, as shown in Figures 4.74b, where the cup sometimes recovers partially toward its original shape. As shown in the second row (for medium force) of Table 4.4, the cup is identified either in the elastic (58.3%) and elasto-plastic stages (41.7%) in individual trials among the 60 repetitions. Lastly, when a strong force is applied, the cup continues to exhibit an elasto-plastic behavior and even a plastic behavior, as shown in Figure 4.74c. When the force is strong enough, the object remains in its elasto-plastic deformation stage, and tends to progressively transition to its plastic stage, as shown in the last row (for strong force) of Table 4.4.

This test scenario demonstrates the flexibility of the proposed robotic probing process and material behavior-based classification system to handle a large diversity of objects.

Chapter 5 Conclusion

5.1 Summary

The objective of this research is to propose an innovative approach for monitoring non-rigid object deformation from RGB-D data to support a decision system for automated characterization of deformable object's physical properties with the assistance of a robotic hand.

The approach developed in the thesis efficiently integrates well-established techniques for background removal through plane segmentation and distance-based clustering from RGB-D data, and proposes an original formulation of the fast level set method in the log-polar domain, in order to detect and track the contour of an object of interest in the RGB-D data stream. It is worth noting that the assumption for background removal is coherent in the context of this thesis. The goal of the thesis is to develop an automated classification process for deformable object elasticity, and not to develop fully autonomous solutions for 3D object detection, localization, and manipulation over cluttered backgrounds. Dynamic time warping is employed with a modified scoring scheme in order to characterize the objects material properties based on contours extracted at strategic stages of the manipulation by the robotic hand. The proposed scoring scheme provides tolerance to translations and rotations that the object may exhibit, beyond its deformation, as a consequence of the manipulation.

As a result, the proposed solution is able to extract the contour of an object of interest by either focusing on robustness to achieve successive contour detection over a series of frames, or on speed to achieve faster active contour tracking, supporting up to 23 Hz update rate, which is close to the 30 Hz frame rate supported by the Kinect sensor used in this research. The proposed solution for object classification achieves an average classification rate of 98.3% over all material types considered when a force of at least 3.9 N is applied on the object, and of 88.8% regardless of the force magnitude.

5.2 Contributions

This research makes the following contributions to the fields of machine vision, especially to the areas of contour detection and tracking, object classification, and handling of deformable materials in the perspective of autonomous robotics systems:

• Design of a formulation of the fast level set method that makes it operate in the log-polar domain to robustly and efficiently detect and track the 2D contour of an object of interest as it evolves under robotic manipulation, independently from the placement of the Kinect sensor, and the shape, color and other visual attributes of the object;

• Design and integration of an original framework for automatically characterizing deformation properties (and stages of deformation) of an object under manipulation by a robotic hand using RGB-D data and an adapted classification approach that relies on a sequence of extracted 2D contours;

• Formulation of an original scoring model that makes dynamic time warping (DTW) more robust to variable length contours comparison, along with a decisional flowchart adapted for material behavior-based classification;

• Development of an innovative distance-based clustering algorithm to identify the object of interest from RGB-D data and a robust color selection scheme, which both contribute to minimize human intervention, and enhance object contour detection.

This research led to the publication of two papers: [91] and [92].

5.3 Future Work

In section 3.3.2, a segmentation algorithm for point clouds based on the idea of fixed-radius near neighbors is proposed. In the current work, the algorithm extracts the object of interest from a point cloud through a user-provided point, using a *k*-d tree data structure to optimize the performance of the segmentation and to minimize processing time. Future work could investigate the improvement of the data structure to support a more efficient cluster segmentation. Alternatively, a direction in which this research could be expanded would be to consider more data features (e.g. normals, colors, or even texture) to support a more accurate segmentation.

In section 3.3.6, an original adaptation of the fast level set method to the log-polar representation of the color map is introduced to detect and track the contour of an object. This process consists of a region-based image segmentation. This research could further be expanded by developing edge-based image segmentation in the log-polar domain in order to analyze and extract contours over objects of interest. Extending the formulation to 3D space also represents a promising path, as the proposed system would then take full advantage of the RGB-D data collected by the Kinect sensor, not only to extract the relevant cluster, but also to track the deformation over the entire surface of the object.

