ABSTRACT

Title of dissertation: REASONING ABOUT GEOMETRIC OBJECT INTERACTIONS IN 3D FOR MANIPULATION ACTION UNDERSTANDING

Konstantinos Zampogiannis
Doctor of Philosophy, 2019

Dissertation directed by: Professor Yiannis Aloimonos
Department of Computer Science

In order to efficiently interact with human users, intelligent agents and autonomous systems need the ability of interpreting human actions. We focus our attention on manipulation actions, wherein an agent typically grasps an object and moves it, possibly altering its physical state. Agent-object and object-object interactions during a manipulation are a defining part of the performed action itself. In this thesis, we focus on extracting semantic cues, derived from geometric object interactions in 3D space during a manipulation, that are useful for action understanding at the cognitive level.

First, we introduce a simple grounding model for the most common pairwise spatial relations between objects and investigate the descriptive power of their temporal evolution for action characterization. We propose a compact, abstract action descriptor that encodes the geometric object interactions during action execution, as captured by the spatial relation dynamics. Our experiments on a diverse dataset

confirm both the validity and effectiveness of our spatial relation models and the discriminative power of our representation with respect to the underlying action semantics. Second, we model and detect lower level interactions, namely object contacts and separations, viewing them as topological scene changes within a dense motion estimation setting. In addition to improving motion estimation accuracy in the challenging case of motion boundaries induced by these events, our approach shows promising performance in the explicit detection and classification of the latter. Building upon dense motion estimation and using detected contact events as an attention mechanism, we propose a bottom-up pipeline for the guided segmentation and rigid motion extraction of manipulated objects. Finally, in addition to our methodological contributions, we introduce a new open-source software library for point cloud data processing, developed for the needs of this thesis, which aims at providing an easy to use, flexible, and efficient framework for the rapid development of performant software for a range of 3D perception tasks.

REASONING ABOUT GEOMETRIC OBJECT INTERACTIONS IN 3D FOR MANIPULATION ACTION UNDERSTANDING

by

Konstantinos Zampogiannis

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2019

Advisory Committee:
Professor Yiannis Aloimonos, Chair/Advisor
Dr. Cornelia Fermüller, Co-Advisor
Professor Shihab Shamma, Dean's Representative
Professor Ramani Duraiswami
Professor Matthias Zwicker

Acknowledgments

I would like to thank my research advisors, Prof. Yiannis Aloimonos and Dr. Cornelia Fermüller, for introducing me to the world of robotic perception and generously supporting me throughout my PhD journey. Their continuous guidance and feedback enabled me to develop a deeper, more fundamental perspective on research problems and directions and provided an endless source of inspiration. The topics in this thesis are the embodiment of some of the countless ideas that came up during our discussions. I would also like to thank my doctoral committee members, Prof. Ramani Duraiswami, Prof. Shihab Shamma, and Prof. Matthias Zwicker for the insightful feedback they provided on my work.

I am also grateful for having the chance to know and work with some amazing people in the UMD CS department and UMIACS. The graduate classes I took by Prof. Michael Hicks, Prof. David Jacobs, and Prof. David Mount were a true inspiration and allowed me to expand my knowledge and background beyond my strict area of study. Being a TA for Fawzi Emad, Prof. David Mount, and Nelson Padua-Perez gave me the opportunity to come in contact with young talent and realize the importance of helping it develop. Last but not least, I would like to thank Brenda Chick, Jodie Gray, Tom Hurst, Janice Perrone, and Jennifer Story for their patience and support, and for turning potentially stressful and laborious administrative processes into a breeze.

I would like to thank my colleagues and lab mates Francisco Barranco, Chinmaya Devaraj, Alex Ecins, Matthew Evanusa, Kanishka Ganguly, Gregory Kramida,

Michael Maynord, Anton Mitrokhin, Austin Myers, Chethan Parameshwara, Jack Rasiel, Behzad Sadrfaridpour, Nitin J. Sanket, Snehesh Shrestha, Chahat Deep Singh, Peter Sutor, and Yezhou Yang. In addition to some of my best collaboration experiences, I will always remember our discussions and the fun we had, both within and outside of the lab. I feel particularly fortunate for the valuable friendships that have emerged from this, which, among other things, lead to the creation of the 'Perpetual Rowdy Gangsters' group, an alternative *hue* variant of our official PRG group.

I also feel lucky to have become friends with a number of awesome people during my years in College Park. Lida Apergi, Ioana Bercea, Jason Filippou, Adi Hajj-Ahmad, Garrett Katz, Udayan Khurana, Antonis Kyprianidis, Vassilis Lekakis, Evripidis Paraskevas, Tommy Pensyl, Thodoris Rekatsinas, Eirini Tsiropoulou, Niki Vazou, Kleoniki Vlachou, Kostas Xirogiannopoulos, and Haytham Yaghi, thank you all for being there for me, being good friends and roommates, dragging me out the lab or house, and helping me remain balanced. To the more recent UMD acquisitions and friends: Eleftheria Briakou, Panos Dimitrellos, Ioanna Galani, Leonidas Lampropoulos, Christos Mavridis, Christos Papageorgakis, Sotiria Stathopoulou, and Leonidas Tsepenekas, thank you all for the awesome time we have spent together having fun. You give me hope that the Erie Street spirit will live on – whatever that may entail.

I cannot thank enough my good old friends and NTUA brothers, Haris Angelidakis, Marios-Stavros Grigoriou, and Petros Kapouleas. Thank you for periodically reminding me that I have a life outside of UMD. No matter where in the world we

are or will be, I will always cherish our friendship and dream of the next reunion!

Finally, I cannot express my gratitude for the people closest to me for their unconditional love and support. That would include my family, and the smartest, most beautiful, and most genuinely sweet person I was blessed to have in my life: Moschoula, thank you for *bearing* with me during our joint adventure and know that I am counting on you to always and forever be the light in my life.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1: Introduction

1.1 Motivation and scope

Recent technological advances have lead autonomous systems and intelligent agents to become an ubiquitous part of our everyday life. In order for such systems to successfully and efficiently interact with human users, they need to be able to interpret human actions. Action understanding from image or video observations has been a very active research topic in computer vision [1–4] and robotics [5–7], as it constitutes a core part for a wide range of applications that include content-based video indexing, video summarization, surveillance, human-computer/robot interaction, and robot imitation learning. A family of actions of particular interest is that of *object manipulations*, wherein an agent interacts with objects in their environment, modifying the physical state of the latter in a potentially meaningful way.

During the course of a manipulation action, an agent typically grasps an object and moves it, possibly altering its physical state. At the same time, the handled object may interact in various ways with other objects that are directly or indirectly involved in the manipulation. For example, an object may be picked and placed *inside* another object, or two separate objects may be *joined* together to form a

new, single one. Agent-object and object-object *interactions* during a manipulation are a defining part of the performed action itself [8–11].

In this thesis, we focus on modeling and extracting two types of *geometric object interactions* that manifest as objects move in 3D space during a manipulation. First, we introduce a simple grounding model for the most common pairwise *spatial relations* between objects and investigate the descriptive power of their dynamic evolution for action characterization. Second, we model and detect lower level interactions, namely object *contacts* and *separations*, viewing them as topological scene changes within a dense motion estimation setting. The ability to extract and reason about these types of interactions directly provides semantic cues for manipulation action understanding at the *cognitive* level. In particular, contact detection in a manipulation video can directly answer *what* was grasped and *when*, whereas spatial relations reveal *where* the object was moved relative to its surroundings. Cognitive-level cues and representations have the advantage of being applicable not only to specific tasks that require some form of action understanding (e.g., a discriminative model for classification), but in a range of applications, from traditional recognition to robot imitation learning [12, 13].

1.2 Thesis contributions and outline

All of our proposed models and algorithms, unless otherwise specified, assume 3D observations as input, either in the form of unorganized 3D point clouds (and sequences thereof) or RGBD videos. We briefly summarize our contributions and

thesis structure in the following.

In Chapter 2, building upon a direct grounding model for the most common spatial relations between objects tracked in 3D space, we introduce an abstract representation for manipulation actions that is based on the temporal evolution of the spatial relations between the involved objects. Our proposed descriptor and time-normalized distance measure give rise to an object-agnostic, timing-invariant action representation that is found to be highly descriptive of the underlying high-level manipulation semantics. As a side-product, our spatial relation grounding model can directly output spatial relation predicate scores for a given object pair in a known state, effectively bridging the gap between raw 3D observations and semantically meaningful relation symbols that correspond to spatial prepositions in natural language. This chapter is based on published prior work [14].

In Chapter 3, we consider the problem of dense motion estimation between point cloud geometries that may undergo topological changes. Standard warp field estimation algorithms tend to produce erratic motion estimates on boundaries associated with 'close-to-open' topology changes (i.e. object separations). We overcome this limitation by exploiting an alternative warp field hypothesis that is derived from backward motion: in the opposite motion direction, a 'close-to-open' (separation) topological event becomes 'open-to-close' (contact), which is by default handled correctly. Our proposed pipeline explicitly detects regions of the deformed geometry that undergo topological changes by means of local deformation criteria and broadly classifies them as 'contacts' or 'separations'. Subsequently, our two motion hypotheses (forward and inverted backward) are seamlessly blended on a local basis,

according to the type and proximity of detected topological event regions, resulting in a topology-aware estimate. Our method is evaluated on motion estimation accuracy on a public dataset, as well as on explicit topological event detection and classification on a custom RGBD dataset with topological event annotations. This chapter is based on prior work [15].

In Chapter 4, we present an active, bottom-up pipeline for the detection of actor-object contacts and the subsequent extraction (segmentation) of moved objects and their rigid motions in RGBD manipulation videos. At the core of our approach lies the non-rigid registration algorithm of Chapter 3: we continuously warp a point cloud model of the observed scene to the current video frame, generating a set of dense 3D point trajectories. Under loose assumptions, we employ simple point cloud segmentation techniques to extract the actor and subsequently detect actor-environment contacts based on the estimated trajectories. For each such interaction, using the detected contact as an attention mechanism, we obtain an initial motion segment for the manipulated object by clustering trajectories in the detected contact vicinity and then we jointly refine the object segment and estimate its 6DOF pose in all observed frames. The content of this chapter is based on prior work [16].

In Chapter 5, we introduce `cilantro`, an open-source software library for geometric and general purpose point cloud data processing that was developed for the needs of this thesis. The library aims at being minimal yet versatile, efficient, and easy to use. At the same time, its comprehensive out-of-the-box functionality and flexible design (generic algorithms) enable the rapid implementation of performant point cloud processing software for a wide spectrum of tasks commonly encountered

by 3D perception practitioners and roboticists. This chapter is based on published prior work [17].

Finally, in Chapter 6, we conclude and discuss possible directions for improvements and future research.

Chapter 2: Learning the Spatial Semantics of Manipulation Actions through Preposition Grounding

In this chapter, we introduce an abstract representation for manipulation actions that is based on the evolution of the spatial relations between involved objects. Object tracking in RGBD streams enables straightforward and intuitive ways to model spatial relations in 3D space. Reasoning in 3D overcomes many of the limitations of similar previous approaches, while providing significant flexibility in the desired level of abstraction. At each frame of a manipulation video, we evaluate a number of spatial predicates for all object pairs and treat the resulting set of sequences (Predicate Vector Sequences, PVS) as an action descriptor. As part of our representation, we introduce a symmetric, time-normalized pairwise distance measure that relies on finding an optimal object correspondence between two actions. We experimentally evaluate the method on the classification of various manipulation actions in video, performed at different speeds and timings and involving different objects. The results demonstrate that the proposed representation is remarkably descriptive of the high-level manipulation semantics.

2.1 Introduction

Intelligent robots built for manipulation tasks need to learn how to manipulate. However, given that there is an infinite number of ways to perform a certain manipulation action, such as for example, making a peanut butter and jelly sandwich [18], the robot should be able to store and organize compact representations effectively, rather than simply a large set of examples that are hard to generalize. Various aspects and levels of representations of manipulation actions have been studied, such as objects and tools [19, 20], manipulator trajectories [21, 22], actions and sub-actions [23], object-wise touch relations [10], action consequences or goals [24], etc. In this work, we focus on another crucial aspect of manipulation actions, which is the object-wise spatial relations in 3D space and the correlation of their temporal evolution to the underlying manipulation semantics.

Why are spatial relations crucial for interpreting manipulations? First of all, while most primitive spatial relations can be inferred directly from the perceptual input, their exploitation allows for more complex types of reasoning about not only geometric, but also the underlying physical relations between objects. For example, by perceiving that an apple is "ABOVE" a plate as well as that these two "TOUCH", we can infer that the plate is the supporting object. Secondly, at a higher level, by grounding the spatial relations for the most commonly used prepositions in natural language, we effectively establish a bridge from observation to language and then from language to execution.

Additionally, a correct understanding of object-wise spatial relations for a

given action is essential for a robot to perform the action successfully. For example, when a robot is asked to "Pour a cup of coffee from the pitcher", it not only needs to recognize "cup" and "pitcher" and generate a sequence of trajectories to perform the "pour" action, but also needs to perform geometric precondition checks on the 3D spatial relations between "pitcher" and "cup": unless the "pitcher" tip is "ABOVE" the "cup", "pour" should not be triggered! In other words, a correct understanding of spatial relations can provide the premise of an accurate execution.

Few work has touched upon object-wise spatial relations for manipulation actions, due to the difficulties inherited from object tracking (inevitable occlusions, etc.). A joint segmentation and tracking technique to reason about "touch" and "contain" relations from top-down 2D views is used in [10]. The limitations of their approach come from reasoning in 2D. For example, their system is not able to differentiate between spatial predicates "above" and "in". Other approaches on spatial reasoning [25] require additional equipment, such as motion capture suits, which makes them impractical for more general purpose applications.

The goal of our work is to develop a system which equips the robot with a fundamental and reliable understanding of 3D spatial relations. During both observing and executing manipulation actions, a robot with our proposed capabilities will be able to understand and reproduce the evolution of the spatial relations between involved objects with high accuracy. We show that a 3D spatial relation based representation is highly descriptive of the underlying action semantics.

At the lower level of this work, we developed a system for RGBD object segmentation and tracking that does not require additional markers on the objects

and allows us to overcome difficulties arising from 2D-only image reasoning. Given point cloud representations for all tracked objects, we adopted a straightforward yet intuitive way to model geometric relations for the most commonly used natural language spatial prepositions by partitioning the space around each object. For a given object pair, our spatial relation predicates effectively capture the *spatial distribution* of the first object with respect to the second. The temporal evolution of each such distribution is encoded in a *Predicate Vector Sequence* (PVS). At a higher level, we *a)* propose a novel action descriptor, which is merely a properly ordered set of PVSes for all involved object pairs, and *b)* introduce its associated distance measure, which relies on finding an optimal, in terms of PVS similarity, object correspondence between two given actions in this representation. Experiments on real manipulation videos for various actions, performed with significant amounts of variation (e.g., different subjects, execution speeds, initial/final object placements, etc.), indicate that our proposed abstract representation successfully captures the manipulation's high-level semantic information, while demonstrating high discriminative performance.

2.2 Related work

From the very beginning of robotics research, a great amount of work has been devoted to the study of manipulation, due to its direct applications in intelligent manufacturing. With the recent development of advanced robotic manipulators, work on robust and accurate manipulation techniques has followed quickly.

For example, [26] developed a method for the PR2 to fold towels, which is based on multiple-view geometry and visual processes of grasp point detection. In [27], they proposed learning object affordance models in multi-object manipulation tasks. Robots searching for objects were investigated in [28], using reasoning about both perception and manipulation. [21] and [22] developed manipulation and perception capabilities for their humanoid robot, ARMAR-III, based on imitation learning for human environments. A good survey on humanoid dual arm manipulation can be found in [29]. These works reached promising results on robotic manipulation, but they focused on specific actions without allowing for generalization. Here, we suggest that the temporal evolution of object-wise 3D spatial relations is a highly descriptive feature for manipulation actions and thus can be potentially used for generalizing learned actions.

Several recent works have studied spatial relations in the context of robot manipulation, either explicitly or implicitly. From a purely recognition point of view, the prepositions proposed in [30] can be directly used for training visual classifiers. Recently, [31] built a joint model of prepositions and objects in order to parse natural language commands, following the method introduced in [32]. In [33] and [34], supervised learning techniques were employed to symbolically ground a specific set of spatial relations, using displacements of likely object contact points and histograms that encode the relative position of elementary surface patches as features respectively. However, in general contexts, most common spatial prepositions do not have ambiguous geometric meanings and can, in principle, be directly modeled. Instead of grounding them via learning from large sets of annotated data, we directly model

spatial relations according to their clear geometric interpretation.

In [10], they developed a 2D video segmentation and tracking system, through which they proposed a compact model for manipulation actions based on the evolution of simple, 2D geometric relations between tracked segments. Reasoning with only 2D segments enforces an unwanted and unnecessary type of abstraction in the representation due to the loss of geometric information. Here, instead, we infer spatial relations between segmented point clouds in 3D space, which enables our system to generate representations of controlled levels of abstraction and discriminative power.

2.3 Our approach

2.3.1 Overview of our method

A very brief description of the processing steps involved in our method is provided here and details are discussed in the following subsections.

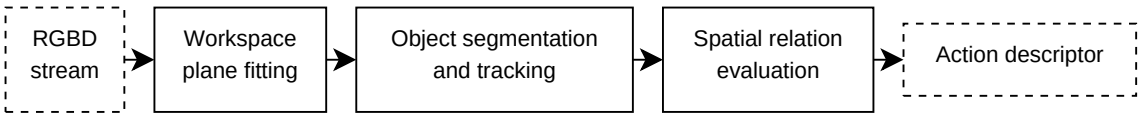


Figure 2.1: Processing steps for our action descriptor extraction.

A schematic of our action descriptor extraction pipeline is given in Fig. 2.1. The input to our algorithm is an RGBD video of a manipulation action. The objects of interest are segmented in the first frame and tracked in 3D space throughout the rest of the sequence (Section 2.3.2). At each time instant, we evaluate a predetermined set of pairwise spatial predicates for each pair of tracked objects

(Section 2.3.3), thus obtaining a sequence of spatial relation descriptors for every object pair (*Predicate Vector Sequence*, PVS). This set of sequences, arranged in a predetermined order, constitutes our proposed action descriptor (Section 2.3.4). Subsequently, we define an appropriate pairwise distance function for this representation, which relies on standard dynamic programming techniques for time series similarity evaluation. Distance computation between two actions is reduced to finding an optimal, in a sense to be defined, object correspondence between the actions (Section 2.3.5).

Assumptions. To simplify certain subtasks, we assume that the action takes place, at least at the beginning, on a planar surface (e.g., on a table) on which all involved objects initially lie. Furthermore, we assume that the depth sensor used to record the manipulation remains still throughout the action duration (no ego-motion). Since our method mostly pertains to applications of robots learning by watching humans or other robots and as long as objects remain visible in order to be reliably tracked, we do not consider the latter to be too restrictive.

2.3.2 Point cloud tracking

The initialization of the tracking procedure involves fitting a plane to the workspace surface in the first RGBD frame of the input sequence. This is done reliably using standard methods, i.e. least squares fitting under RANSAC. The points that lie a certain threshold distance above the estimated plane are clustered based on which connected component of their k-NN graph they belong to. Assuming

that the maximum distance between points that belong to the same object is smaller than the minimum distance between points of different objects, this simple procedure yields an effective segmentation of all the objects in the initial frame.

Treating these initial segments (point clouds) as object models, we initiate one tracker for each object and perform rigid object tracking using the KLD-sampling adaptive Particle Filtering algorithm [35] that is implemented in [36]. This results in a point cloud for each object, for each video frame. We denote this set of point clouds at time index t by $\{X_1^t, \ldots, X_{N_o}^t\}$, where N_o is the number of objects in the action. In the following, we will use the terms X_i^t and "object i (at time t)", for some $i = 1, \ldots, N_o$, interchangeably. A sample output of our point cloud tracker for a two object manipulation recording is depicted in Fig. 2.2.

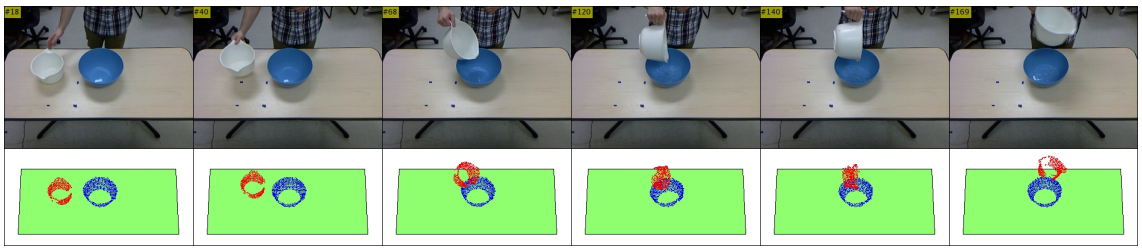


Figure 2.2: Point cloud tracking for a `Pour` action. First row: RGB frames from input video. Second row: workspace plane and point clouds for the two tracked objects.

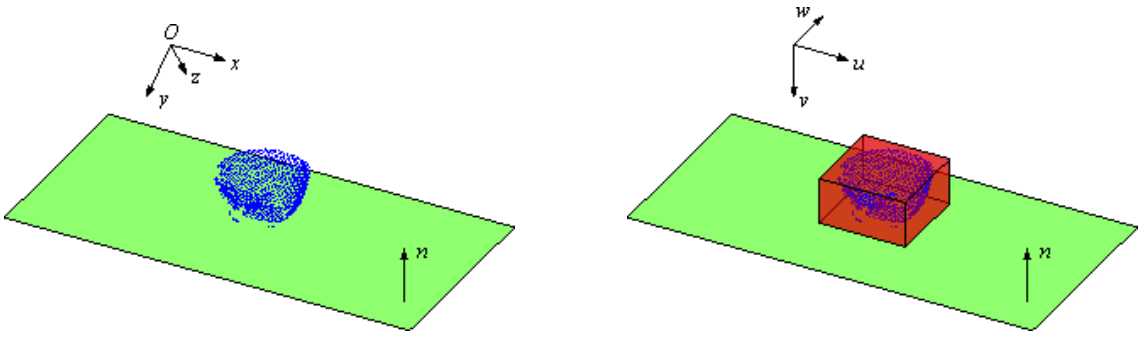
We must note that object segmentation and tracking are not the primary objectives of this study. The approach described here is the one we used in our experiments (Section 2.4) and primarily serves as a showcase for the feasibility of subsequent processing steps using readily available tools. We believe that any reasonably performing point cloud tracking algorithm is adequate for our purposes, as spatial relations can, in general, be quite insensitive to tracking accuracy.

2.3.3 Spatial relations

2.3.3.1 Relative spaces

Here, we describe a straightforward yet intuitive approach to modeling pairwise spatial relations between objects, given their point clouds in 3D space. We begin by defining an auxiliary coordinate frame, whose axes will directly determine the `left/right`, `above/below` and `front/behind` directions. As mentioned previously, we assume a still camera (robot's eyes) looking at the planar surface where the manipulation takes place.

The auxiliary axes directions are calculated using general prior knowledge about the axis orientations of our RGBD sensor world coordinate frame, in which the object point clouds are in known positions, and the workspace normal vector that was estimated during the plane fitting step of segmentation/tracking. Naturally, since the planar workspace (initially) supports all the objects involved in the manipulation, we would want the `above/below` direction to be directly defined by its normal vector n.



(a) (b)

Figure 2.3: Coordinate frames.

Table 2.1: Spatial relation defining directions.

Direction	`left`	`right`	`front`	`behind`	`above`	`below`
Reference vector	$-\hat{u}$	$+\hat{u}$	$-\hat{w}$	$+\hat{w}$	$-\hat{v}$	$+\hat{v}$

Our sensor coordinate frame is drawn in Fig. 2.3a. As we can see, the z-axis corresponds to perceived depth, in roughly the front/behind direction, while the x and y axes point approximately to the right and downwards, respectively. This frame can be rotated so that the new axes, u, v and w, are aligned with the workspace and the aforementioned directions with respect to it (Fig. 2.3b). Axis v can be set to be parallel to n (e.g., pointing downwards). Axis w can then be defined as the projection of the original z-axis to the orthogonal complement of v. Axis u is then uniquely determined as being orthogonal to both v and w (e.g., so that the resulting frame is right-handed). Let $\hat{x}$, $\hat{y}$ and $\hat{z}$ be the unit length vectors that are codirectional with the sensor frame axes, $\hat{n}$ be a unit length workspace normal and $\hat{u}$, $\hat{v}$, $\hat{w}$ be the unit length vectors codirectional with the "workspace-aligned" auxiliary frame axes. The above construction, with the axis directions chosen as in Fig. 2.3b, is captured by the following equations:

$$\hat{v} = \operatorname{sgn}(\hat{y} \cdot \hat{n})\, \hat{n},$$

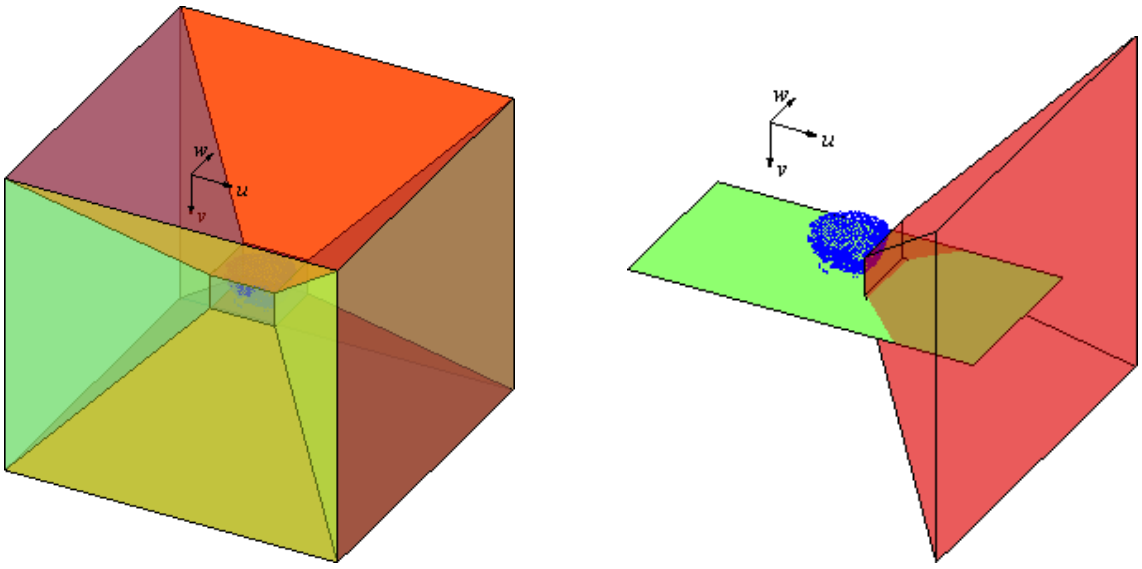
$$\hat{w} = (\hat{z} - (\hat{v} \cdot \hat{z})\, \hat{v}) \,/\, \|\hat{z} - (\hat{v} \cdot \hat{z})\, \hat{v}\|,$$

$$\hat{u} = \hat{v} \times \hat{w},$$

where $a \cdot b$ is the dot product of vectors a and b, $a \times b$ is their cross product, sgn is the signum function and $\|x\|$ is the Euclidean norm of x. Table 2.1 defines the 6

spatial relation defining directions in terms of the auxiliary frame axes.

To infer spatial relations between objects, we will first build models for the regions of 3D space *relative* to an object. For example, to reason about some object being on the `right` of object X at some given time, we first explicitly model the space region on the `right` of X. We will consider 7 space regions with respect to a given object: one that will represent the object interior (for the `in` relation) and 6 around the object, aligned to the directions of Table 2.1. We will represent all these relative spaces of X as convex polyhedra and denote them as $S_r(X)$, for $r \in \{$`in`, `left`, `right`, `front`, `behind`, `below`, `above`$\}$, so that, for example, $S_{\texttt{right}}(X)$ is the space region on the right of X.



(a) (b)

Figure 2.4: Directional relative spaces.

We model $S_{\texttt{in}}(X)$ simply as the smallest bounding box (cuboid) of X that is aligned with the u, v, w axes (Fig. 2.3b for the blue point cloud). The 6 "directional" relative spaces are built upon this one, as indicated in Fig. 2.4a. Consider

a uvw-aligned cube, concentric to $S_{\texttt{in}}(X)$, of edge length significantly larger than the maximum workspace dimension. Clearly, each face of $S_{\texttt{in}}(X)$ with the closest face of the surrounding cube that is parallel to it can uniquely define an hexahedron that also has these two as faces. We will use these 6 hexahedra as models for our directional relative spaces. In Fig. 2.4b, we draw $S_{\texttt{right}}(X)$, where X is represented by the blue point cloud.

A few remarks are in order. First, all relative spaces are represented as sets (conjunctions) of linear constraints, so checking if a point lies in them is very easy. Second, one would expect the directional relative spaces to be unbounded (e.g., polyhedral cones). They can be modeled this way by simply dropping one of their defining linear constraints. Finally, while more elaborate models can be considered for $S_{\texttt{in}}(X)$, like the convex hull of X, the construction of Fig. 2.4a has the nice property of *partitioning* the space around X. This will enable us to easily reason about object relations in a probabilistic manner, in the sense that our (real-valued) relation predicates will define the *spatial distribution* of an object relative to another. Clearly, this provides a more flexible framework than modeling using binary-valued predicates, which can be simply inferred anyway.

