

Semantic Scene Understanding for Prediction of Action Effects in Humanoid Robot Manipulation Tasks

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik des
Karlsruher Instituts für Technologie (KIT)

genehmigte
Dissertation

von

M. Sc. Fabian Paus

aus Coesfeld

Tag der mündlichen Prüfung: 17. Mai 2022
Referent: Prof. Dr.-Ing. Tamim Asfour
Korreferent: Prof. Danica Kragic Jensfelt



This document is licensed under a Creative Commons Attribution 4.0 International License
(CC BY 4.0): <https://creativecommons.org/licenses/by/4.0/deed.en>

Deutsche Zusammenfassung

Menschen sind in der Lage, komplexe Szenen mit mehreren übereinander liegenden Objekten intuitiv zu verstehen und zu handhaben. Das semantische Verständnis der Szene, d. h. deren Zerlegung in die enthaltenen Objekte und die Analyse ihrer 3D-Struktur, des Szenenlayouts und der Beziehungen zwischen Objekten ermöglicht es dem Menschen, gelernte Konzepte auf zuvor ungesene Szenen mit einer unterschiedlichen Anzahl von Objekten und Beziehungen zu übertragen. Dieses Konzept aus den Neurowissenschaften wird kombinatorische Generalisierung genannt und erlaubt Menschen, ihr Verständnis über Objekte und Beziehungen bzw. Relationen zwischen diesen zu kombinieren, um neue Situationen konzeptionell zu erfassen. Auf Basis dieses relationalen Szeneverständnisses können Menschen effektive, intuitive physikalische Prädiktionsmodelle für Interaktionen mit der Welt erlernen und bei der Ausführung von Aufgaben verwenden. Analysiert man menschliche Vorhersagen für Objektinteraktionen, kann man feststellen, dass diese eher mit intuitiven Modellen basierend auf Objektrelationen übereinstimmen als mit genauen Physiksimulationen. Kleinkinder erwerben die Fähigkeit zum Verständnis von Objektrelationen und der Vorhersage mit intuitiven Modellen während der ersten zwei Lebensjahre. Zunächst entwickelt sich ein Verständnis von einfachen Objektrelationen wie Kontakt und Nicht-Kontakt. Über die Zeit wird diese Fähigkeit erweitert, um in komplexen Szenen zu verstehen, welche Objekte in Kontakt sind und von anderen unterstützt werden.

Ziel dieser Arbeit ist es, einen humanoiden Roboter mit einem Szenenverständnis basierend auf Objektrelationen und einem intuitiven Prädiktionsmodell auszustatten, um damit Aufgaben wie Greifen und Platzieren von Objekten in komplexen Szenen zu unterstützen. Dazu wird zunächst eine neue probabilistische Repräsentation von Objekten und deren Unterstützungsrelationen er-

arbeitet. Dann wird eine Methode zur Extraktion dieser Repräsentation aus Punktwolken, die von den Tiefenkameras des Roboters aufgenommen wurden, implementiert und evaluiert. Anschließend werden Prädiktionsmodelle für Interaktionen zwischen mehreren starren und verformbaren Objekten (am Beispiel einer Stofftasche) entwickelt. Diese Modelle können mit Szenengraphen umgehen, die eine beliebige Anzahl von Objekten in den Knoten und Objektrelationen in den Kanten darstellen können. Sowohl die Extraktion von Unterstützungsrelationen als auch die Prädiktionsmodelle werden in Experimenten mit den humanoiden Robotern ARMAR-III und ARMAR-6 validiert. Im Folgenden werden die drei Einzelbeiträge dieser Arbeit vorgestellt:

Repräsentation und Extraktion von Unterstützungsrelationen: Aus einer Punktwolke der Szene, die von der Tiefenkamera des Roboters aufgenommen wurde, sollen Objektgeometrie und Unterstützungsrelationen extrahiert werden. Dabei soll die Unsicherheit aus der Wahrnehmung der Szene sowohl in der Geometrie als auch in den Relationen repräsentiert werden. Dazu wird eine neuartige probabilistische Repräsentation entwickelt, die Objektposen, Formen und Unterstützungsrelationen umfasst. Die Objektgeometrien werden als parametrisierbare Primitive, wie z. B. Boxen, Zylinder und Kugeln repräsentiert. Die Wahrnehmungsunsicherheit wird über eine Wahrscheinlichkeitsverteilung über den Primitivtyp und die geometrischen Parameter modelliert. Für diese Repräsentation wird eine Extraktionsmethode aus Punktwolken implementiert. Zunächst wird die Wahrscheinlichkeitsverteilung über die Objektgeometrien mittels eines modifizierten RANSAC-Algorithmus (Random Sample Consensus) berechnet. Dann schätzt eine Monte-Carlo-Simulation die Wahrscheinlichkeit des Vorhandenseins einer Unterstützungsrelation durch eine Analyse der wirkenden Kräfte und der Szenengeometrie. Durch eine Analyse der Stützpolygone kann potenzielle Unterstützung von oben erkannt werden, welche von existierenden Verfahren bisher nicht betrachtet wurde. Die Extraktionsmethode wird auf zwei Datensätzen, die komplexe Szenen mit mehreren Objekten enthalten, evaluiert. Die Ergebnisse zeigen, dass sowohl die Analyse der Stützpolygone als auch die probabilistische Repräsentation die Erkennungsraten für Unterstützungsrelationen deutlich verbessern.

Graphenbasierte Prädiktion von Aktionseffekten: Aus dem aktuellen Zustand einer Szene und den Parametern für eine Aktion, die der Roboter ausführen wird, soll der Zustand der Szene nach Aktionsausführung vorhergesagt werden. Dazu werden Methoden zur Vorhersage von Aktionseffekten auf der Grundlage von neuronalen Graphnetzen (engl. *neural graph networks*) für Interaktionen zwischen mehreren starren und verformbaren Objekten entwickelt. Zunächst wird eine Methode zur Prädiktion der Effekte von Schiebeaktionen auf mehrere starre Objekte implementiert. Der Zustand der Szene wird als Graph dargestellt, in dem die Knoten die Objekteigenschaften und die Kanten die räumlichen Relationen zwischen den Objekten repräsentieren. Die Trainingsdaten wurden durch die Simulation von Aktionen in zufällig generierten Szenen erzeugt. Anschließend wurde ein neuronales Graphnetz trainiert, dessen Eingabe und Ausgabe Graphen mit einer beliebigen Anzahl von Knoten und Kanten sind. Um diese Methode auf Interaktionen mit deformierbaren Objekten anwenden zu können, wurde die Repräsentation des Szenengraphen erweitert. Dabei werden relevante Punkte auf der Oberfläche von deformierbaren Objekten ausgewählt und als Knoten im Graph aufgenommen. Darüber hinaus wurde eine zweistufige Prädiktionemethode implementiert, die zunächst die sich bewegenden Knoten klassifiziert und sie dann mithilfe eines Regressionsmodells aktualisiert, falls sie als bewegend klassifiziert wurden. Schließlich wurde ein Modell konzipiert und implementiert, das Vorhersagemodelle mit unterschiedlichen Zeitschritten kombiniert, um Vorhersagen über längere Zeithorizonte zu ermöglichen. Die Evaluation der Prädiktion von Schiebeefferkten erfolgt mit simulierten und realen Daten und zeigt, dass die Methode in der Lage ist, die Interaktionen mehrerer starrer Objekte präzise und effizient vorherzusagen. Im Gegensatz zu bestehenden datengetriebenen Methoden ist dieser Ansatz unabhängig von der Anzahl der beteiligten Objekte. Bei Interaktionen mit verformbaren Objekten werden Ablationsstudien durchgeführt, um zu zeigen, dass die zweistufige Methode sporadisch prädizierte Bewegungen reduziert, da der Klassifikator unbeteiligte Knoten herausfiltert. Es wird gezeigt, dass das Modell mit unterschiedlichen Zeitschritten die Vorhersagegenauigkeit über längere Zeithorizonte verbessert.

Anwendung auf Manipulationsaufgaben von humanoiden Robotern: Die in der Arbeit entwickelten Methoden sollen auf Manipulationsaufgaben im Kontext der humanoiden Robotik angewendet und in Experimenten validiert werden. Zunächst wird eine Methode zur Bestimmung einer sicheren Manipulationsreihenfolge implementiert, die die Wahrscheinlichkeit maximiert, dass die einzelnen Aktionen erfolgreich ausgeführt werden können. D. h., dass keine anderen Objekte bewegt oder zu Fall gebracht werden. Anschließend wird eine zweihändige Manipulationsstrategie implementiert, die die zweite Hand des Roboters nutzt, um potenziell fallende Objekte zu sichern. Darüber hinaus werden Methoden zur Prädiktion von Aktionseffekten verwendet, um Aktionssequenzen zu parametrisieren, die von einem symbolischen Planer erzeugt worden sind. Durch die Suche im Raum der Aktionsparameter und der Bewertung der Parameter auf Basis der vorhergesagten Effekte kann ein Satz von Aktionsparametern identifiziert werden, der die spezifizierten symbolischen Effekte erzeugt. Zur Validierung der vorgeschlagenen Methoden werden Experimente mit den humanoiden Robotern ARMAR-III und ARMAR-6 durchgeführt.

Contents

1. Introduction	1
1.1. Problem Statement	2
1.2. Contributions	3
1.3. Outline	5
2. State of the Art	7
2.1. Relational Scene Representations	7
2.1.1. Spatial Relations	8
2.1.2. Support Relations	10
2.1.3. Summary and Review	12
2.2. Support Relation Extraction	14
2.2.1. Methods	14
2.2.2. Summary and Review	20
2.3. Learning with Relational Representations	22
2.3.1. Graph Neural Networks	23
2.3.2. Predicting Physics with Graph Neural Networks	27
2.3.3. Summary and Review	27
2.4. Action Effect Prediction	29
2.4.1. Methods	30
2.4.2. Summary and Review	33
2.5. Deformable Object Dynamics	36
2.5.1. Simulation Environments	36
2.5.2. Prediction Methods	38
2.5.3. Summary and Review	39
2.6. Discussion	41

3. Representation and Extraction of Support Relations	43
3.1. Probabilistic Object and Support Relation Representation	45
3.2. Probabilistic Geometric Primitive Fitting	48
3.3. Extraction of Probabilistic Support Relations	55
3.3.1. Deterministic Support Relation Extraction	55
3.3.2. Probabilistic Support Relation Extraction	61
3.4. Evaluation	63
3.4.1. Deterministic Support Relation Analysis	63
3.4.2. Probabilistic Support Relation Extraction	66
3.5. Summary and Review	71
4. Graph-based Prediction of Action Effects	73
4.1. Action Effect Prediction for Rigid Objects	75
4.1.1. Scene Representation	76
4.1.2. Data Generation	79
4.1.3. Graph Learning Problem	81
4.1.4. Evaluation	84
4.2. Action Effect Prediction for Deformable Objects	87
4.2.1. Task Description	88
4.2.2. Data Generation	90
4.2.3. Scene Representation	91
4.2.4. Prediction Models	94
4.2.5. Evaluation	96
4.3. Summary and Review	101
5. Application to Robot Manipulation Tasks	103
5.1. Support Relations for Manipulation Tasks	104
5.1.1. Deriving Manipulation Order from Support Relations . . .	104
5.1.2. Bimanual Manipulation Strategy	106
5.1.3. Validation Experiments	112
5.2. Parametrizing Actions using Prediction	114
5.2.1. Determining Action Parameters based on Prediction . . .	115
5.2.2. Integration into the Robot Software Architecture	118
5.2.3. Push Execution	119

5.2.4. Validation Experiments	121
5.3. Summary and Review	122
6. Conclusion	125
6.1. Scientific Contributions	125
6.2. Discussion and Future Work	127
Appendix	131
A. KIT Semantic Relation Dataset	131
B. Deformable Bag Interaction Dataset	136
C. Normality Test for Geometric Primitive Distributions	140
Acronyms	143
List of Figures	146
List of Tables	147
List of Algorithms	149
Bibliography	165

1. Introduction

Humans have the ability to intuitively understand and manipulate complex scenes with multiple stacked and overlapping objects. Understanding the scene semantically, i. e. decomposing it into the encountered objects, analyzing their 3D structure, scene layout, and relations, enables humans to transfer learned concepts to previously unseen scenes with a varying number of objects and relations (Kemp et al., 2006; Kemp and Tenenbaum, 2008). Equipped with this relational scene understanding, humans build effective, intuitive physics prediction models for interactions with the environment. Battaglia et al. (2013) compared predictions made by accurate physics simulations with intuitive models based on object relations. They showed that human predictions align with intuitive models rather than accurate physics simulations. Infants acquire this ability gradually during their first two years. They start by understanding simple contact/no-contact relations and expand to reasoning about supported surface areas (Baillargeon, 2002).