In section 3.4, the proposed decision system to characterize an object's deformation properties is based on a Cartesian distribution of the object contour. That contour is obtained by using inverse log-polar mapping from the cortical image where it is actually extracted. Future research could develop a decision system to classify the object's deformation properties directly from the contours estimated in the log-polar domain in order to avoid the need, and computation time, for this inverse log-polar transformation.

Another extension of this work would be to optimize the code of the contour tracking using the CUDA parallel processing platform on GPGPUs to achieve the faster update rates for the planar segmentation, clustering, and contour extraction stages, especially the calculation of the data-dependent speed function defined in section 3.3.6.

References

[1] D. Henrich and H. Wörn, Eds., *Robot Manipulation of Deformable Objects*. Springer London, 2000.

[2] M.-H. Choi, S. C. Wilber, and M. Hong, "Estimating material properties of deformable objects by considering global object behavior in video streams," *Multimed. Tools Appl.*, vol. 74, no. 10, pp. 3361–3375, May 2015.

[3] L. Zaidi, B.-C. Bouzgarrou, L. Sabourin, and Y. Mezouar, "Interaction modeling in the grasping and manipulation of 3D deformable objects," in *2015 International Conference on Advanced Robotics (ICAR)*, 2015, pp. 504–509.

[4] 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)*, 2015, pp. 3914–3921.

[5] J. Schulman, A. Lee, J. Ho, and P. Abbeel, "Tracking deformable objects with point clouds," in *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 1130–1137.

[6] M. Krainin, P. Henry, X. Ren, and D. Fox, "Manipulator and object tracking for in-hand 3D object modeling," *Int. J. Rob. Res.*, vol. 30, no. 11, pp. 1311–1327, 2011.

[7] J. Stückler and S. Behnke, "Perception of Deformable Objects and Compliant Manipulation for Service Robots," in *Soft Robotics*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 69–80.

[8] C. Elbrechter, R. Haschke, and H. Ritter, "Folding paper with anthropomorphic robot hands using real-time physics-based modeling," in *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, 2012, pp. 210–215.

[9] C. M. Mateo, P. Gil, D. Mira, and F. Torres, "Analysis of Shapes to Measure Surfaces," *Proc. IEEE Conf. Informatics Control. Autom. Robot.*, pp. 60–65, 2015.

[10] C. Mateo, P. Gil, and F. Torres, "3D Visual Data-Driven Spatiotemporal Deformations for Non-Rigid Object Grasping Using Robot Hands," *Sensors*, vol. 16, no. 5, p. 640, 2016.

[11] D. Kraft, N. Pugeault, E. Baseski, M. Popovic, D. Kragic, S. Kalkan, F. Wörgötter, and N. Krüger, "Birth of the Object: Detection of Objectness and Extraction of Object Shape through Object-Action complexes," *Int. J. Humanoid Robot.*, vol. 5, no. 2, pp. 247–265, Jun. 2008.

[12] J. Hur, H. Lim, and S. C. Ahn, "3D Deformable Spatial Pyramid for Dense 3D Motion Flow of Deformable Object," in *Advances in Visual Computing*, 2014, pp. 118–127.

[13] D. Navarro-Alarcon, H. M. Yip, Z. Wang, Y.-H. Liu, F. Zhong, T. Zhang, and P. Li, "Automatic 3-D Manipulation of Soft Objects by Robotic Arms With an Adaptive Deformation Model," *IEEE Trans. Robot.*, vol. 32, no. 2, pp. 429–441, Apr. 2016.

[14] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision in C++ with the OpenCV Library*, 2nd ed. O'Reilly Media, Inc., 2013.

[15] J. Kramer, N. Burrus, F. Echtler, H. C. Daniel, and M. Parker, *Hacking the Kinect*. Apress, 2012.

[16] "kinect— ROS.org." [Online]. Available: http://wiki.ros.org/kinect. [Accessed: 10-Apr-2017].

[17] M. Sonka, V. Hlavac, and R. Boyle, *Image Processing, Analysis, and Machine Vision (International student edition)*, 3rd ed. Thomson, 2008.

[18] S. W. Zucker, "Region growing: Childhood and adolescence," *Comput. Graph. Image Process.*, vol. 5, no. 3, pp. 382–399, Sep. 1976.

[19] C. R. Brice and C. L. Fennema, "Scene analysis using regions," *Artif. Intell.*, vol. 1, no. 3–4, pp. 205–226, Jan. 1970.