2.3.3.2 Spatial relation predicates

We are now ready to define our models for a set of basic pairwise spatial object relations. We will model 7 of them (the inclusion relation and the 6 "directional" ones) directly based on their respective relative space, and an additional one that is

indicative of whether two objects are in physical contact (`touch`). Let $\mathcal{R}^f = \{$`in`, `left`, `right`, `front`, `behind`, `below`, `above`, `touch`$\}$ be our full set of spatial relations. We note that these primitive relations correspond to the spatial prepositions most commonly used in natural language and can be used to model more complex relations (e.g., we can reason about `on` based on `above` and `touch`). For each $r \in \mathcal{R}^f$, we will define a pairwise predicate $R_r(X_i^t, X_j^t)$ that quantifies exactly whether X_i^t is positioned relatively to X_j^t according to relation r. For example, $R_{\texttt{left}}(X_i^t, X_j^t)$ indicates to what degree object i is on the left of object j at time t.

Let $\mathcal{R}^s = \mathcal{R}^f \setminus \{$`touch`$\}$ be the set of relations that can be defined by an explicit space region with respect to X_j^t (reference object). Instead of making hard decisions, i.e. using binary values, we let the predicates $R_r(X_i^t, X_j^t)$, for $r \in \mathcal{R}^s$, assume real values in $[0, 1]$ that represent the fraction of X_i^t that is positioned according to r with respect to X_j^t. A reasonable way to evaluate $R_r(X_i^t, X_j^t)$, for $r \in \mathcal{R}^s$, is then simply as the fraction of points of X_i^t that lie in the relative space $S_r(X_j^t)$:

$$R_r(X_i^t, X_j^t) = \left| X_i^t \cap S_r(X_j^t) \right| / \left| X_i^t \right|, \tag{2.1}$$

where by $|A|$ we denote the cardinality of set A. Since $S_r(X_j^t)$, for $r \in \mathcal{R}^s$, are mutually disjoint by construction, the predicates of (2.1) have the property:

$$\sum_{r \in \mathcal{R}^s} R_r(X_i^t, X_j^t) = 1,$$

i.e. they define a distribution of the points of X_i^t to the relative spaces of X_j^t.

The `touch` relation can be useful in capturing other, non-primitive spatial relations. For example, by having models for both `above` and `touch`, one can express the `on` relation as their "conjunction". More expressive power is desirable, as it can translate to better discriminative performance for our representation. We let our contactual predicate $R_{\texttt{touch}}(X_i^t, X_j^t)$ assume binary values in $\{0, 1\}$. Whenever either of $R_{\texttt{in}}(X_i^t, X_j^t)$ or $R_{\texttt{in}}(X_j^t, X_i^t)$ is greater than zero, we can assume that X_i^t is touching X_j^t and, therefore, $R_{\texttt{touch}}(X_i^t, X_j^t) = 1$. If this were the defining condition for $R_{\texttt{touch}}$, incorporating the contactual predicate to our set of spatial relations would be redundant, in the sense that we already model $R_{\texttt{in}}$. However, there are cases where both $R_{\texttt{in}}(X_i^t, X_j^t)$ and $R_{\texttt{in}}(X_j^t, X_i^t)$ are equal to zero, while X_i^t is touching X_j^t. We may fail to detect this situation using the above intersection test for various reasons. For example, the two objects could simply be extremely close to each other (touching), without part of one lying inside the other. The precision of our sensor and the accuracy of our tracker can also cause the above condition to falsely fail. To compensate for these situations, whenever both $R_{\texttt{in}}(X_i^t, X_j^t) = 0$ and $R_{\texttt{in}}(X_j^t, X_i^t) = 0$, in which case X_i^t and X_j^t are guaranteed to be linearly separable point sets, we perform an additional "proximity" test. We train a linear binary SVM on $X_i^t \cup X_j^t$, with the class labels given by the object ownership for each data point, and use the classifier margin, d_m, as a measure of distance between X_i^t and X_j^t. If d_m falls below a preset threshold d_T, we set $R_{\texttt{touch}}(X_i^t, X_j^t) = 1$. Our complete model of the

contactual predicate is then given by:

$$R_{\texttt{touch}}(X_i^t, X_j^t) = \begin{cases} 1 & \text{if } R_{\texttt{in}}(X_i^t, X_j^t) > 0 \\ & \text{or } R_{\texttt{in}}(X_j^t, X_i^t) > 0 \\ & \text{or } d_m < d_T, \\ 0 & \text{otherwise.} \end{cases}$$

The value of d_T depends, among other things, on point cloud precision related parameters (e.g., sensor and tracking errors) and was set to a few millimeters in our trials. We note that, of all relations in $\mathcal{R}^f$, only `touch` is symmetric.

2.3.3.3 Spatial abstraction

At this point, we have defined our models for all relations $r \in \mathcal{R}^f$. Our goal is to define an action representation using the temporal evolution of spatial object relations. However, tracking all relations in $\mathcal{R}^f$ can yield a representation that is viewpoint-specific or unnecessarily execution-specific. For example, disambiguating between `left` and `right` or `front` and `behind` or, actually, any two of these is clearly dependent on the sensor viewpoint and might not be informative about the actual manipulation semantics. As an example, consider a "stir the coffee" action that involves a cup and a spoon. Picking up the spoon from the left of the cup, then stirring the coffee and finally leaving the spoon on the right of the cup is expected to have the same high-level semantics as picking up the spoon from the right of the cup, stirring and then leaving it in front of the cup. For this reason, we combine

the relations {`left`, `right`, `front`, `behind`} into one, which we can simply name `around`, to obtain a desirable kind of spatial abstraction that, in most cases we can think of, does not leave out information that is actually manipulation-specific. The new relation can be viewed as the disjunction of the 4 ones it replaces and its predicate is given by:

$$R_{\texttt{around}}(X_i^t, X_j^t) = R_{\texttt{left}}(X_i^t, X_j^t) + R_{\texttt{right}}(X_i^t, X_j^t) + R_{\texttt{front}}(X_i^t, X_j^t) + R_{\texttt{behind}}(X_i^t, X_j^t).$$

This makes $\mathcal{R}^a$ = {`in`, `around`, `below`, `above`, `touch`} the set of relations upon which we will build our action descriptors.

2.3.4 Action descriptors

Let $\Phi^t(i,j)$ be the $|\mathcal{R}^a|$-dimensional vector of all relation predicates $R_r(X_i^t, X_j^t)$, $r \in \mathcal{R}^a$, for object i relative to object j at time t, arranged in a fixed relation order, e.g., (`in`, `around`, `below`, `above`, `touch`), so that:

$$\Phi^t(i,j) \equiv \left(R_{\texttt{in}}(X_i^t, X_j^t), \ldots, R_{\texttt{touch}}(X_i^t, X_j^t)\right), \tag{2.2}$$

where $i, j = 1, \ldots, N_o$ and $i \neq j$. Let $\Phi(i,j)$ denote the sequence of the predicate vectors (2.2), for $t = 1, \ldots, T$:

$$\Phi(i,j) \equiv \left(\Phi^1(i,j), \ldots, \Phi^T(i,j)\right).$$

The latter captures the temporal evolution of all spatial relations in $\mathcal{R}^a$ of object i with respect to object j throughout the duration of the manipulation execution. We will call $\Phi(i,j)$ a *Predicate Vector Sequence* (PVS). PVSes constitute the building block of our action descriptors and can be represented as $|\mathcal{R}^a| \times T$ matrices.

Our proposed action descriptors will contain the PVSes for all object pairs in the manipulation. As will become clear in the next subsection, comparing two action descriptors reduces to finding an optimal correspondence between their involved objects, e.g., infer that object i_1 in the first action corresponds to object i_2 in the second. To facilitate this matching task, we require our proposed descriptors to possess two properties.

The first has to do with the fact that the spatial relations we consider are not symmetric. A simple solution to fully capture the temporally evolving spatial relations between objects i and j, where none of the objects acts as a reference point, is to include both $\Phi(i,j)$ and $\Phi(j,i)$ in our descriptor, for $i,j = 1,\ldots,N_o$ and $i \neq j$. This might seem redundant, but, given our predicate models, there is generally no way to exactly infer $\Phi^t(j,i)$ from $\Phi^t(i,j)$ (e.g., due to different object dimensions). Our descriptor will then consist of $N_r = N_o(N_o-1)$ PVSes, as many as the *ordered* object pairs.

Finally, we need to be able to identify which (ordered) object pair a PVS refers to, i.e. associate every PVS in our representation with a tuple (i,j) of object indices. We opted to do this implicitly, by introducing a reverse indexing function and encoding the mapping information in the *order* in which the PVSes are stored. Any bijective function I_{N_o} from $\{1,\ldots,N_r\}$, the set of PVS indices, onto

$\{(i,j) \mid i,j = 1,\ldots,N_o \ \wedge \ i \neq j\}$, the set of ordered object index pairs, is a valid choice as an indexing function. Our proposed action descriptors are then ordered sets of the form:

$$A \equiv (\Phi_1, \ldots, \Phi_{N_r}),$$

where, for $k = 1,\ldots,N_r$, $\Phi_k = \Phi(i,j)$ and $(i,j) = I_{N_o}(k)$. Function I_{N_o} can be embedded in the descriptor extraction process. Utilizing the knowledge of the PVS ordering within an action descriptor will simplify the formulation of establishing an object correspondence between two actions in the following subsection.

2.3.5 Distance measure

To complete our proposed representation, we now introduce an appropriate distance function d on the descriptors we defined above. If A^1 and A^2 are two action descriptors, we design $d(A^1, A^2)$ to be a symmetric function that gives a *time-normalized* distance between the two actions. Additionally, we allow comparisons between manipulations that involve different numbers of objects. This will enable us to reason about the similarity between an action and a *subset* (in terms of the objects it involves) of another action. In the following, for $k = 1, 2$, let N_r^k be the number of PVSes in A^k and N_o^k be the number of objects in manipulation A^k, so that $N_o^k(N_o^k - 1) = N_r^k$.

At the core of our action distance evaluation lies the comparison between PVSes. Each PVS is a time series of $|\mathcal{R}^a|$-dimensional feature vectors that captures the temporal evolution of all spatial relations between an ordered object pair. Dif-

ferent action executions have different durations and may also differ significantly in speed during the course of manipulation: e.g., certain subtasks may be performed at different speeds in different executions of semantically identical manipulations. To compensate for timing differences, we use the Dynamic Time Warping (DTW) [37] algorithm to calculate time-normalized, pairwise PVS distances. We consider the symmetric form of the algorithm in [37], with no slope constraint or adjustment window restriction.

Let $\Phi^1_{r^1}$ and $\Phi^2_{r^2}$ be PVSes in A^1 and A^2, respectively, where $r^1 = 1, \ldots, N^1_r$ and $r^2 = 1, \ldots, N^2_r$. We form the $N^1_r \times N^2_r$ matrix $C = (c_{r^1 r^2})$ of all time-normalized distances between some PVS in A^1 and some PVS in A^2, where:

$$c_{r^1 r^2} = \mathrm{DTW}(\Phi^1_{r^1}, \Phi^2_{r^2}).$$

In the following, we will calculate $d(A^1, A^2)$ as the total cost of an optimal correspondence of PVSes between A^1 and A^2, where the cost of assigning $\Phi^1_{r^1}$ to $\Phi^2_{r^2}$ is given by $c_{r^1 r^2}$.

One could simply seek a minimum cost matching of PVSes between two action descriptors by solving the linear assignment combinatorial problem, with the assignment costs given in C (e.g., by means of the Hungarian algorithm [38]). This would yield an optimal cost *PVS correspondence* between the two actions. However, it is clear that the latter does not necessarily translate to a valid *object correspondence*. Instead, we directly seek an object correspondence between A^1 and A^2 that induces a minimum cost PVS correspondence. The object correspondence between A^1 and

A^2 can be represented as an $N_o^1 \times N_o^2$ binary-valued assignment matrix $X = (x_{ij})$, where $x_{ij} = 1$ if and only if object i in A^1, $i = 1, \ldots, N_o^1$, is matched to object j in A^2, $j = 1, \ldots, N_o^2$. We require that every row and every column of X has at most one nonzero entry and that the sum of all entries in X is equal to $\min(N_o^1, N_o^2)$. This ensures that X defines an one-to-one mapping from the objects in the action involving the fewest objects to the objects in the other action. An object assignment X is then evaluated in terms of the PVS correspondence it defines. We denote the latter by $Y_X = (y_{r^1 r^2})$, where $y_{r^1 r^2} = 1$ if and only if PVS r^1 in A^1 is mapped to r^2 in A^2. The $N_r^1 \times N_r^2$ matrix Y_X has the same structure as X, with $\min(N_r^1, N_r^2)$ nonzero entries. The cost of assignment X, in terms of its induced PVS assignment Y_X, is then given by the sum of all individual PVS assignment costs:

$$J(Y_X) = \sum_{r^1=1}^{N_r^1} \sum_{r^2=1}^{N_r^2} c_{r^1 r^2} y_{r^1 r^2}. \tag{2.3}$$

According to our descriptor definition in the previous subsection, the PVS with index r^1 in A^1 refers to the ordered object pair $(o_1^1, o_2^1) = I_{N_o^1}(r^1)$ and, similarly, r^2 in A^2 refers to $(o_1^2, o_2^2) = I_{N_o^2}(r^2)$ (superscripts indicate action). Clearly, $y_{r^1 r^2}$ is nonzero if and only if object o_1^1 in A^1 is assigned to o_1^2 in A^2 and o_2^1 in A^1 is assigned to o_2^2 in A^2:

$$y_{r^1 r^2} = x_{o_1^1 o_1^2} x_{o_2^1 o_2^2}.$$

Using the above, we can rewrite and optimize (2.3) in terms of the object assignment

variables x_{ij}, for $i = 1, \ldots, N_o^1$ and $j = 1, \ldots, N_o^2$:

$$
\begin{aligned}
&\underset{X}{\text{Minimize}} && J(X) = \sum_{r^1=1}^{N_r^1} \sum_{r^2=1}^{N_r^2} c_{r^1 r^2} x_{o_1^1 o_1^2} x_{o_2^1 o_2^2} \\
&\text{where} && (o_1^1, o_2^1) = I_{N_o^1}(r^1),\ (o_1^2, o_2^2) = I_{N_o^2}(r^2) \\
&\text{subject to} && \textstyle\sum_{j=1}^{N_o^2} x_{ij} \leq 1, \quad i = 1, \ldots, N_o^1 \\
& && \textstyle\sum_{i=1}^{N_o^1} x_{ij} \leq 1, \quad j = 1, \ldots, N_o^2 \\
& && \textstyle\sum_{i=1}^{N_o^1} \sum_{j=1}^{N_o^2} x_{ij} = \min(N_o^1, N_o^2) \\
& && x_{ij} \in \{0, 1\}, \quad i \in \{1, \ldots, N_o^1\}, \\
& && \qquad\qquad\qquad\quad j \in \{1, \ldots, N_o^2\}.
\end{aligned}
$$

The binary quadratic program above encodes an instance of the quadratic assignment problem (QAP), which is NP-hard. QAP instances of size (number of objects) $N > 30$ are considered intractable [39]. However, most manipulations of practical interest involve a number of objects well below that limit. In our implementation, we used the SCIP (constraint integer programming) solver [40]. To evaluate the correctness and running time behavior of our optimization scheme, we ran a small number of tests. For various numbers of objects N, we built an action descriptor A^1 by randomly generating $N(N-1)$ PVSes. We constructed A^2 from A^1 based on an arbitrary object permutation (assignment), by rearranging the PVSes of A^1 (according to I_N and the known object permutation) and adding Gaussian noise to them. Minimization of $J(X)$ gave back the correct object assignment, even for significant amounts of noise variance. Running time for $N = 10$ objects was in

the order of a few (≈ 10) seconds on a laptop machine.

The minimum value of $J(X)$, over all possible object assignments, directly defines the distance between actions A^1 and A^2:

$$d(A^1, A^2) = \min_X(J(X)).$$

The function $d(A^1, A^2)$ is symmetric and, being a sum of DTW distances, gives a time-normalized measure of action dissimilarity. As noted before, $d(A^1, A^2)$ is also defined when A^1 and A^2 involve different numbers of objects ($N_o^1 \neq N_o^2$). For example, if $N_o^1 < N_o^2$, $d(A^1, A^2)$ is expected to be exactly the same as if A^2 only involved the N_o^1 of its objects to which the objects in A^1 are assigned. This flexibility can be useful in sub-action matching scenarios.

2.4 Experiments

2.4.1 Data description

All our experiments were performed on a set of 21 RGBD sequences of manipulation executions. All actions involve 2 objects and are partitioned in 4 distinct semantic classes:

- `Pour`: water poured from a pitcher into a bowl (8 executions).
- `Transfer`: small object placed inside a bowl (6 executions).
- `Stack`: a book placed on top of another (2 executions).
- `Stir`: bowl content stirred using a ladle (5 executions, one of them performed

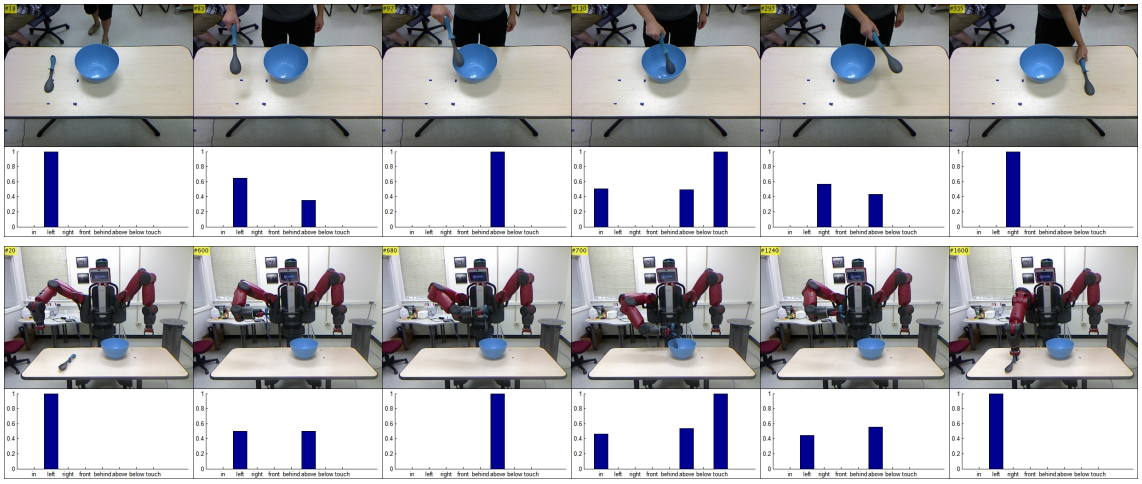


Figure 2.5: Spatial relations evolution: ladle relative to bowl for two instances of `Stir`.

by our robot).

Executions within each semantic class were performed by various individuals and with various initial and final positions of the manipulated objects. For example, in some instances of `Stir`, the ladle was initially picked from the left of the bowl and was finally placed on its left, in others, it was picked from the right and then placed on the left, etc. Naturally, there were significant timing differences across instances of the same semantic class (different overall durations and execution speeds at each action phase).

We also included a robot execution for an instance of `Stir` (Fig. 2.5, bottom rows) that took roughly 4 times the average human execution duration to complete and demonstrated disproportionately long "idle" phases throughout the manipulation. Our robot platform is a standard Baxter humanoid with parallel grippers. To generate trajectories, we used predefined dynamic movement primitives [41]. The trajectory start and end points were given from the point cloud segmentation and transferred onto the robot via a standard inverse kinematics procedure. We also

used visual servoing to ensure a firm grasp of the tool.

2.4.2 Spatial relations evaluation

We begin with a quick evaluation of the performance of our spatial relation models. To establish our ground truth, we sampled 3 time instances from each execution sequence. To evaluate a rich enough set of spatial relations, we sampled at roughly the beginning, the middle and the end of each manipulation. For each sample frame, we picked one of the objects to act as reference (say X_1) and picked the relation $r \in \mathcal{R}^s = \{\texttt{in}, \texttt{left}, \texttt{right}, \texttt{front}, \texttt{behind}, \texttt{below}, \texttt{above}\}$ that best described the position of the second object, X_2, relative to X_1. For testing, we calculated the spatial predicates $R_r(X_2, X_1)$, for all $r \in \mathcal{R}^s$ for all 63 annotated frames and labeled each according to the relation of maximum predicate value. This gave a classification error rate of $3/63 \approx 4.8\%$. However, 2 of the errors were due to tracking issues and not the spatial predicates per se: in both `Stack` executions, significant part of the book at the bottom overlapped the top one (the tracked point clouds were barely distinguishable), so `in` dominated `above`. The third error was from a `Stir` instance (during the stirring phase), where we decided `in` with a ground truth of `above`, which was the second largest predicate. Overall, we believe that, up to severe tracking inaccuracy, our spatial relation estimation can be considered reliable.

In Fig. 2.5, we depict part of the temporal evolution of the spatial predicate vector of the ladle relative to the bowl for two executions of `Stir` (samples from

the corresponding PVS for all relations in $\mathcal{R}^f$): one performed by a human and one by our robot. From the figure, we can see that our system can reliably track the temporal evolution of spatial relations in both observation and execution scenarios.

2.4.3 Action classification

To evaluate the discriminative performance of our proposed representation, we begin by forming the matrix $D = (d_{ij})$ of all pairwise distances for our $N_a = 21$ manipulation executions, where $d_{ij} = d(A^i, A^j)$, for $i, j = 1, \ldots, N_a$. We depict the values of D in Fig. 2.6. As expected, D is symmetric with zero diagonal. Manipulation executions of the same semantic class were grouped together to consecutive indices (e.g., 1-8 for `Pour`, 9-14 for `Transfer`, 15-16 for `Stack` and 17-21 for `Stir`). Given this, the evident block-diagonal structure of low distance values in D (blue regions) suggests that the proposed representation can be quite useful in classification tasks.

To confirm this intuitive observation, we considered a clustering scenario by applying the Affinity Propagation algorithm [42] on our data. Affinity Propagation is an unsupervised clustering algorithm that does not assume prior knowledge of the number of classes. Instead, the resulting number of clusters depends on a set of real-valued "preference" parameters, one for each data point, that express how likely the point is to be chosen as a class "exemplar" (cluster centroid). A common choice [42] is to use the same preference value for all points, equal to the median of all pairwise similarities. A similarity measure between actions A^i and A^j, for $i, j = 1, \ldots, N_a$, is

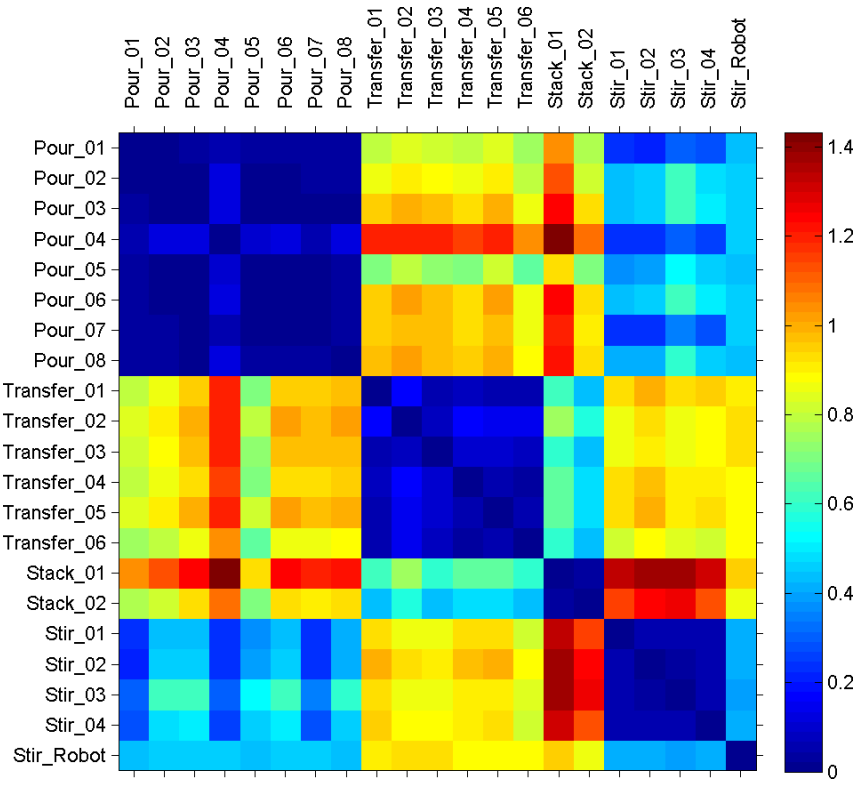


Figure 2.6: Matrix $D = (d_{ij})$ of pairwise distances.

directly given by $s_{ij} = -d_{ij}$. Clustering using this scheme resulted in 4 clusters that correctly corresponded to our 4 semantic classes and there were no classification errors. In Fig. 2.7, we plot a 2-dimensional embedding of the action descriptors for all executions, where we use the same color for all data points of the same cluster and mark the cluster centroids. The outcome of our simple clustering experiment confirms our intuition about matrix D (Fig. 2.6), suggesting that our proposed abstract representation is indeed descriptive of the actual high-level manipulation semantics.

It is worth noting that the descriptor for the robot stirring scenario was correctly classified as a `Stir` instance (marked in Fig. 2.7). This empirically shows that, even when the specific movement trajectories are quite different, e.g., between

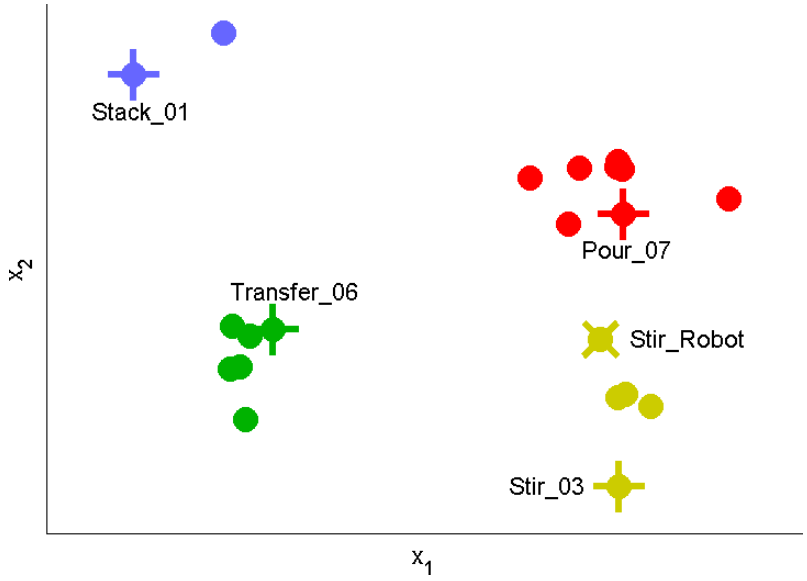


Figure 2.7: Clustering and embedding of our action descriptors in 2 dimensions, based on our similarity/distance measure.

human trials and robot executions, our PVS-based representations remain relatively invariant under the proposed distance function. Thus, our learned from observation manipulation representations could be used to provide additional constraints for robot control policies.

2.5 Summary

In this chapter, we introduced our direct take on grounding spatial relations, by properly partitioning the space around an object to a set of relative spaces, and then proposed a novel compact representation that captures the geometric object interactions during the course of a manipulation. Experiments conducted on both human and robot executions validate that *1)* our relative space models successfully capture the geometric interpretation of their respective relation and the corresponding predicates can be reliably evaluated; *2)* the temporal evolution of object-wise

spatial relations, as encoded in our abstract representation, is indeed descriptive of the underlying manipulation semantics.

Knowledge of models for spatial relations for common natural language prepositions is a very strong capability by itself. First, it enables certain types of nontrivial spatial reasoning, which is a fundamental aspect of intelligence. Second, it narrows the gap between observation and execution, with language acting as the bridge. Particularly, from the human-robot interaction perspective, a robot equipped with these models will be able to answer a rich repertoire of spatial queries and understand commands such as: "pick up the object on the left of object `X` and in front of object `Y`". Another such task, closer in spirit to this work, is the automatic generation of natural language descriptions for observed actions.

In this work, the matching of our spatial-relation sequence representations is performed at once on whole sequences. Currently, we are investigating the possibility of modifying the matching algorithm into an online one. An online action matching algorithm is needed if we want a fast system that observes actions and is able to *predict* during their course (e.g., for monitoring the correctness of execution). Additionally, here, we only took into account one aspect of manipulation actions: object-wise spatial relations. A complete action model, that would attempt to bridge the gap between observation and execution, needs to be a multi-layer combination of spatial relations and many other aspects, such as movement trajectories, objects, goals, etc.