In humans, relational scene understanding and intuitive prediction models build the basis for goal-directed manipulation of complex scenes. This thesis investigates how these abilities can be developed, implemented, and evaluated in the context of humanoid robot manipulation tasks. Figure 1.1 shows an example of a complex scene that requires physical reasoning about object relations, the ability to predict action effects, and the strategies to prevent undesired action side effects.

This chapter starts with the problem statement and research question formulation. Then, the main contributions of this thesis addressing the research questions are presented. Finally, an outline of the thesis is given.

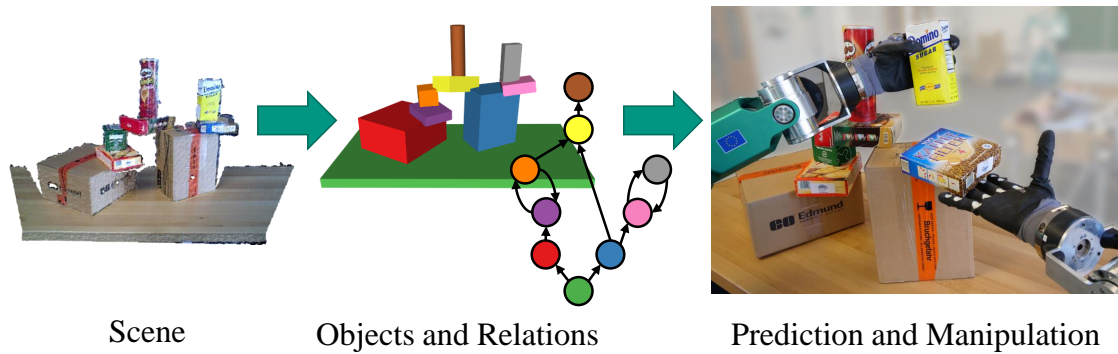


Figure 1.1.: A scene with multiple stacked objects is perceived. By extracting the contained objects and their relations, we can predict possible action effects. For example, before grasping the yellow sugar box with its left hand, the robot is able to reason about physical support between objects and predict that the blue coffee filters box might fall. Therefore, the right hand supports the filters during manipulation.

1.1. Problem Statement

This thesis approaches the problem of relational semantic scene understanding, action effect prediction based on intuitive physics models, and their application to humanoid robot manipulation tasks. To show that interpreting a scene as objects and their relations enables a robot to predict action effects and manipulate complex scenes, the following three research questions will be investigated:

1. How can a robot understand a scene semantically and decompose it into objects and their relations?

A humanoid robot perceives its environment through head-mounted cameras. In this work, we assume that the robot has the ability to perceive 3D information either via active depth cameras or stereo camera systems. The robot needs to extract object shapes, poses, and relations from this scene perception. Sensor noise, object occlusion, and partial views require a suitable object and relation representation, which captures these uncertainties.

2. How can a robot learn to predict the effects of its actions using an intuitive physics model?

This thesis considers actions like pushing, lifting, or moving an object, potentially interacting with multiple other objects in the scene. One key challenge is to represent scenes with a varying number of interacting objects in a unified way suitable for machine learning methods. Another challenge is the generation of training data. On the one hand, generating data to learn action effect prediction models on the real robot is time-consuming and expensive. On the other hand, generating training data in simulation requires transferring it to the real robot due to differences between the real and simulated world.

3. How can relational scene understanding and action effect prediction models facilitate humanoid robot manipulation tasks?

Relevant tasks in this work are grasping an object in complex scenes while preventing other objects from falling and bringing a scene into a specified goal configuration. To achieve this, both scene understanding and action effect prediction need to be part of a humanoid robots' functional and cognitive software architecture. Therefore, the extraction of object relations and the action effect prediction models need to be integrated into existing perception pipelines and memory systems.

1.2. Contributions

There are two areas to which this thesis makes contributions: semantic scene understanding and action effect prediction. In the area of semantic scene understanding, this work proposes a novel probabilistic representation and a method for extracting support relations. In the area of action effect prediction, graph-based representations and prediction models are used to learn effects for scenes with multiple interacting objects. Both contributions are validated in the context of humanoid robot manipulation tasks.

Representation and Extraction of Support Relations: The first contribution of this thesis is a novel probabilistic representation and extraction method for *support relations*. Support relations encode which object is supported by which

other objects in a scene. Both object and relation representations are modeled as probability distributions to capture the uncertainty in perception. In this thesis, we assume that objects in the scene are unknown, i. e. no knowledge about their class, shape, or texture is known before. Therefore, a method for extracting probabilistic object shape and pose representations from point clouds of the scene is developed. Furthermore, state-of-the-art extraction methods for support relations are extended and improved via a *support polygon analysis*, which detects support relations previously overlooked by existing approaches. Based on the probabilistic object representation, existence probabilities for support relations are then estimated via a *Monte Carlo simulation*.

Graph-based Prediction of Action Effects: The second contribution of this thesis is an action effect prediction method, which uses *graph-based representations and learning methods*. By encoding each involved object as a vertex in a graph, and their relations as edges, these methods are order-invariant and independent of the number of interacting objects. This work investigates action effect prediction for both *rigid* and *deformable* objects. First, a push effect prediction for multiple rigid objects is developed and trained on a large number of simulated, randomized scenes. This prediction model is then extended via a *sparse keypoint representation* to handle deformable objects and multiple new actions, e. g. lifting or opening cloth-like bags. Furthermore, a *mixed-horizon* model is proposed, which supports prediction over multiple time frames while reducing the accumulation of errors.

Application to Humanoid Robot Manipulation Tasks: The ability to understand support relations between objects enables a humanoid robot to predict which objects might fall during manipulation tasks. For example, lifting a particular object can lead to another object underneath it falling. To prevent undesired side effects, this work proposes a method to derive a *safe manipulation order*, which takes into account uncertainties about object shape, poses, and support relations. Furthermore, the action effect prediction enables *goal-directed manipulation* of scenes. Given a description of the scene’s goal state, the robot can generate action sequences and use the learned model to predict whether the

actions lead to the desired effects. The proposed strategies have been implemented and validated on the humanoid robots ARMAR-III (Asfour et al., 2006) and ARMAR-6 (Asfour et al., 2019).

1.3. Outline

This thesis is structured into six chapters, discussing the state-of-the-art, presenting and evaluating the main contributions, and summarizing the results.

Chapter 2 discusses related work concerning the contributions in this thesis. This includes *relational scene representations*, focusing on spatial and support relations. One central aspect of the state-of-the-art is concerned with methods for *extracting support relations* from images or point clouds. The discussion of *graph-based learning methods* is the basis for the proposed prediction models. Another body of related work is action effect prediction, from which model-based, data-driven, and hybrid methods are reviewed. Furthermore, simulation environments and prediction methods for *deformable object dynamics* are discussed.

Chapter 3 introduces a *probabilistic representation* of objects and support relations with physical grounding. This representation captures perception uncertainty in object shapes and poses as well as support relation existence. A *geometric primitive fitting* algorithm is developed that extracts the probabilistic object representation from point clouds. Based on the object representation, a method for support relation extraction is proposed, which extends existing force-based methods with a *support polygon analysis*. Finally, a Monte Carlo method is used, which computes existence probabilities for support relations by propagating the uncertainty in object shapes and poses. The proposed extraction method is evaluated on two datasets. To this end, we record the novel *KIT Support Relation (KIT-SR)* dataset containing various table-top scenes with multiple stacked objects and extend an existing dataset with support relation annotations.

Chapter 4 describes graph-based action effect prediction. This work considers scenes in which *multiple interacting objects* can be affected by an action executed by a robot. As a first scenario, we investigate *push effect prediction* for multiple rigid objects. To this end, a graph-based scene representation encodes objects

and their relations. Furthermore, *graph neural networks* are trained to predict the scene state after executing an action. To extend this prediction model for more actions and deformable objects, we propose a sparse keypoint representation for both rigid and deformable objects. A two-stage graph neural network model first classifies which keypoints potentially move and then predicts keypoint motion based on this representation. Furthermore, a *mixed-horizon model* combines prediction models trained with different time steps to prevent the accumulation of prediction errors over longer time horizons. The proposed methods are evaluated on large datasets generated in simulation.

Chapter 5 demonstrates the application of the support relation extraction and the action effect prediction from the previous chapters to *humanoid robot manipulation tasks*. First, a method to derive a *safe manipulation order* from the probabilistic representation of support relations proposed in chapter 3 is presented. Lifting a grasped object might unintentionally cause other objects in the scene to fall. A *bimanual manipulation strategy* is implemented to prevent unintentional effects, which uses a second hand of the robot to secure a potentially falling object. Additionally, we propose to use internal simulation based on the action effect prediction for *planning action sequences* to achieve a specific goal configuration of multiple objects. Using the prediction models presented in chapter 4, the robot can select suitable action parameters based on their predicted effects.

Chapter 6 concludes the thesis by summarizing the contributions and the obtained results. Furthermore, it reviews the strengths and weaknesses of the proposed methods. Finally, possible extensions for future work are presented.

2. State of the Art

This thesis aims at developing methods for semantic scene understanding and action effect prediction to facilitate humanoid robot manipulation tasks. To this end, it contributes to three research areas in robotics: *relational scene representation*, *support relation analysis*, and *action effect prediction*. This chapter presents the fundamentals and discusses the contributions and limitations of related work in these three research areas.

The chapter begins with a discussion of relational scene representations in section 2.1, focusing on spatial and support relations. Then, existing methods for extracting support relations from sensor data are presented in section 2.2. After reviewing graph-based machine learning methods in section 2.3, we discuss approaches for action effect prediction in section 2.4. Finally, section 2.5 presents simulation and prediction methods for deformable object dynamics.

2.1. Relational Scene Representations

One key ingredient of human intelligence is *combinatorial generalization* (Kemp and Tenenbaum, 2008). This is the ability to create infinitely many and arbitrarily complex structures from a finite set of building blocks and relations. Humans use this ability to construct sentences, acquire new skills, and understand the world around them (Humboldt, 1999). In the area of visual scene understanding, humans can transfer knowledge and skills to scenes with a previously unseen number of objects and relations by deconstructing perceived scenes into objects and their relations.

Deconstructing scenes into objects and their relations allows humans to learn a particular relation and generalize to a diverse set of scenes. For example, after



Figure 2.1.: Example scenes containing different objects which support each other. Understanding support relations between objects helps to manipulate them safely in such scenes. (*Images licensed under CC0*)

learning the relation "object A supports object B", humans are able to understand and manipulate a stack of dishes in the kitchen, a Jenga tower, or a heap of cardboard boxes (see Figure 2.1). This is evidenced by the fact that relational models are better at explaining human predictions than accurate physics simulations (Battaglia et al., 2013).

There are different kinds of object relations. *Spatial* relations between objects are based on the scene geometry, i. e. relative positions and orientations between objects. *Support* relations take into account physical object interactions and encode whether an object is supported by other objects.

2.1.1. Spatial Relations

Spatial relations encode geometric information about the relative position and orientation of objects, e.g. the cup is *to the left of* the plate, and both are *on* the table. Representations of spatial relations differ in the way how they encode geometric information and whether they consider change over time. The encoding can be symbolic or subsymbolic. A symbolic encoding uses a discrete set of spatial relations existing between objects in a scene, e.g. *left of*, *right of*, *in front of*, and *behind*. A subsymbolic encoding uses continuous values to represent spatial relations, e.g. the euclidean distance between objects. If relations consider change over time, e.g. the cup is *getting closer to* the plate, they are dynamic. In contrast, relations that only consider a single point in time are static. Figure 2.2 shows examples of spatial relations and their categorization into static, dynamic, symbolic, and subsymbolic representations.

2.1. Relational Scene Representations

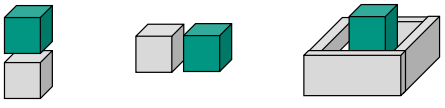
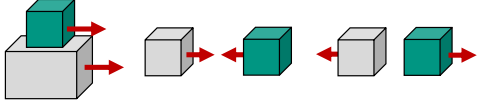
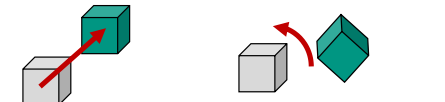
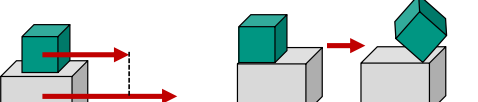
	Static	Dynamic
Symbolic	 <p>On To the Right of Contained in</p>	 <p>Moving Together Getting Closer Moving Apart</p>
Subsymbolic	 <p>Euclidean Distance Relative Orientation</p>	 <p>Relative Velocity Transformation Change</p>

Figure 2.2.: Categorization of different spatial relation representations according to the consideration of time (static vs. dynamic) and the encoding of geometric information (symbolic vs. subsymbolic).