[20] J. A. Feldman and Y. Yakimovsky, "Decision theory and artificial intelligence: I. A semantics-based region analyzer," *Artif. Intell.*, vol. 5, no. 4, pp. 349–371, Dec. 1974.

[21] S. Beucher and C. Lantuéjoul, "Use of Watersheds in Contour Detection," *Work. Publ.*, 1979.

[22] L. Vincent and P. Soille, "Watersheds in digital spaces: an efficient algorithm based on immersion simulations," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 13, no. 6, pp. 583–598, Jun. 1991.

[23] F. Meyer, "Color image segmentation," *1992 Int. Conf. Image Process. its Appl.*, pp. 303–306, 1992.

[24] F. Meyer, "Topographic distance and watershed lines," *Signal Processing*, vol. 38, no. 1, pp. 113–125, Jul. 1994.

[25] C. Xu, D. L. Pham, and J. L. Prince, "Chapter 3: Image segmentation using deformable models," in *Handbook of Medical Imaging: Medical image processing and analysis*, M. Sonka and J. M. Fitzpatrick, Eds. SPIE Press, 2000.

[26] E. Angelini, Y. Jin, and A. Laine, "State of the Art of Level Set Methods in Segmentation and Registration of Medical Imaging Modalities," in *Handbook of Biomedical Image Analysis*, Boston, MA: Springer US, 2005, pp. 47–101.

[27] S. Osher and J. A. Sethian, "Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations," *J. Comput. Phys.*, vol. 79, no. 1, pp. 12–49, Nov. 1988.

[28] H.-K. Zhao, T. Chan, B. Merriman, and S. Osher, "A Variational Level Set Approach to Multiphase Motion," *J. Comput. Phys.*, vol. 127, no. 1, pp. 179–195, Aug. 1996.

[29] T. F. Chan, B. Y. Sandberg, and L. A. Vese, "Active Contours without Edges for Vector-Valued Images," *J. Vis. Commun. Image Represent.*, vol. 11, no. 2, pp. 130–141, Jun. 2000.

[30] T. F. Chan and L. A. Vese, "A level set algorithm for minimizing the Mumford-Shah functional in image processing," in *Proceedings IEEE Workshop on Variational and Level Set Methods in Computer Vision*, 2001, pp. 161–168.

[31] T. F. Chan and L. A. Vese, "Active contours without edges," *IEEE Trans. Image Process.*, vol. 10, no. 2, pp. 266–277, 2001.

[32] T. F. Chan and L. A. Vese, "Active Contour and Segmentation Models using Geometric PDE's for

Medical Imaging," in *Geometric Methods in Bio-Medical Image Processing*, 2002, pp. 63–75.

[33] L. A. Vese and T. F. Chan, "A Multiphase Level Set Framework for Image Segmentation Using the Mumford and Shah Model," *Int. J. Comput. Vis.*, vol. 50, no. 3, pp. 271–293, 2002.

[34] S. Osher and R. P. Fedkiw, "Level Set Methods: An Overview and Some Recent Results," *J. Comput. Phys.*, vol. 169, no. 2, pp. 463–502, May 2001.

[35] M. Thida, K. L. Chan, and H.-L. Eng, "An Improved Real-Time Contour Tracking Algorithm Using Fast Level Set Method," in *Advances in Image and Video Technology*, 2006, pp. 702–711.

[36] Y. Shi and W. C. Karl, "Real-Time Tracking Using Level Sets," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 2005, vol. 2, pp. 34–41.

[37] Y. Shi and W. C. Karl, "A Real-Time Algorithm for the Approximation of Level-Set-Based Curve Evolution," *IEEE Trans. Image Process.*, vol. 17, no. 5, pp. 645–656, May 2008.

[38] X. Sun, H. Yao, S. Zhang, and D. Li, "Non-Rigid Object Contour Tracking via a Novel Supervised Level Set Model," in *IEEE Transactions on Image Processing*, 2015, vol. 24, no. 11, pp. 3386–3399.

[39] A. Yilmaz, O. Javed, and M. Shah, "Object tracking," *ACM Comput. Surv.*, vol. 38, no. 4, Dec. 2006.

[40] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981.

[41] M. Chessa and F. Solari, "Local Feature Extraction in Log-Polar Images," in *Image Analysis and Processing — ICIAP 2015*, 2015, pp. 410–420.

[42] A. Mishra and Y. Aloimonos, "Active Segmentation," *Int. J. Humanoid Robot.*, vol. 6, no. 3, pp. 361–386, Sep. 2009.