Chapter 3: Topology-Aware Non-Rigid Point Cloud Registration

In this chapter, we introduce a non-rigid registration pipeline for pairs of unorganized point clouds that may be topologically different. Standard warp field estimation algorithms, even under robust, discontinuity-preserving regularization, tend to produce erratic motion estimates on boundaries associated with 'close-to-open' topology changes. We overcome this limitation by exploiting backward motion: in the opposite motion direction, a 'close-to-open' event becomes 'open-to-close', which is by default handled correctly. At the core of our approach lies a general, topology-agnostic warp field estimation algorithm, similar to those employed in recently introduced dynamic reconstruction systems from RGBD input. We improve motion estimation on boundaries associated with topology changes in an efficient post-processing phase. Based on both forward and (inverted) backward warp hypotheses, we explicitly detect regions of the deformed geometry that undergo topological changes by means of local deformation criteria and broadly classify them as 'contacts' or 'separations'. Subsequently, the two motion hypotheses are seamlessly blended on a local basis, according to the type and proximity of detected events. Our method achieves state-of-the-art motion estimation accuracy on the MPI Sintel dataset. Experiments on a custom dataset with topological event annotations

demonstrate the effectiveness of our pipeline in estimating motion on event boundaries, as well as promising performance in explicit topological event detection.

3.1 Introduction

Motion estimation in 3D is a problem of great importance in computer vision, robotics, and computer graphics, playing a central role in a wide range of applications that include 3D scene reconstruction/modeling, human and object pose tracking, robot localization, augmented reality, human-computer interfaces and deformable shape manipulation. The advent of affordable, commercial depth sensors has caused significant research effort on 3D motion estimation from 3D input, leading to the development of RGBD algorithms for fast visual odometry [43,44], efficient and accurate scene flow estimation [45, 46], as well as notable SLAM systems for both static [47, 48] and dynamic [49, 50] environments.

Given the availability of 3D input, dense non-rigid registration is the most general motion estimation problem and it is particularly challenging. In its general form, the problem can be described as computing a motion field, densely supported on the surface of a 3D shape, that deforms the latter in order to geometrically align it to another, fixed "template" shape. This process of non-rigid 3D registration shares fundamental similarities with 2D image registration, known in the computer vision community as optical flow estimation: both problems pose similar challenges in deriving formulations that lead to accurate alignment while encoding reasonable prior constraints (regularization) to overcome ill-posedness.

A classical problem variant that is closely related to 3D non-rigid registration is that of RGBD scene flow. Given a pair of images, scene flow refers to the per-pixel 3D motion of observed points in space from the first frame to the second; optical flow refers to the per-pixel 2D projected motion. There have been a number of successful recent works on scene flow estimation from RGBD frame pairs, following both classical (variational) [45, 46, 51–53] and deep learning [54] frameworks. While of great relevance to a number of motion reasoning tasks, RGBD scene flow targets a specific instance of dense 3D motion estimation, as it inherently registers pairs of 2.5D geometries (depth maps). This hinders its application in scenarios that require alignment of *fully 3D* geometries, such as *model-to-frame* registration for dynamic reconstruction or *model-to-model* shape deformation.

Recently introduced dynamic reconstruction pipelines from RGBD input [49, 50, 55, 56] solve a more general problem by implementing warp field optimization algorithms for their *model-to-frame* registration step. Despite adopting different approaches for their model representations and surface fusion steps, they all rely on similar, point cloud based formulations for non-rigid registration. Scenes with dynamic topology are a challenging case for dynamic reconstruction systems: [49] and [50] make no provisions at all for these cases, while [55] and [56] deal with registration errors that occur because of dynamic topology at a subsequent stage, by discarding problematic regions and reinitializing model tracking. The fully volumetric approaches of [57] and [58] do not use point representations for registration, directly aligning Signed Distance Fields (SDFs) [59] instead. While they intrinsically handle topological changes, significant scalability limitations are introduced by

relying on volumetric representations. To the best of our knowledge, there exists no non-rigid *point cloud* registration algorithm producing warp fields that are error-free on motion boundaries induced by dynamic scene topology.

We note that, throughout our discussion, we use the term 'point cloud' to refer to a geometry representation by a *discrete* point set sample of the underlying surface, as opposed to a volumetric 3D image. Our broad definition does not preclude additional per-point attributes. Therefore, oriented point clouds (point sets equipped with per-point normals) and surfel clouds (oriented point clouds with per-point radii) both fall within what we refer to simply as point cloud based representations.

Contributions. We introduce a complete pipeline for the non-rigid registration of unorganized, oriented 3D point cloud pairs, which explicitly detects topology changes between the input point sets and produces piecewise-smooth warp fields that respect motion boundaries that result from these events. At the core of our approach lies a general warp field estimation algorithm (Section 3.3.3), inspired by those employed in recent dynamic reconstruction systems from RGBD input. We improve motion estimation on motion boundaries associated with topology changes in an efficient post-processing phase (Section 3.3.4) that exploits the different properties of warp fields that are estimated in different directions (i.e. forward and backward) with respect to different types of topological events (i.e. 'contact' or 'separation', Section 3.3.2). After explicitly detecting regions of topology change events by means of simple, intuitive tests of local deformation, our method blends the forward and inverted backward motion hypotheses on a local basis, based on

the type and proximity of detected events, ensuring smooth, seamless hypothesis transitions on the deformed surface. This stage makes no assumptions about the underlying registration engine and can be easily adapted for integration into existing pipelines. The implementation of our warp field estimation module (without the topology event handling) is openly available as part of our point cloud processing library [17]. Furthermore, the ability to detect and classify motion boundaries associated with dynamic topology is a byproduct of our pipeline that may be useful in tasks beyond geometric registration.

After discussing related work, we present our proposed method in detail in Section 3.3. In Section 3.4, we provide two kinds of quantitative evaluation of our approach, focusing on our registration accuracy and the effectiveness of our topology event handling framework, respectively.

3.2 Related work

RGBD scene flow estimation. The term 'scene flow' was introduced in [60] to refer to "the three-dimensional motion field of points in the world, just as optical flow is the two-dimensional motion field of points in an image." Since then, significant research focus has shifted towards scene flow estimation from RGBD input. The formulation of [51] couples an L^1-norm data term derived from the *optical flow* and *range flow* [61] constraints with weighted TV regularization. In [52], the authors follow a similar variational approach but use a rigid motion parameterization of the flow field, computing 6DoF per-pixel transformations and enforcing a local rigidity

prior on the solution. A 6DoF local parameterization is also used in [62], which introduces a correspondence search mechanism that relies on 3D spheres rather than image plane patches, and effectively handles large displacements. In [63], a probabilistic approach for joint segmentation and motion estimation method is proposed; a depth-based segmentation is used for motion estimation, which is in turn regularized based on the mean rigid motion of each layer. A joint segmentation and scene flow estimation method is also presented in [53], which assumes that the scene movement can be described by a small number of latent rigid motions. Starting with a spatial k-means clustering for the motion label initialization, the algorithm iterates between motion estimation and segmentation (soft labeling), merging labels in the process. In [45], the first real-time RGBD variational scene flow method is introduced, achieving state-of-the-art accuracy. An efficient joint odometry and piecewise-rigid scene flow estimation method is proposed in [46], where the scene is segmented into 'static' and 'moving' geometric clusters, from which odometry and independent non-rigid motions are computed.

As mentioned in our introduction, scene flow solves a somewhat restricted problem in the context of dense 3D registration, as the support of the computed motion field is image bound.

Dynamic scene reconstruction. General non-rigid 3D registration algorithms have been developed in the context of online reconstruction of dynamic scenes from RGBD input. Most of them are formulated within a non-rigid Iterative Closest Point (ICP) framework, similar to the one introduced in [64], with the goal of registering a point cloud representation of the scene model to the current

frame, while there also exist purely volumetric approaches [57] that align Signed Distance Field (SDF) geometry representations. DynamicFusion [49] was the first system to achieve high quality, real-time dense reconstructions from RGBD input. While it performs volumetric (SDF) fusion [59], its warp field estimation algorithm is based on oriented point cloud renderings of the model geometry. The estimated warp field is defined on a sparse 'Embedded Deformation' (ED) graph [65], with a 6DoF transformation attached to each node, and its evaluation on arbitrary points is performed via interpolation. The registration objective consists of a point-to-plane ICP cost, coupled with an 'As-Rigid-As-Possible' (ARAP) [66], hierarchically defined regularization term, both under robust loss functions. The non-rigid tracker of VolumeDeform [50] does not rely on an ED graph and estimates individual 6DoF transformations for every source geometry point. Its cost function consists of a dense point-to-plane cost, a sparse point-to-point term derived from SIFT [67] correspondences, and an ARAP prior based on a 'flat' neighborhood graph, with all terms being quadratic. Fusion4D [55] combines the input of multiple range cameras for the task of dynamic reconstruction, using an ED warp field parameterization and following a similar registration objective formulation that additionally includes a 'visual hull' term. CoFusion [68] and MaskFusion [69] segment, using semantic and motion cues, and reconstruct multiple moving objects in real-time, assuming that every object moves rigidly. SurfelWarp [56] is a purely point (surfel) cloud based approach that also relies on an ED motion field representation and uses the same registration costs as DynamicFusion, but under the quadratic loss function. On the other end of the spectrum, KillingFusion [57] and SobolevFusion [58] are purely vol-

umetric approaches that rely on direct SDF-to-SDF alignment [70] via variational minimization under novel regularizers that enforce the motion field to be isometric and preserve level set geometry.

All of the above systems produce results of remarkable quality, especially given their real-time budget. However, with the exceptions of [55], [56], and [57], they cannot handle scenes with dynamic topology, with the 'close-to-open' case ('separation', in our terminology) being particularly problematic. According to our introductory discussion, [55] and [56] deal with these cases essentially by discarding affected regions, while the volumetric registration approaches of [57] and [58] are inherently immune to these events. Our proposed method is the first to tackle dynamic topology within the context of *motion estimation* and within a scalable *point-based* representation framework.

3.3 Our approach

3.3.1 Problem statement

Given a pair of unorganized 3D point sets, our goal is to estimate a warping function that non-rigidly deforms the first point cloud (source geometry) towards the second one (target) in a piece-wise smooth manner.

Let $S = \{x_i^s\} \subset \mathbb{R}^3$ and $D = \left\{x_i^d\right\} \subset \mathbb{R}^3$ be the source and target geometry point sets, respectively, and $\mathcal{W} : \mathbb{R}^3 \mapsto \mathbb{R}^3$ be a warping function. In our non-rigid alignment setting, $\mathcal{W}$ is required to have the following properties:

- The image of point set S via $\mathcal{W}$, $\mathcal{W}[S]$, should be aligned as close to the target

geometry D as possible. Typically, this is formulated as the minimization of the sum of residuals between points in $\mathcal{W}[S]$ and their corresponding points (e.g., nearest neighbors) in D.

- Local transformations of neighboring points that lie on the same moving surface in S should be similar; i.e. $\mathcal{W}$ should be *smooth*. At the same time, motion discontinuities should be preserved: neighboring points in S that lie on independently moving surfaces should be allowed to have different local transformations. This combined prior is known as *piecewise-smoothness*.

In a typical registration objective minimization formulation, the first property is expressed by the sum of registration residuals (e.g., point-to-point and/or point-to-plane distances) over corresponding point pairs in the objective (*data term*), while the second one renders the otherwise under-constrained problem well-posed by introducing terms that penalize differences in local transformations of neighboring points (*regularization term*).

The loss function used to model the regularization penalty terms, plays an important role in the behavior of the warping function in motion boundary regions. For example, it is well known from the optical flow literature that quadratic regularization tends to oversmooth motion boundaries. On the other hand, robust loss functions (e.g., L^1-norm approximations for the penalty terms) are more effective in producing piecewise-smooth motion fields that preserve discontinuities.

In this work, we focus on estimating warp fields that respect motion boundaries resulting from changes in scene *topology*. Our notion of topology is directly derived from object-level connectivity: a change in scene topology can occur either

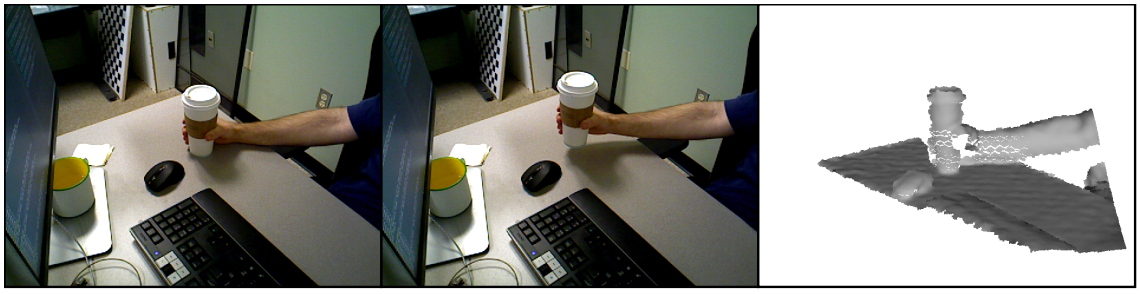
when two or more separate objects come into contact or when two or more initially connected objects separate. We note that we use the term 'object' simply to refer to independently moving scene surface regions, without attaching to them any semantic meaning or assuming prior knowledge thereof.

In the following, we show that simply adopting a robust loss function for regularization still produces visible warping artifacts in motion discontinuity regions that result from scene topology changes, and we present a complete registration pipeline that effectively and efficiently solves this problem.

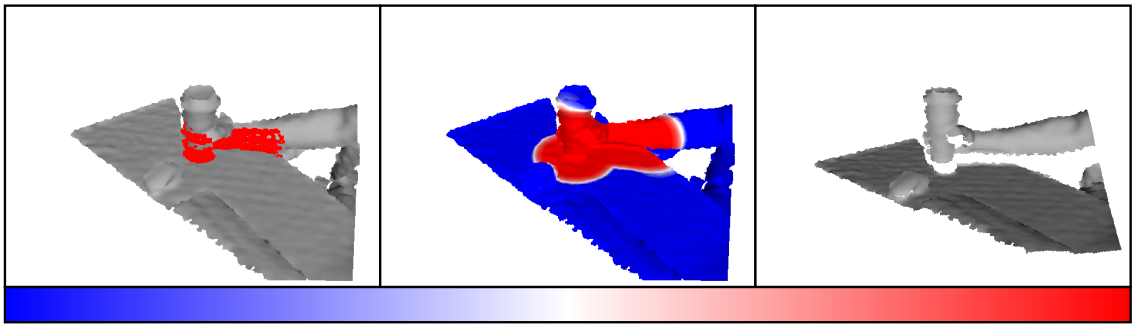
3.3.2 Motivation and overview of our approach

Motion estimation errors on motion boundaries typically manifest as oversmoothing of the warp field because of excessive regularization and can be suppressed by eliminating regularization penalty terms for points in S that lie on different sides of the discontinuity. However, without any knowledge about S and its motion (e.g., some form of segmentation into independently moving objects), we cannot obtain a "correct" regularization graph a priori. Instead, the common choice is to use a k-NN graph of points in S to define the regularization terms. It is easy to see that this choice is particularly problematic in cases where connected objects in S move apart in D, as k-NN regularization over S will introduce penalty terms that relate points that lie on different objects, resulting in some amount of motion field oversmoothing over the separation boundary.

Such a challenging scenario that involves object 'separations' is depicted in



(a) Left, middle: color frames (source and target) captured by an RGBD camera. Right: warped source frame by the result of a standard non-rigid registration algorithm.



(b) Results of our proposed method. Left: detected (red) topological event regions (Section 3.3.4.1). Middle: blending weight $w_b^i \in [0, 1]$ for the inverted backward hypothesis using a 'blue-to-red' colormap (blue: forward, red: inverted backward) shown at the bottom (Section 3.3.4.2). Right: source frame transformed by our topology-aware warp field.

Figure 3.1: Non-rigid registration under a 'close-to-open' topology change.

Fig. 3.1a, where the general, topology-agnostic warp field estimation algorithm described in Section 3.3.3 was used to non-rigidly align two RGBD frames. Despite the fact that the algorithm's regularization term is formulated based on the robust, discontinuity-preserving Huber-L^1 loss function (see Section 3.4.1 for parameter details), the registration result (warped source geometry) shows visible artifacts near the object separation areas. Quadratic regularization is known to induce even more excessive smoothing on motion boundaries. Since quadratic and L^1-norm approximation regularization types are the most commonly used ones in the literature, most current non-rigid registration algorithms are expected to exhibit a very similar behavior on these types of motion boundaries.

At the same time, it is clear that any change in topology between S and D will be directly reflected on the (different) nearest-neighbor graph structures of the source and target geometries. We exploit this fact in the following way. Consider the case where two or more objects that are connected in S become separate in D. As discussed above, estimating the warp field $\mathcal{W}_S^f$ that aligns S to D using the source geometry's k-NN graph to define the regularization penalties is expected to result in some amount of oversmoothing over the motion boundary of the separation. However, in the *backward* motion direction (from D to S), the same topology change manifests as a connection of separate objects. Estimating the backward warp field $\mathcal{W}_D^b$ that aligns D to S using the target (D) geometry's k-NN graph to define the regularization penalties should not exhibit any oversmoothing over the connection boundary, because the corresponding k-NN graph edges that would define regularization terms over the discontinuity are not there in the first place. Inverting the warping function $\mathcal{W}_D^b$ yields another *forward* warp field hypothesis, $\mathcal{W}_S^b$, that will be free of oversmoothing over separation motion boundaries. Of course, the latter, being derived from $\mathcal{W}_D^b$, may suffer from oversmoothing over motion boundaries that correspond to object separations in the backward motion direction (from D to S), or, equivalently, to objects coming into contact from S to D. These cases are expected to be handled correctly in the first place by the standard forward warp, $\mathcal{W}_S^f$.

Based on the above observations, the standard *forward* warp $\mathcal{W}_S^f$ is expected to exhibit good behavior over *contact* boundaries, but to oversmooth *separation* boundaries. On the other hand, the *inverted backward* warp $\mathcal{W}_S^b$ is expected to

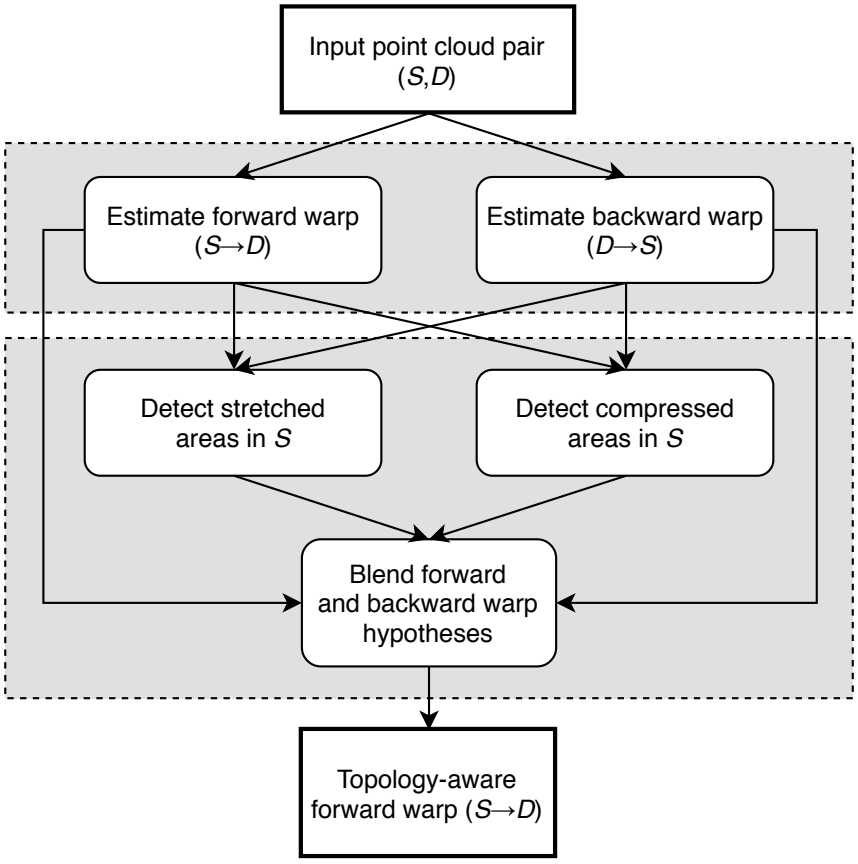


Figure 3.2: Overview of our topology-aware non-rigid registration pipeline.

behave the opposite way, preserving *separation* discontinuities, but possibly blurring motion estimates in *contact* areas. Our proposed registration pipeline builds upon this idea by first detecting regions in S that are likely to be contact or separation boundaries, and then locally blending the warp hypotheses $\mathcal{W}_S^f$ and $\mathcal{W}_S^b$ accordingly in a seamless manner. The final result is a piecewise-smooth warp field that aligns S to D and respects motion boundaries because of changes in scene topology.

An overview of our approach is provided in Fig. 3.2. The (topology-agnostic) warp field estimation algorithm used to obtain the initial forward and backward warp hypotheses is described in detail in Section 3.3.3. Our topology event detection mechanism, as well as our local hypothesis blending approach, are presented in

Section 3.3.4.

3.3.3 Warp field estimation

We implement our warp field estimation algorithm within a non-rigid Iterative Closest Point (ICP) framework [64], similarly to the non-rigid trackers used in the recently introduced dynamic reconstruction pipelines of [49], [50], and [56].

Algorithm 1 $\mathcal{W} = \text{NonRigidICP}(D, S, \mathcal{W}_0)$

1: $\mathcal{W} \leftarrow \mathcal{W}_0$
2: **repeat**
3: $\quad S' \leftarrow \mathcal{W}[S]$
4: $\quad \mathcal{C} \leftarrow \text{FindCorrespondences}(D, S')$
5: $\quad \mathcal{W}_{\text{iter}} \leftarrow \text{OptimizeWarpField}(D, S', \mathcal{C})$
6: $\quad \mathcal{W} \leftarrow \mathcal{W}_{\text{iter}} \circ \mathcal{W}$
7: **until** $\mathcal{W}_{\text{iter}}$ is close to the identity warp

Given the source and target geometries S and D, represented as oriented point clouds, as well as an initial estimate $\mathcal{W}_0$ of the unknown warp field $\mathcal{W}$ (usually taken as the identity warp), the algorithm iteratively refines the latter until convergence has been reached. At the top level, the process iterates between a point correspondence search step between the warped source $\mathcal{W}[S]$ (according to the current $\mathcal{W}$ estimate) and D, and a warp field optimization step that updates $\mathcal{W}$ given the established point correspondences (Algorithm 1). The two algorithm phases are presented in detail in the following.

3.3.3.1 Correspondence association

Our framework supports two complementary types of point correspondences between the (warped) source and the target geometries: *dense* correspondences that are established based on spatial point proximity, and *sparse* correspondences that result from keypoint matching. Each individual correspondence is represented as a pair of point indices, whose first component indexes a point in S and its second one a point in D: $\mathcal{C} = \{\mathcal{C}_{\text{dense}}, \mathcal{C}_{\text{sparse}}\}$, where $\mathcal{C}_{\text{dense}}, \mathcal{C}_{\text{sparse}} \subseteq \{1, \dots, |S|\} \times \{1, \dots, |D|\}$.

We support two mechanisms to establish dense correspondences. By default, we assume that both S and D have a fully 3D structure and we establish dense correspondences by finding the nearest-neighbor in D, in terms of Euclidean distance, of each point in $\mathcal{W}[S]$, with the search being performed efficiently by parallel kd-tree queries. For certain applications that only require frame-to-frame (2.5D-to-2.5D) or model-to-frame (3D-to-2.5D) registration, we can further speed up the process by obtaining *projective* correspondences. This amounts to projecting S and D onto the target frame image and extracting correspondences based on points that are projected to the same pixel. This is the mechanism adopted in most real-time reconstruction pipelines [47, 49, 50].

In many common situations, dense geometric/depth correspondences alone are not enough to disambiguate the underlying motion. For example, tracking points on flat surfaces that lack geometric texture and slide parallel to each other may exhibit drift. Establishing robust keypoint correspondences between the source and target geometries can effectively mitigate this problem. We assume that our input

geometries are equipped with sparse interest points; our sparse correspondences are established by the interest point descriptor matches between S and D. In our implementation, we focus on input geometries that are either RGBD frames or 3D reconstructions from RGBD input. The availability of regular images along with (registered) geometry allows us to adopt SIFT keypoint [67] (lifted to 3D) matches for our sparse correspondences.

To make optimization more stable, we discard correspondence candidates from the above mechanisms that do not meet some basic proximity and local similarity criteria. Let $\{n_i^s\}$, $\{n_i^d\}$, and $\{c_i^s\}$, $\{c_i^d\}$ be the surface normals and colors (e.g., RGB value 3-vectors) of the source and target geometries, indexed in the same way as their support points in S and D. A correspondence candidate $(i, j) \in \{1, \ldots, |S|\} \times \{1, \ldots, |D|\}$ is considered valid and used in the optimization if all of the following hold:

- $\|x_i^s - x_j^d\|_2 < \theta_d$
- $\arccos\left(n_i^{s\top} n_j^d\right) < \theta_n$
- $\|c_i^s - c_j^d\|_2 < \theta_c$

In the above, θ_d is a point distance threshold, θ_n is a normal angle threshold, and θ_c is a color "distance" threshold.

3.3.3.2 Warp field optimization

Given a set of dense and sparse point correspondences, we shall now describe our warp field optimization step. Modeling the warp field using locally affine [64]

or locally rigid [49] transformations provides better motion estimation results than adopting a simple translational local model, due to more effective regularization. In our approach, we adopt a locally rigid (6DoF) model.

Instead of computing a unique rigid transformation for each point in S, we use the more efficient embedded deformation graph representation [65] for the warp field $\mathcal{W}$, similarly to [49] and [55]. Let $\mathcal{G} = \{(g_i, \sigma_i, T_i)\}$ be the set of virtual deformation nodes, where $g_i \in \mathbb{R}^3$ is the position of the ith node, σ_i is a radius parameter that controls the ith node's area of effect, and $T_i \in SE(3)$ is the 6DoF rigid transformation attached to the ith node. The deformation node positions are obtained by downsampling the source geometry S by means of a voxel grid of bin size r_b. A reasonable choice for σ_i that ensures sufficient area of effect overlap among neighboring nodes is $\sigma_i = \sigma_{\text{def}} \equiv r_b/2$, for $i = 1, \ldots, |\mathcal{G}|$. As in [50], each local transformation T_i is parameterized during optimization by a 6D vector θ_i of 3 Euler angles and 3 translational offsets. The effect of the warp field $\mathcal{W}$, represented by $\mathcal{G}$, on a point $x \in \mathbb{R}^3$ is given by interpolating the local node deformations in the neighborhood of x. Let $\mathcal{N}(x) \subseteq \{1, \ldots, |\mathcal{G}|\}$ be the indices of the k-nearest neighbors of x in $\mathcal{G}$. The local transformation parameter vector at x is given by:

$$\theta(x) \equiv \frac{\sum_{i \in \mathcal{N}(x)} w_i(x)\theta_i}{\sum_{i \in \mathcal{N}(x)} w_i(x)}, \tag{3.1}$$

where $w_i(x) = \exp\left(-\|x - g_i\|_2^2 / (2\sigma_i^2)\right)$. The image of x via $\mathcal{W}$ is then:

$$\mathcal{W}(x) \equiv \text{Rot}(\theta(x))x + \text{Trans}(\theta(x)), \tag{3.2}$$

where $\text{Rot}(\theta)$ and $\text{Trans}(\theta)$ extract the rotation matrix and translation vector from our 6D parameterization.

We note that the above 6D parameterization is only used within optimization (line 5 of Algorithm 1) and that both the estimated *incremental* warp $\mathcal{W}_{\text{iter}}$ and the final composite estimate $\mathcal{W}$ have their node transformations T_i expressed in terms of $SE(3)$ transformation matrices. The fact that we continuously warp S and compute $\mathcal{W}_{\text{iter}}$ starting from the identity warp, combined with the smoothness prior imposed on the warp field (shown below), allows us to overcome any problems associated with Euler angle parameterizations of rotation.