Related works can be categorized using these criteria. First, static representations, which only consider a single point in time, are discussed. In Gupta et al. (2010), the authors segment a point cloud into object hypotheses and represent each object as a cuboid (block). Based on the relative position of these blocks, they define three discrete spatial relations between them: *in front of*, *behind*, *no relationship*. Rosman and Ramamoorthy (2011) derive symbolic spatial relations between objects based on contact points, i.e. they are only interested in contact relations like *on*, *under*, and *adjacent*. Kasper et al. (2011) use a subsymbolic encoding for euclidean distance in 3D and the projection onto the ground plane, as well as relative orientation between objects. After segmenting the scene and annotating oriented bounding boxes manually, they compute statistics about the occurrence of objects and spatial relations. In the works of Aksoy et al. (2011, 2015), the relations *touching* and *overlapping* are extracted from segmented images. Here, the authors track the changes in spatial relations during a human manipulation task and store them into transition matrices called *semantic event chains* (SECs). While changes in the spatial relations over time are tracked, the relational representation is still static. Sui et al. (2017) estimate a probability distribution over possible scene graphs consisting of static relations *in*, *on* and *has* (indicating that the robot grasped an object).

Dynamic representations express change over time directly in relational form. Meißner et al. (2013) represent subsymbolic relations as six Degrees-of-Freedom (DoF) pose transformations. The transformations between the objects as well as the transformations over time are tracked. In Ziaeetabar et al. (2017, 2020), the authors introduce a symbolic representation for dynamic spatial relations, *moving together*, *halting together*, or *moving apart*. Objects are approximated as cuboids, and relations are derived based on hand-crafted geometric rules.

2.1.2. Support Relations

Support relations encode which object is supported by another object and allow reasoning about the stability of a scene. When representing support relations in scenes with multiple interacting objects, one can either consider the whole scene, single objects, or object pairs. This determines the relational *structure* of the support representation. When the whole scene is considered, the representation only stores whether the scene is stable or not (Battaglia et al., 2013; Sallami et al., 2019; Paxton et al., 2022). Some models have *unidirectional* structure, i. e. they represent whether an object is supported by other objects but not which the supporting objects are (Liu et al., 2015).

Most relevant for this work are *bidirectional* support relations. We denote that bidirectional support exists between two objects X and Y as $(X, Y) \in \text{SUPP}$ or short $\text{SUPP}(X, Y)$. The existence of a support relation $\text{SUPP}(X, Y)$ means that if the supporting object X is removed, the supported object Y will start to move. Mojtahedzadeh et al. (2015) give a formal definition of a support relation:

For two objects X and Y in a static configuration, we say that X supports Y if removing X from the configuration causes Y to lose its motionless state (e.g., Y will fall down.). We denote this relation as $\text{SUPP}(X, Y)$. A support relation can hold if the two objects are in direct or indirect contact with each other.
(Mojtahedzadeh et al., 2015)

Support relations can either be grounded in *physical* or *semantic* principles. Physical grounding requires knowledge or assumptions about interaction forces and physical object properties, e.g. mass, friction, and inertia. Semantic grounding

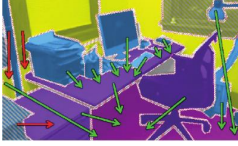
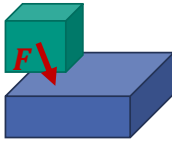
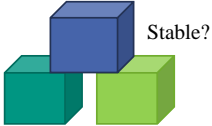
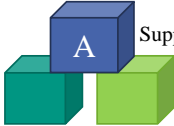
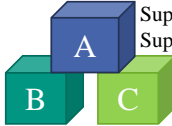
Grounding	<p style="text-align: center;">Semantic</p>  <p style="text-align: center;">Silberman et al., 2012</p>		<p style="text-align: center;">Physical</p> 
Structure	<p style="text-align: center;">Stability</p>  <p style="text-align: center;">Stable?</p>	<p style="text-align: center;">Unidirectional</p>  <p style="text-align: center;">Supported(A)</p>	<p style="text-align: center;">Bidirectional</p>  <p style="text-align: center;">Supports(B, A) Supports(C, A)</p>
Model	<p style="text-align: center;">Deterministic</p> $(A, B) \in SUPP \vee (A, B) \notin SUPP$		<p style="text-align: center;">Probabilistic</p> $P((A, B) \in SUPP) \in [0,1]$

Figure 2.3.: Categorization scheme for support relation representations according to three criteria: grounding, structure, and model.

is mostly used in the computer vision community, e. g. „mugs are often supported by tables, but rarely by walls“ (Silberman et al., 2012). The existence of a semantic support relation is based on statistics about object categories and not on physical object properties. Since physically accurate support relations are necessary for robotic manipulation tasks, most robotic applications prefer the physical definition over the semantic one.

A support relation representation can either be *deterministic* or *probabilistic*. In a deterministic representation, a support relation between two objects either exists or not. A probabilistic representation assigns an existence probability to every object pair (X, Y) :

$$P(\text{SUPP}(X, Y)) \in [0, 1] \subset \mathbb{R}$$

In the following, related works that propose or use support relation representations will be classified according to their structure (stability, unidirectional, or bidirectional), grounding (semantic or physical), and model (deterministic or probabilistic). Figure 2.3 illustrates this categorization scheme.

Works from the computer vision community predominantly use a semantic grounding for support relations based on object features. As a seminal work,

Silberman et al. (2012) assign structure classes to each object in the scene (*ground, furniture, prop, or structure*). Based on geometric features and these structure classes, they define support relation probabilities. In the works of Xue et al. (2015), the authors use a semantic grounding based on structure classes as well. However, their representation of support relations is deterministic, i.e. relations are represented as either support from the bottom, behind, the ground, or no support. Similarly, Jia et al. (2014), Zheng et al. (2015) and Yang et al. (2017) use a deterministic representation with semantic grounding.

A physical grounding is preferred for robotics applications since it allows reasoning about the scene's stability and consequences of actions. The definitions in Mojtahedzadeh et al. (2013) and Zhang et al. (2019) are based on a static equilibrium analysis, which computes support relations based on the acting forces and torques. In the work of Zheng et al. (2013), the authors use physical plausibility based on stability under gravity. All these works employ a deterministic representation. An example of a probabilistic representation with robotic application is the work by Panda et al. (2013, 2016). However, this work uses semantic grounding similar to the works by Silberman et al. (2012).

2.1.3. Summary and Review

Table 2.1 categorizes related works according to their grounding, relational structure, and model for support relations. While the probabilistic representations have been exploited for support relations with semantic grounding, their combination with physical grounding remains unaddressed. Since physical grounding is necessary for robot manipulation tasks and a probabilistic representation enables robots to propagate uncertainty from sensors through perception and up to high-level planning, this research gap is addressed in this thesis. To this end, this thesis proposes a probabilistic scene representation with physical grounding, which uses bidirectional support relations.

Table 2.1.: Comparison of Support Relation Representations

Reference	Grounding		Structure			Model		Application Area
	Semantic	Physical	Bidirectional	Unidirectional	Stability	Deterministic	Probabilistic	
Silberman et al. (2012)	✓		✓				✓	Computer Vision
Panda et al. (2013, 2016)	✓		✓				✓	Robotics
Jia et al. (2014)	✓		✓			✓		Computer Vision
Xue et al. (2015)	✓		✓			✓		Computer Vision
Zheng et al. (2015)	✓		✓			✓		Computer Vision
Yang et al. (2017)	✓		✓			✓		Computer Vision
Liu et al. (2015)	✓			✓			✓	Computer Vision
Battaglia et al. (2013)		✓			✓	✓		Neuroscience
Sallami et al. (2019)		✓			✓	✓		Robotics
Paxton et al. (2022)		✓			✓	✓		Robotics
Zheng et al. (2013)		✓	✓			✓		Robotics
Mojtahedzadeh et al. (2013)		✓	✓			✓		Robotics
Zeng et al. (2018)		✓	✓			✓		Robotics
Zhang et al. (2019)		✓	✓			✓		Robotics
This thesis		✓	✓				✓	Robotics

2.2. Support Relation Extraction

Methods for extracting support relations aim at determining the support relations between objects given a perception of the scene, e.g. image, point cloud, or voxel grid. There are five main methods to analyze support relations:

- **Feature-based** methods define hand-crafted features to determine support between image parts. Supervised models are trained to predict support relations based on these features.
- **Rule-based** methods determine support relations based on a set of hand-crafted rules. These rules usually include geometric properties like height, orientation, or density.
- **Data-driven** methods learn the correlation between input scene and support relations directly from data.
- **Simulation-based** methods estimate the stability of the scene by running a physics simulation and analyzing object movement.
- **Force analysis** methods compute the forces which act on objects after extracting object poses and shapes from the scene.

We first describe different methods for support relation extraction. Then, the advantages and disadvantages of the presented methods and the research gap addressed in this thesis are discussed.

2.2.1. Methods

Feature-based methods extract features from input images of the scene and learn regression models to predict both segmentation and support relations. Most of the time, these methods try to improve image segmentation results by incorporating knowledge about support relations between objects. All feature-based methods use semantic grounding, i. e. they are not physically accurate.

In Gupta et al. (2010), the authors define geometric features like the objects' shape and density and derive potential energy terms for each feature. This potential energy is not a physical quantity but rather an optimization term, which is lower if the scene is more stable than a scene with higher potential energy. By



Figure 2.4.: Top row: Images of different indoor scenes. Bottom row: Segmentation results and inferred support relations (Correct relations are green arrows, wrong relations are red arrows). (taken from Silberman et al., 2012))

minimizing this potential energy during image segmentation, regions are split in a way that prefers stable scene arrangements. The resulting support relations between image segments have improved the image segmentation results but are not evaluated for their physical accuracy.

Silberman et al. (2012) uses 2D and 3D features to infer support relations which improve image segmentation results for indoor scenes. The authors first oversegment the image. Then a hierarchical segmentation merges regions with low boundary strength (computed based on 3D and 2D features and the region's class label). Based on this segmentation, structure classes (ground, furniture, prop, and structure) and support relations for each region are inferred simultaneously. This is done by defining a function that assigns an energy value to each assignment of support relations and structure classes. By optimizing this energy function, they are able to find the most probable scene structure given existing 3D and 2D features in each region. Figure 2.4 shows the results of the approach for selected indoor scenes.

Jia et al. (2014) extend the work of Silberman et al. (2012) by fitting boxes to

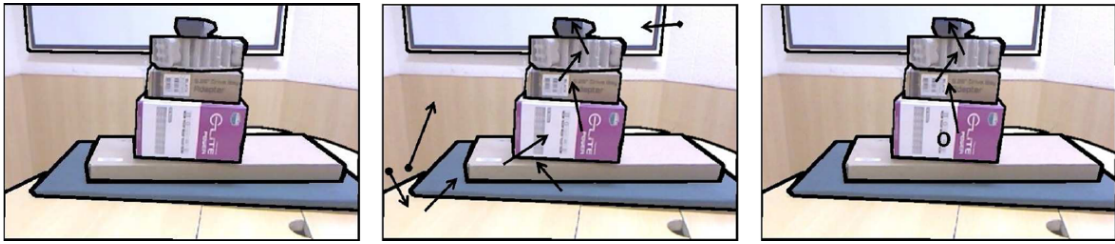


Figure 2.5.: Left image: Segmented scene. Middle image: Support relation between all segments. Right image: Relevant support relations for manipulating an object of interest O . (taken from Panda et al., 2016)

the segmented regions. Based on these boxes, the authors extract features like the separating orientation, the center of mass, and the supporting area. Then, an energy function is defined and optimized using these features. Xue et al. (2015) use SIFT features to infer structure classes and add features describing the stability of the scene. Yang et al. (2017) extend the work of Xue et al. (2015) by incorporating prior knowledge about support relations between object classes. All these works aim to use support relations to improve image segmentation results.