[43] A. Mishra, Y. Aloimonos, and C. L. Fah, "Active segmentation with fixation," in *2009 IEEE 12th International Conference on Computer Vision*, 2009, pp. 468–475.

[44] M. Chessa, S. P. Sabatini, F. Solari, and F. Tatti, "A Quantitative Comparison of Speed and Reliability for Log-Polar Mapping Techniques," in *ICVS 2011: Computer Vision Systems*, 2011, pp. 41–50.

[45] V. J. Traver and F. Pla, "Log-polar mapping template design: From task-level requirements to geometry parameters," *Image Vis. Comput.*, vol. 26, no. 10, pp. 1354–1370, Oct. 2008.

[46] R. B. Rusu, "Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments," PhD thesis, Technische Universität München, 2009.

[47] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, Sep. 1975.

[48] P. Harrington, *Machine Learning in Action*, Illustrate. Manning Publications Company, 2011.

[49] I. H. Witten, E. Frank, and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd ed. Morgan Kaufmann, 2011.

[50] D. Conway and J. M. White, *Machine Learning for Hackers*. O'Reilly Media, 2012.

[51] J. L. Bentley, D. F. Stanat, and E. H. Williams, "The complexity of finding fixed-radius near neighbors," *Inf. Process. Lett.*, vol. 6, no. 6, pp. 209–212, Dec. 1977.

[52] R. F. Sproull, "Refinements to nearest-neighbor searching in k-dimensional trees," *Algorithmica*, vol. 6, no. 1–6, pp. 579–589, Jun. 1991.

[53] M. Muja and D. G. Lowe, "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration," *Int. Conf. Comput. Vis. Theory Appl.*, pp. 331–340, 2009.

[54] M. Muja and D. G. Lowe, "Fast Matching of Binary Features," in *2012 Ninth Conference on Computer and Robot Vision*, 2012, pp. 404–410.

[55] M. Muja and D. G. Lowe, "Scalable Nearest Neighbor Algorithms for High Dimensional Data," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 11, pp. 2227–2240, Nov. 2014.

[56] J. H. Freidman, J. L. Bentley, and R. A. Finkel, "An Algorithm for Finding Best Matches in Logarithmic Expected Time," *ACM Trans. Math. Softw.*, vol. 3, no. 3, pp. 209–226, Sep. 1977.

[57] M. de Berg, M. van Kreveld, M. Overmars, and O. C. Schwarzkopf, *Computational Geometry*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000.

[58] R. A. Brown, "Building a Balanced k-d Tree in O(kn log n) Time," *J. Comput. Graph. Tech.*, vol. 4, no. 1, pp. 50–68, 2015.

[59] K. B. Sun and B. J. Super, "Classification of Contour Shapes Using Class Segment Sets," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2, pp. 727–733.

[60] X. Bai, W. Liu, and Z. Tu, "Integrating contour and skeleton for shape classification," in *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, 2009, pp. 360–367.

[61] A.-M. Cretu, P. Payeur, and E. M. Petriu, "Learning and Prediction of Soft Object Deformation Using Visual Analysis of Robot Interactions," in *Advances in Visual Computing - 6th International Symposium, ISVC 2010*, 2010, pp. 232–241.

[62] E. Gonzalez-Sosa, R. Vera-Rodriguez, J. Fierrez, and J. Ortega-Garcia, "Comparison of Body Shape Descriptors for Biometric Recognition Using MMW Images," in *2014 22nd International Conference on Pattern Recognition*, 2014, pp. 124–129.

[63] M. Müller, *Information Retrieval for Music and Motion*, Illustrate. Springer Berlin Heidelberg, 2007.

[64] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Trans. Acoust.*, vol. 26, no. 1, pp. 43–49, Feb. 1978.

[65] T. M. Rath and R. Manmatha, "Word image matching using dynamic time warping," in *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, vol. 2, p. II-521-II-527.

[66] I. Bartolini, P. Ciaccia, and M. Patella, “WARP: accurate retrieval of shapes using phase of Fourier descriptors and time warping distance,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 1, pp. 142–147, Jan. 2005.

[67] S. H. Kim, Y. Mun, and H. I. Choi, “Improved Curvature Scale Space Based on Image Retrieval Using Dynamic Time Warping,” in *Computational Science and Its Applications – ICCSA 2008*, pp. 32–42.