Our registration objective, as a function of the unknown warp field $\mathcal{W}$, which in the context of Algorithm 1 is the incremental warp $\mathcal{W}_{\text{iter}}$, and the point correspondences $\mathcal{C}$ between S and D, which are fixed for this step, is formulated as:

$$E\left(D, S, \mathcal{C}, \mathcal{W}\right) = E_{\text{data}}\left(D, S, \mathcal{C}, \mathcal{W}\right) + \lambda_{\text{stiff}} E_{\text{stiff}}\left(\mathcal{W}\right). \tag{3.3}$$

Our data term, $E_{\text{data}}\left(D, S, \mathcal{C}, \mathcal{W}\right)$, is a weighted sum of a point-to-plane and a point-to-point ICP cost:

$$E_{\text{data}}\left(D, S, \mathcal{C}, \mathcal{W}\right) = E_{\text{plane}}\left(D, S, \mathcal{C}, \mathcal{W}\right) + \lambda_{\text{point}} E_{\text{point}}\left(D, S, \mathcal{C}, \mathcal{W}\right). \tag{3.4}$$

Pure point-to-plane metric optimization generally converges faster and to better solutions than pure point-to-point [71], and is the standard trend in recent rigid [47,48] and non-rigid [49, 50, 56] registration pipelines. However, as discussed in Section

3.3.3.1, integrating a point-to-point term for robust point matches into the registration cost can effectively disambiguate motion estimation in cases where surfaces that lack geometric texture slide parallel to each other. Similarly to [50], we use our dense geometric correspondences $\mathcal{C}_{\text{dense}}$ to define our point-to-plane cost and our sparse keypoint correspondences $\mathcal{C}_{\text{sparse}}$ for our point-to-point cost:

$$E_{\text{plane}}\left(D, S, \mathcal{C}, \mathcal{W}\right) = \sum_{(i,j)\in\mathcal{C}_{\text{dense}}} \left(n_j^{d\top}\left(\mathcal{W}\left(x_i^s\right) - x_j^d\right)\right)^2, \tag{3.5}$$

$$E_{\text{point}}\left(D, S, \mathcal{C}, \mathcal{W}\right) = \sum_{(i,j)\in\mathcal{C}_{\text{sparse}}} \left\|\mathcal{W}\left(x_i^s\right) - x_j^d\right\|_2^2. \tag{3.6}$$

Our regularization term $E_{\text{stiff}}\left(\mathcal{W}\right)$ directly penalizes differences in transformation parameters of neighboring virtual nodes of $\mathcal{G}$ under the robust Huber-L^1 loss function. If $\mathcal{N}(i) \subseteq \{1, \ldots, |\mathcal{G}|\}$ is the set of indices of the k-nearest neighbors of g_i in $\mathcal{G}$, our regularization term is formulated as:

$$E_{\text{stiff}}\left(\mathcal{W}\right) = \sum_{i=1}^{|\mathcal{G}|} \sum_{j\in\mathcal{N}(i)} w_{ij}\psi_\delta\left(\theta_i - \theta_j\right), \tag{3.7}$$

where $w_{ij} = \exp\left(-\left\|g_i - g_j\right\|^2 / (2\sigma_{\text{def}}^2)\right)$ weights the pairwise penalties based on node distance, and $\psi_\delta(\Delta\theta)$ denotes the sum of the Huber loss function values over the 6 parameter residual components. Parameter δ controls the point at which the loss function behavior switches from quadratic (squared L^2-norm) to absolute-linear (L^1-norm). Since L^1-norm regularization is known to better preserve solution discontinuities, we use a small value of $\delta = 10^{-4}$. Given our locally rigid motion model, this regularization scheme enforces an 'As-Rigid-As-Possible' [66] prior to

the estimated warp field.

The registration objective of equation (3.3) is non-linear in the $6|G|$ unknowns. We minimize $E(D, S, \mathcal{C}, \mathcal{W})$ by performing a small number of Gauss-Newton iterations. As in [55], we handle non-quadratic terms using the square-rooting technique of [72]. At every step, we linearize E around the current solution $\theta \in \mathbb{R}^{6|G|}$ (vector concatenation of all node transformation parameters θ_i) and obtain a solution increment $\hat{\theta}$ by solving the system of normal equations $J^\top J\hat{\theta} = J^\top r$, where J is the Jacobian matrix of the residual terms in E and r is the vector of (negative) residual values. We solve this sparse system iteratively, using the Conjugate Gradient algorithm with a diagonal preconditioner.

3.3.4 Handling topology changes

In our post-processing phase for handling topology change motion boundaries, we first explicitly detect likely *contact* and *separation* regions in the source geometry, and proceed by appropriately blending our (default) *forward* and *inverted backward* warp hypotheses in a local yet seamless manner. It follows from the discussion in Section 3.3.2 that, as far as topology changes are concerned, the forward warp is only problematic in separation areas. However, instead of only focusing on and amending separations, it is beneficial to also explicitly consider contact events. As will become clear in the following, considering both types of events and treating them symmetrically robustifies their detection and allows for a less biased hypothesis blending scheme.

Using the same notation as in Section 3.3.2, let $\mathcal{W}_S^f$ be the warp field that aligns S to D and $\mathcal{W}_D^b$ the backward warp that aligns D to S, both computed by Algorithm 1. We will be using the 'S' subscript for forward ($S \rightarrow D$) motion entities and the 'D' subscript for backward ($D \rightarrow S$) ones. For the needs of the following discussion, we will consider these motion fields to be represented by the *per-point* local rigid transformations of their support geometries, so that:

$$\mathcal{W}_S^f = \{T_{Si}^f,\ i = 1, \ldots, |S|\}, \tag{3.8}$$

$$\mathcal{W}_D^b = \{T_{Di}^b,\ i = 1, \ldots, |D|\}. \tag{3.9}$$

To invert $\mathcal{W}_D^b$ in order to obtain an alternative forward warp, we appropriately rebase its inverse local transformations on S. To that end, we first compute the target geometry's image $\mathcal{W}_D^b[D]$ (which should be closely aligned to S) and, to each point $x_i^s \in S$, we assign the transformation ${T_{Dj}^b}^{-1}$, where j is the index of the nearest neighbor of x_i^s in the point set $\mathcal{W}_D^b[D]$. Of course, we assume that the latter is indexed in the same way as D. The inverted backward warp is then represented as:

$$\mathcal{W}_S^b = \{T_{Si}^b,\ i = 1, \ldots, |S|\}, \tag{3.10}$$

where $T_{Si}^b = {T_{Dj}^b}^{-1}$ and j is the nearest neighbor index of x_i^s in $\mathcal{W}_D^b[D]$. Analogously, we obtain an alternative backward warp, by inverting our forward hypothesis:

$$\mathcal{W}_D^f = \{T_{Di}^f,\ i = 1, \ldots, |D|\}, \tag{3.11}$$

where $T^f_{Di} = T^{f^{-1}}_{Sj}$ and j is the nearest neighbor index of x^d_i in $\mathcal{W}^f_S[S]$. To summarize, we have two forward motion ($S \rightarrow D$) warp hypotheses ($\mathcal{W}^f_S$ and $\mathcal{W}^b_S$) and two backward motion ($D \rightarrow S$) ones ($\mathcal{W}^b_D$ and $\mathcal{W}^f_D$).

3.3.4.1 Detecting topology change events

We detect topology change regions in S based on how our warp estimates affect local neighborhoods of the source geometry.

Naturally, we expect that if $x^s_i \in S$ is close to a separation boundary, its distance to some of its neighbors in S should increase after applying the correct warp to S. We shall refer to this effect as neighborhood *stretching*. The dual case of a contact event manifests exactly the same way in the backward motion direction (stretching of neighborhoods of D), in which the event is perceived as a separation. In the following, we will use a local measure of stretch over points in S to detect separation areas, and map the same measure over D in the backward direction onto S to obtain a dual measure of "compression" that will allow us to detect contacts.

We quantify the above intuition by defining a local "stretch" operator for point $x_i \in X \subseteq \mathbb{R}^3$ under the warp $\mathcal{W}$ as the maximum ratio of the distance to its neighbors before and after applying $\mathcal{W}$:

$$\textbf{Stretch}\,(i, X, \mathcal{W}) \equiv \max_{j \in \mathcal{N}(i)} \frac{\|\mathcal{W}(x_i) - \mathcal{W}(x_j)\|_2}{\|x_i - x_j\|_2}, \tag{3.12}$$

where $\mathcal{N}(i) \subseteq \{1, \ldots, |X|\}$ indexes the neighbors of x_i in X that lie within ρ_s distance from it. The choice of the neighborhood radius value ρ_s depends on the

scale and resolution of the input geometries. For close-range point clouds acquired with Kinect-like cameras, we use $\rho_s = 1.5$cm.

To each point in S, we associate one stretch value for each of our two forward warp hypotheses, according to definition (3.12). For $i = 1, \ldots, |S|$, we have:

$$\textsc{Stretch}_S^f(i) = \textbf{Stretch}\left(i, S, \mathcal{W}_S^f\right), \text{ and} \tag{3.13}$$

$$\textsc{Stretch}_S^b(i) = \textbf{Stretch}\left(i, S, \mathcal{W}_S^b\right). \tag{3.14}$$

We also compute the local stretch of the target geometry D under each of the two backward warps:

$$\textsc{Stretch}_D^f(i) = \textbf{Stretch}\left(i, D, \mathcal{W}_D^f\right), \text{ and} \tag{3.15}$$

$$\textsc{Stretch}_D^b(i) = \textbf{Stretch}\left(i, D, \mathcal{W}_D^b\right), \tag{3.16}$$

which we subsequently map onto S, interpreting them as a compression measure (contact indicator), according to:

$$\textsc{Compress}_S^f(i) = \textsc{Stretch}_D^f\left(\text{NN}\left(\mathcal{W}_S^f\left(x_i^s\right), D\right)\right), \tag{3.17}$$

$$\textsc{Compress}_S^b(i) = \textsc{Stretch}_D^b\left(\text{NN}\left(\mathcal{W}_S^b\left(x_i^s\right), D\right)\right), \tag{3.18}$$

where $\text{NN}(x, X) \in \{1, \ldots, |X|\}$ is the index of the nearest neighbor of point x in point set X.

Using the above point-wise stretch/compress values on S, we extract subsets

Algorithm 2 ExtractTopologyEvents

Input: $S, \textsc{Stretch}_S^f, \textsc{Stretch}_S^b, \textsc{Compress}_S^f, \textsc{Compress}_S^b$
Output: $\textsc{Con}, \textsc{Sep}$

$\textsc{Con} \leftarrow \varnothing$
$\textsc{Sep} \leftarrow \varnothing$
for $i = 1, \ldots, |S|$ **do**
    stretch $\leftarrow \max\left\{\textsc{Stretch}_S^f(i), \textsc{Stretch}_S^b(i)\right\}$
    compress $\leftarrow \max\left\{\textsc{Compress}_S^f(i), \textsc{Compress}_S^b(i)\right\}$
    if stretch $> \tau$ **and** stretch $> \alpha \cdot$ compress **then**
        $\textsc{Sep} \leftarrow \textsc{Sep} \cup \{x_i^s\}$
    end if
    if compress $> \tau$ **and** compress $> \alpha \cdot$ stretch **then**
        $\textsc{Con} \leftarrow \textsc{Con} \cup \{x_i^s\}$
    end if
end for

of the source geometry that are likely to lie on topology change motion boundaries. Let $\textsc{Sep}, \textsc{Con} \subseteq S$ be the sets of candidate separation and contact boundary points, respectively. According to the above discussion, points on a separation boundary are expected to have high stretch scores, while points on a contact boundary should exhibit high local compression. To decide whether a point in S is a boundary candidate, we perform two symmetric tests per case that rely on two threshold values, an absolute score threshold τ, and a relative (ratio) threshold α. A point of S is a member of $\textsc{Sep}$ ($\textsc{Con}$) if and only if the maximum of its two stretch (compress) scores is greater than τ and also greater than α times its maximum compress (stretch) score. The process is summarized in Algorithm 2. A sample output is shown in Fig. 3.1b (left), marked in red; note that $\textsc{Con} = \varnothing$ in this case.

As we will show in Section 3.4.3, the above procedure is very effective at detecting and classifying topology changes, but, because of the continuous nature

of our local stretch/compression measures and depending on the selected threshold values, it may produce "false positives" (e.g., in areas of actual deforming surface stretching or compression but constant topology). However, under the assumption that our two forward warp hypotheses behave similarly in the false positive areas and, as will become clear in the next section, this does not affect our final warp estimate.

3.3.4.2 Local hypothesis blending

Our blending scheme produces a topology-aware warp field $\mathcal{W}_S$ by combining the forward warp hypotheses $\mathcal{W}_S^f$ and $\mathcal{W}_S^b$ on a per-point basis. Our objective is to assign a higher weight to $\mathcal{W}_S^b$ (inverted backward warp) near separation areas, and ensure that $\mathcal{W}_S^f$ (forward warp) has a stronger weight near contact areas. At the same time, it is desirable that point weights vary smoothly on S, so that our warp blending does not introduce seam artifacts on $\mathcal{W}_S[S]$ due to differences in our original warp hypotheses.

We achieve the above by attaching a smoothly decaying kernel on each of our detected event points in Con and Sep and locally computing the weight for each event class. Assuming a maximum radius of effect ρ_e (free parameter) for our event points, we model the influence of each event with an RBF kernel of bandwidth $\sigma = \rho_e/3$. The weights w_f^i and w_b^i of $\mathcal{W}_S^f$ and $\mathcal{W}_S^b$ for the source point x_i^s are computed by accumulating influences of the event points in $\text{Con} = \{c_i\}$ and

Algorithm 3 LocalHypothesisBlending

Input: $S, \mathcal{W}_S^f, \mathcal{W}_S^b, \textsc{Con} = \{c_i\}, \textsc{Sep} = \{s_i\}, \rho_e$
Output: $\mathcal{W}_S = \{T_{Si},\ i = 1, \ldots, |S|\}$

$\sigma \leftarrow \rho_e/3$
for $i = 1, \ldots, |S|$ **do**
  $\mathcal{N} \leftarrow \textsc{RadiusSearch}\,(x_i^s, \textsc{Con}, \rho_e)$
  $w_f \leftarrow 1$
  for $j \in \mathcal{N}$ **do**
    $w_f \leftarrow w_f + \exp\left(-\left\|x_i^s - c_j\right\|^2/(2\sigma^2)\right)$
  end for
  $\mathcal{N} \leftarrow \textsc{RadiusSearch}\,(x_i^s, \textsc{Sep}, \rho_e)$
  $w_b \leftarrow 0$
  for $j \in \mathcal{N}$ **do**
    $w_b \leftarrow w_b + \exp\left(-\left\|x_i^s - s_j\right\|^2/(2\sigma^2)\right)$
  end for
  $w \leftarrow w_f + w_b$
  $w_f \leftarrow w_f/w$
  $w_b \leftarrow w_b/w$
  $T_{Si} \leftarrow \mathrm{SE3}\left(w_f T_{S\,i}^f + w_b T_{Si}^b\right)$
end for

$\textsc{Sep} = \{s_i\}$ respectively that are within a ρ_e-radius from x_i^s:

$$w_f^i = \frac{1}{Z}\left(1 + \sum_{j \in \mathcal{N}_C(i)} \exp\left(-\left\|x_i^s - c_j\right\|^2/(2\sigma^2)\right)\right), \text{ and} \tag{3.19}$$

$$w_b^i = \frac{1}{Z}\left(\sum_{j \in \mathcal{N}_S(i)} \exp\left(-\left\|x_i^s - s_j\right\|^2/(2\sigma^2)\right)\right), \tag{3.20}$$

where Z is a normalizing constant ensuring that $w_f^i + w_b^i = 1$, and $\mathcal{N}_C(i)$, $\mathcal{N}_S(i)$ index the ρ_e-radius neighbors of x_i^s in Con and Sep respectively. In the absence of any topology event influence (e.g., for x_i^s at least ρ_e from any event point), the above defaults to $w_f^i = 1$ and $w_b^i = 0$, giving full weight to the standard forward hypothesis $\mathcal{W}_S^f$. The local transformation of our final, topology-aware warp field

estimate $\mathcal{W}_S = \{T_{Si},\ i = 1, \ldots, |S|\}$ at location x_i^s is then given by:

$$T_{Si} = \text{SE3}\left(w_f^i T_{S\,i}^f + w_b^i T_{Si}^b\right), \tag{3.21}$$

where SE3$(\cdot)$ converts the linear blend of the two transformation matrices back to a valid $SE(3)$ transformation matrix. The complete blending process is summarized in Algorithm 3 (see also equations (3.8) and (3.10)). A visualization of the blending weights is given in Fig. 3.1b (middle), where each source geometry point is colored according to its inverted backward warp hypothesis weight w_b^i.

We note that, for source points close to topology events, one of w_f^i and w_b^i will dominate the other, effectively rendering the blending of (3.21) a binary selection. As we move farther from topology events, it is possible that the two weights assume comparable values (e.g., at points lying between two events of different type). For our blended output (3.21) to be seamless and error-free in that case, it is expected that $\mathcal{W}_S^f$ and $\mathcal{W}_S^b$ do not differ significantly in areas that are "far enough" from event points. This highlights the importance of parameter ρ_e, which should be of adequate magnitude to cover event regions; we have found that values $\rho_e \geq 3r_b$ work well in practice, where r_b is the resolution of our virtual deformation graph (Section 3.3.3.2).

As a concluding remark, we observe that the most costly operation of our post-processing phase is the calculation, based on radius-neighborhoods, of the stretch/-compress values of Section 3.3.4.1. Algorithm 3 also performs radius queries on point sets Con and Sep, but the latter are typically very small in size. In our experience,

the overall running time of the entire phase is significantly smaller than a single warp field estimation. Furthermore, in the case of RGBD input, image structure can be easily exploited in order to accelerate the extraction of point neighborhoods.

3.4 Experiments

We conduct two sets of experiments for the evaluation of our registration pipeline. The first one is performed on a public dataset and examines our algorithm's motion estimation accuracy, both with and without the topology handling phase (Section 3.4.2). For the second one, we use a custom dataset with topology event annotations and evaluate our event detection performance, as well as our estimated warp field quality in the presence of separation events (Section 3.4.3).

3.4.1 General setup details

For both sets of experiments, the input is RGBD data, either synthetic (first set) or captured by a Kinect-like camera (second set).

Point cloud generation. RGBD frames are converted to point clouds that are equipped with surface normal and color information, as well as a sparse set of interest points derived from SIFT features. In all cases, the full resolution of the input depth map is used, which is 1024×436 for the synthetic sequences and 640×480 for the camera data. We use a fixed maximum depth of 5m for all sequences in the first set, and vary the cut-off value in the range of 0.8m to 1.5m for the camera data, depending on the sequence. For normal estimation, we use k-NN neighborhoods with

$k = 30$ in our first set of experiments, and ρ-radius neighborhoods with $\rho = 1.5$cm in our second set. SIFT keypoints are extracted from the RGB images and lifted to 3D, discarding the ones that lie on depth boundaries.

Warp field estimation. In the correspondence association step (Section 3.3.3.1) of our non-rigid ICP algorithm, we set the maximum correspondence distance to $\theta_d = 15$cm for our first set of experiments and $\theta_d = 5$cm for our second one, while we use common values $\theta_n = 15°$ and $\theta_c = 0.4$ for the maximum normal angle and color difference (colors are RGB triplets in $[0, 1]^3$). The embedded deformation graph $\mathcal{G}$ for our warp field parameterization (Section 3.3.3.2) has a resolution of $r_b = 2.5$cm, with each node's area of effect being controlled by $\sigma_{\text{def}} = r_b/2$. To evaluate the local transformation for each point in the source geometry (equation (3.1)), we use its 4 nearest neighbors in $\mathcal{G}$. The point-to-point weight in our data term (3.4) is set to $\lambda_{\text{point}} = 2$. To favor L^1-norm behavior by our regularization term (3.7), we use a small Huber loss parameter value of $\delta = 10^{-4}$, while we set the term's weight to $\lambda_{\text{stiff}} = 200$ (equation (3.3)). Regularization topology is given by the 6 nearest neighbor nodes of $\mathcal{G}$. We perform a maximum of 10 top-level ICP iterations, while the process typically converges in less. Within each optimization step, we perform a maximum of 5 Gauss-Newton iterations.

Dynamic topology handling. In all experiments, local stretch is computed on neighborhoods of radius $\rho_s = 1.5$cm (Section 3.3.4.1). To detect and classify topology change events, we use an asbsolute score threshold of $\tau = 2.2$ and a relative ratio of $\alpha = 1.5$. For our blending step (Section 3.3.4.2), we assume that every detected event has a radius of effect equal to $\rho_e = 7.5$cm.

3.4.2 Motion estimation accuracy evaluation

Due to the lack of publicly available datasets with ground truth dense 3D motion, we perform our accuracy assessments on MPI Sintel [73], a synthetic optical flow evaluation dataset. The dataset contains multiple sequences of (typically) 50 frames that capture motions ranging from slow, almost rigid to very large, highly non-rigid ones. In addition to ground truth optical flow, metric ground truth depth and camera intrinsics are provided, which we use to emulate RGBD input.

We base our evaluation on two classical optical flow performance measures, the endpoint error (EPE) and the angular error (AE) [74]. If $\tilde{f} = (\tilde{u}, \tilde{v})$ is an optical flow estimate at a given pixel whose ground truth value is $f = (u, v)$, EPE is computed as:

$$e_{\text{EPE}}\left(\tilde{f}, f\right) = \left\|\tilde{f} - f\right\|_2. \tag{3.22}$$

The angular error AE is defined as the angle between the 3D space-time vectors $h(\tilde{f}) = (\tilde{u}, \tilde{v}, 1)$ and $h(f) = (u, v, 1)$, as:

$$e_{\text{AE}}\left(\tilde{f}, f\right) = \arccos \frac{h(\tilde{f})^\top h(f)}{\|h(\tilde{f})\|_2 \|h(f)\|_2}, \tag{3.23}$$

effectively enabling evaluation at pixels of zero flow. We convert 3D motion estimates to optical flow by first warping the source points in 3D and then computing the 2D point/pixel displacements as differences of the projected endpoints onto the image plane.

We evaluate and compare three methods: PD-Flow [45], a state-of-the-art

scene flow algorithm, F-Warp, our general warp field estimation algorithm defined in Section 3.3.3 (without the topology change handling phase), and FB-Warp, our complete topology-aware warp field estimation pipeline. We run the three algorithms on the entire duration of 12 Sintel sequences and compute the average EPE and AE values per consecutive frame pair. We report the median and mean frame-level average errors over each sequence in Table 3.1. Median values are not easily affected by extreme values, often providing a better picture of how accurate estimation is "half of the time". We also report overall mean and median error values for each method, computed over the total number of frames from all sequences.

Our FB-Warp method overall achieves the highest accuracy in terms of both error metrics, followed closely by our baseline, F-Warp. PD-Flow is very accurate in estimating slow motions (e.g., in the `sleeping_2` sequence), but falls behind in most cases, producing particularly large errors in sequences that contain very fast motions, such as `ambush_5` and `ambush_6`. We also refer the reader to the Sintel-based evaluation of MC-Flow [53] (Table 2 of that paper), another state-of-the-art scene flow algorithm with significantly better performance than PD-Flow in estimating large motions. We were unable to evaluate MC-Flow ourselves, because its implementation has not been released. While 6 of the Sintel sequences in our experiments and the ones in [53] are common, there are significant differences between our evaluation setups: *1)* their reported EPE and AE values are computed on one specific frame pair per sequence, not over whole sequences as in here, *2)* they downsample their input to half its original resolution per dimension (to 512×218), whereas we use full resolution images, and *3)* they consider only non-occluded pixels, while we

compute errors on all valid ones. The above prohibit reaching definitive conclusions. However, the fact that we adopt an arguably more difficult evaluation strategy (at *full resolution*, which means smaller pixel size for EPE interpretation, and evaluating over *whole sequences*) and still obtain comparable EPE and AE absolute values to the ones reported in [53] (averages of 1.203 and 6.559, respectively), leads us to argue that FB-Warp compares favorably to MC-Flow.

Table 3.1: Evaluation on the MPI Sintel Dataset.

Sequence	Frame-wise EPE over entire sequence						Frame-wise AE over entire sequence					
	PD-Flow		F-Warp		FB-Warp		PD-Flow		F-Warp		FB-Warp	
	Median	Mean	Median	Mean	Median	Mean	Median	Mean	Median	Mean	Median	Mean
alley_1	0.774	0.964	0.485	0.519	**0.485**	**0.516**	**7.005**	**7.319**	8.256	8.379	8.285	8.361
ambush_5	12.651	21.172	1.714	7.017	**1.639**	**6.411**	30.435	29.505	5.558	6.920	**5.262**	**6.386**
ambush_6	25.467	28.981	**4.409**	**7.535**	5.106	8.028	37.813	39.470	7.165	7.520	**7.083**	**7.457**
ambush_7	1.861	4.969	0.633	0.925	**0.584**	**0.891**	11.517	27.146	11.377	10.885	**11.270**	**10.763**
bamboo_1	1.104	1.221	**0.677**	**0.712**	0.677	0.713	8.159	8.560	**5.517**	**5.705**	5.517	5.713
bamboo_2	0.889	2.628	0.687	1.027	**0.686**	**1.014**	**7.134**	11.924	8.315	8.729	8.357	**8.669**
bandage_1	1.683	2.160	0.973	0.957	**0.936**	**0.935**	15.201	15.244	10.111	9.829	**9.862**	**9.681**
bandage_2	0.769	1.062	**0.329**	0.334	0.330	**0.331**	9.035	10.754	6.525	6.554	**6.523**	**6.542**
shaman_2	**0.286**	0.317	0.294	0.284	0.292	**0.284**	**5.571**	**5.643**	7.801	7.838	7.779	7.831
shaman_3	0.485	0.552	**0.178**	**0.291**	0.179	0.298	8.125	8.115	3.155	**4.559**	**3.153**	4.604
sleeping_1	0.502	**0.514**	0.149	0.541	**0.149**	0.541	5.939	6.110	1.954	**6.487**	**1.954**	6.488
sleeping_2	**0.146**	**0.143**	0.170	0.171	0.170	0.171	**2.508**	**2.520**	3.352	3.335	3.352	3.335
Overall	0.701	4.122	0.488	1.379	**0.487**	**1.336**	7.899	13.009	6.826	7.213	**6.815**	**7.136**

3.4.3 Topology change event handling

To evaluate the performance of the topology-handling phase of our pipeline, we collected, using a Kinect-like camera, an RGBD dataset of 7 sequences that contain changes in scene topology. The regions of visible topology changes were manually annotated on the color image as binary masks, drawn in freehand mode, and classified as either contacts or separations. Our collected sequences capture the following diverse set of actions:

- Hand-clapping (`clap` sequence)
- Fast hand-drumming on a desk (`drum` sequence)
- Two pick-and-place actions on objects lying on a flat surface or on top of each other (`stack` and `unstack` sequences)
- A separation of two touching objects using both hands (`separate` sequence)
- Two drawer opening action sequences (`top_drawer` and `bot_drawer`)

The annotated ground truth events capture all visible instances of hand-hand, hand-object, and object-object interaction. Snapshots of our sequences are shown in Fig. 3.3 (first two columns).

Our evaluation of this phase is twofold. First, we assess how well our detected event points in Con and Sep (Section 3.3.4.1) relate to the ground truth events. Second, we qualitatively and quantitatively evaluate our topology-aware motion estimate (FB-Warp), including comparisons with our baseline algorithm (F-Warp), as well as two scene flow estimation methods.

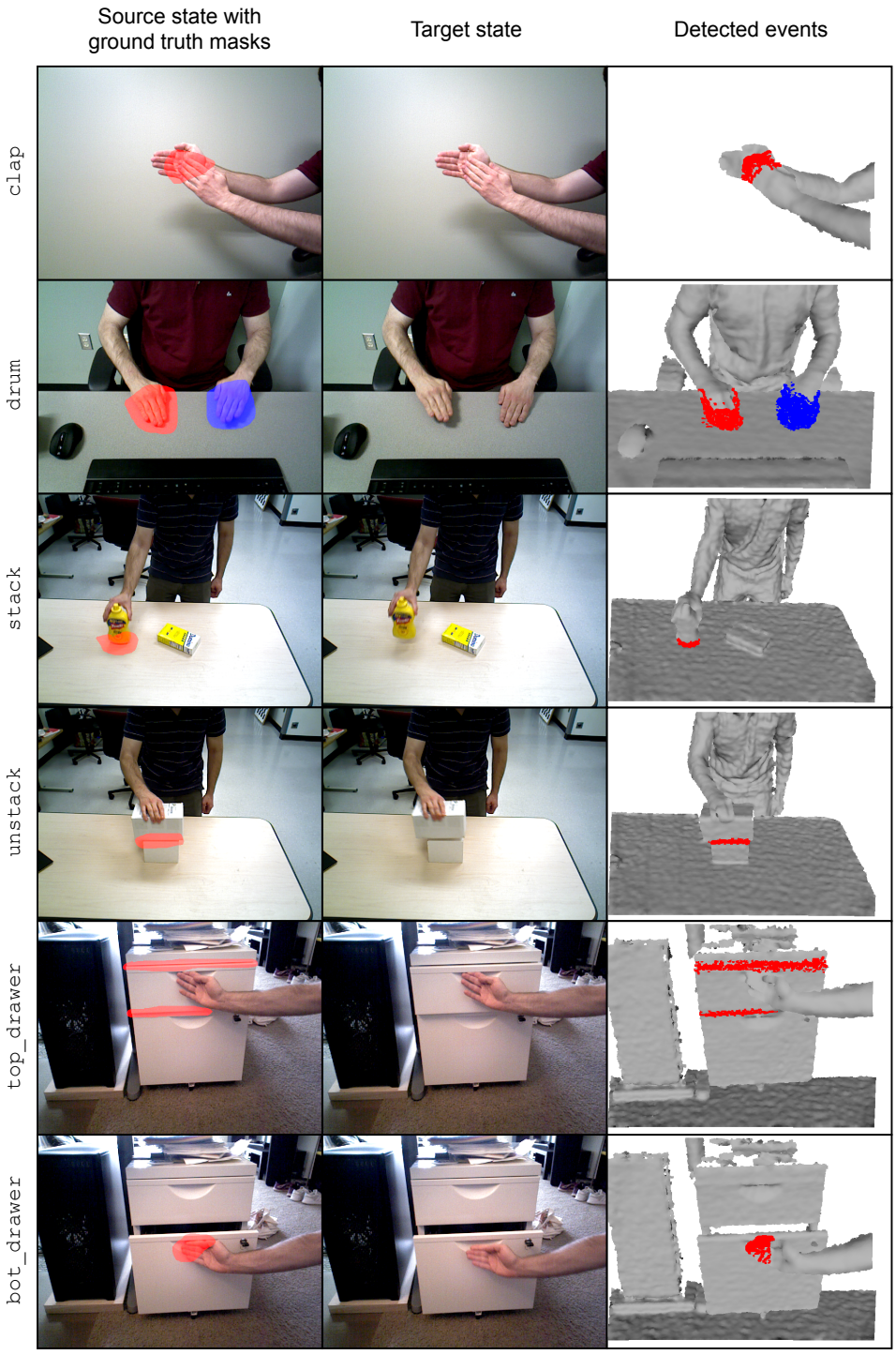


Figure 3.3: Topological event detections on our dataset. Each row corresponds to a different sequence. First column: source color frame with event mask overlays (blue for contact, red for separation). Second column: target color frame. Third column: our topological event detections overlaid on the source geometry (blue for contact, red for separation).