Panda et al. (2013, 2016) apply a very similar segmentation and optimization approach to Silberman et al. (2012). However, the authors define different features, e. g. boundary ratio (overlap between objects in contact), distance, and containment. In contrast to the other feature-based works, the goal of these works is to infer a manipulation order that a robot can execute. Figure 2.5 shows extracted support relations and manipulation order for an object of interest.

Rule-based methods take an already segmented scene as input and extract support relations via a set of hand-crafted rules. These rules are usually defined on the basis of symbolic predicates, which have in turn been extracted from the scene before. Liu et al. (2015) derive symbolic predicates for object relations (e. g. $\text{contact}(A, B)$, $\text{higher}(A, B)$, and $\text{intersect}(A, B)$) from 6D object poses. A sophisticated set of rules is used to infer support relations based on the symbolic predicates. Zeng et al. (2018) define a heuristic based on object height and overlapping areas in the image, which determines whether support exists between two image regions. Wu et al. (2019) extract object bounding boxes from images

and, additionally, incorporate depth data to define rules for support relation extraction.

Data-driven methods learn the correlation between perceived scene and support relations. To this end, they need large datasets annotated with support relations. These methods are often based on Convolutional Neural Networks (CNNs), which are used to extract features from images. Additionally, a pairwise support predictor determines support between image regions.

Zhang et al. (2018) predict manipulation order from input images. Their proposed visual manipulation relationship network consists of a Convolutional Neural Network, an object detector, and a relationship predictor. The CNN extracts features from the input images, which are used by both the object detector and the relationship predictor. The object detector generates 2D bounding boxes for detected objects. For each detected object pair, the relationship predictor determines the manipulation order based on the extracted features. The predicted manipulation order is equivalent to inverted support relations.

Yang et al. (2018) extend the visual manipulation relationship network of Zhang et al. (2018) by correcting false support relation predictions. They use Conditional Random Fields (CRFs) to predict where no support exists. In Ren et al. (2018), the authors use the same method, i. e. CNN for feature extraction, CRFs for support relation improvement, in the context of exoskeletons. Their goal is to use the support relations to find a region on the floor that can safely support the person wearing a leg exoskeleton.

Simulation-based methods run physics simulations and analyze object movement to estimate the stability of the scene. Knowledge about geometry, poses and physical properties is required to create object models for the simulation. The stability of the scene and support relations between objects can be analyzed in simulation by observing which objects start to move and which objects stay still.

Battaglia et al. (2013) investigate how humans understand stability and support relations scenes with stacked blocks. The authors compare an intuitive physics

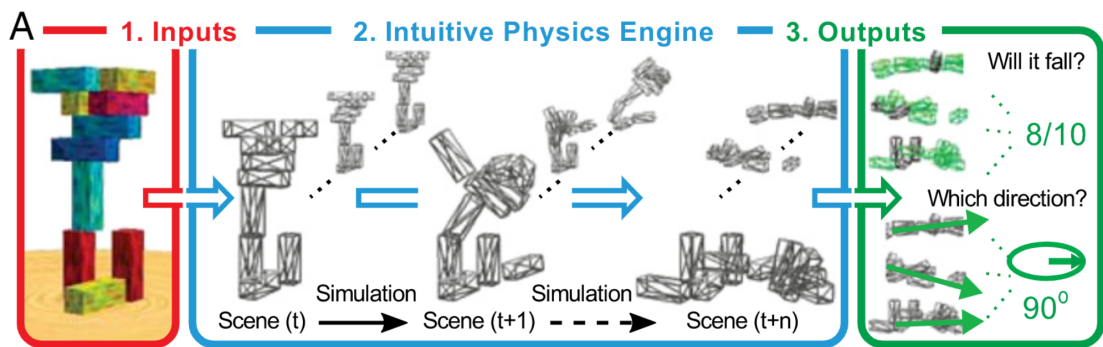


Figure 2.6.: The intuitive physics engine takes an input scene and simulates how the scene evolves over time. The results are aggregated to determine which object will fall and in which direction (taken from Battaglia et al. (2013)).

model based on object relations with an accurate model based on a physics simulation. By simulating scenes with the Bullet physics simulator, they predict whether a scene is stable or not and in which direction the objects will fall (see Figure 2.6). As input, a distribution of scene geometries is used. Then, by sampling from this distribution and simulating each sampled scene, the stability of the scene can be predicted.

In Sallami et al. (2019), the goal is to estimate a consistent scene model from an input point cloud. First, the authors compute initial support relations using geometric features from bounding boxes. Then, the Bullet physics simulator is used to correct inconsistencies due to penetrations or collisions.

Desingh et al. (2016) propose a simulation-informed particle filter for physically plausible scene estimation taking into account stacked objects with partial support. By assigning scene structures with plausible support relations a higher weight, the estimation of object poses can be improved.

Force analysis methods compute support relations based on acting forces between objects. As input, these methods expect object poses and shapes, which need to be extracted from the perceived scene. Due to the physical grounding, the results can be effectively used to manipulate scenes with stacked objects in robotic manipulation tasks.

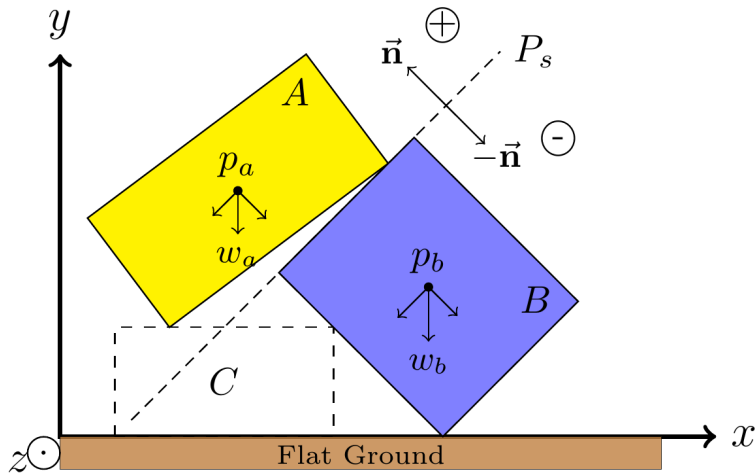


Figure 2.7.: The separating plane P_s between two objects A and B in contact can be used to infer the ACT relation. Object A is on the separating plane's positive side (regarding gravity). Therefore, $ACT(A, B)$ holds. (taken from Mojtahedzadeh et al., 2013)

Zheng et al. (2013) use a two-step approach. First, during geometric reasoning, shape primitives are extracted from an input point cloud and physical properties like volume, mass, and supporting areas are estimated. As a next step for physical reasoning, the authors define a physical stability function, which takes into account the minimal work necessary to change the scene configuration. In Zheng et al. (2015), they extend their work to find the most stable scene segmentation.

Mojtahedzadeh et al. (2013) first extract box primitives from input point clouds. Based on the extracted boxes, contact points are computed. Between each box pair in contact, the *acting* force is calculated, i. e. the force that one object exerts on the other due to gravity. To this end, separating planes are introduced at each contact point. The normal vector of this separating plane in combination with the gravity vector determines which object is on the positive side of the plane. This object acts on the object on the negative side of the separating plane. Figure 2.7 shows an example of the ACT relation extraction. In order to extract support relations $SUPP(B, A)$, the relation $ACT(A, B)$ is inverted. This method only considers cases where an object supports another object above it.

Zhang et al. (2019) register multiple views of the same scene before segmenting it. Then, they use static equilibrium analysis, i. e. the object state is analyzed in terms of the forces that act upon the object. This results in a system of equations for each contact point. The system is stable if the equations can be solved.

2.2.2. Summary and Review

Table 2.2 categorizes the works related to support relation analysis according to their method, support relation representation, and input. In the following, the advantages and disadvantages of the different methods are discussed.

Feature-based methods offer a way to extract support relations from images. Both deterministic as well as probabilistic representations are available. However, these methods are based on a semantic grounding, i. e. the extracted support relations are not necessarily physically plausible. They are optimized to improve image segmentation results.

Instead of learning a regression model based on image features, **rule-based methods** define a set of rules that derive support relations based on geometric object properties. The rules can be implemented in a straightforward way and can be evaluated efficiently. However, these hand-crafted rules cannot cover all relevant cases and require a deterministic object and relation representation.

Data-driven methods do not rely on hand-crafted features or rules but learn to extract relevant features for support relation extraction from images. Since these methods employ supervised machine learning, they require large image datasets annotated with support relations. They do not consider 3D geometric information since only color images are used.

Simulation-based methods allow modeling accurate physical interactions between objects with complex geometries. However, they require knowledge about object geometry, pose, and physical properties.

Most suitable for robotic manipulation tasks are **force analysis methods** since they allow physically accurate predictions while not requiring a full physical simulation. Furthermore, some methods can cope with partial knowledge about object properties. Nevertheless, existing methods still rely on deterministic rep-

Table 2.2.: Comparison of Methods for Extracting Support Relations

Reference	Method	Repr.	Input
Gupta et al. (2010)	Feature-based	Prob.	RGB-D
Silberman et al. (2012)	Feature-based	Prob.	RGB-D
Panda et al. (2013, 2016)	Feature-based	Prob.	RGB-D
Jia et al. (2014)	Feature-based	Det.	RGB-D
Xue et al. (2015)	Feature-based	Det.	RGB-D
Deng et al. (2015)	Feature-based	Det.	RGB-D
Chen et al. (2015)	Feature-based	Det.	RGB-D
Zheng et al. (2013, 2015)	Feature-based	Det.	RGB-D
Yang et al. (2017)	Feature-based	Det.	RGB-D
Ahmadi and Khotanlou (2017)	Feature-based	Det.	RGB-D
Liu et al. (2015)	Rule-based	Det.	Object Poses
Huang et al. (2015)	Rule-based	Det.	Object Poses
Zeng et al. (2018)	Rule-based	Det.	RGB-D
Wu et al. (2019)	Rule-based	Det.	RGB-D
Zhang et al. (2018)	Data-driven	Prob.	RGB
Yang et al. (2018)	Data-driven	Prob.	RGB
Ren et al. (2018)	Data-driven	Prob.	RGB
Ahuja et al. (2020)	Data-driven	Det.	RGB
Battaglia et al. (2013)	Simulation-based	Det.	Object Poses
Desingh et al. (2016)	Simulation-based	Det.	Object Poses
Sallami et al. (2019)	Simulation-based	Det.	Object Poses
Zheng et al. (2013)	Force Analysis	Det.	Voxel Grid
Mojtahedzadeh et al. (2013, 2015)	Force Analysis	Det.	Point Cloud
Jonathan et al. (2017)	Force Analysis	Det.	RGB-D
Zhang et al. (2019)	Force Analysis	Det.	Voxel Grid
This thesis	Force Analysis	Prob.	Point Cloud

representations. To capture the uncertainty in perception, a method for extracting support relations with a probabilistic representation is needed.

This thesis proposes a novel probabilistic representation and extraction method for support relations. The extraction method extends existing force analysis methods by incorporating uncertainty about object geometries and poses.

2.3. Learning with Relational Representations

This thesis makes extensive use of relational scene representations for understanding and manipulating scenes with multiple interacting objects. In order to learn a model, which predicts action effects, relational scene representations are both the input and output of the model. Since the number of objects and their relations in a scene can vary, this learning problem challenges traditional machine learning approaches.

Traditional machine learning methods, e. g. neural networks, can learn models with fixed sized input and output data. Furthermore, the order of the input and output data is relevant, e. g. the first three features contain position information ordered $x - y - z$. Consider the scenes in Figure 2.8 with three and respectively four objects on a table. One could decide design a neural network architecture, where the feature vectors of each object are stacked on top of each other to create an input vector. However, this input vector size needs to be fixed before training the neural network. Therefore, predictions for scenes with additional objects beyond this limit cannot be done. Furthermore, by stacking object feature vectors, this approach is inherently order-dependent, i. e. a first, second, and third object need to be defined.

To overcome these problems, relational representations for machine learning should meet the following requirements:

- **Variable input size:** Since the number of objects in a scene can vary, the input size of the model should accommodate for this.
- **Order independence:** There is no inherent order among objects, therefore the model needs to be order invariant.