[68] K. C. Santosh, “Use of Dynamic Time Warping for Object Shape Classification through Signature,” *Kathmandu Univ. J. Sci. Eng. Technol.*, vol. 6, no. 1, pp. 33–49, Jun. 2010.

[69] V. Palazón-González and A. Marzal, “On the dynamic time warping of cyclic sequences for shape retrieval,” *Image Vis. Comput.*, vol. 30, no. 12, pp. 978–990, Dec. 2012.

[70] J. Romero, D. Kragic, V. Kyrki, and A. Argyros, “Dynamic time warping for binocular hand tracking and reconstruction,” in *2008 IEEE International Conference on Robotics and Automation*, 2008, pp. 2289–2294.

[71] S. Majed, “Robust Face Localization Using Dynamic Time Warping Algorithm,” in *Reviews, Refinements and New Ideas in Face Recognition*, InTech, 2011.

[72] D. Lemire, “Faster retrieval with a two-pass dynamic-time-warping lower bound,” *Pattern Recognit.*, vol. 42, no. 9, pp. 2169–2180, Sep. 2009.

[73] G. Al-Naymat, S. Chawla, and J. Taheri, “SparseDTW: A Novel Approach to Speed up Dynamic Time Warping,” *Knowl. Inf. Syst.*, vol. 7, no. 3, pp. 358–386, Jan. 2012.

[74] D. F. Silva and G. E. A. P. A. Batista, “Speeding Up All-Pairwise Dynamic Time Warping Matrix Calculation,” in *Proceedings of the 2016 SIAM International Conference on Data Mining*, 2016, pp. 837–845.

[75] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, 2004.

[76] “pcl::visualization::PCLVisualizer Class Reference — Point Cloud Library (PCL).” [Online]. Available: http://docs.pointclouds.org/trunk/classpcl_1_1visualization_1_1_p_c_l_visualizer.html. [Accessed: 10-Apr-2017].

[77] M. Bolduc and M. D. Levine, “A Review of Biologically Motivated Space-Variant Data Reduction Models for Robotic Vision,” *Comput. Vis. Image Underst.*, vol. 69, no. 2, pp. 170–184, Feb. 1998.

[78] W. Burger and M. J. Burge, *Principles of Digital Image Processing*. London: Springer London, 2009.

[79] A. Leon-Garcia, *Probability, Statistics, and Random Processes For Electrical Engineering*, 3rd ed. Prentice Hall, 2008.

[80] A.-M. Cretu, “Experimental data acquisition and modeling of three-dimensional deformable objects using neural networks,” PhD thesis, University of Ottawa, 2009.

[81] “Confusion matrix — Wikipedia.” [Online]. Available:

https://en.wikipedia.org/wiki/Confusion_matrix. [Accessed: 10-Apr-2017].

[82] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System," *ICRA Work. open source Softw.*, vol. 3, no. 3.2, p. 5, 2009.

[83] "BarrettHand BH8-262 — Barrett Technology, LLC." [Online]. Available: http://support.barrett.com/wiki/Hand/262. [Accessed: 10-Apr-2017].

[84] "What is PCL? — Point Cloud Library (PCL)." [Online]. Available: http://pointclouds.org/about/. [Accessed: 10-Apr-2017].

[85] "About — OpenCV." [Online]. Available: http://opencv.org/about.html. [Accessed: 17-Apr-2017].

[86] "freenect_camera — ROS.org." [Online]. Available: http://wiki.ros.org/freenect_camera. [Accessed: 10-Apr-2017].

[87] "Module filters — Point Cloud Library (PCL)." [Online]. Available: http://docs.pointclouds.org/trunk/group__filters.html. [Accessed: 10-Apr-2017].

[88] "Plane model segmentation — PCL." [Online]. Available: http://www.pointclouds.org/documentation/tutorials/planar_segmentation.php. [Accessed: 10-Apr-2017].

[89] "How to use a KdTree to search — Point Cloud Library (PCL)." [Online]. Available: http://pointclouds.org/documentation/tutorials/kdtree_search.php. [Accessed: 17-Apr-2017].

[90] "cv::LogPolar_Interp Class Reference — OpenCV." [Online]. Available: http://docs.opencv.org/ref/2.4/db/de4/classcv_1_1LogPolar__Interp.html. [Accessed: 10-Apr-2017].

[91] 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, no. 1, art. 5, Mar. 2017.

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