3.4.3.1 Topology event detection

We uniformly represent topology change events as triplets $e_i = (l_i, t_i, X_i)$, where $l_i \in \{0, 1\}$ is a binary label indicating contact or separation, $t_i \in \mathbb{N}$ is the time (frame index) of the event, and $X_i \subset \mathbb{R}^3$ is the subset of points in the source geometry that lie very close to event motion boundaries. We use the superscript 'gt' to denote ground truth event entities, and 'det' to denote the ones associated with detections by our algorithm. Let $\mathcal{E}^{\text{gt}} = \{e_i^{\text{gt}}\}$ and $\mathcal{E}^{\text{det}} = \{e_i^{\text{det}}\}$ denote the sets of ground truth and detected events, respectively. Given an annotated sequence, these are populated according to the following:

- Labels and timestamps for events in $\mathcal{E}^{\text{gt}}$ come directly from the annotation data. Ground truth event point clouds X_i^{gt} are obtained by the image annotation binary masks, which directly mask regions of the t_i-th frame's point cloud (input color and depth maps are registered).
- Detected events $\mathcal{E}^{\text{det}}$ are derived from the per-frame outputs of Algorithm 2. At time k (frame pair index), we interpret the *connected components*, in the Euclidean sense, of point sets Con and Sep as separate, meaningful events. We insert all triplets $(0, k, \text{Con}_i)$ and $(1, k, \text{Sep}_i)$ into $\mathcal{E}^{\text{det}}$, where $\{\text{Con}_i\}$ and $\{\text{Sep}_i\}$ denote the respective connected component sets. We use a distance threshold of 2cm for the Euclidean segmentation; to avoid noisy detections, we discard components that contain less than 75 points.

Adopting a 3D point set based representation for the spatial extent of topology events enables reasoning about event similarity in terms of metric distances.

Our assessments on spatial overlap of events will be based on the 'ρ-overlap' metric, defined for a pair of point clouds X_1 and X_2 as:

$$\textsc{Overlap}_\rho(X_1, X_2) \equiv \frac{\left|S_\rho^{X_2}(X_1)\right| + \left|S_\rho^{X_1}(X_2)\right|}{|X_1| + |X_2|}, \tag{3.24}$$

where $S_\rho^B(A) \subseteq A$ contains exactly the points in A that lie within distance ρ from their nearest neighbor in set B. Clearly, $\textsc{Overlap}_\rho(X_1, X_2) \in [0, 1]$. It is easy to verify that this metric is simply the *intersection-over-union* ratio for the sets $X_1 \cup S_\rho^{X_1}(X_2)$ and $X_2 \cup S_\rho^{X_2}(X_1)$. We use a radius value of $\rho = 3$cm.

We derive a many-to-many matching between sets $\mathcal{E}^{\text{gt}}$ and $\mathcal{E}^{\text{det}}$ by associating events in the two sets that have a significant spatiotemporal overlap. A ground truth event e_i^{gt} is matched to a detected event e_j^{det} if and only if they both belong to the same class (contact or separation), their timestamps are very close, and they share a substantial spatial overlap. If $\mathcal{M} = \{(i, j)\}$ is the set of event matches, then $\mathcal{M}$ contains all pairs (i, j), for $i = 1, \ldots, |\mathcal{E}^{\text{gt}}|$ and $j = 1, \ldots, |\mathcal{E}^{\text{det}}|$, that satisfy all three conditions:

- $l_i^{\text{gt}} = l_j^{\text{det}}$
- $|t_i^{\text{gt}} - t_j^{\text{det}}| \leq 2$
- $\textsc{Overlap}_\rho(X_i^{\text{gt}}, X_j^{\text{det}}) \geq 0.2$

Based on the set of all valid spatiotemporal matches $\mathcal{M}$, we derive two interesting event mappings: one that maps each ground truth event to a single detected one, and an 'inverse' one that maps each detected event to a ground truth one. Both our 'ground truth to detected' and 'detected to ground truth' mappings associate

an event in the first set with its match in the second one that maximizes overlap:

$$\textsc{GtToDet}(i) = \underset{j:(i,j)\in\mathcal{M}}{\arg\max}\, \textsc{Overlap}_\rho(X_i^{\text{gt}}, X_j^{\text{det}}), \tag{3.25}$$

$$\textsc{DetToGt}(j) = \underset{i:(i,j)\in\mathcal{M}}{\arg\max}\, \textsc{Overlap}_\rho(X_i^{\text{gt}}, X_j^{\text{det}}). \tag{3.26}$$

Of course, the above are only defined for events (ground truth/detected) that have valid matches, i.e. $(i, j) \in \mathcal{M}$.

We present our detection results on our custom dataset in Table 3.2. On average, our pipeline extracts three times more events than the annotated ones (columns 2-3). This is normal, as topology changes may manifest gradually in continuous video sequences, while our annotation process treats them as being instantaneous. In columns 4-6 and 7-9, we evaluate each of the mappings GtToDet and DetToGt in terms of event coverage (fraction of $\mathcal{E}^{\text{gt}}$ and $\mathcal{E}^{\text{det}}$, respectively, that was matched), average spatial overlap, and average detection delay (signed difference $t_i^{\text{gt}} - t_j^{\text{det}}$). All ground truth events are covered by our detections (column 4), while an average of 66.47% of the detected events have a valid ground truth match (column 7). These coverage fractions directly correspond to *recall* and *precision*, yielding an F-score of 0.8. At the same time, the spatial overlap of matched events is high (almost 80% on average for the covered ground truth events) and within the error margins of our freehand annotation, while average detection delays are very small. A small number of our detections (mostly separations) are depicted in the third column of Fig. 3.3, with the respective ground truth events shown in the first column.

We note that, in our context of non-rigid registration, high recall is more important than high accuracy, because missing a topology change event is very likely to result in motion estimation errors. At the same time, as discussed in the concluding remarks of Section 3.3.4.1, a small number of false positive detections have essentially no impact on motion estimation under reasonable assumptions. Therefore, our topology change detection mechanism has desirable properties from a motion estimation perspective, while, at the same time, being able to detect contacts and separations with reasonable accuracy.

Table 3.2: Topology Change Event Detection.

Sequence	**Number of events**		**Ground truth to detected mapping** GtToDet			**Detected to ground truth mapping** DetToGt		
	Ground truth	Detected	Matched fraction %	Mean overlap	Mean delay	Matched fraction %	Mean overlap	Mean delay
`clap`	11	32	100.00	0.832	1.000	78.12	0.633	0.800
`drum`	21	63	100.00	0.763	-0.190	76.19	0.616	-0.042
`stack`	4	15	100.00	0.820	-0.250	40.00	0.713	0.000
`unstack`	4	20	100.00	0.689	-0.250	40.00	0.527	0.000
`separate`	5	17	100.00	0.661	-0.400	70.59	0.527	-0.583
`top_drawer`	4	11	100.00	0.951	0.000	54.55	0.819	-0.667
`bot_drawer`	3	12	100.00	0.882	0.000	66.67	0.757	-0.375
Overall:	52	170	100.00	0.788	0.058	66.47	0.630	0.035

3.4.3.2 Registration under dynamic topology

We now quantitatively evaluate our motion estimation performance in the presence of dynamic topology. As discussed before, object separation events tend to induce warp field artifacts when not accounted for, while standard 'forward' warp estimates properly handle contacts. Therefore, we focus our assessments on areas of separation events.

For each ground truth separation event, we compute the average point-to-point distance between points in the *warped* source geometry and their nearest neighbors in the target frame. In our comparisons, we include four different motion field estimation algorithms: VO-SF [46], PD-Flow [45], F-Warp (our baseline), and FB-Warp (our proposed method). In order to avoid obscuring the differences between the two estimates in the areas of interest, instead of averaging over the whole source frame, we only consider points within our ground truth annotation masks, which provide reasonable approximations of the true motion boundaries. Furthermore, we discard occluded warped points using a simple depth test against the target geometry's depth map, with a tolerance threshold of $\Delta z_{\text{occ}} = 1\text{cm}$.

We report per-sequence average registration errors (in mm) over separation areas in Table 3.3. FB-Warp is more accurate in most sequences, achieving an average error reduction of about 30% over our F-Warp baseline, with PD-Flow being a very close second. VO-SF produces significantly less accurate results, because of the coarse pre-segmentation step on which its piecewise-rigid model is based. We also provide qualitative registration results for a subset of our ground truth separation

Table 3.3: Registration Under Close-To-Open Topology.

Sequence	**Registration error** (in mm)			
	VO-SF	PD-Flow	F-Warp	FB-Warp
`clap`	4.652	1.155	1.326	**1.135**
`drum`	2.678	**1.006**	1.608	1.207
`stack`	3.209	1.565	1.996	**1.452**
`unstack`	6.457	2.371	3.138	**2.192**
`separate`	2.249	**1.735**	2.418	1.985
`top_drawer`	1.995	1.383	2.305	**1.325**
`bot_drawer`	5.680	1.773	2.380	**1.222**
Overall:	3.846	1.570	2.167	**1.503**

events in Fig. 3.4. VO-SF introduces seam artifacts to the warped geometry, as a result of its pre-segmentation step. As the latter is highly unlikely to align with separation boundaries, the algorithm does not preserve motion discontinuities. F-Warp, as expected, significantly oversmooths separation motion boundaries. FB-Warp and PD-Flow exhibit the best performance, with the former producing appreciably cleaner surface separations.

3.5 Summary

We presented a complete pipeline for the non-rigid registration of arbitrary, unorganized point clouds that may be topologically different. Building upon a general warp field estimation algorithm, we introduced an efficient topology event handling post-processing phase that detects and classifies object contact and separation events, and, by exploiting the different qualities of forward and backward motion estimates with respect to different event types, locally selects the most appropriate one, in a seamless manner. We evaluated the motion estimation accuracy of our method on the MPI Sintel dataset, achieving state-of-the-art performance. Our

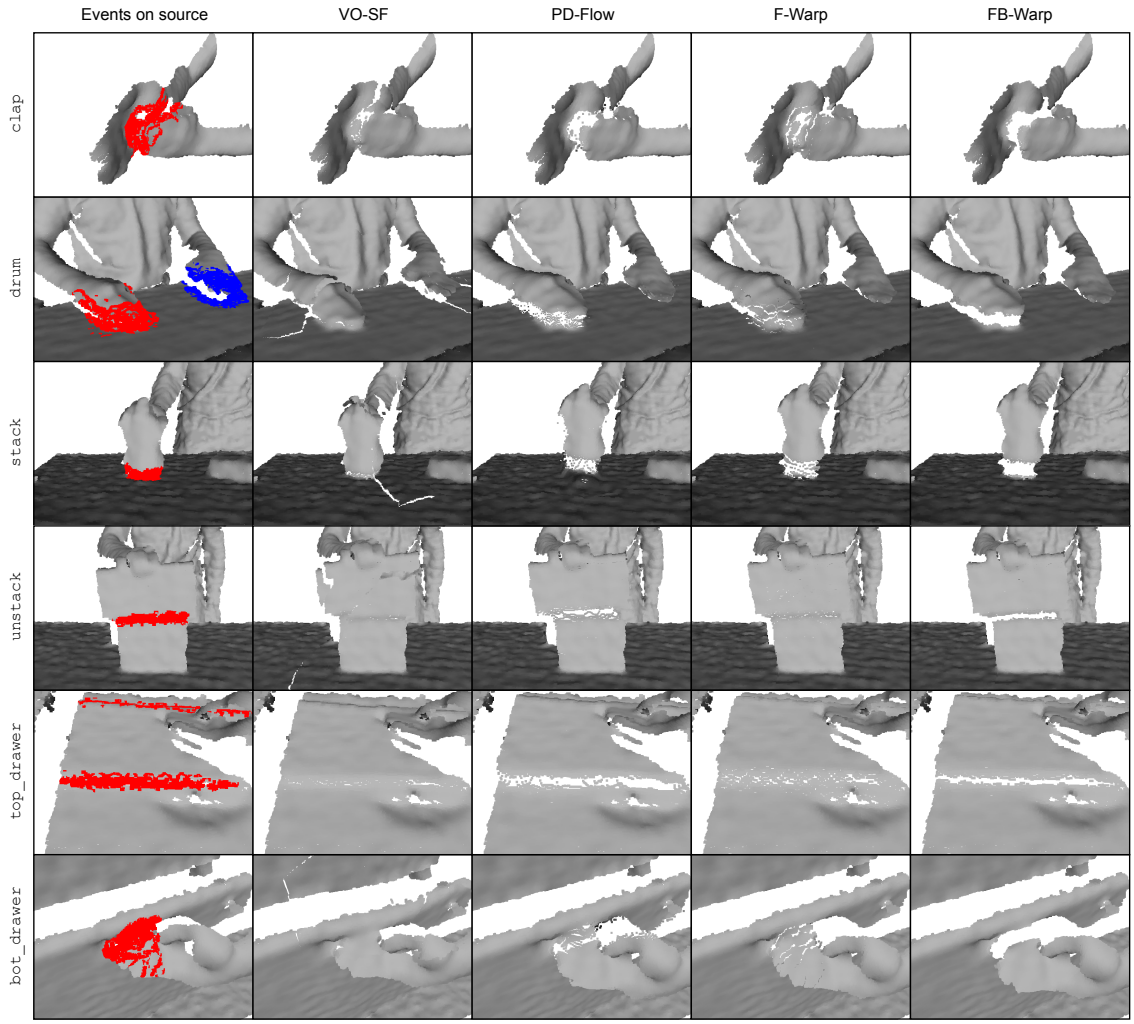


Figure 3.4: Warping results on our dynamic topology dataset, with focus given to separation events. Each row corresponds to a different sequence. First column: our topological event detections overlaid on the source geometry (same as last column of Fig. 3.3, but rendered from a different viewpoint). Columns 2-5: warped source geometry under VO-SF, PD-Flow, F-Warp (our topology-agnostic baseline algorithm), and FB-Warp (our proposed approach).

evaluation on a custom dataset with sequences of highly dynamic scene topology demonstrated the success of our method in estimating motion on topological event boundaries, and showed promising performance in event detection. To the best of our knowledge, this is the first approach to handle dynamic topology in the context of raw point cloud registration. Furthermore, we openly release the implementation of our baseline warp field estimation algorithm as part of our point cloud processing library [17].

In this work, we focused on improving dense motion estimation on separation boundaries by reasoning about two specific types of dynamic topology: 'open-to-close' and 'close-to-open'. There exist, however, object interactions that induce different types of topological changes, which our method is not equipped to handle. One such interesting example is the case of an object *sliding* on its supporting surface. In this case, while our deformation criteria might give us some hints regarding the problematic areas, our inverted backward estimate is expected to share similar oversmoothing properties as a standard, forward warp field. We are currently investigating insights that would allow us to efficiently tackle those situations, ideally without attacking the more general and (possibly) much harder problem of joint motion estimation and motion segmentation.

Chapter 4: Extracting Contact, Objects, and Object Motions from Manipulation Videos

When we physically interact with our environment using our hands, we touch objects and force them to move: contact and motion are defining properties of manipulation. In this chapter, we present an active, bottom-up method for the detection of actor-object contacts and the extraction of moved objects and their motions in RGBD videos of manipulation actions. At the core of our approach lies non-rigid registration: we continuously warp a point cloud model of the observed scene to the current video frame, generating a set of dense 3D point trajectories. Under loose assumptions, we employ simple point cloud segmentation techniques to extract the actor and subsequently detect actor-environment contacts based on the estimated trajectories. For each such interaction, using the detected contact as an attention mechanism, we obtain an initial motion segment for the manipulated object by clustering trajectories in the contact area vicinity and then we jointly refine the object segment and estimate its 6DoF pose in all observed frames. Because of its generality and the fundamental, yet highly informative, nature of its outputs, our approach is applicable to a wide range of perception and planning tasks. We qualitatively evaluate our method on a number of input sequences and present a

comprehensive robot imitation learning example, in which we demonstrate how our outputs facilitate developing action representations/plans from observations.

4.1 Introduction

A manipulation action, by its very definition, involves the handling of objects by an intelligent agent. Every such interaction requires physical contact between the actor and some object, followed by the exertion of forces on the manipulated object, which typically induce motion. When we open a door, pick up a coffee mug, or pull a chair, we invariably touch an object and cause it (or parts of it) to move. This obvious observation demonstrates that *contact* and *motion* are two fundamental aspects of manipulation.

Contact and motion information alone are often sufficient to describe manipulations in a wide range of applications, as they naturally encode crucial information regarding the performed action. Contact encodes *where* the affected object was touched/grasped, as well as *when* and for how long the interaction took place. Motion conveys *what* part of the environment (i.e. which object or object part) was manipulated and *how* it moved.

The ability to automatically extract contact and object motion information from video either directly solves or can significantly facilitate a number of common perception tasks. For example, in the context of manipulation actions, knowledge of the spatiotemporal extent of an actor-object contact automatically provides action *detection/segmentation* in the time domain, as well as *localization* of the detected

action in the observed space [2, 3]. At the same time, motion information bridges the gap between the observation of an action and its semantic grounding. Knowing what part of the environment was moved effectively acts as an attention mechanism for the manipulated *object recognition* [75, 76], while the extracted motion profile provides invaluable cues for *action recognition*, in both "traditional" [2, 3, 77] and deep learning [78] frameworks.

Robot imitation learning is rapidly gaining attention. The use of robots in less controlled workspaces and even domestic environments necessitates the development of easily applicable methods for robot programming: autonomous robots for manipulation tasks must efficiently *learn* how to manipulate. Exploiting contact and motion information can largely automate robot replication of a wide class of actions. As we will discuss later, the detected contact area can effectively bootstrap the grasping stage by guiding primitive fitting and grasp planning, while the extracted object and its motion capture the trajectory to be replicated as well as any applicable kinematic/collision constraints. Thus, the components introduced in this work can play an essential role in building more complex, hierarchical models of action (e.g., behavior trees, activity graphs) as they appear in the recent literature [10, 12–14, 79–81].

In this chapter, we present an unsupervised, bottom-up method for estimating the human-object contacts and object motions in RGBD video observations of manipulation actions. Our approach reasons about contact interactions and object motions in 3D by relying on dense motion estimation. We first initialize a point cloud model of the observed scene, which captures both the human actor and the

part of their environment they will interact with, and then continuously warp/update it throughout the duration of the video. Building upon our estimated dense 3D point trajectories, we segment the actor, using simple approaches that are valid under reasonable assumptions, and detect actor-environment contact locations and time intervals. Subsequently, we exploit the detected contact to guide the motion segmentation of the manipulated object. Finally, we estimate a refined object segment and its 6DoF pose in all observed video frames. An overview of our process in shown in Fig. 4.1; our intermediate and final results are summarized in Table 4.1. The generality of our framework, combined with the highly informative nature of our outputs, renders our approach applicable to a wide spectrum of perception and planning tasks.

It is worth noting that we do not treat contact detection and object motion segmentation/estimation independently: we use the detected contact as an *attention mechanism* to guide the extraction of the manipulated object and its motion. This *active* approach provides an elegant and effective solution to our joint motion estimation and motion segmentation task. A passive approach to our problem would typically segment the whole observed scene into an *unknown* (i.e. to be estimated) number of motion clusters. By exploiting contact, we avoid having to solve a much larger and less constrained problem and gain significant improvements in terms of both computational efficiency and segmentation/estimation accuracy.

After discussing related work, we provide a detailed technical description of our method in Section 4.3. In Section 4.4, we demonstrate our intermediate results and final outputs for a number of input sequences. In Section 4.5, we present a

comprehensive example of how our outputs can be used to enable an one-shot robot imitation learning task.

4.2 Related work

We focus our literature review on recent works in three areas that are most relevant to our problem and the major processes/components upon which we build. We are not including works from the action recognition literature; the scope of this chapter is the extraction of contacts, moving manipulated objects, and their motions.

Dense 3D motion estimation. We include a comprehensive review of RGBD scene flow estimation methods, as well as more general warp field estimation algorithms used in recent systems for dynamic reconstruction from RGBD input, in Section 3.2. While being of great relevance in a number of motion reasoning tasks, plain scene flow cannot be directly integrated into our pipeline, which requires *model-to-frame* motion estimation: the scene flow motion field has a 2D support (i.e. the image plane), effectively warping the 2.5D geometry of an RGBD frame, while we need to continuously warp a *fully* 3D point cloud model. In this work, we use the non-rigid registration algorithm of [15], described in detail in Chapter 3.

Contact detection. A CNN-based method for grasp recognition is introduced in [82]. A 2D approach for detecting "touch" interactions between a caregiver and an infant is presented in [83]. To the best of our knowledge, there is no prior work on explicitly determining the spatiotemporal extent of human-environment

contact.

Motion segmentation. A very large volume of works on motion segmentation have casted the problem as subspace clustering of 2D point trajectories, assuming an affine camera model [84–89]. In [90], an active approach for the segmentation and kinematic modeling of articulated objects is proposed, which relies on the robot manipulation capabilities to induce object motion. In [91], object segmentation is performed from two RGBD frames, one before and one after the manipulation of the object, by rigidly aligning and 'differencing' the two views and robustly estimating rigid motion between the 'difference' regions. The same method is used in [51], where scene flow is used to obtain motion proposals, followed by an MRF inference step. In [68], joint tracking and reconstruction of multiple rigidly moving objects is achieved by combining two segmentation/grouping strategies with multiple surfel fusion [48] instances. A naive integration of a generic motion segmentation algorithm for the extraction of the manipulated object into our pipeline would be suboptimal in multiple ways. For instance, given the fact that there may exist an unknown number of other object motions that are irrelevant to the manipulation, we would be potentially solving an unnecessarily hard problem. For the same reason (unknown number of motion components), we would have little control over the segmentation granularity, which could cause the manipulated object to be over/under-segmented. Instead, we leverage the detected contact and bootstrap our segmentation by an informed trajectory clustering approach that is otherwise in the same spirit as that of [92].

4.3 Our approach

4.3.1 Overview

We present a system that, given a video of a human performing a manipulation task as input, *detects* and *tracks* the parts of the environment that participate in the manipulation. More specifically, our system is able to visually detect physical contact between the actor and their environment, and, using contact as an attention mechanism, eventually segment the manipulated object and estimate its 6DoF pose in every observed video frame. Our pipeline, as well as the interconnections of the involved processes, are sketched in Fig. 4.1 and followed by a more detailed description. We summarize our proposed system's expected inputs, final outputs, and some useful generated intermediate results in Table 4.1.

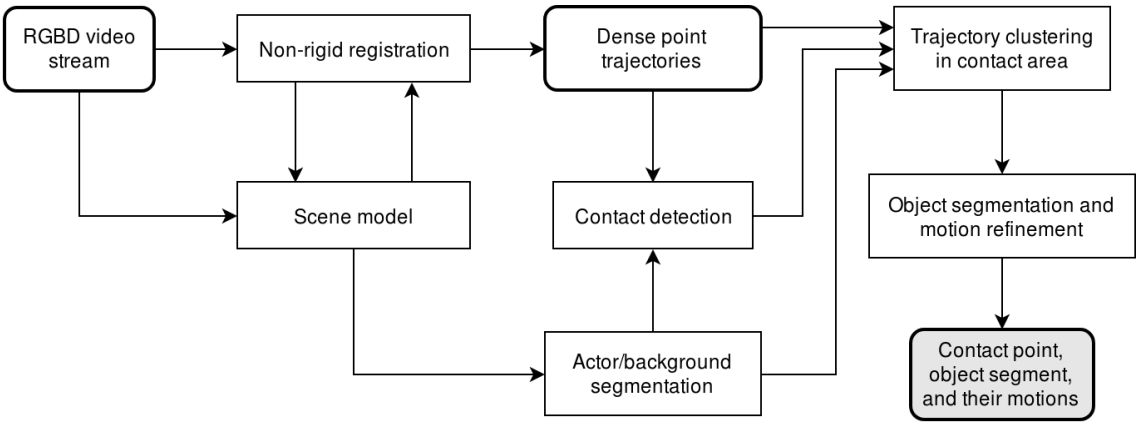


Figure 4.1: A high-level overview of our modules and their relations in the proposed pipeline.

The input to our system is an RGBD frame sequence, captured by a commodity depth sensor, of a human actor performing a task that involves the manipulation of objects in their environment. We assume that the input depth images are reg-

Table 4.1: List of our inputs, intermediate results, and final outputs.

Input	Intermediate results	Final outputs
RGBD video of manipulation	• Dense **point trajectories** in 3D that span the whole sequence duration • **Actor-environment segmentation** labels for every point at all times	• Detected actor-environment **contact points** and their 3D **trajectories** • Handled **object segments** and their **6DoF poses** for every time point

istered to and in sync with their color counterparts. Using estimates of the color camera intrinsics (e.g., from the manufacturer provided specifications), all input RGBD frames are back-projected to 3D point clouds (colored, with estimated surface normals), on which all subsequent processing is performed.

At the core of our method lies non-rigid point cloud registration; we use the algorithm introduced in [15] and described in detail in Chapter 3. An initial point cloud model of the observed scene is built from the first observed RGBD frame and is then consecutively transformed to the current observation based on the estimated *model-to-frame* warp field at every time instance. This process generates a dense set of point trajectories, each associated with a point in the initial model. In order to keep the presentation clean, we opted to obtain the scene model from the first frame and keep it fixed in terms of its point set. Non-rigid reconstruction techniques for updating the model over time [49, 50] can be easily integrated to our pipeline if required.

To perform actor/background segmentation, we follow the semi-automatic approach described in Section 4.3.2. The obtained binary labeling is propagated to the whole temporal extent of the observed action via our estimated dense point tra-

jectories, and enables us to easily detect human-environment *contacts* as described in Section 4.3.3.

Using our extracted dense scene point trajectories, actor/background labels, and detected contact interaction locations and time intervals, our ultimate goal is, for each such interaction, to *segment* the manipulated object and estimate its *rigid motion* (6DoF pose) for every time instance. Our contact-guided motion segmentation approach for this task is described in Section 4.3.4.

4.3.2 Actor-environment segmentation

We follow a semi-automatic approach to perform actor-background segmentation that relies on simple point cloud segmentation techniques. First, we construct a proximity (radius-based nearest neighbor) graph over the scene model points in the initial state, in which each node is a model point and two nodes are connected if and only if their Euclidean distance falls below a predefined threshold. Assuming that the actor is *initially* not in contact with any other part of the scene (i.e. the minimum distance of an actor point to a background point is at least our predefined distance threshold) and the observed actor points are not too severely disconnected in the initial state, the actor points will be exactly defined by a single *connected component* of this proximity graph.

The selection of the correct (actor) component can be automated by filtering all the extracted components based on context-specific criteria (e.g., rough size, shape, location, etc.) or by picking the component whose image projection exhibits

maximum overlap with the output of a 2D human detector [93, 94]. Alternatively, in order to avoid extracting all the Euclidean segments of the scene, we may begin by selecting a seed point known to belong to the actor and then perform region growing on the model point cloud until our maximum distance constraint is no longer satisfied. The selection of the seed point can also be automated by resorting to standard 2D means (e.g., by picking the point with the strongest skin color response [95, 96] within a 2D human detector output [93, 94]).

We consider the assumptions imposed by our Euclidean clustering based approach for the actor segmentation task to be not particularly restricting, in the sense that they can be easily enforced either directly, during data capture, or in a post-processing step that uses standard common-sense and/or context-specific prior knowledge. For example, we can easily ensure disconnectedness between the actor and a manipulated door in the initial state of a door-opening action by segmenting and removing the floor points that may be connecting the actor and the door. This is particularly true in the typically controlled setting of human demonstrations for robot imitation learning.

We note that, since we opted to keep the scene model point set fixed and track it throughout the observed action, the obtained actor-environment segmentation automatically becomes available at all time points.

4.3.3 Contact detection

The outputs of our full scene model tracking and actor segmentation processes are a dense set of *point trajectories* and their respective actor/background *labels*. Given this information, it is straightforward to reason about actor-environment *contact*, simply by examining whether the minimum distance between parts of the two clusters is small enough at any given time. In other words, we can easily infer both *when* the actor comes into/goes out of contact with part of their environment and *where* this interaction takes place.

Some of the contact interactions detected using this simple proximity criterion may, of course, be semantically irrelevant to the performed action. Assuming a reasonably controlled setting, we expect that the detected contacts are established by the actor *hands*, with the goal of manipulating an *object* in their environment.

4.3.4 Manipulated object segmentation and motion

Knowing the dense scene point trajectories, labeled as either actor or background, as well as the actor-environment contact locations and temporal extents, our next goal is to infer what part of the environment is being manipulated, or, in other words, which object was moved. We assume that every contact interaction involves the movement of a *single* object, and that the latter undergoes *rigid* motion. In the following, we only focus on the *background* part of the scene around the contact point area, ignoring the human point trajectories. We follow a two-step approach.