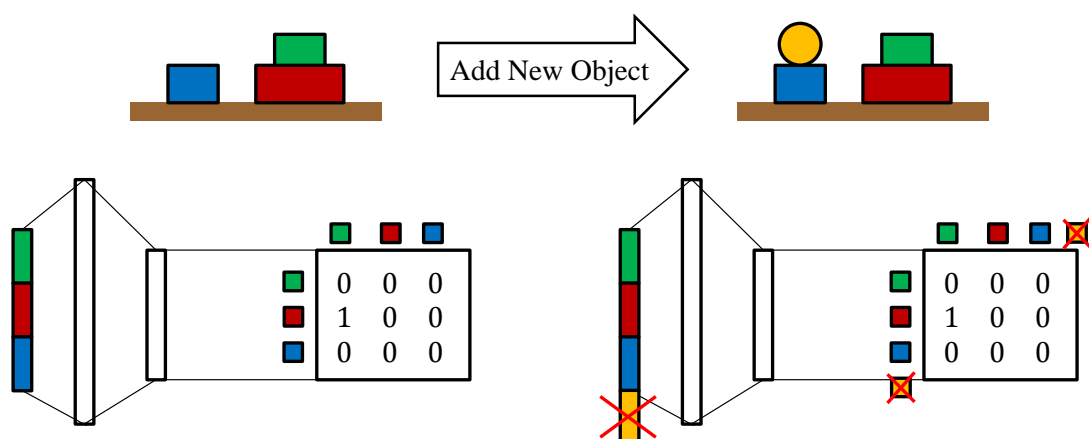


Figure 2.8.: Example task of predicting support relations between objects. Top: Table-top scenes with three objects (left) and four objects (right). Bottom: Neural network with stacked object feature vector as input and output relation matrix as output. Once the input and output size have been fixed, new objects cannot be handled by the neural network without modifying the structure and retraining.

Recently, effective learning methods for relational structure have been proposed. They are mostly referred to as graph neural networks (see Wu et al., 2020b, for a survey), although they are not tied to using neural networks. First, this section reviews how graph neural networks work. Then, their application to physics prediction is presented.

2.3.1. Graph Neural Networks

One step towards achieving variable input size and order independence, are Deep Sets (Zaheer et al., 2017). They support a variable number of input objects, over which shared object features are computed and aggregated in a symmetric way. Symmetric aggregation functions (e. g. mean) provide a permutation invariant way to achieve the same results independent of order. Deep Sets consider an arbitrary number of objects but do not consider relations between objects. Furthermore, they can be used for classification and regression to produce aggregated results with a fixed size, but not variably sized output sets. The idea

of symmetric aggregation can be extended from sets to graphs by implementing interactions between related sets (Hartford et al., 2018). By aggregating over related objects, the features of an object in the output set can be computed.

This can be rephrased and extended to directed graphs as both input and output of a neural network, which is often called a graph neural network (Battaglia et al., 2018). This model combines learnable update functions (usually neural networks) and symmetric aggregation functions for both vertices and edges.

Formally, graph neural networks operate on a directed, attributed graph $G = (V, E, \mathbf{u})$ consisting of N_v vertices V , N_e edges E and global features \mathbf{u} :

- Vertices $V = \{\mathbf{v}_i \mid i \in [1, N_v]\}$ with d_v -dimensional attributes $\mathbf{v}_i \in \mathbb{R}^{d_v}$
- Edges $E = \{(\mathbf{e}_k, r_k, s_k) \mid k \in [1, N_e], r_k, s_k \in [1, N_v]\}$ with d_e -dimensional attributes $\mathbf{e}_k \in \mathbb{R}^{d_e}$, receiver vertex index r_k , and sender vertex index s_k , where the receiver and sender index define the vertices that are connected through the k -th edge
- Global feature vector $\mathbf{u} \in \mathbb{R}^{d_u}$ with d_u dimensions

The attribute dimensions d_v , d_e and d_u are model parameters that can be chosen freely.

A GN block transforms an input graph $G = (V, E, \mathbf{u})$ into an output graph $G' = (V', E', \mathbf{u}')$ in three steps. As a first step, the updated edges are computed according to sender and receiver vertices as well as the global state using the update function Φ_e (see Figure 2.9b).

$$\mathbf{e}'_k = \Phi_e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u})$$

The second step updates each vertex \mathbf{v}_i depending on the global state and an aggregation $\rho_{e \rightarrow v}$ over incoming edges E'_i to \mathbf{v}_i , i.e. $r_k = i$ using the update function Φ_v (see Figure 2.9c).

$$\mathbf{v}'_i = \Phi_v(\mathbf{v}_i, \mathbf{u}, \rho_{e \rightarrow v}(E'_i))$$

As a third step, the global feature vector is updated according to an aggregation over all vertices $\rho_{v \rightarrow u}$ and all edges $\rho_{e \rightarrow u}$ using the update function Φ_u (see Figure 2.9d).

$$\mathbf{u}' = \Phi_u(\mathbf{u}, \rho_{e \rightarrow u}(E'), \rho_{v \rightarrow u}(V'))$$

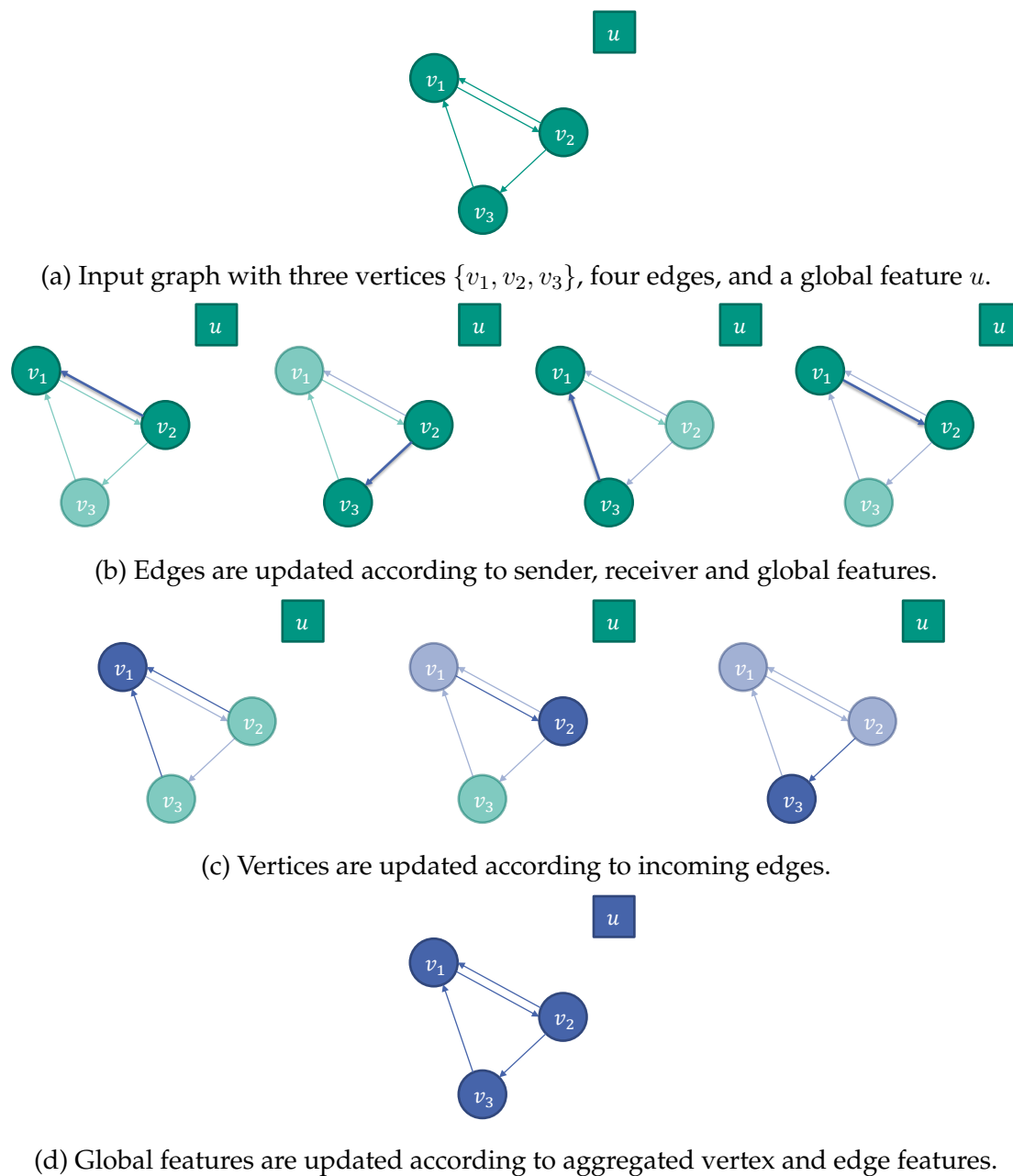
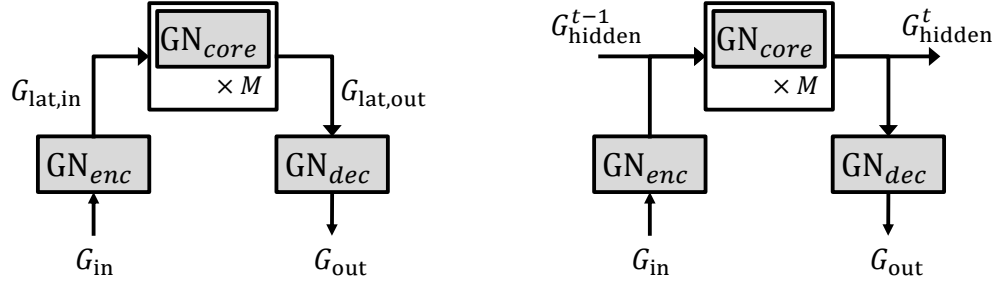


Figure 2.9.: A complete update step of a Graph Network (GN) block transforms an input graph (a) into an output graph (d). This consists of three steps: edge update (b), vertex update (c), and global update (d). Vertices and edges not involved in the update step are grayed out.



(a) Encode-Process-Decode architecture

(b) Recurrent GN architecture

Figure 2.10.: Different network architectures can be created by concatenating GN blocks. The white box with $\times M$ indicates that the block is concatenated to itself M times, i. e. the inner block runs repeatedly (taken from Battaglia et al., 2018).

The vertex, edge and global update functions Φ_v, Φ_e, Φ_u are learnable models and often implemented as neural networks. The aggregation functions $\rho_{e \rightarrow v}, \rho_{v \rightarrow u}$ and $\rho_{e \rightarrow u}$ are fixed functions which need to be symmetric, i. e. invariant to permutations of the input, and accept a variable number of arguments, e. g. sum or mean. Figure 2.9 illustrates the three update steps on an example.

GN blocks can be concatenated into more complex network architectures, e. g. the Encode-Process-Decode architecture or the Recurrent GN architecture, see Figure 2.10. The Encode-Process-Decode architecture (Hamrick et al., 2018) first encodes the input graph into a latent representation using a GN block G_{enc} , see Figure 2.10a. Then, a GN block G_{core} is run repeatedly on this latent representation until a fixed iteration limit M is reached. Finally, the result is produced by running a separate GN block G_{dec} as a decoder. The recurrent GN architecture (Sanchez-Gonzalez et al., 2018) shares encoder, decoder and core GN blocks with the Encode-Process-Decode architecture, see Figure 2.10b. However, it adds a hidden representation G_{hidden}^t that is updated on each recurrent update step t . This hidden representation is input and output of the core GN block.

2.3.2. Predicting Physics with Graph Neural Networks

Due to their ability to handle arbitrary number of objects and relations, graph neural networks are a natural fit for predicting physics of multiple interacting objects. For example, forces are transferred between objects in a contact relationship. The Encode-Process-Decode and Recurrent GN architectures mimic the way how physics simulators work. The core GN block is run repeatedly similar to an integration or update step in physics simulators. The idea is that each update step brings the prediction closer to the ground truth.

Graph neural networks have been applied to physics predictions. In Battaglia et al. (2016), the authors show simple planar examples, including n-body systems interacting through gravity, balls bouncing in a box, and strings colliding with an object. Two use cases are evaluated, physics prediction and estimating potential energy in the system. Battaglia et al. (2018) have investigated mass-spring systems with an arbitrary number of masses and springs, which are fixed at both ends.

Janner et al. (2019) consider a scenario, where an object is dropped onto stack of blocks. The input is an image of the rendered scene. The proposed approach segments the objects from the image and derive a latent object representation. This latent object representation is used as the vertex data, while the pairwise relations between objects form the edges. A graph neural network predicts the latent object representation after the object has been dropped. From this predicted object representation, Janner et al. (2019) reconstruct the image after action execution. Figure 2.11 shows this mechanism.

2.3.3. Summary and Review

This section introduced **graph neural networks**, which allow arbitrarily sized graphs as input and output of machine learning methods. This way, limitations of traditional machine learning approaches like fixed-sized input and output vectors as well as dependence on order are overcome. Since graphs are a natural encoding for scenes with multiple interacting objects, i. e. objects are vertices and relations are edges, graph neural networks are suitable for predicting physical interactions between multiple interacting objects.

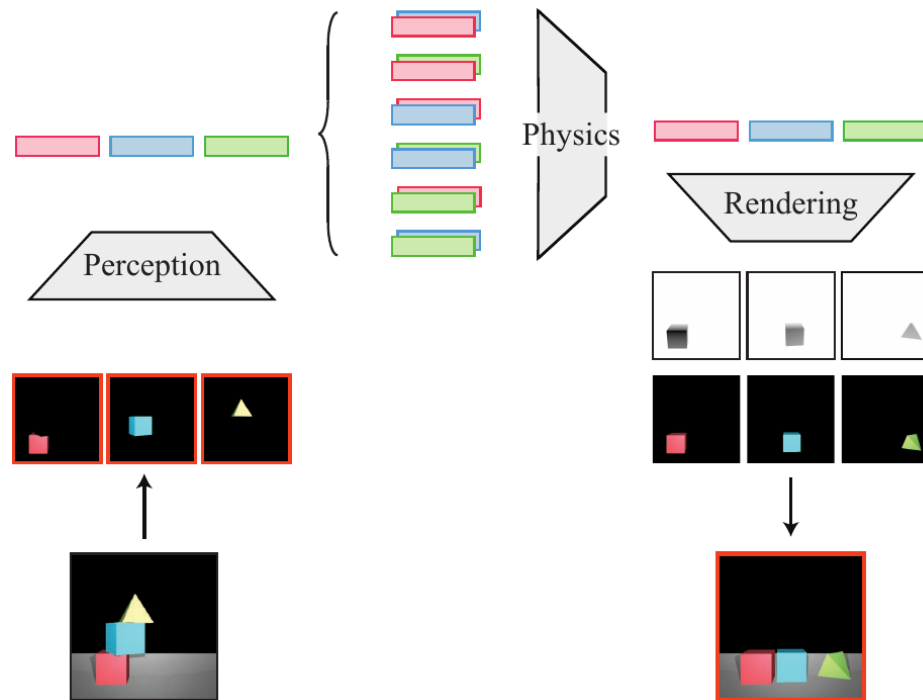


Figure 2.11.: The perception model extracts a latent object representation from the input image. Pairwise relations are fed into the physics prediction model, which outputs predicted object representations. The rendering model creates images of the scene containing only a single object. These images are merged into the output image (taken from Janner et al., 2019).

Examples of learning to **predict physics** of multiple interacting objects implemented using graph neural networks were presented. For simple scenarios like mass-spring systems, colliding balls, gravitational orbits, and falling blocks, graph neural networks have been shown to be able to learn the involved physics and generalize to numbers of involved objects not seen during training.

A remaining open question is, whether graph neural networks can be used in more complex interaction scenarios and whether their predictions are useful in the context of robot manipulation tasks. This thesis will investigate action effect prediction models for multiple interacting rigid and deformable objects based on graph neural networks.

2.4. Action Effect Prediction

Given a perceived current scene state and an action to execute, the goal of action effect prediction is to predict the *scene state after execution* of that action. Figure 2.12 illustrates input and output of action effect prediction methods. The robot can perceive the current scene state on a sensory level through its cameras in the form of images, point clouds, or voxel grids. On a higher level, scene state perception can include object detection, localization and extraction of spatial and support relations. Actions include but are not limited to pushing, grasping, lifting, and dropping objects. They are parametrizable, e. g. pushing into a direction or dropping an object at a specified location, and executable, i. e. the robot can execute an action given concrete parameters.

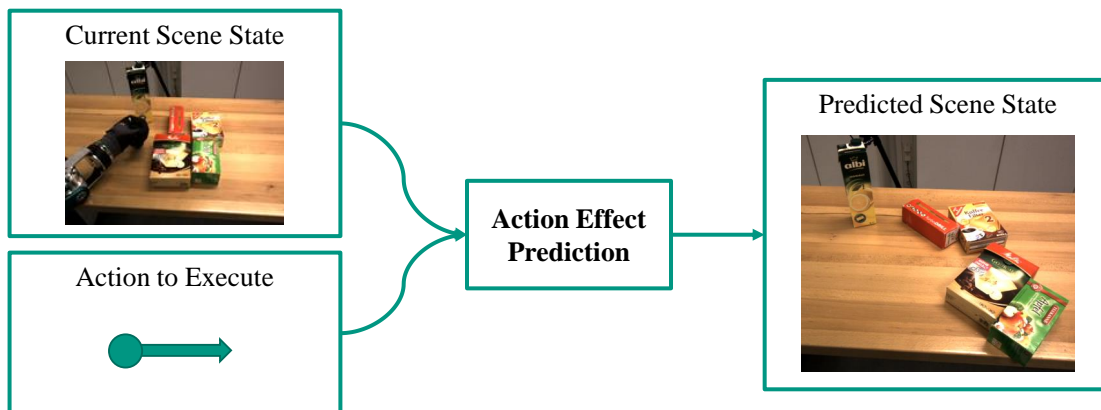


Figure 2.12.: Conceptual input and output of action effect prediction methods. As an example, the scene state is perceived as an image, showing objects on a table and the robot’s hand. The action is pushing to the right and the output is an image of the scene after the push has been executed.

The area of action effect prediction can be divided into four main approaches. **Model-based** methods build an analytical model of the physics involved including physical object properties like mass, friction coefficients and inertial moments. Instead of building an analytical model from scratch, **simulation-based** methods use existing physics simulators to predict action effects. **Data-driven** methods learn the correlation between scene perception, action parame-

ters and perceived effects. There exist also **hybrid** methods, which follow mostly the model-based or simulation-based approach but try to learn parts of the model, e. g. the friction model.

2.4.1. Methods

Model-based methods build analytical models based on physical object properties, which need to be known a-priori. These approaches consider the effects on a single object and often are limited to planar motions.

Early work on analytical models for pushing has been done, predicting the object motion based on frictional forces (Mason, 1986; Lynch et al., 1992). Current model-based approaches (Zhou et al., 2018; Hogan and Rodriguez, 2020; Yu et al., 2016) are focused on planar pushing and predict object motion based on end-effector motion.

Simulation-based methods reconstruct the perceived scene in a physics simulator and simulate actions to predict their effects on a sub-symbolic level. For the scene reconstruction, knowledge about physical object properties is necessary, which either requires prior knowledge or the ability to estimate these properties from the robot's perception.

Weitnauer et al. (2010) predict planar object motion on a table using a physics simulator. Furthermore, physics parameters are optimized to more accurately resemble the real world action effects. Mösenlechner and Beetz (2011) want to find suitable put down locations for objects during planning. They use a physics simulator to predict action effect predicates for planning like contact, stability, and occlusion. In Kunze and Beetz (2017), this concept is extended to predict action effects for a complete planned manipulation sequence. On a sub-symbolic level, the simulator predicts action effects, which are then translated into symbolic predicates. Fromm and Birk (2016) minimize unintended object motion while planning manipulation sequences. A cost function, which quantifies the unintended object motion during action execution, is defined and evaluated using a physics simulator.

Hybrid methods incorporate learning for different parts of a physics model. In Zhou et al. (2018), the authors learn parameters of an even-degree homogeneous polynomial as a friction model. Another approach trains a convolutional neural network to estimate object poses from depth images (Kloss et al., 2017, 2020). An analytical model then uses this object position to predict the effect of a pushing action. Similarly, Wu et al. (2017) train a CNN to estimate object properties (position, velocity, mass, and friction), then predict how the scene evolves over time using a physics simulation.

Data-driven methods learn the relationship between scene perception, action parameters and expected outcome. They require a way of generating training data, which can either happen on the robot or be augmented with data from simulation. Additionally, some methods predict action effects directly on the visual scene data (e.g. images or point clouds) while others rely on existing localizers to extract object poses.

Early data-driven approaches have only considered the motion of a single object. Omrčen et al. (2009) predict planar motion of a single object using binary segmentation masks of the target object as input. In this work, the authors learn the effects of pushing actions on different object classes separately. Kopicki et al. (2011) use a visual object tracker to extract object poses from input images. Then, a regression based method predicts relative pose changes of objects due to a pushing action. In Elliott et al. (2016), an image-based prediction model is used to predict planar motion of a single object. The goal is to find a push action that makes the object graspable.

Instead of considering the motion of a single object, more recent approaches consider a fixed amount of objects. By learning action effects in the image or point cloud space, the limitation to a single object can be lifted. However, most approaches rely on a constant number of object masks, which must be pre-defined before training. These approaches cannot generalize beyond this *fixed* number of objects. For example, Finn et al. (2016) use a fixed number of image masks in their prediction model. Byravan and Fox (2017) and Byravan et al. (2017) also use a fixed number of masks but their prediction model is based on point clouds (Figure 2.13). The proposed model predicts full $SE(3)$ transfor-

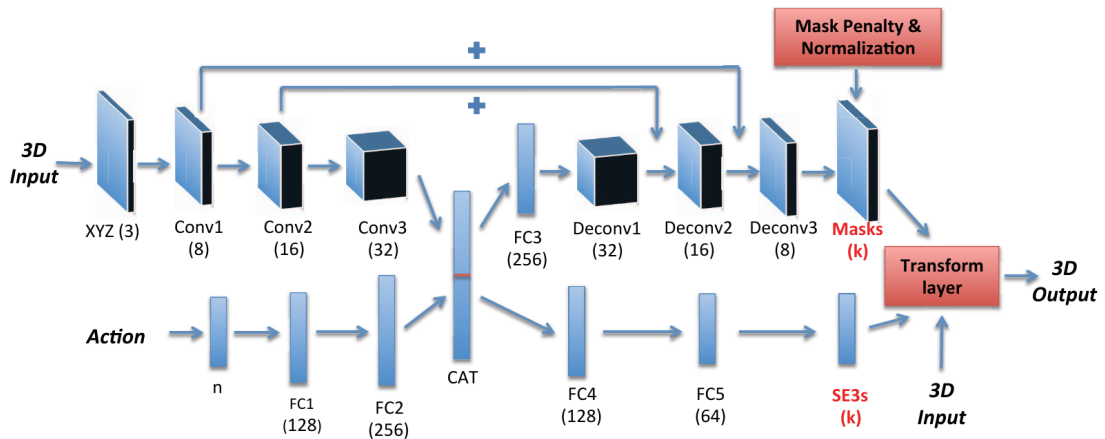


Figure 2.13.: The SE3-Net prediction model from Byravan and Fox (2017) uses point cloud and action data as input. The prediction output is generated for a fixed number k of masks (taken from Byravan and Fox, 2017).

mations, i. e. 3D translation and rotation, for each masked object. Nematollahi et al. (2020) learn interaction dynamics during pushing actions with a similar model.

By working only on images, the inherent limitations of a fixed number of objects can be circumvented (Mottaghi et al., 2016; Agrawal et al., 2016; Eitel et al., 2017). While these approaches are able to learn efficient models for predicting effects on images, including non-planar object interactions, they are still limited to the perceived 2D image plane. Zeng et al. (2018) use a height map with color information as input and can, therefore, incorporate depth as well.

Recent works have applied implicit models effectively to determine action parameters using start and goal state as input. The goal of such reinforcement learning models is to determine actions that yield the desired effects instead of predicting action effects explicitly. Discriminative approaches rate the fitness of a specific action vector to transform a given initial state into the desired goal state. An action then can be chosen by sampling actions and maximizing this score (Li et al., 2018; Eitel et al., 2017; Zeng et al., 2018). Contrary, a regression approach predicts the action vector itself given start and goal state (Agrawal et al., 2016).

In order to handle an arbitrary number of objects, graph neural networks as discussed in section 2.3 have been used in a few works. However, these approaches still only consider planar movement or are limited to object movement in the image plane. Janner et al. (2019) learn an object-centric physics model by training a graph neural network using a physics simulation. The input is a rendered 2D image of blocks, where one block is suspended in the air. Then, the physics model predicts how the image will look after the block has fallen. In the experiments, the robot lets a block fall onto other blocks, mimicking the simulations. Tekden et al. (2020) consider planar motion of multiple connected rigid objects. A graph neural network is used, where the vertices encode an arbitrary number of objects and the edges represent the connection information.