First, we bootstrap our segmentation task by finding a coarse/partial mask of the moving object, using standard unsupervised clustering techniques. Specifically, we cluster the background point trajectories that lie within a fixed radius of the detected contact point at the beginning of the interaction into two groups. We adopt a spectral clustering approach, using the 'random walk' graph Laplacian [97] and a standard k-means final step. Our pairwise trajectory similarities are given by $s_{ij} = \exp\left(-(d_{\max} - d_{\min})^2/(2\sigma^2)\right)$, where $d_{\min}$ and $d_{\max}$ are the minimum and maximum Euclidean point distance of trajectories i and j over the duration of the interaction, respectively. This similarity metric enforces similar trajectories to exhibit approximately constant point-wise distances, i.e. it promotes clusters that undergo rigid motion. From the two output clusters, one is expected to cover the object being manipulated or a part of it. Operating under the assumption that only interaction can cause motion in the scene, we pick the cluster that exhibits the largest average motion over the duration of contact as our object segment candidate. In the above, we restricted our focus within a fixed-radius region of the contact point, in order to *1)* avoid having our binary classification influenced by other motions in the scene that are not relevant to the current interaction, and *2)* make the classification problem more computationally efficient by drastically reducing its size. The exact value of the radius of interest is unimportant, as long as it is selected to be sufficiently large to cover at least part of the manipulated object.

Subsequently, we obtain a refined, more accurate segment of the moving object by requiring that the latter undergoes a rigid motion that is, at every time point, consistent with that of the previously found motion cluster. Let B^t denote the

background (non-actor) part the scene model point cloud at time t, for $t = 0, \ldots, T$, and $\hat{M}^t \subseteq B^t$ be the initial motion cluster state at the same time instance. For all $t = 1, \ldots, T$, we robustly estimate the rigid motion between point sets $\hat{M}^0$ and $\hat{M}^t$ (i.e. relative to the first frame), using the closed form solution of [98] under a RANSAC scheme, and then find the set of points in *all* of B^t that are potential images of points in B^0 under this motion model. Let I^t, for $t = 1, \ldots, T$, denote this set of motion inliers, so that I^t contains exactly the indices of the points (equivalently, trajectories) in B^t whose motion from B^0 to B^t is consistent with the estimated segment motion from $\hat{M}^0$ to $\hat{M}^t$. We obtain our final object segment for this interaction as the intersection of inlier index sets I^t for all time instances $t = 1, \ldots, T$:

$$I \equiv \bigcap_{t=1}^{T} I^t. \tag{4.1}$$

The subset of the background points indexed by I, as well as the per-frame RANSAC motion (pose) estimates of this last step, are the final outputs of our pipeline for the given interaction.

4.4 Experiments

We provide a qualitative evaluation of our method for video inputs recorded in different settings, covering three different scenarios: *1)* a tabletop object manipulation that involves flipping a pitcher, *2)* opening a drawer, and *3)* opening a room door. All videos were captured from a static viewpoint, using a standard RGBD camera.

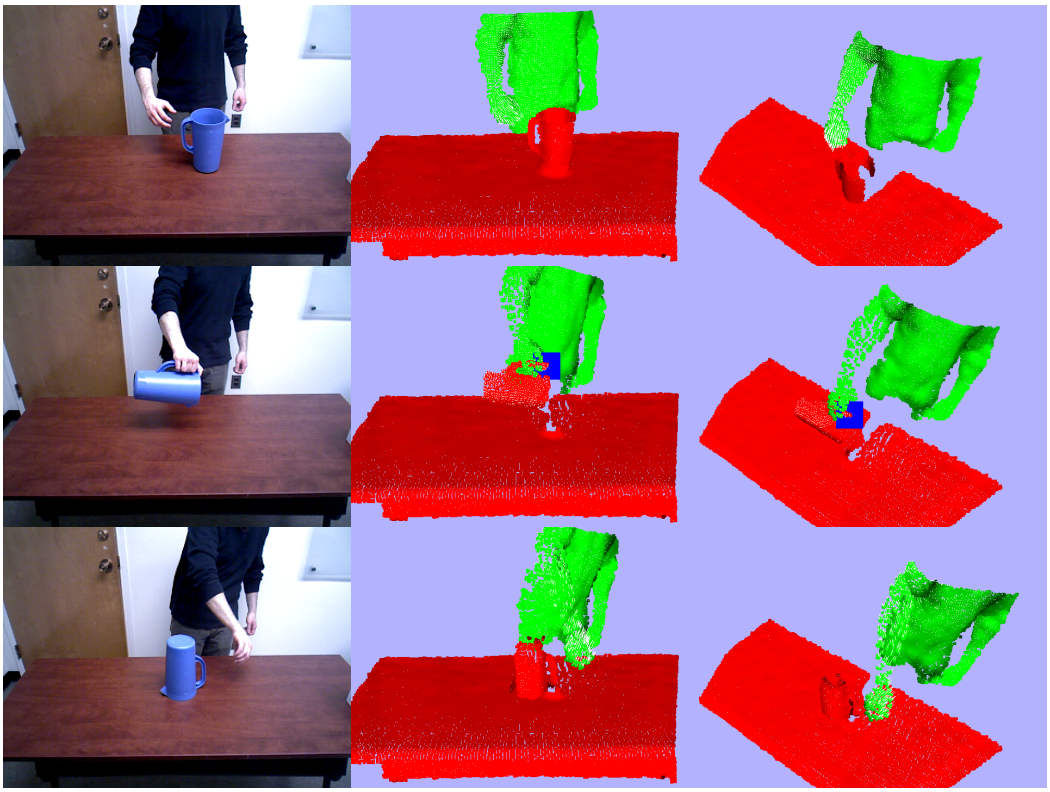


Figure 4.2: Flipping a pitcher: scene tracking, labeling, and contact detection.

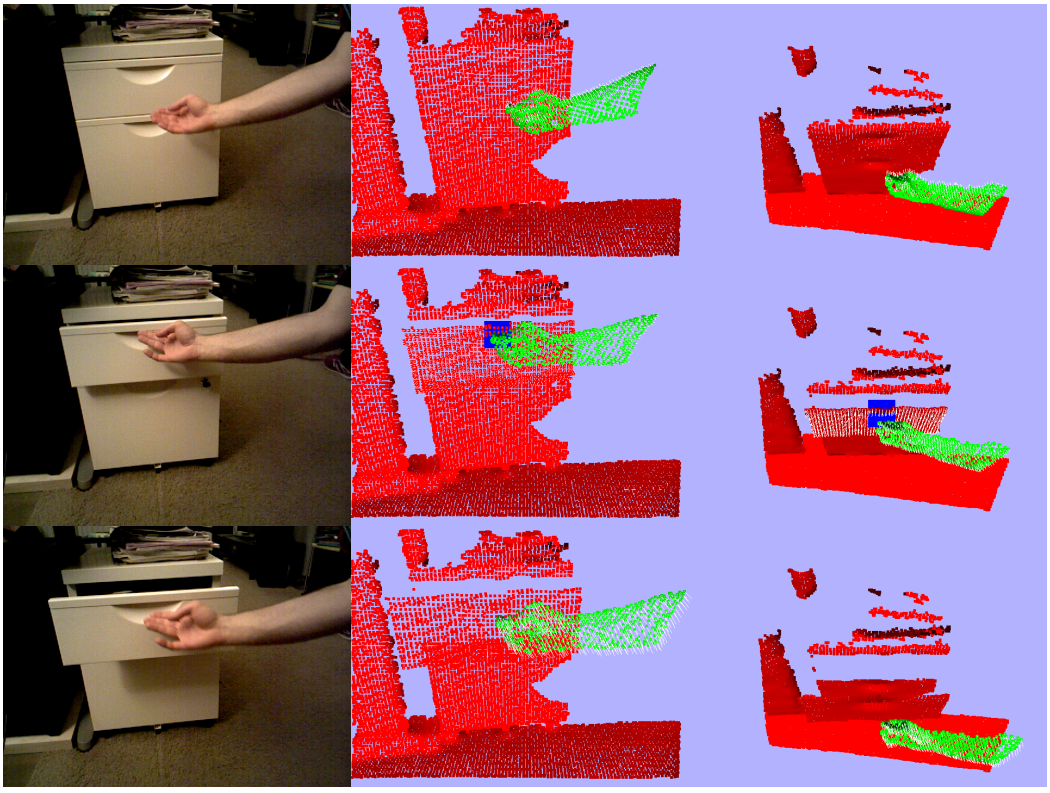


Figure 4.3: Opening a drawer: scene tracking, labeling, and contact detection.

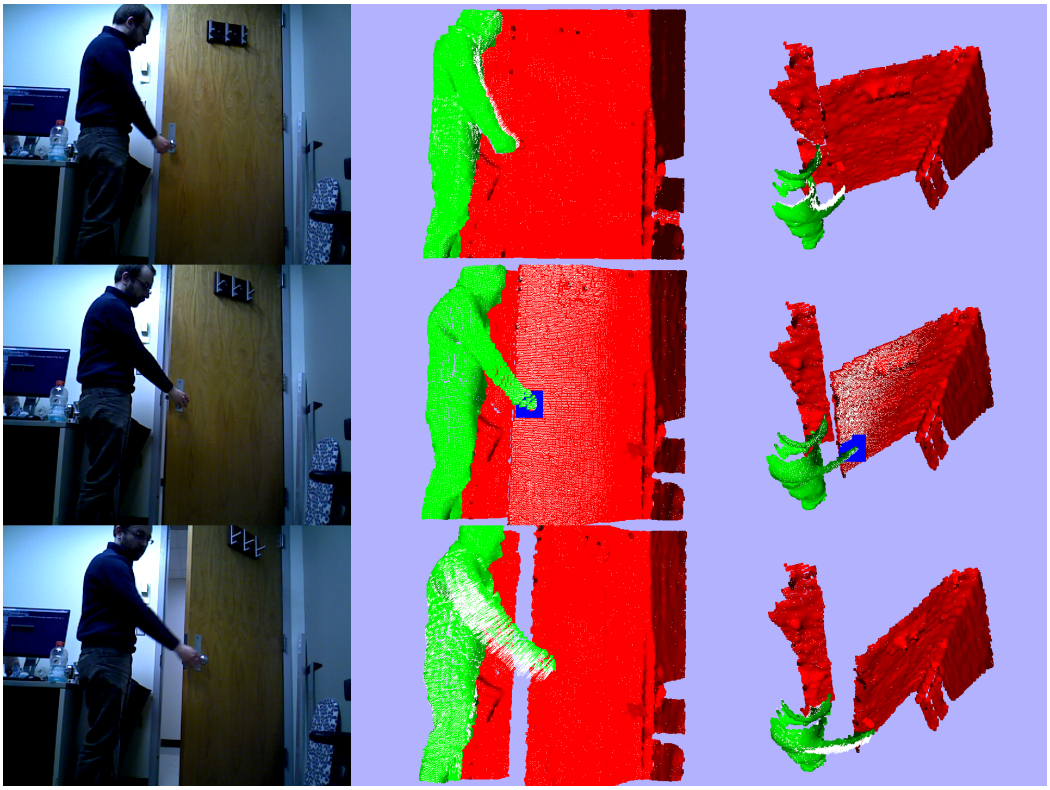
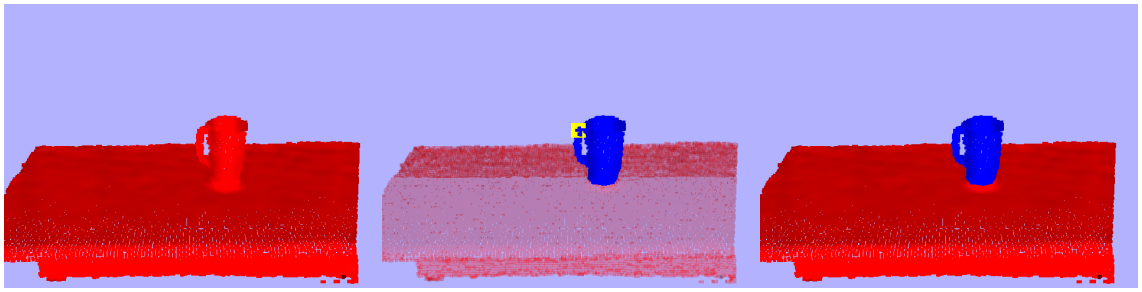


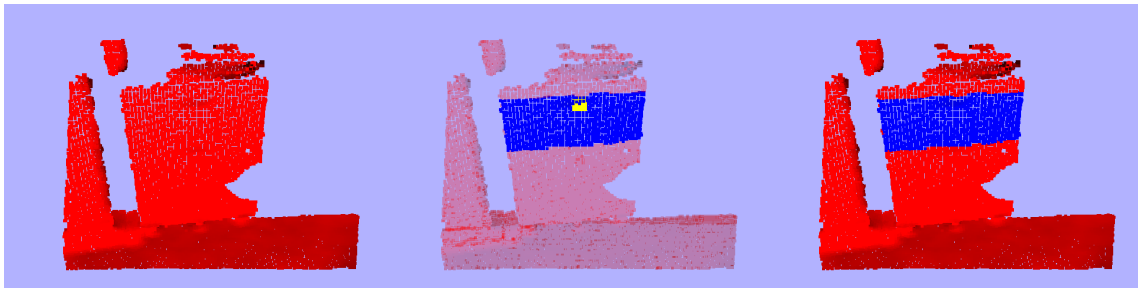
Figure 4.4: Opening a door: scene tracking, labeling, and contact detection.

For each scenario, in Fig. 4.2, 4.3, and 4.4, we depict the scene model point cloud state at three time snapshots: one right before, one during, and one right after the manipulation. For each time point (row), we show the corresponding color image and render the tracked point cloud from two viewpoints. The actor segment is colored green, the background is red, and the detected contact point is marked by blue. We also render the estimated warp field induced point displacements from the currently visible state to its next as white lines (mostly visible in areas that exhibit large motion). The outputs displayed in these figures are in direct correspondence with the processes described in Sections 4.3.2 and 4.3.3.

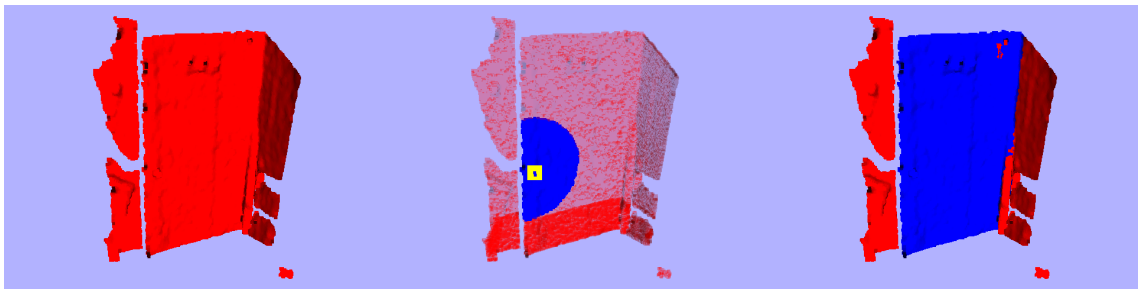
Next, we demonstrate our attention-driven motion segmentation and 6DoF pose estimation of the manipulated object. In Fig. 4.5, we render the background



(a) Flipping a pitcher.



(b) Opening a drawer.



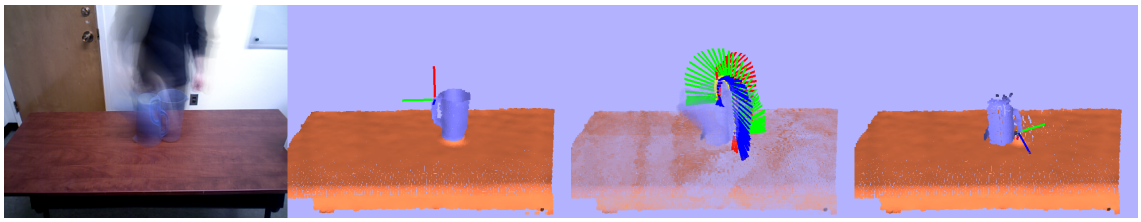
(c) Opening a door.

Figure 4.5: Motion segmentation of the manipulated object. First column: scene background points (the actor is removed). Second column: initial motion segment (blue) obtained by spectral clustering of point trajectories around contact area (yellow). Third column: final, refined motion segment (blue).

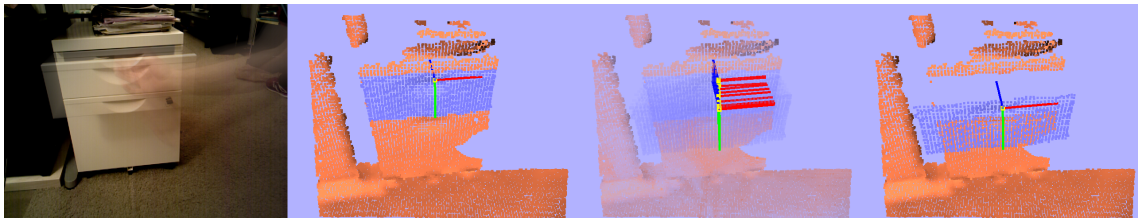
part of the scene model in its initial state with the actor removed (left column) and show the two steps of our segmentation method described in Section 4.3.4. In the middle column, the blue segment corresponds to the initial motion segment, obtained by clustering trajectories in the vicinity of the contact point (propagated back to the initial state and highlighted in yellow). In the right column, we show the refined, final motion segment. We note that, because of our choice of the attention

radius around the contact point in the first segmentation step, the initial segment in the first two cases covers the whole object and is the same as the final one.

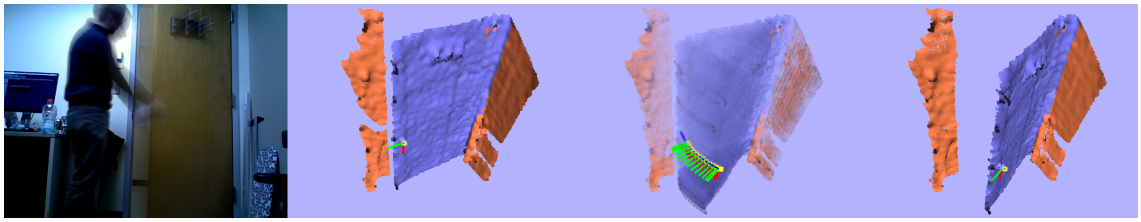
In Fig. 4.6, we show the estimated rigid motion (series of 6DoF poses) of the segmented object. To more clearly visualize the evolution of object pose over time, we attach a local coordinate frame to the object, at the location of the contact point, whose axes were chosen as the principal components of the extracted object point cloud segment.



(a) Flipping a pitcher.



(b) Opening a drawer.



(c) Opening a door.

Figure 4.6: Estimated rigid motion of the manipulated object. A coordinate frame is attached to the object segment (blue) at the contact point location (yellow). First column: temporal accumulation of color frames for the whole action duration. Second column: object state before manipulation. Third column: object trajectory as a series of 6DoF poses. Fourth column: object state after manipulation.

The above illustrations provide a qualitative demonstration of the successful application of our proposed pipeline to three different manipulation observations. In

all cases, contacts were detected correctly and the manipulated object was accurately segmented and rigidly tracked. A more thorough, quantitative evaluation of our segmentation outputs on an extended set of inputs is in our future plans.

4.5 Application: replication from observation by a robot

For any human-environment task to be successful, there is a well-defined process involved, demarcated into distinct phases based on human-environment contact and consequent motion. This allows us to generate a graph representation for actions, such as that shown in Fig. 4.7 for the task of opening a refrigerator door. Given this general representation of tasks, we demonstrate how our algorithm allows grounding of the *grasping* process, based on contact detection, and also of the *feedback loop* for opening the door, based on motion analysis of segmented objects. This type of representation can form the basis for a principled method of bridging perception with planning, enabling robots to replicate observed human actions.

In the following, we present a comprehensive application of our method to a real-world task, where a robot observes a human operator opening a refrigerator door and learns the process for replication. Our setup is shown in Fig. 4.8, where a calibrated RGBD sensor mounted on the robot's manipulator is used for observation. Using our methods elucidated in Section 4.3, our process involves the segmentation of the human from the environment, followed by the contact interaction detection (in this case, between the hand and the refrigerator door handle) and the joint segmentation and rigid motion tracking of the manipulated object (door).

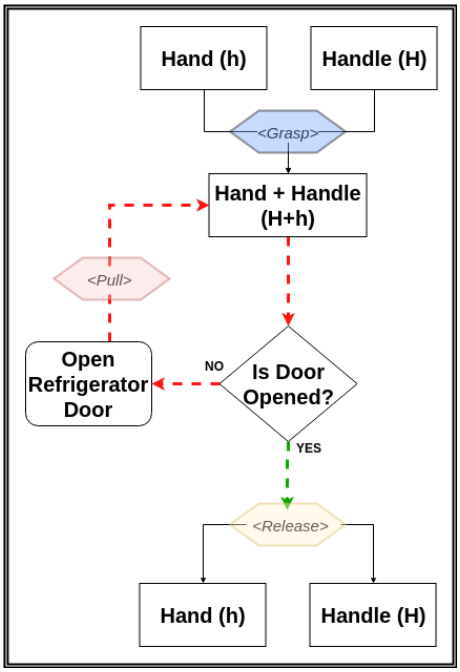


Figure 4.7: High-level representation of opening a refrigerator door.

Figure 4.8: Robot observing a human opening a refrigerator door.

Using prior domain knowledge about our scenario, the outputs of these analyses are then converted into an intermediate graph-like representation, which encodes both semantic labeling of regions of interest, such as doors and handles in our case, as well as motion trajectories computed from the observed manipulation. As we will show, the combination of the above provides the robot with the ability of low-level imitation learning (i.e. trajectory replication) given a single demonstration. The

combined graph-like representation is visually described in Fig. 4.9, which separates our process into three phases: *preprocessing*, *planning*, and *execution*. In the following, we provide more detailed descriptions of each of these three phases involved in our application.

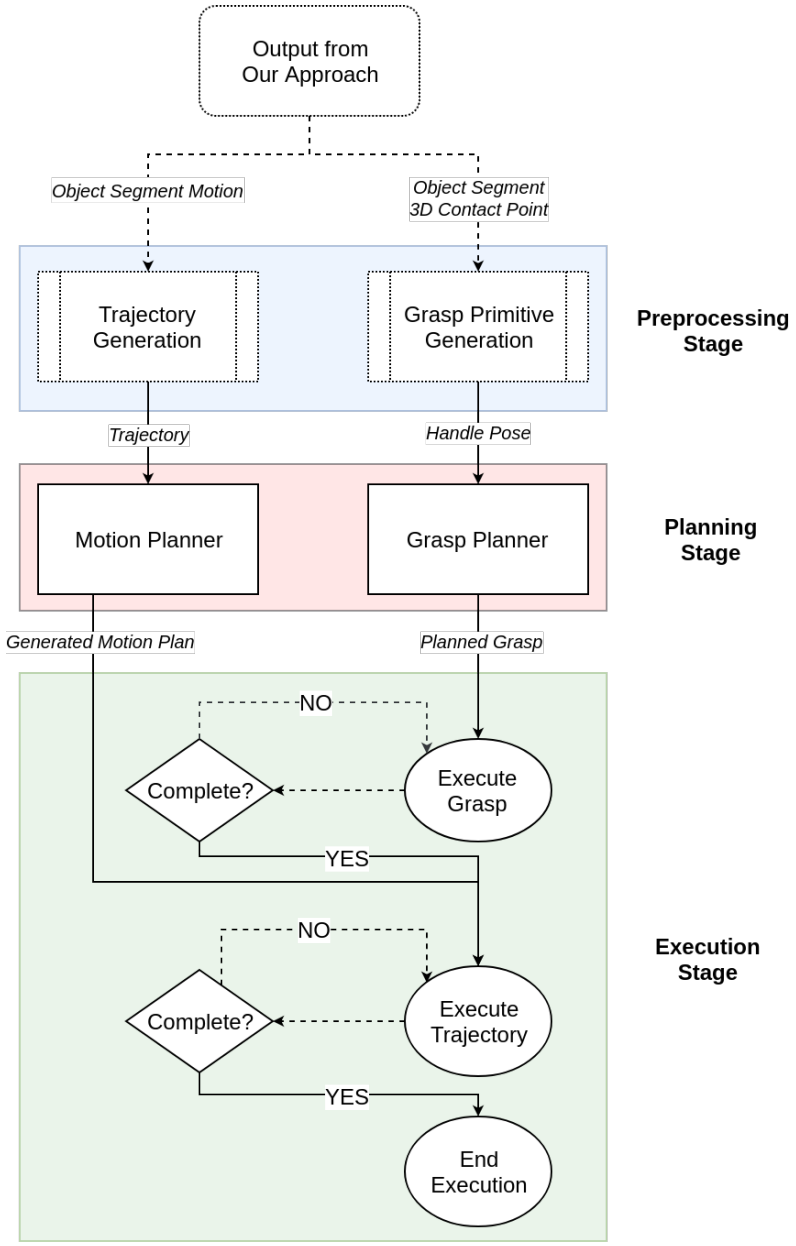


Figure 4.9: State transition diagram of our process.

4.5.1 Preprocessing stage

The preprocessing stage is responsible for taking the outputs of our approach described in Section 4.3 (contact point, object segments and their motion trajectories, as summarized in the last column of Table 4.1) and converting them into robot-specific grasp poses and trajectories that will be used for planning and execution. A visualization of the input entities to this stage for our door opening task is provided in Fig. 4.10.

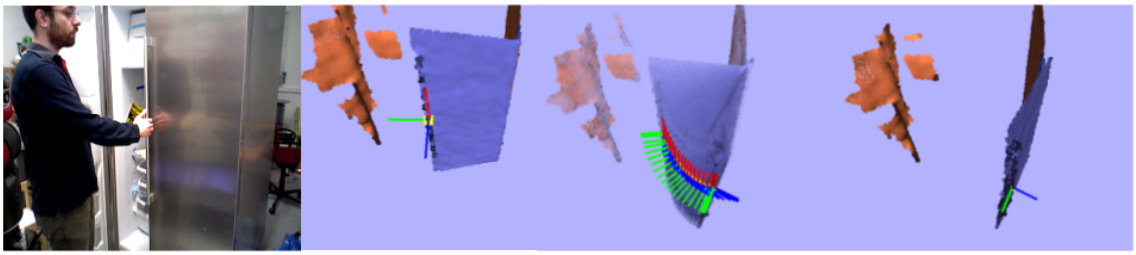
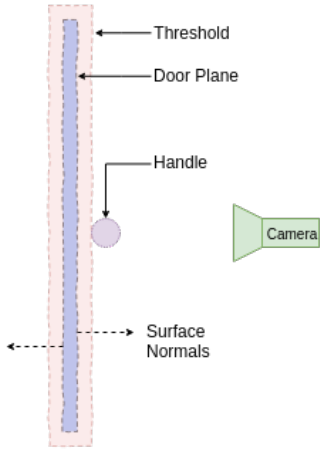


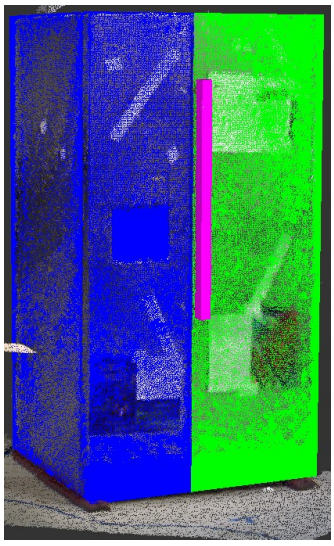
Figure 4.10: Input to the preprocessing stage from our algorithm. (a) RGB frame captured at the beginning of the door manipulation. (b) Extracted contact point (yellow), object segment (blue), and initial object pose (frame axes). (c) 6DoF pose trajectory of door during its manipulation. (d) Final object state, after opening was completed.

In this stage, we exploit domain knowledge to ground and interpret contact points, as well as object segments and trajectories, in a way that will enable the replication of the manipulation task by the robot. For instance, since we know that our task involves opening a refrigerator door, we can make the assumptions that the contact point between the human agent and the environment will happen at the door handle and any consequent motion will likely only involve the door. Furthermore, we know that the manipulated object (door) segment essentially follows a planar surface model, with the exception of the door handle part.

Based on this scenario-specific information, we proceed to robustly fit a plane to the points of the extracted door segment, using standard least squares fitting under RANSAC, and obtain a set of points for the door handle as the major Euclidean cluster of the door plane outliers (Fig. 4.11a). We subsequently fit to these points a cuboid block of consistent orientation with that of the door plane and major axes (Fig. 4.11b), in order to generate a grasp primitive with a known 6DoF pose for robot *grasp planning* (Section 4.5.2.1). The estimated trajectories of the object segment, as depicted in Fig. 4.10, are not directly utilized by the robot execution system, but are instead converted to a robot-specific representation before replication can take place. In particular, the visually extracted trajectory poses and their relative timestamps are used to define waypoint poses and time constraints for the robot's end effector *trajectory planning* (Section 4.5.2.2).



(a) Diagram depicting refrigerator handle detection.



(b) Point cloud of refrigerator with detected handle and door.