2.4.2. Summary and Review

Table 2.3 gives an overview and compares related works in the area of action effect prediction. It illustrates the differences between related approaches regarding the following aspects:

- **Method:** *Model*-based approaches use an analytical physics model to describe and predict object interactions and motions. In contrast, *data*-driven approaches learn a model from training data and do not rely on an explicit physics model. If major parts of an approach require a physics model, but other parts are learned, we call it a *hybrid* method.
- **Explicit:** An *explicit* prediction model produces the state after action execution as an output, whereas an *implicit* approach learns an internal prediction model, which does not produce human-interpretable results.
- **Number of Objects:** Many approaches are designed to predict the motion of a *single* object. If the approach can handle more than one object but still limits the number of objects it can handle a priori, either by design or during training time, we categorize the number of objects as *fixed*. If the maximum number of objects is not inherently limited, we say that such an approach can handle *multiple* objects.
- **Dimension:** We call an approach *2D* if it constrains objects to planar motion. If an approach explicitly models non-planar motions, we categorize it

as *3D*. Note, that this does not require the approach to model full 6D transformations. Some approaches only process 2D images, which are nonetheless able to handle specific non-planar cases. We label such approaches *2D**.

- **Input and Output:** Common representations for the input and output of an approach include images (color and/or depth), object segmentation masks (short: *mask*), object poses and pose changes. We use the term *action vector* to refer to the encoding for parametrizing an action, e. g. start point, direction, and length of a push.

While model-based and simulation-based methods give very accurate results, they require *a-priori knowledge* about the relevant physical properties. In this work, we do not assume that knowledge about physical properties like friction or inertial moment is known. Hybrid approaches are able to deal with some unknown object properties by learning them during interaction. However, similar to model-based approaches, existing methods are still limited to actions applied to a *single object*.

Implicit data-driven approaches use reinforcement learning to *choose actions* in complex manipulation scenarios. However, these approaches do not predict the scene state after action execution directly. Therefore, they lack interpretability and explainability.

Explicit data-driven approaches are promising, since they are able to handle more complex scenes with more than one object. Often, these approaches are still limited to a *fixed number of objects* due to the use of a constant number of segmentation masks. Image-based approaches are able to handle an arbitrary number of objects, but the motion of objects is constrained to the image plane. Furthermore, the object state needs to be extracted from the predicted image.

There are a few data-driven approaches, which have used graph neural networks to predict object motion. This way, the limitation to a fixed number of objects can be lifted. Existing works have used graph neural networks to successfully predict planar object motion.

Table 2.3.: Comparison of Action Effect Prediction Approaches

Reference	Method	Explicit	# Objects	Dim.	Input	Output
Mason (1986)	Model-based	Yes	Single	3D	Push velocity	Direction of rotation
Lynch et al. (1992)	Model-based	Yes	Single	2D	Push velocity	Object velocity
Hogan and Rodriguez (2020)	Model-based	Yes	Single	2D	End-effector motion	Object velocity
Zhou et al. (2018)	Hybrid	Yes	Single	2D	End-effector motion	Object velocity
Wu et al. (2017)	Hybrid	Yes	Multiple	2D*	Color images and action vector	Color images
Kloss et al. (2020)	Hybrid	Yes	Single	2D	Depth image and action vector	Object pose
Weitnauer et al. (2010)	Simulation	Yes	Single	2D	Object Poses	Object Poses
Mösenlechner and Beetz (2011)	Simulation	No	Multiple	3D	Object Poses	Stability
Kunze and Beetz (2017)	Simulation	Yes	Single	3D	World State	World State
Fromm and Birk (2016)	Simulation	No	Multiple	3D	Object Poses	Manipulation Cost
Omrčen et al. (2009)	Data-driven	Yes	Single	2D	Mask	Object velocity
Kopicki et al. (2011)	Data-driven	Yes	Single	3D	Object Poses	Relative pose change
Elliott et al. (2016)	Data-driven	Yes	Single	2D	Object pose and action vector	New object pose
Byravan et al. (2017)	Data-driven	Yes	Fixed	3D	Point cloud and action vector	Point cloud
Finn et al. (2016)	Data-driven	Yes	Fixed	2D*	Image	Image
Tekden et al. (2020)	Data-driven	Yes	Multiple	2D	Object Poses	Object Poses
Mottaghi et al. (2016)	Data-driven	Yes	Multiple	2D*	Image and force vector	Image
Janner et al. (2019)	Data-driven	Yes	Multiple	2D*	Image	Image
Li et al. (2018)	Data-driven	No	Single	2D	Masks and action vector	Similarity score
Eitel et al. (2017)	Data-driven	No	Multiple	2D*	Image and action vector	Success probability
Agrawal et al. (2016)	Data-driven	No	Multiple	2D*	Images	Action vector
Zeng et al. (2018)	Data-driven	No	Multiple	3D	Height map	Dense expected reward
Nematollahi et al. (2020)	Data-driven	Yes	Fixed	3D	Point cloud and action vector	Point cloud
This thesis	Data-driven	Yes	Multiple	3D	Object Poses	Object Poses

In this thesis, an explicit action effect prediction is required, which also increases interpretability of the prediction model. Data-driven methods seem most promising due to their ability to learn complex object interactions. In the area of data-driven methods, graph neural networks can handle multiple interacting objects without being limited to a fixed number of objects. Contrary to existing works, this thesis considers full 3D object motions, which has not been addressed in the context of explicit action effect prediction for multiple interacting objects.

2.5. Deformable Object Dynamics

The previous section discussed action effect prediction in general. However, most of the presented approaches only deal with rigid objects. Deformable object like cloths, bags and ropes require additional modeling, since a single pose is not sufficient to describe their configuration. This section first discusses existing simulation environments that were designed to handle deformable objects explicitly. Then, prediction methods tailored for deformable object interactions are presented.

2.5.1. Simulation Environments

Multiple simulation environments are capable of modeling and simulating deformable objects. They were either directly designed for or have been extended to handle deformable objects.

SoftGym: SoftGym¹ is a collection of open-source simulation benchmarks for manipulation of deformable objects (Lin et al., 2020). The goal of SoftGym is to facilitate reproducible research in the area of Reinforcement Learning (RL) for deformable objects. Internally, SoftGym is based on the Nvidia FleX physics simulator, which models deformable objects in a particle and position based dynamical system (Müller et al., 2007; Macklin et al., 2014).

¹<https://sites.google.com/view/softgym>

The SoftGym benchmark suite contains different tasks, in which the goal is to manipulate either a rope, a cloth/towel or fluids. The manipulators range from points on the deformable object, which can be controlled freely, to a robot arm that manipulates the object.

SOFA: Simulation Open Framework Architecture (SOFA)² is an open-source framework for implementing physics simulations. Its first application was simulation of the human organs during surgery (Allard et al., 2007). SOFA contains different methods for modeling deformable objects: mass-spring systems, Finite Element Method (FEM) and free-form deformation grids.

Although SOFA has mostly been used for simulating soft tissue for medical applications (Faure et al., 2012), these methods can also be used to model robotic manipulation of other deformable objects like cloths (Yin et al., 2021), dismantling electronic devices (Suárez-Hernández et al., 2020), or tracking the state of foam-like objects (Petit et al., 2015) and plush toys (Lagneau et al., 2020).

MuJoCo: Multi-Joint dynamics with Contact (MuJoCo)³ is a physics simulator designed for model-based optimization of contact-rich manipulation tasks (Todorov et al., 2012). While it is mostly used to model dynamics of robot arms and rigid objects, it can also simulate deformable objects as collection of regular rigid bodies connected through joints, tendons or soft constraints. In MuJoCo, these composite objects can be used to model ropes, cloth or other objects (e. g. Wu et al., 2020a).

Unity with Obi Cloth: Unity⁴ is a 3D engine focused on real-time rendering applications, e. g. games, animation, architecture, etc. Various extensions for specific use cases are available for Unity, including the Obi Cloth⁵ extension, which focuses on the simulation of deformable cloth-like objects. Figure 2.14 shows examples of a cloth interacting with rigid objects. Obi Cloth uses a position and particle based simulation similar to SoftGym. It is fully integrated into

²<https://www.sofa-framework.org>

³<https://mujoco.org/>

⁴<https://unity.com>

⁵<http://obi.virtualmethodstudio.com/>

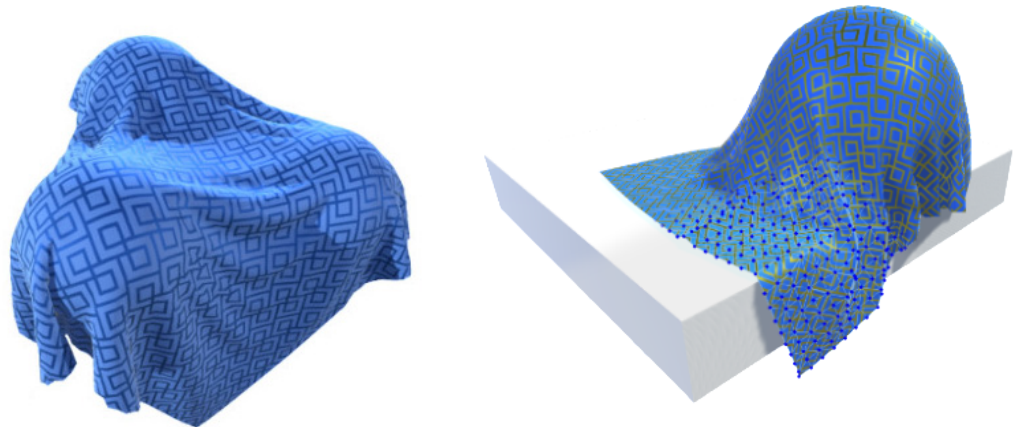


Figure 2.14.: Examples of deformable objects rendered in Unity and simulated using the Obi Cloth extension. The left image shows a cloth that covers multiple spheres. On the right, the cloth covers a single sphere on a block. *(Images are taken from the Obi Cloth website)*

Unity and therefore allows to use the integrated editor tools for creation and manipulation of scenes containing deformable and rigid objects.

2.5.2. Prediction Methods

There are two types of methods for predicting complex dynamics of deformable objects as a result of an action. On the one hand, **model-based** methods try to model the dynamics of deformable objects analytically, determine the model parameters, and use the parametrized model for prediction. On the other hand, **data-driven** methods collect data about action effects from simulation or real interactions, learn the dynamics of deformable object interactions from this data, and use the learned model for prediction.

Model-based Methods: Traditional methods for cloth dynamics are based on analytical modeling (see Hou et al. (2019) and Yin et al. (2021) for a review). One popular approach for analytical modeling is FEM, which mostly applies to fabrics of simple shapes if real-time simulations are required (Sanchez et al., 2018).

Another approach is to construct a particle-based simulation system based on the measured physical properties like friction, mass, elasticity, bending, etc. (Luiblé and Magnenat-Thalmann, 2008). However, these methods are computationally expensive, especially when the geometric and topological structure is complex.

Data-driven Methods: Data-driven methods rely on learning the dynamics of deformable object interactions from data. Here, the dynamics are captured without explicitly measuring and modeling the object properties. Recently, many research works addressed rope dynamics modeling. Battaglia et al. (2016) investigate graph neural networks and learn the dynamics of a simulated rope environment. Watters et al. (2017) use a front-end network to encode the visual input as latent representations and build a dynamics estimator based on interaction network structure. Yan et al. (2020) take images as input and use a neural network to encode the rope state as a set of connected nodes, and apply a bi-directional LSTM to capture the dynamics based on the node representations. For 2D cloth-like objects, physics simulators and deep neural networks have been combined into a prediction model (Oh et al., 2018; Lee et al., 2019). These works use the physics simulation to predict coarse motion and refine the prediction results using deep neural networks. Hafner et al. (2019) encode input images to a latent representation using an autoencoder and predict the future state based on a recurrent neural network structure. The proposed approach is evaluated in SoftGym (Lin et al., 2020).

2.5.3. Summary and Review

Now, we summarize and review the results of the previous sections and identify promising methods that will be utilized in this thesis. Concretely, simulation environments and prediction methods for deformable objects will be discussed.

Simulation Environments: The presented simulation environments use different methods to model and simulate deformable objects. Common methods include FEM and particle-based simulations.

The purpose of SoftGym is to serve as a benchmark for further research about reinforcement learning for manipulation tasks. New tasks and scenes can be designed by hand. However, creating many randomized scenes is not the main focus of SoftGym and therefore not well supported. SOFA is used for both medical and robotic applications. Medical applications include modeling soft tissue and organs in the human body, while robotic applications include the manipulation of various soft objects and dismantling electronic devices. The framework is flexible and extensible, so that deformable objects like cloths or bags can be modeled and simulated but require a lot of parameter tuning to achieve realistic simulation behavior. MuJoCo is designed to handle contact-rich interactions between actuated robots and rigid objects. While it can model deformable objects via joints, tendons and soft constraints, it has been found that the simulated behavior for deformable objects does not accurately model reality in more complex manipulation tasks (Hoque et al., 2020). Unity with the Obi Cloth extension offers good customization capabilities and tooling due to Unity's wide spread use for different applications. It is easy to automate the process of generating new randomized scenes. Therefore, we have chosen Unity with the Obi Cloth extension as a simulator for deformable object interactions in this thesis.