Figure 4.11: Door handle detection.

4.5.2 Planning stage

The outputs from the preprocessing stage, namely the robot-specific 6DoF end effector poses and the grasp primitive parameters for the door handle, are passed to the planning stage of our pipeline, for both *grasp* and *trajectory* planning. The RViz [99] package in the Robot Operating System (ROS) [100] allows for simulation and visualization of the robot during planning and execution, via real-time feedback from the robot's state estimator. It also has point cloud visualization capabilities, which can be overlaid over virtual shapes. We use this tool for the planning stage; an RViz screenshot of our Baxter robot model and point cloud observations overlaid over fitted grasp/collision primitive shapes is shown in Fig. 4.12.

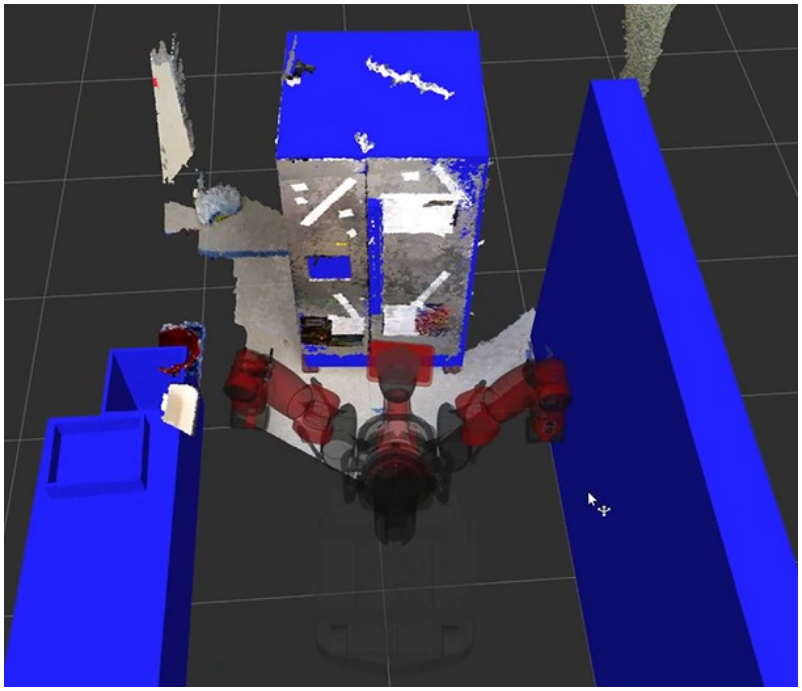


Figure 4.12: Visualization of planning stage.

4.5.2.1 Grasp planning

Given the location, orientation, and size parameters of the primitive block shape fitted to the door handle, we use the "MoveIt!" Simple Grasps [101] package to generate grasp candidates for the parallel gripper mounted on the Baxter robot. The package integrates with the "MoveIt!" [102] library's 'pick-and-place' pipeline to simulate and generate multiple potential grasp candidates (i.e. approach poses). This process also involves a grasp filtering stage, which uses task and configuration specific constraints to remove kinematically infeasible grasps, by performing feasibility tests via inverse kinematics solvers. At the end of the grasp planning pipeline, we have a set of candidate grasps, sorted by a grasp quality metric, of which one is selected for execution.

4.5.2.2 Trajectory planning

The ordered set of the poses over time obtained from the preprocessing stage is used to generate a Cartesian path, using the ROS "MoveIt!" [102] motion planning library. The abstractions provided by the library allow us to input a set of poses through which the end effector must pass, along with constraint parameters for path validity and obstacle avoidance. The "MoveIt!" library then uses sampling-based planning algorithms, such as Rapidly-Exploring Random Trees (RRT) [103], which takes into account inverse kinematics constraints for the specified manipulator configuration, to generate a trajectory for the robot to execute.

4.5.3 Execution stage

The execution stage takes as input the grasp and trajectory plans generated in the planning stage and executes the task on the robot. First, the generated grasp candidate is used to move the end effector to a pre-grasp pose, in which the parallel gripper is aligned to the fitted door handle block primitive. The grasp is executed based on a feedback control loop, with the termination condition decided by collision avoidance and force feedback. Upon successful grasp of the handle, our pipeline transitions into the trajectory execution stage, which attempts to follow the generated plan based on feedback from the robot's state estimation system. Once the trajectory has been successfully executed, the human motion replication pipeline is complete. A robot execution instance is demonstrated in Fig. 4.13, beginning with the robot grasping the handle (top-left frame) and ending with the robot releasing the handle (bottom-left frame), with intermediate frames showing

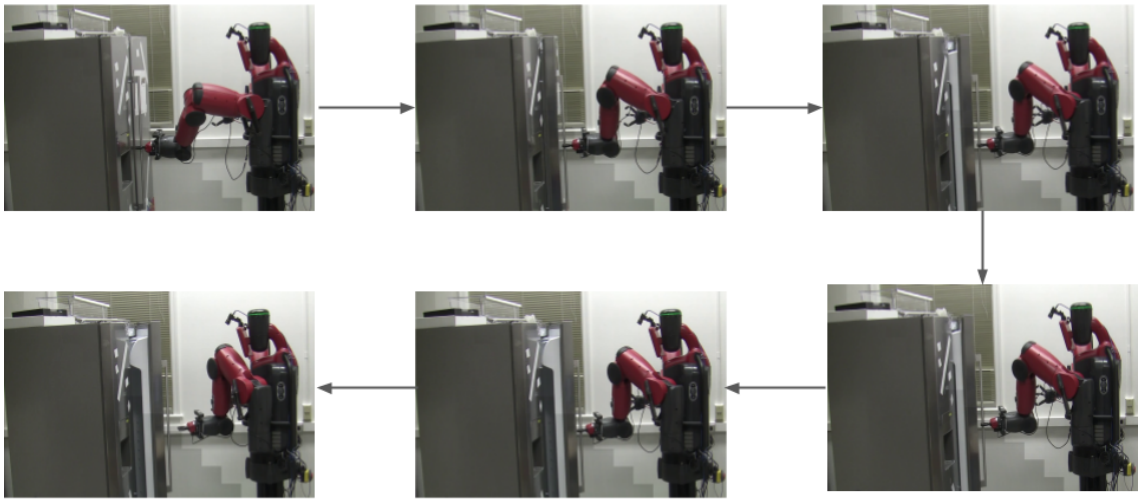


Figure 4.13: Robot imitating human in opening refrigerator door.

the robot imitating the demonstrated trajectory by the human. In future work, we plan to implement a Dynamic Motion Primitives (DMP) [104] based approach, which will allow more accurate and robust trajectory following by the robot.

4.6 Summary

In this chapter, we have introduced an active, bottom-up method for the extraction of two fundamental features of an observed manipulation, namely the actor-environment contact points and the manipulated object segments and rigid motion trajectories. We have qualitatively demonstrated the success of our approach on a set of video inputs and described in detail its enabling role in a robot imitation scenario. Owing to its general applicability and the manipulation-defining nature of its output features, our method can significantly facilitate bridging the gap between observation and the development of action representations and executable plans.

There are many possible directions for future work. At a lower level, we plan to integrate *dynamic reconstruction* into our pipeline to obtain a more complete model for the manipulated object; currently, this can be achieved by introducing a step of static scene reconstruction before the manipulation happens, after which we run our algorithm. We also plan to extend our method so that it also can handle *articulated* manipulated objects, as well as objects that are *indirectly* manipulated (e.g., via the use of tools). On the planning end, one of our future goals is to release a software component for the fully automated replication of *door opening* tasks given only a single human demonstration (Section 4.5). This module will be hardware

agnostic up to the final execution stage of the pipeline, so that the generated plan can be easily adapted to become executable by any robot agent, given their specific manipulator and end effector configurations.

Chapter 5: `cilantro`: A Lean, Versatile, and Efficient Library for Point Cloud Data Processing

We introduce `cilantro`, an open-source C++ library for geometric and general-purpose point cloud data processing. The library provides functionality that covers low-level point cloud operations, spatial reasoning, various methods for point cloud segmentation and generic data clustering, flexible algorithms for robust or local geometric alignment, model fitting, as well as powerful visualization tools. To accommodate all kinds of workflows, `cilantro` is almost fully templated, and most of its generic algorithms operate in arbitrary data dimension. At the same time, the library is easy to use and highly expressive, promoting a clean and concise coding style. `cilantro` is highly optimized, has a minimal set of external dependencies, and supports rapid development of performant point cloud processing software in a wide variety of contexts.

Availability: the project source code, with usage examples and sample data, can be found at `https://github.com/kzampog/cilantro`.

5.1 Introduction

Processing geometric input plays a crucial role in a number of machine perception scenarios. Robots use stereo cameras or depth sensors to create 3D models of their environment and/or the objects with which they interact. Autonomous mobile agents are typically equipped with LiDAR sensors to map their surroundings, localize themselves, and avoid collisions. Consumer electronics are increasingly adopting the integration of depth cameras to identify users and enable "natural" user interfaces. The output signals of these sensors are either inherently or directly convertible to 2D or 3D point clouds, highlighting the need for usable and efficient tools for processing raw geometric data.

Thanks to the central role of geometry in the fields of computational theory and computer graphics, a number of notable 3D data processing open-source software libraries have been developed. The Computational Geometry Algorithms Library (CGAL) [105] implements algorithms and data structures for an extensive set of tasks, including triangulations, shape analysis, meshing, and various geometric decompositions. The Visualization and Computer Graphics (VCG) library [106] provides a collection of tools for processing and visualizing 3D meshes and constitutes the back-end of the popular MeshLab [107] GUI-based mesh editor. To simplify working with mesh data, libigl [108] drops complex data structures in favor of raw data matrices and vectors for shape representations, and supports computation of discrete differential quantities and operators, shape deformation, and remeshing functionalities. While mature and feature-rich in their respective contexts, these

software packages have had limited adoption by the machine perception and engineering communities.

The Point Cloud Library (PCL) [36] was introduced to fill this gap and became the standard for unorganized point cloud processing among roboticists and machine vision practitioners. It implements numerous algorithms for filtering, feature extraction, geometric registration, reconstruction, segmentation, and model fitting. Partly due to its templated nature, PCL exhibits a steep learning curve; its user-friendliness is further affected by its verbose coding style and typically long application compilation times. Furthermore, its performance in very common tasks is lacking by today's standards (e.g., due to lack of parallelization in many of its modules) and the project has been in an essentially dormant state for a long time. The Open3D [109] library was recently introduced as a potential alternative. It implements 3D point cloud primitive operations (neighbor queries, downsampling, surface normal estimation), as well as useful tools for geometric registration and 3D reconstruction. The library is easy to use and performs better than PCL in common tasks. However, its supported functionality is quite limited, as it essentially only targets 3D reconstruction workflows.

We introduce `cilantro`, a versatile, easy to use, and efficient C++ library for generic point cloud data processing. We have implemented a concise set of algorithms that cover primitive point cloud operations, spatial reasoning based on convex polytopes, various methods for point cloud segmentation and generic data clustering, flexible algorithms for both robust and local (iterative) geometric alignment, model fitting, as well as powerful visualization tools. The library is written

in C++11, is highly templated and optimized, and makes efficient use of multi-threading for computationally demanding tasks. Significant effort has been put into making every component customizable by the user, so that the library does not hinder the development of "non-standard" workflows. At the same time, we provide useful convenience functions and aliases for the most common algorithm variants, ensuring that our adaptable design does not get in the way of productivity.

By virtue of its flexibility, ease of use, and comprehensive out-of-the-box functionality, `cilantro` is a great fit for a wide spectrum of workflows, enabling the rapid development of performant point cloud processing software. In the following, we briefly state our main design principles (Section 5.2), give a more detailed overview of our supported functionality (Section 5.3), and demonstrate `cilantro`'s performance in some ordinary tasks in comparison to equivalent implementations in PCL and Open3D (Section 5.4).

5.2 Design overview

We outline `cilantro`'s main design principles and how they are reflected on the library implementation.

Simple data representations. We rely on `Eigen` [110] matrices and common STL containers to represent point sets and most other entities used/generated by our algorithms. In particular, the input to almost all of our algorithms is a *matrix view* of a point set: a lightweight wrapper, based on the `Eigen::Map` class template. Our `ConstDataMatrixMap` universal input template is parameterized by the scalar

numerical type (typically `float` or `double`) and the point dimensionality (integer value), while dynamic (runtime) dimensionality settings are also supported. The resulting matrix view contains one data point per column. `ConstDataMatrixMap` is constructible from many common point set representations, such as `Eigen` matrix variants, STL vectors of fixed-size vector/array objects, raw data pointers, etc. The only requirement is that the mapped data should be contiguously stored in memory, in column-major order. This mechanism is transparent to the user; as a result, users can in most cases directly pass their data to `cilantro` functions, without the need for type casts or data copying.

Generic algorithms. Almost all of the algorithms implemented in `cilantro` operate in arbitrary dimension, according to the nature of the input data. Furthermore, data dimensionality can be either known at compile time or determined and set dynamically. To accommodate this feature, the library is fully templated, with the exception of visualization and some of the I/O functions. At the most basic level, template parameters typically include the input data numerical type and dimensionality. More complex algorithms are adaptable to the user's needs, by being parameterized by the entities responsible for the simpler tasks involved. For example, our ICP base class only implements an interface for the top-level EM-style iterations of the algorithm and is parameterized by a correspondence search mechanism and an estimator entity. This way, we can easily generate ICP variants for different metrics (e.g., point-to-point or point-to-plane) and different correspondence search engines (e.g., kd-tree based or projective), while maintaining a common, general interface. For performance reasons, we have opted to implement our "modular"

high-level algorithms as *static* interfaces, by making use of the Curiously Recurring Template Pattern (CRTP) idiom.

Ease of use. Generality should not come at the cost of usability. For this reason, we tackle template parameter verbosity by providing type aliases and convenience functions for the most standard variants of our algorithms. We also increase `cilantro`'s expressiveness by enabling method chaining in almost all of our classes. Furthermore, the library code is maintained in a clean, consistent style, with class and function names that are descriptive of the underlying functionality. The following code snippet showcases the library usage in a very simple pipeline that involves reading a 3D point cloud from a file, downsampling it by means of a voxel grid, estimating its surface normals, and saving the result to a file:

```
#include <cilantro/point_cloud.hpp>

int main(int argc, char ** argv) {
    cilantro::PointCloud3f(argv[1]).gridDownsample(0.005f).
       estimateNormalsRadius(0.02f).toPLYFile(argv[2]);
    return 0;
}
```

The code is very concise, with minimal boilerplate. Readers familiar with PCL will reckon that an equivalent PCL implementation would require a significantly larger amount of code.

Performance. The library is highly optimized and makes use of OpenMP parallelization for computationally demanding operations. Significant effort has been put into benchmarking alternative implementations of common operations in order to find the fastest variant and/or parallelization pattern. As a result, `cilantro`

exhibits the lowest running times in the benchmarks of Section 5.4.

Minimal dependencies. `cilantro` was built upon a carefully selected set of third party libraries and has minimal external dependencies. The library comes bundled with: `nanoflann` [111] for fast kd-tree queries, Spectra [112], an ARPACK-inspired library for large scale eigendecompositions, Qhull [113] for convex hull and halfspace intersection computations, and `tinyply` [114] for PLY format geometry I/O. The only external dependencies are:

- `Eigen` [110], an elegant and efficient linear algebra library on which we rely for most of our numerical operations, and
- Pangolin [115], a lightweight OpenGL viewport manager and image/video I/O abstraction library, on which our visualization modules were built.

`cilantro` uses the CMake build system and can be compiled with all major C++ toolchains (GCC, Clang).

5.3 Functionality

For most of our supported functionality, we have adopted an object-oriented approach, implementing each of our algorithms as a class template, while also providing free functions for simpler operations. The library components can be conceptually divided in the following categories.

Core operations. We support a wide set of primitive point cloud operations by implementing:

- An optimized, user-friendly `KDTree` template that supports general dimension

k-NN, radius, and hybrid queries, under all of the distance metrics supported by `nanoflann`.

- Arbitrary dimension surface normal and curvature estimation.
- Principal Component Analysis (PCA).
- A generic, optimized `GridAccumulator` template and its appropriate instantiations for general dimension grid-based point cloud downsampling.
- I/O functions for general matrices and 3D point clouds.
- Utility functions for RGBD to/from 3D point cloud conversions.

All the above algorithms operate on "raw", `ConstDataMatrixMap`-wrapped data. To facilitate common workflows, we also provide a convenience `PointCloud` class template that encapsulates point coordinates, normals, and colors, and provides basic point selection functionality, as well as a large number of helper methods for normal estimation, downsampling, and geometric transformations. In the 3D case, `PointCloud` instances are directly constructible from PLY files and RGBD image pairs.

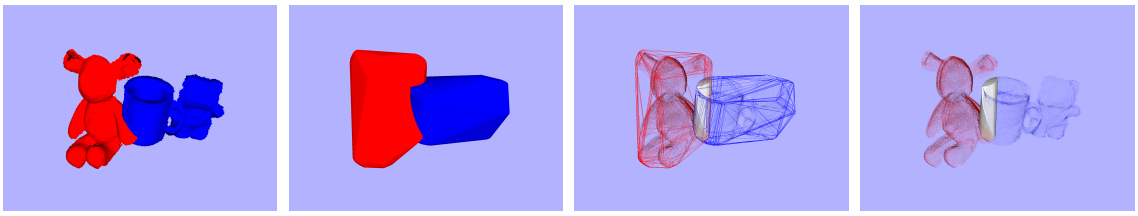


Figure 5.1: Convex hulls of two object scans and their intersection.

Spatial reasoning. Building on Qhull's facilities and a simple feasibility solver, we provide a `ConvexPolytope` template that is constructed as either the convex hull of an input point set or the halfspace intersection defined by an in-

put set of linear inequality constraints, enabling seamless representation switching. Our `SpaceRegion` class represents arbitrary space regions as unions of convex polytopes. Both space representations implement set operations (intersection for `ConvexPolytope` (Fig. 5.1), all common ones for `SpaceRegion`), interior point tests, volume calculation, and can be transformed geometrically.

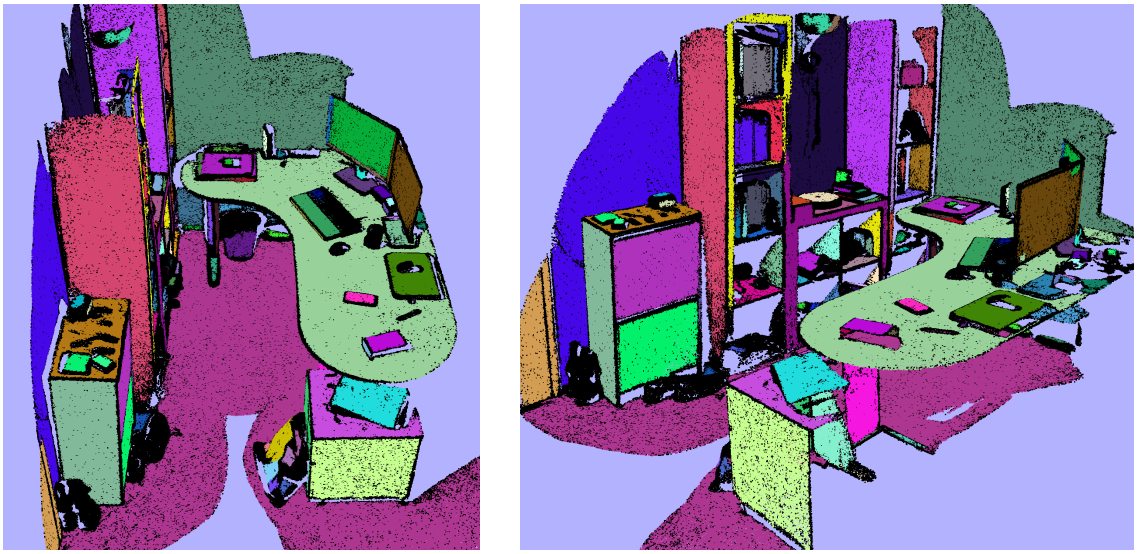


Figure 5.2: Scene segmentation into flat surfaces.

Clustering/segmentation. `cilantro` provides four standard clustering algorithms: a parallel, optimized k-means implementation that can optionally use kd-trees for large numbers of clusters, three common spectral clustering variants [97] that rely on Spectra, an arbitrary kernel mean-shift implementation, and a parallelized, generic connected component segmentation algorithm that supports arbitrary point-wise similarity functions. We note that common segmentation tasks such as extracting Euclidean (see PCL's `EuclideanClusterExtraction`) or smooth (see PCL's `RegionGrowing`) segments can be straightforwardly cast as connected component segmentation under different similarity metrics. A sample result of smooth

segment extraction is shown in Fig. 5.2, where using a very small normal angle similarity threshold effectively results in the segmentation of the scene into flat surfaces.

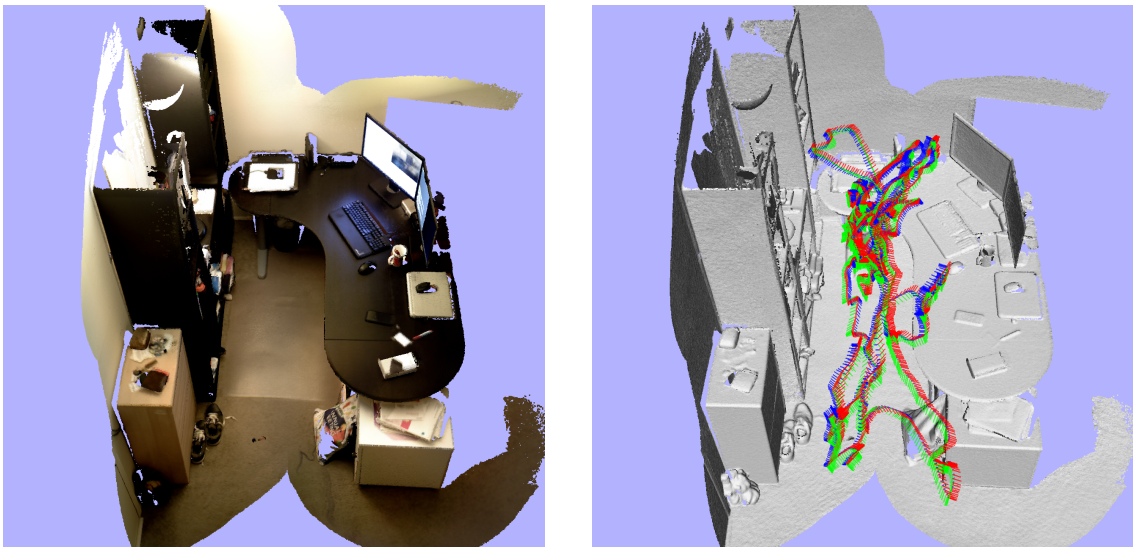


Figure 5.3: A reconstruction obtained in real time using an RGBD camera. Left: colored 3D point cloud model. Right: path of camera poses for all fused frames.

Iterative geometric registration. We provide a CRTP base class template that implements the top-level loop logic (alternating between correspondence estimation and transformation parameter optimization) of the Iterative Closest Point (ICP) algorithm. The base template is parameterized by the correspondence estimation mechanism and the transform estimator. We implement a standard, kd-tree based correspondence search engine, which can operate on arbitrary point feature spaces, as well as one for correspondences by projective data association for the 3D case. We provide optimizers for *rigid* [116, 117], *affine*, as well as *non-rigid* (by means of a robustly regularized, *locally-rigid* [66] or *locally-affine* [64], densely or sparsely [65] supported warp field) pairwise registration under the the point-to-

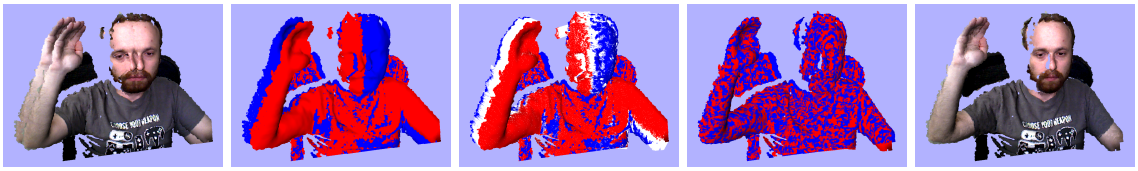


Figure 5.4: Non-rigid point cloud registration. First two images: overlaid source (red) and target (blue) geometries. Middle image: computed per-point displacements (white lines). Last two images: warped source geometry overlaid on the target one.

point and point-to-plane metrics or weighted combinations thereof. Our modular design can be easily extended to accommodate less common ICP processes (e.g., for multi-way registration). At the same time, `cilantro` provides useful shortcut wrappers/definitions for the more common registration workflows. In Fig. 5.3, we depict a point cloud reconstruction from RGBD video, obtained in real time on CPU using `cilantro`'s model-to-frame registration via projective point-to-plane ICP and a point-based fusion approach similar to that of [118]. In Fig. 5.4, we show a non-rigid surface registration result obtained using one of the library's non-rigid ICP implementations.

Robust estimation. The library comes with a RANSAC template that is meant to be used as a CRTP base class and only implements the top-level (random sampling/inlier evaluation) loop logic. We currently provide two general dimension RANSAC estimator instances: one for robust (hyper)plane fitting and one for rigid pairwise point cloud alignment given (noisy) point correspondences.

Visualization. `cilantro` implements a fully customizable, interactive, and easy to use 3D `Visualizer` class that supports: an extensible set of renderable entities with adjustable rendering options, complete control over all projection parameters, both perspective and orthographic projection modes, image capturing and

video recording of the viewport, and custom keyboard callbacks. We also provide a convenience `ImageViewer` class. Both our visualization facilities are based on Pangolin. To facilitate the visualization of data of potentially unknown dimension, for which we are only given pairwise distances, we have included a classical Multidimensional Scaling (MDS) implementation that can be used to compute low dimensional, distance-preserving Euclidean embeddings.

5.4 Performance

We compare `cilantro`'s performance in common 3D point cloud processing tasks against PCL and Open3D. Our benchmarking was done on a standard Linux (Ubuntu 16.04 based) desktop machine, equipped with an Intel® Core i7-6700K CPU (4 cores, 8 threads). All three libraries were compiled from source in 'Release' configuration, with compiler (GCC 5) optimizations set to the highest level (`-O3`).

Our first test involves the segmentation of an apartment-scale 3D reconstruction into smooth segments. We compare PCL's `RegionGrowing` with `cilantro`'s `ConnectedComponentSegmentation`, in an essentially equivalent configuration. We use the `apt0` model (Fig. 5.5, left) as input, available on the BundleFusion [119] project website, which consists of roughly 7.8 million points. After downsampling the input using a voxel grid of bin size equal to 5mm, using kNN neighborhoods with $k = 30$ and a normal angle threshold of 2.8 radians, `cilantro` produces the result of Fig. 5.5 (right) in 3.65 seconds, while, for the same parameters, the PCL implementation took 17.34 seconds. Open3D offers no equivalent/similar function-

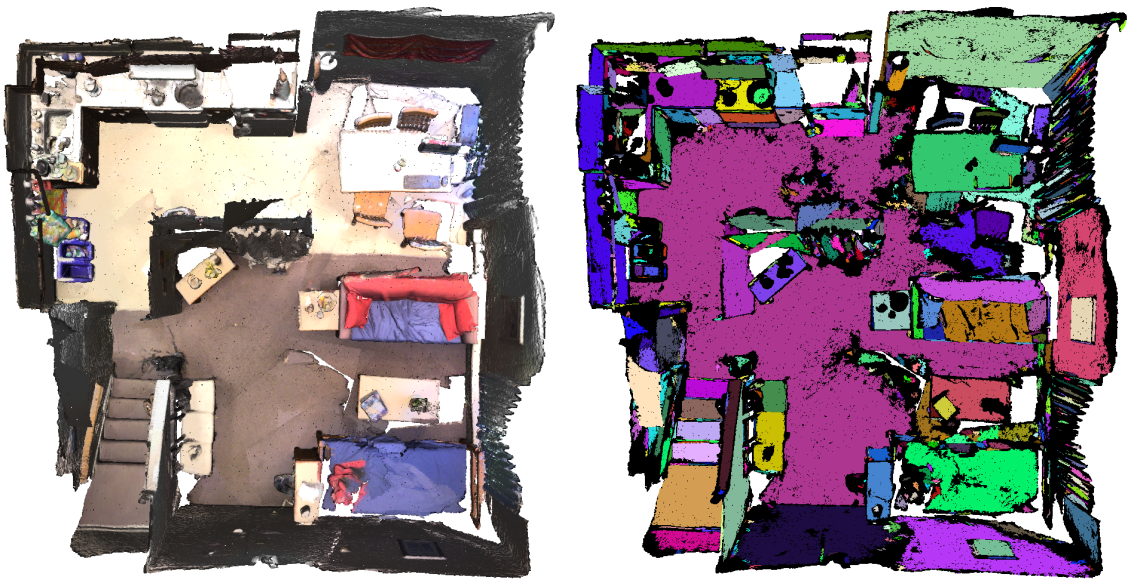


Figure 5.5: Point cloud segmentation into smooth segments. Black points correspond to very small clusters that were discarded.

ality.

PCL's standard point types use single precision floating point numbers, while Open3D works only with double precision. For all subsequent computations, we use double precision for `cilantro`.

In our next test, we compare our normal estimation performance against equivalent implementations in Open3D (parallelized by default) and PCL (the multi-threaded `NormalEstimationOMP` variant). We use the `fr1/desk` sequence of the TUM RGBD dataset [120] as input (about 600 pre-registered RGBD image pairs at VGA resolution). We run all three implementations on all input frames at full resolution, using two types of local neighborhoods: one defined by the 10 nearest neighbors and one defined by a radius of 1cm. As can be seen in the two top rows of Fig. 5.6, `cilantro` consistently outperforms the other two implementations. In the kNN case in particular, it exhibits speedups of 1.58× and 1.85× over Open3D and

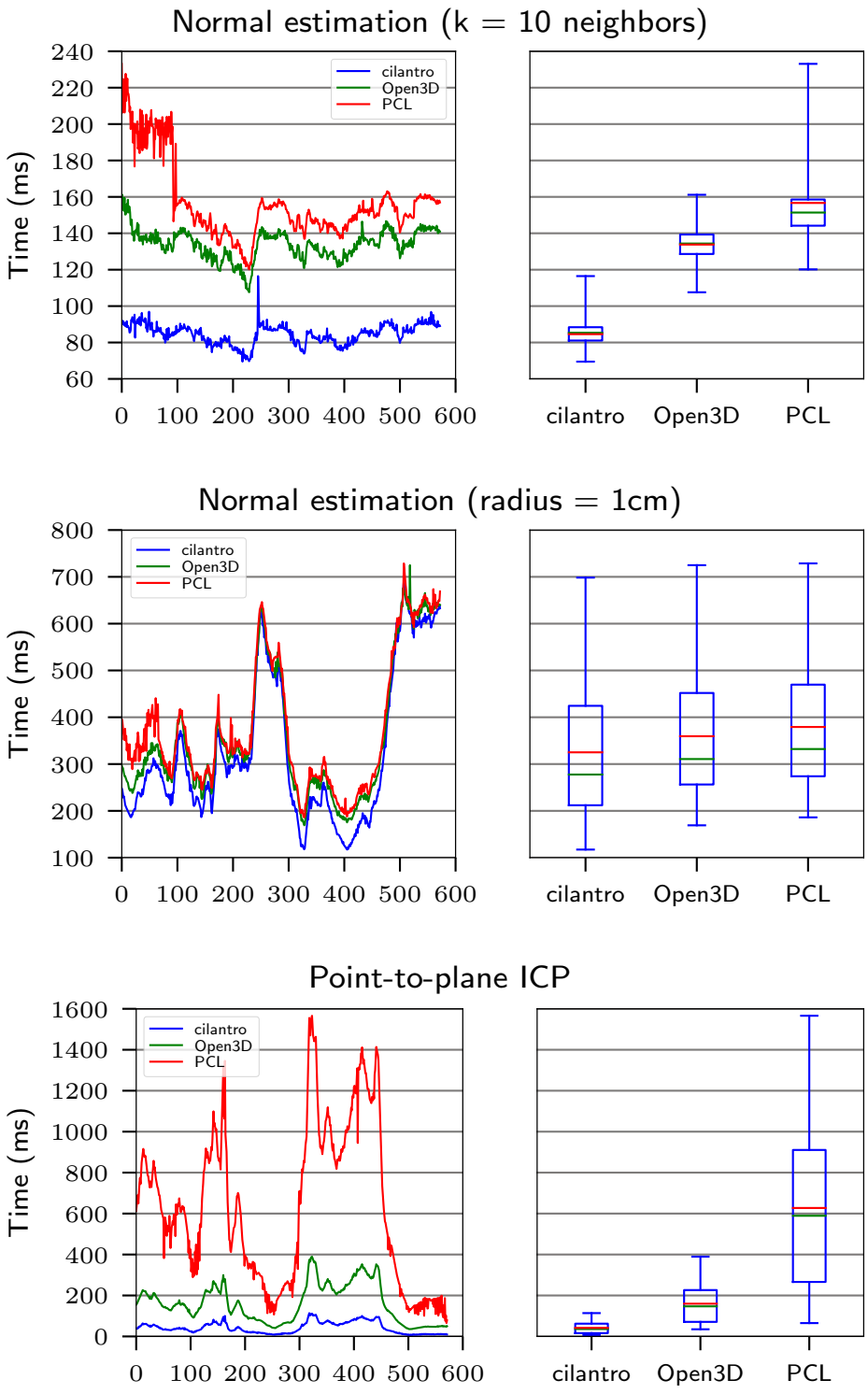


Figure 5.6: Performance comparisons against PCL and Open3D in common operations. Left column: running time as a function of the input video frame. Right column: box and whisker plots of running times per library (red lines are means, green lines are medians).

PCL, respectively. We note that `cilantro` and PCL enforce viewpoint-consistent normals during estimation, while in Open3D this has to be done in a separate pass, which was not performed in our experiments.

In our last test, we compare point-to-plane ICP performance among the three libraries. We used the same `fr1/desk` sequence as before, aligning each frame to its previous one. All point clouds were downsampled using a voxel grid of bin size equal to 1cm, and normals were estimated based on 10 nearest neighbor local neighborhoods. To account for different termination criteria conventions, we forced all three implementations to perform a fixed number of 15 iterations. In all three cases, correspondences were established by means of nearest neighbor kd-tree queries, with a rejection distance threshold of 5cm. Registration times are reported in the third row of Fig. 5.6. We note that, at all times, all three implementations converged to the same transformation, up to numerical precision. `cilantro`'s ICP implementation outperforms the other two by a significant margin, demonstrating speedups of 3.82× and 14.99× over Open3D and PCL, respectively.

5.5 Software release

`cilantro` is released under the MIT license and its source code is openly available at `https://github.com/kzampog/cilantro`. Contributions from the open-source community are welcome, via the GitHub issues/pull request mechanisms.

The library is under active, continuous development, undergoing frequent API improvements, functionality additions, and performance optimizations. Future ad-

ditions will include structures and algorithms for surface merging/reconstruction, more point cloud resampling strategies, and support for geometry I/O in other file formats. We are also investigating GPU implementations for some of our algorithms, focusing on improving our non-rigid registration performance.

Chapter 6: Conclusions and Future Work

6.1 Summary

In this thesis, we focused on modeling and extracting certain types of geometric object interactions in RGBD videos of manipulations, with the goal of providing useful semantic cues for cognitive action interpretation. We focused on two different types of interactions.

At a higher level, in Chapter 2, we directly modeled pairwise spatial relations based on their intuitive geometric interpretation and proposed a novel abstract action representation that encodes the geometric object interactions during action execution, as captured by the spatial relation dynamics. Our experiments on a diverse set of inputs confirm both the validity and effectiveness of our spatial relation models and the discriminative power of our representation with respect to the underlying action semantics.

At a lower level, in Chapter 3, we defined a method for the bottom-up detection of object contacts and separations, with the ultimate goal of discontinuity-preserving non-rigid registration. In addition to state-of-the-art motion estimation accuracy and measurable improvements over existing approaches in motion estimates on topological event boundaries, our approach shows promising performance

in the explicit detection of contact and separation events.

Furthermore, in Chapter 4, we presented a simple bottom-up pipeline that builds upon our topology-aware dense motion estimation approach and uses detected contact interactions as an attention mechanism in order to obtain a motion-based segmentation of the manipulated object and track its motion throughout the duration of an observed manipulation.

The ability to extract and reason about these types of geometric object interactions provides useful semantic cues for the cognitive interpretation of manipulation actions, as it either fully enables or significantly facilitates answering a number of fundamental questions about the action at hand. For example, contact detection in a manipulation video provides strong hints to *what* was grasped and *when*; combined with the guided motion segmentation of Chapter 4, both of these questions, as well as *how* the object moved, can be fully answered. At the same time, spatial relations and their dynamics reveal *where* the manipulated object was moved with respect to its surroundings.

In addition to our methodological contributions, in Chapter 5, we presented our publicly released open-source software library for generic point cloud data processing that includes implementations of the core functionality required by the components of this thesis.

6.2 Future work

There are several paths for improvements and extensions of our work.

The abstract, spatial relation based action representation that we introduced in Chapter 2 is currently equipped with a distance measure that requires temporally complete observations. Given the compactness of our descriptor, an *online* action matching mechanism, applicable also to partial manipulation observations, would provide the means for efficient action *prediction*, which would be particularly useful in quick correctness assessments of monitored executions.

The topology-aware non-rigid registration algorithm of Chapter 3 only considers the cases of contacts and separations. However, there exist interactions, such as that of an object sliding on a surface, which induce different types of topological changes that our method does not handle. Investigating ways of reasoning about *other types of interactions* would not only be beneficial for improving dense motion estimation, but also possibly enable the explicit detection and classification of said interactions, providing additional semantic cues for action interpretation.

Finally, the guided motion segmentation pipeline of Chapter 4 can be adapted to also integrate *dynamic reconstruction*, enabling the extraction of a more complete model for the manipulated object. Other possible extensions include the ability to extract handled objects that are *non-rigid* (e.g., articulated) or *indirectly* manipulated (e.g., via the use of tools).

Bibliography

[1] P. Turaga, R. Chellappa, V.S. Subrahmanian, and O. Udrea. Machine recognition of human activities: A survey. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(11):1473–1488, 2008.

[2] Ronald Poppe. A survey on vision-based human action recognition. *Image and vision computing*, 28(6):976–990, 2010.

[3] Daniel Weinland, Remi Ronfard, and Edmond Boyer. A survey of vision-based methods for action representation, segmentation and recognition. *Computer vision and image understanding*, 115(2):224–241, 2011.

[4] Samitha Herath, Mehrtash Harandi, and Fatih Porikli. Going deeper into action recognition: A survey. *Image and vision computing*, 60:4–21, 2017.

[5] Volker Krüger, Danica Kragic, Aleš Ude, and Christopher Geib. The meaning of action: A review on action recognition and mapping. *Advanced robotics*, 21(13):1473–1501, 2007.

[6] Yezhou Yang, Anupam Guha, C Fermüller, and Yiannis Aloimonos. A cognitive system for understanding human manipulation actions. *Advances in Cognitive Sysytems*, 3:67–86, 2014.

[7] Yezhou Yang, Yi Li, Cornelia Fermüller, and Yiannis Aloimonos. Robot learning manipulation action plans by "watching" unconstrained videos from the world wide web. In *AAAI*, pages 3686–3693, 2015.

[8] Abhinav Gupta, Aniruddha Kembhavi, and Larry S Davis. Observing human-object interactions: Using spatial and functional compatibility for recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(10):1775–1789, 2009.

[9] Vincent Delaitre, Josef Sivic, and Ivan Laptev. Learning person-object interactions for action recognition in still images. In *Advances in neural information processing systems*, pages 1503–1511, 2011.

[10] E.E. Aksoy, A. Abramov, J. Dörr, K. Ning, B. Dellen, and F. Wörgötter. Learning the semantics of object-action relations by observation. *The International Journal of Robotics Research*, 30(10):1229–1249, 2011.

[11] Fatemeh Ziaeetabar, Eren Erdal Aksoy, Florentin Wörgötter, and Minija Tamosiunaite. Semantic analysis of manipulation actions using spatial relations. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4612–4619. IEEE, 2017.

[12] Yezhou Yang. *Manipulation Action Understanding for Observation and Execution*. PhD thesis, University of Maryland, College Park, 2015.

[13] Garrett Ethan Katz. *A Cognitive Robotic Imitation Learning System Based on Cause-Effect Reasoning*. PhD thesis, University of Maryland, College Park, 2017.

[14] Konstantinos Zampogiannis, Yezhou Yang, Cornelia Fermüller, and Yiannis Aloimonos. Learning the spatial semantics of manipulation actions through preposition grounding. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 1389–1396. IEEE, 2015.

[15] Konstantinos Zampogiannis, Cornelia Fermuller, and Yiannis Aloimonos. Topology-aware non-rigid point cloud registration. *arXiv preprint arXiv:1811.07014*, 2018.

[16] Konstantinos Zampogiannis, Kanishka Ganguly, Cornelia Fermuller, and Yiannis Aloimonos. Extracting contact and motion from manipulation videos. *arXiv preprint arXiv:1807.04870*, 2018.

[17] Konstantinos Zampogiannis, Cornelia Fermuller, and Yiannis Aloimonos. Cilantro: A lean, versatile, and efficient library for point cloud data processing. In *Proceedings of the 26th ACM International Conference on Multimedia*, MM '18, pages 1364–1367, New York, NY, USA, 2018. ACM.

[18] Weilie Yi and Dana Ballard. Recognizing behavior in hand-eye coordination patterns. *International Journal of Humanoid Robotics*, 6(03):337–359, 2009.

[19] H. Kjellström, J. Romero, D. Martínez, and D. Kragić. Simultaneous visual recognition of manipulation actions and manipulated objects. *Proceedings of the 2008 IEEE European Conference on Computer Vision*, pages 336–349, 2008.

[20] A. Gupta and L.S. Davis. Objects in action: An approach for combining action understanding and object perception. In *Proceedings of the 2007 IEEE International Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.

[21] Tamim Asfour, Pedram Azad, Florian Gyarfas, and Rüdiger Dillmann. Imitation learning of dual-arm manipulation tasks in humanoid robots. *International Journal of Humanoid Robotics*, 5(02):183–202, 2008.

[22] T. Asfour, P. Azad, N. Vahrenkamp, K. Regenstein, A. Bierbaum, K. Welke, J. Schrder, and R. Dillmann. Toward humanoid manipulation in human-centred environments. *Robotics and Autonomous Systems*, 56:54–65, 2008.

[23] Anupam Guha, Yezhou Yang, Cornelia Fermüller, and Yiannis Aloimonos. Minimalist plans for interpreting manipulation actions. In *Proceedings of the 2013 International Conference on Intelligent Robots and Systems*, pages 5908–5914, Tokyo, 2013. IEEE.

[24] Yezhou Yang, Cornelia Fermüller, and Yiannis Aloimonos. Detection of manipulation action consequences (MAC). In *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2563–2570, Portland, OR, 2013. IEEE.

[25] Mirko Wächter, Sebastian Schulz, Tamim Asfour, Eren Aksoy, Florentin Wörgötter, and Rüdiger Dillmann. Action sequence reproduction based on automatic segmentation and object-action complexes. In *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, pages 0–0, 2013.

[26] Jeremy Maitin-Shepard, Marco Cusumano-Towner, Jinna Lei, and Pieter Abbeel. Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. In *Proceedings of the 2010 IEEE International Conference on Robotics and Automation*, pages 2308–2315. IEEE, 2010.

[27] Bogdan Moldovan, Plinio Moreno, Martijn van Otterlo, José Santos-Victor, and Luc De Raedt. Learning relational affordance models for robots in multi-object manipulation tasks. In *Proceedings of the 2010 IEEE International Conference on Robotics and Automation*, pages 4373–4378. IEEE, 2012.

[28] Mehmet R Dogar, Michael C Koval, Abhijeet Tallavajhula, and Siddhartha Srinivasa. Object search by manipulation. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013.

[29] Christian Smith, Yiannis Karayiannidis, Lazaros Nalpantidis, Xavi Gratal, Peng Qi, Dimos V Dimarogonas, and Danica Kragic. Dual arm manipulation: A survey. *Robotics and Autonomous Systems*, 2012.

[30] Abhinav Gupta and Larry S. Davis. Beyond nouns: Exploiting prepositions and comparative adjectives for learning visual classifiers. In David A. Forsyth, Philip H. S. Torr, and Andrew Zisserman, editors, *ECCV (1)*, volume 5302 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2008.

[31] S. Guadarrama, L. Riano, D. Golland, D. Gohring, Y. Jia, D. Klein, P. Abbeel, and T. Darrell. Grounding spatial relations for human-robot interaction. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, November 2013.

[32] Dave Golland, Percy Liang, and Dan Klein. A game-theoretic approach to generating spatial descriptions. In *Proceedings of the 2010 conference on empirical methods in natural language processing*, pages 410–419. Association for Computational Linguistics, 2010.

[33] Benjamin Rosman and Subramanian Ramamoorthy. Learning spatial relationships between objects. *I. J. Robotic Res.*, 30(11):1328–1342, 2011.

[34] S. Fichtl, A. McManus, W. Mustafa, D. Kraft, N. Kruger, and F. Guerin. Learning spatial relationships from 3D vision using histograms. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 501–508, May 2014.

[35] Dieter Fox. KLD-Sampling: Adaptive Particle Filters. In *Advances in Neural Information Processing Systems 14 (NIPS 2001)*, pages 713–720. MIT Press, 2001.

[36] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[37] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 26(1):43–49, Feb 1978.

[38] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[39] Eliane Maria Loiola, Nair Maria Maia de Abreu, Paulo Oswaldo Boaventura-Netto, Peter Hahn, and Tania Querido. A survey for the quadratic assignment problem. *European Journal of Operational Research*, 176(2):657–690, 2007.

[40] Tobias Achterberg. SCIP: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, July 2009.

[41] Auke Jan Ijspeert, Jun Nakanishi, and Stefan Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 2, pages 1398–1403. IEEE, 2002.

[42] Brendan J. Frey and Delbert Dueck. Clustering by passing messages between data points. *Science*, 315:972–976, 2007.

[43] Albert S Huang, Abraham Bachrach, Peter Henry, Michael Krainin, Daniel Maturana, Dieter Fox, and Nicholas Roy. Visual odometry and mapping for autonomous flight using an rgb-d camera. In *Robotics Research*, pages 235–252. Springer, 2017.

[44] Christian Kerl, Jürgen Sturm, and Daniel Cremers. Robust odometry estimation for rgb-d cameras. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 3748–3754. IEEE, 2013.

[45] Mariano Jaimez, Mohamed Souiai, Javier Gonzalez-Jimenez, and Daniel Cremers. A primal-dual framework for real-time dense rgb-d scene flow. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 98–104. IEEE, 2015.

[46] Mariano Jaimez, Christian Kerl, Javier Gonzalez-Jimenez, and Daniel Cremers. Fast odometry and scene flow from rgb-d cameras based on geometric clustering. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3992–3999. IEEE, 2017.

[47] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pages 127–136. IEEE, 2011.

[48] Thomas Whelan, Stefan Leutenegger, Renato Salas Moreno, Ben Glocker, and Andrew Davison. Elasticfusion: Dense slam without a pose graph. In *Proceedings of Robotics: Science and Systems*, Rome, Italy, July 2015.

[49] Richard A Newcombe, Dieter Fox, and Steven M Seitz. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 343–352, 2015.

[50] Matthias Innmann, Michael Zollhöfer, Matthias Nießner, Christian Theobalt, and Marc Stamminger. Volumedeform: Real-time volumetric non-rigid reconstruction. In *European Conference on Computer Vision*, pages 362–379. Springer, 2016.

[51] Evan Herbst, Xiaofeng Ren, and Dieter Fox. Rgb-d flow: Dense 3-d motion estimation using color and depth. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 2276–2282. IEEE, 2013.

[52] Julian Quiroga, Thomas Brox, Frédéric Devernay, and James Crowley. Dense semi-rigid scene flow estimation from rgbd images. In *European Conference on Computer Vision*, pages 567–582. Springer, 2014.

[53] Mariano Jaimez, Mohamed Souiai, Jörg Stückler, Javier Gonzalez-Jimenez, and Daniel Cremers. Motion cooperation: Smooth piece-wise rigid scene flow from rgb-d images. In *3D Vision (3DV), 2015 International Conference on*, pages 64–72. IEEE, 2015.

[54] Nikolaus Mayer, Eddy Ilg, Philip Hausser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4040–4048, 2016.

[55] Mingsong Dou, Sameh Khamis, Yury Degtyarev, Philip Davidson, Sean Ryan Fanello, Adarsh Kowdle, Sergio Orts Escolano, Christoph Rhemann, David Kim, Jonathan Taylor, Pushmeet Kohli, Vladimir Tankovich, and Shahram Izadi. Fusion4d: Real-time performance capture of challenging scenes. *ACM Trans. Graph.*, 35(4):114:1–114:13, July 2016.

[56] Wei Gao and Russ Tedrake. Surfelwarp: Efficient non-volumetric single view dynamic reconstruction. In *Robotics: Science and Systems*, 2018.

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

[58] Miroslava Slavcheva, Maximilian Baust, and Slobodan Ilic. Sobolevfusion: 3d reconstruction of scenes undergoing free non-rigid motion. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2646–2655, 2018.

[59] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312. ACM, 1996.

[60] Sundar Vedula, Simon Baker, Peter Rander, Robert Collins, and Takeo Kanade. Three-dimensional scene flow. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 722–729. IEEE, 1999.

[61] Hagen Spies, Bernd Jähne, and John L Barron. Range flow estimation. *Computer Vision and Image Understanding*, 85(3):209–231, 2002.

[62] Michael Hornacek, Andrew Fitzgibbon, and Carsten Rother. Sphereflow: 6 dof scene flow from rgb-d pairs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3526–3533, 2014.

[63] Deqing Sun, Erik B Sudderth, and Hanspeter Pfister. Layered rgbd scene flow estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 548–556, 2015.

[64] Brian Amberg, Sami Romdhani, and Thomas Vetter. Optimal step nonrigid icp algorithms for surface registration. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.

[65] Robert W. Sumner, Johannes Schmid, and Mark Pauly. Embedded deformation for shape manipulation. *ACM Transactions on Graphics*, 26:80, 2007.

[66] Olga Sorkine and Marc Alexa. As-rigid-as-possible surface modeling. In *Symposium on Geometry processing*, volume 4, page 30, 2007.

[67] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[68] Martin Rünz and Lourdes Agapito. Co-fusion: Real-time segmentation, tracking and fusion of multiple objects. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4471–4478, May 2017.

[69] Martin Rünz, Maud Buffier, and Lourdes Agapito. Maskfusion: Real-time recognition, tracking and reconstruction of multiple moving objects. In *2018 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 10–20. IEEE, 2018.

[70] Miroslava Slavcheva, Wadim Kehl, Nassir Navab, and Slobodan Ilic. Sdf-2-sdf registration for real-time 3d reconstruction from rgb-d data. *International Journal of Computer Vision*, 126(6):615–636, Jun 2018.

[71] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the icp algorithm. In *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on*, pages 145–152. IEEE, 2001.

[72] Chris Engels, Henrik Stewénius, and David Nistér. Bundle adjustment rules. *Photogrammetric computer vision*, 2(2006), 2006.

[73] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. In A. Fitzgibbon et al. (Eds.), editor, *European Conf. on Computer Vision (ECCV)*, Part IV, LNCS 7577, pages 611–625. Springer-Verlag, October 2012.

[74] Simon Baker, Daniel Scharstein, JP Lewis, Stefan Roth, Michael J Black, and Richard Szeliski. A database and evaluation methodology for optical flow. *International Journal of Computer Vision*, 92(1):1–31, 2011.

[75] Ueli Rutishauser, Dirk Walther, Christof Koch, and Pietro Perona. Is bottom-up attention useful for object recognition? In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–II. IEEE, 2004.

[76] Jimmy Ba, Volodymyr Mnih, and Koray Kavukcuoglu. Multiple object recognition with visual attention. *arXiv preprint arXiv:1412.7755*, 2014.

[77] Heng Wang and Cordelia Schmid. Action recognition with improved trajectories. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 3551–3558. IEEE, 2013.

[78] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in neural information processing systems*, pages 568–576, 2014.

[79] Norbert Krüger, Christopher Geib, Justus Piater, Ronald Petrick, Mark Steedman, Florentin Wörgötter, Aleš Ude, Tamim Asfour, Dirk Kraft, Damir Omrčen, et al. Object–action complexes: Grounded abstractions of sensory–motor processes. *Robotics and Autonomous Systems*, 59(10):740–757, 2011.

[80] Karinne Ramirez Amaro, Michael Beetz, and Gordon Cheng. Understanding human activities from observation via semantic reasoning for humanoid robots. In *IROS Workshop on AI and Robotics*, 2014.

[81] Douglas Summers-Stay, Ching L Teo, Yezhou Yang, Cornelia Fermüller, and Yiannis Aloimonos. Using a minimal action grammar for activity understanding in the real world. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4104–4111. IEEE, 2012.

[82] Yezhou Yang, Cornelia Fermuller, Yi Li, and Yiannis Aloimonos. Grasp type revisited: A modern perspective on a classical feature for vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 400–408, 2015.

[83] Qingshuang Chen, He Li, Rana Abu-Zhaya, Amanda Seidl, Fengqing Zhu, and Edward J Delp. Touch event recognition for human interaction. *Electronic Imaging*, 2016(11):1–6, 2016.

[84] Jingyu Yan and Marc Pollefeys. A general framework for motion segmentation: Independent, articulated, rigid, non-rigid, degenerate and non-degenerate. In *European conference on computer vision*, pages 94–106. Springer, 2006.

[85] Roberto Tron and René Vidal. A benchmark for the comparison of 3-d motion segmentation algorithms. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.

[86] Joao Costeira and Takeo Kanade. A multi-body factorization method for motion analysis. In *Computer Vision, 1995. Proceedings., Fifth International Conference on*, pages 1071–1076. IEEE, 1995.

[87] Ken-ichi Kanatani. Motion segmentation by subspace separation and model selection. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 2, pages 586–591. IEEE, 2001.

[88] Shankar Rao, Roberto Tron, Rene Vidal, and Yi Ma. Motion segmentation in the presence of outlying, incomplete, or corrupted trajectories. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(10):1832–1845, 2010.

[89] René Vidal and Richard Hartley. Motion segmentation with missing data using powerfactorization and gpca. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–II. IEEE, 2004.

[90] Dov Katz, Moslem Kazemi, J Andrew Bagnell, and Anthony Stentz. Interactive segmentation, tracking, and kinematic modeling of unknown 3d articulated objects. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 5003–5010. IEEE, 2013.

[91] Evan Herbst, Xiaofeng Ren, and Dieter Fox. Object segmentation from motion with dense feature matching. In *ICRA Workshop on Semantic Perception, Mapping and Exploration*, volume 2, 2012.

[92] Peter Ochs, Jitendra Malik, and Thomas Brox. Segmentation of moving objects by long term video analysis. *IEEE transactions on pattern analysis and machine intelligence*, 36(6):1187–1200, 2014.

[93] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *CVPR*, 2017.

[94] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[95] Michael J Jones and James M Rehg. Statistical color models with application to skin detection. *International Journal of Computer Vision*, 46(1):81–96, 2002.

[96] Vladimir Vezhnevets, Vassili Sazonov, and Alla Andreeva. A survey on pixel-based skin color detection techniques. In *Proc. Graphicon*, volume 3, pages 85–92. Moscow, Russia, 2003.

[97] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[98] Shinji Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on pattern analysis and machine intelligence*, 13(4):376–380, 1991.

[99] Dave Hershberger, David Gossow, and Josh Faust. RViz. `https://github.com/ros-visualization/rviz`, 2012.

[100] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[101] Dave T. Coleman. "MoveIt!" Simple Grasps. `https://github.com/davetcoleman/moveit_simple_grasps`, 2016.

[102] S. Chitta, I. Sucan, and S. Cousins. MoveIt! [ROS Topics]. *IEEE Robotics Automation Magazine*, 19(1):18–19, March 2012.

[103] Steven M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, Iowa State University, 1998.

[104] Stefan Schaal. Dynamic movement primitives - a framework for motor control in humans and humanoid robotics, 2002.

[105] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 4.12 edition, 2018.

[106] Visual Computing Lab of the Italian National Research Council - ISTI. The vcg library, 2018. http://www.vcglib.net/.

[107] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. Meshlab: an open-source mesh processing tool. In *Eurographics Italian Chapter Conference*, volume 2008, pages 129–136, 2008.

[108] Alec Jacobson, Daniele Panozzo, et al. libigl: A simple C++ geometry processing library, 2017. http://libigl.github.io/libigl/.

[109] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.

[110] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[111] Jose Luis Blanco and Pranjal Kumar Rai. nanoflann: a C++ header-only fork of FLANN, a library for nearest neighbor (NN) wih kd-trees. `https://github.com/jlblancoc/nanoflann`, 2014.

[112] Yixuan Qiu. Spectra: A header-only c++ library for large scale eigenvalue problems. `https://spectralib.org/index.html`, 2018.

[113] C Bradford Barber, David P Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software (TOMS)*, 22(4):469–483, 1996.

[114] Dimitri Diakopoulos. tinyply. `https://github.com/ddiakopoulos/tinyply`, 2018.

[115] Steven Lovegrove. Pangolin. `https://github.com/stevenlovegrove/Pangolin`, 2018.

[116] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Sensor Fusion IV: Control Paradigms and Data Structures*, volume 1611, pages 586–607. International Society for Optics and Photonics, 1992.

[117] Kok-Lim Low. Linear least-squares optimization for point-to-plane icp surface registration. *Chapel Hill, University of North Carolina*, 4, 2004.

[118] Maik Keller, Damien Lefloch, Martin Lambers, Shahram Izadi, Tim Weyrich, and Andreas Kolb. Real-time 3d reconstruction in dynamic scenes using point-based fusion. In *2013 International Conference on 3D Vision-3DV 2013*, pages 1–8. IEEE, 2013.

[119] Angela Dai, Matthias Nießner, Michael Zollöfer, Shahram Izadi, and Christian Theobalt. Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface re-integration. *ACM Transactions on Graphics 2017 (TOG)*, 2017.

[120] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.