Prediction Methods: Model-based and data-driven prediction methods for deformable object dynamics have been presented. Model-based methods allow analytical modeling of different cloth-like and deformable objects. However, they require a priori knowledge about physical parameters and are often not real-time viable. Data-driven methods learn prediction models from data. They do not require complete knowledge about physical properties and can be evaluated efficiently. However, they may require a lot of training data, which often is generated in simulation.

All presented methods have their limitations when faced with more complex scenarios. First, most of these works are devoted solely to modeling 1D linear objects like ropes or cables. When 2D cloth-like objects are considered, the authors implicitly assume that their topology is simple. Second, the considered actions are typically restricted to picking and placing. In this thesis, we study the dynamics of complicated deformable objects interacting with multiple rigid objects, as well as a rich set of actions.

2.6. Discussion

This chapter presented and reviewed the related work regarding central topics of this thesis. Several promising approaches, which will be applied, extended or evaluated in this work, were discussed and research gaps addressed by this thesis were identified.

Relational Scene Representation: Representations for object relations can describe geometric properties like spatial relation or encode physical properties like support relations. Relation models can be deterministic or probabilistic, i. e. assign existence probabilities to relations. Representations with semantic and physical grounding were discussed. For robotic applications, physical grounding is necessary to plan robust manipulation sequences. This thesis contributes a probabilistic representation for support relations with physical grounding.

Support Relation Extraction: The review of different support relation extraction methods included feature-based, rule-based, data-driven, simulation-based and force analysis approaches. Out of these methods, simulation-based and force analysis are grounded physically, making them suitable in the context of robot manipulation tasks. A remaining challenge is the incorporation of uncertainty about object geometries and poses. To this end, this thesis implements and evaluates a probabilistic extraction method for support relations extending existing force analysis approaches.

Learning with Relational Representations: Using relational representations as input and output poses a challenge for traditional machine learning methods, which only support fixed sized input and output and whose results are dependent on the order of the input data. Graph neural networks address these problems. The input and output of graph neural networks are attributed graphs, with a variable number of vertices and edges. Through careful choice of aggregation functions, order-invariance can be achieved. Graph neural networks have been successfully applied to physics prediction tasks with varying number

of interacting objects. We have identified graph neural networks and in particular the Encode-Process-Decode architecture as a promising avenue for action effect prediction with multiple interacting objects.

Action Effect Prediction: We discussed model-based, simulation-based, hybrid and data-driven action effect prediction methods. Existing works are either limited to scenes with a fixed number of objects or only consider planar object motion. To address this research gap, this thesis proposes an explicit action effect prediction for 3D motions and multiple interacting rigid objects.

Deformable Object Dynamics: Different simulation environments for deformable object dynamics were discussed. This thesis will use Unity with the Obi Cloth extension as simulator for deformable object interactions due to its customization capabilities and the ability to programmatically generate randomized scenes easily. Furthermore, model-based and data-driven prediction methods for deformable objects have been compared. In this thesis, we propose a graph-based representation for deformable object interactions and learn an action effect prediction model for various tasks.

In summary, this chapter has reviewed relevant topics, analyzed promising methods and tools, and identified existing research gaps, which lead to the contributions in the following chapters of this thesis. A novel probabilistic relational scene representation and support relation extraction method are proposed and implemented in chapter 3. Chapter 4 presents action effect prediction for both rigid and deformable objects based on relational scene representations using graph neural networks. In chapter 5, applications of these methods to humanoid robotics are presented.

3. Representation and Extraction of Support Relations

Robots operating in dynamic and cluttered environments must be able to infer a physically plausible scene representation, which allows to leverage the environment for manipulation tasks and ensures successful task execution. In cluttered scenes, pure geometric reasoning about the scene is insufficient to plan and execute actions. Therefore, the robot must be able to acquire and utilize knowledge about the scene’s structure, i. e. how objects physically interact with each other, in order to generate feasible and safe action plans.

Human perception automatically combines visual features, prior knowledge, and basic physical constraints into a plausible model of the scene. The latter part is known as naive physics (Hayes, 1978; Vosniadou, 2002) since the human brain does not accurately model all physical phenomena at work. Instead, a simplified model based on prior knowledge about action-effect relations is employed when acting in the world. Furthermore, by identifying ambiguities and uncertainties in their scene understanding, humans are able to interact with the environment to verify hypotheses about objects, their relations as well as possible interactions with them, and to acquire new knowledge about the scene structure. Inspired by the concept of naive physics, we have investigated how a robot can extract physically grounded support relations between entities in the scene based on geometric reasoning. Since the proposed approach does not require prior knowledge about the physical properties of involved objects, we need to explicitly model the uncertainty of objects and support relations.

In this chapter, we propose a representation and extraction method for probabilistic support relations, see Figure 3.1. The pipeline starts with an input point cloud of the scene, which is segmented using state-of-the-art segmentation meth-

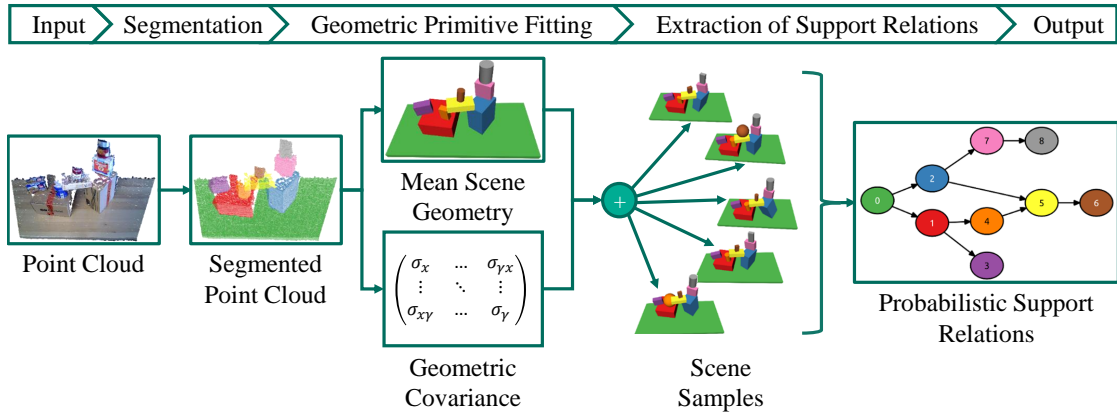


Figure 3.1.: Overview of the pipeline for extracting probabilistic support relations from an input point cloud of a scene.

ods, i. e. Locally Convex Connected Patches (LCCP) (Stein et al., 2014). Then, geometric primitive fitting generates a probability distribution over object shapes and poses in the scene. By sampling from this probability distribution and analyzing forces between objects, probabilistic support relations can be extracted. The chapter addresses the following three problems:

- **Probabilistic Object and Support Relation Representation:** How to represent objects and their support relations while capturing the uncertainty in object shapes, poses, and relation existence?
- **Probabilistic Geometric Primitive Fitting:** How to fit probabilistic geometric shape models to a point cloud of a scene captured by the robot?
- **Extraction of Probabilistic Support Relations:** How to extract probabilistic support relations from geometric shape models?

The chapter is structured as follows. First, section 3.1 describes a novel probabilistic representation for object shapes, poses and support relations. Section 3.2 presents a method for fitting geometric primitives to a point cloud while taking into account shape and pose uncertainty. In section 3.3, a method for extracting physically plausible support relations from a probabilistic object representation is described. Finally, section 3.4 evaluates the proposed methods based on existing and newly recorded datasets. Parts of this chapter have been published in Paus and Asfour (2021) and Kartmann et al. (2018).

3.1. Probabilistic Object and Support Relation Representation

We decompose a scene into a set of geometric primitives. To capture the uncertainty about the parameters of these primitives, we define a probability distribution over primitive shapes and their poses. Furthermore, we define physically grounded support relations between objects. Objects and relations are then combined into a support graph, whose nodes contain the object parameters and whose edges contain the existence probability for a support relation.

Geometric Primitives: We assume that the shape of an object in a scene can be approximated using geometric primitives. In this work, a geometric primitive can be a box, a cylinder, or a sphere, but the set of primitives can be extended to other shape primitives. Each primitive $p_i = (t_i, \mathbf{x}_i)$ is defined by a primitive type $t_i \in \{\text{Box}, \text{Cylinder}, \text{Sphere}\}$, and a state vector $\mathbf{x}_i \in \mathbb{R}^{N(t_i)}$. The state vector parametrizes the geometry of a primitive, e. g. a sphere is parameterized by four variables $\mathbf{x}_i = (c_x, c_y, c_z, r)^T$: a three-dimensional center point $(c_x, c_y, c_z)^T$ and a radius r . The size $N(t_i)$ of the state vector depends on the primitive type:

- $N(\text{Box}) = 10$ (position, orientation, extents)
- $N(\text{Cylinder}) = 8$ (position, direction, radius, height)
- $N(\text{Sphere}) = 4$ (position, radius)

A scene consists of a set of n objects $\mathcal{O} = \{o_1, o_2, \dots, o_n\}$.

Probability Distribution over Geometric Primitives: The geometry of each object o_i is represented as a joint probability distribution $P(t_i, \mathbf{x}_i \mid o_i)$ over the primitive type t_i and state vector \mathbf{x}_i with $i \in [1, n]$. Since the primitive type and state vector are dependent variables, we can represent the joint probability distribution as the product of a discrete distribution over primitive types $P(t_i \mid o_i)$ and a dependent distribution over state vectors $P(\mathbf{x}_i \mid t_i, o_i)$:

$$P(t_i, \mathbf{x}_i \mid o_i) = P(t_i \mid o_i) \cdot P(\mathbf{x}_i \mid t_i, o_i)$$

We choose a multi-variate Gaussian distribution to represent $P(\mathbf{x}_i | t_i, o_i)$:

$$P(\mathbf{x}_i | t_i = \text{Box}, o_i) \sim \mathcal{N}(\mu_{i,\text{Box}}, \Sigma_{i,\text{Box}}) \quad (3.1)$$

$$P(\mathbf{x}_i | t_i = \text{Cylinder}, o_i) \sim \mathcal{N}(\mu_{i,\text{Cylinder}}, \Sigma_{i,\text{Cylinder}}) \quad (3.2)$$

$$P(\mathbf{x}_i | t_i = \text{Sphere}, o_i) \sim \mathcal{N}(\mu_{i,\text{Sphere}}, \Sigma_{i,\text{Sphere}}) \quad (3.3)$$

Note that the mean $\mu_{i,t_i} \in \mathbb{R}^{N(t_i)}$ and the covariance $\Sigma_{i,t_i} \in \mathbb{R}^{N(t_i) \times N(t_i)}$ have different dimensions corresponding to the primitive type t_i and the corresponding state vector dimension. Furthermore, the state vector \mathbf{x}_i includes both pose and shape parameters, i. e. we do not assume that the shape and pose are independent.

Given a scene with a set of objects \mathcal{O} , we represent the probability distribution over the complete scene geometry as a joint distribution over independent object geometries. Therefore, we can express the geometric scene distribution as the product of distributions for the set of primitives $\mathcal{P} = \{p_1, p_2, \dots, p_n\}$:

$$P(\mathcal{P} | \mathcal{O}) = \prod_{i=1}^n P(p_i | o_i) = \prod_{i=1}^n P(t_i, \mathbf{x}_i | o_i)$$

Definition of support relations: A support relation is a binary relation between objects. Since we want to analyze which objects are physically supported by other objects in order to determine safe manipulation sequences, we rely on physical grounding for our definition (similar to Mojtahedzadeh et al. (2015)):

Given an object set \mathcal{O} , a support relation is a binary relation $\text{SUPP} \subseteq \mathcal{O} \times \mathcal{O}$. For two objects $A, B \in \mathcal{O}$, if A supports B then $(A, B) \in \text{SUPP}$. Object A supports object B if and only if removing A causes B to lose its motionless state.

We use the terminology “object A supports object B” and “support exists between object A and object B” interchangeably. For shorter notation, we define

$$\text{SUPP}(A, B) = (A, B) \in \text{SUPP}, \quad A, B \in \mathcal{O}.$$

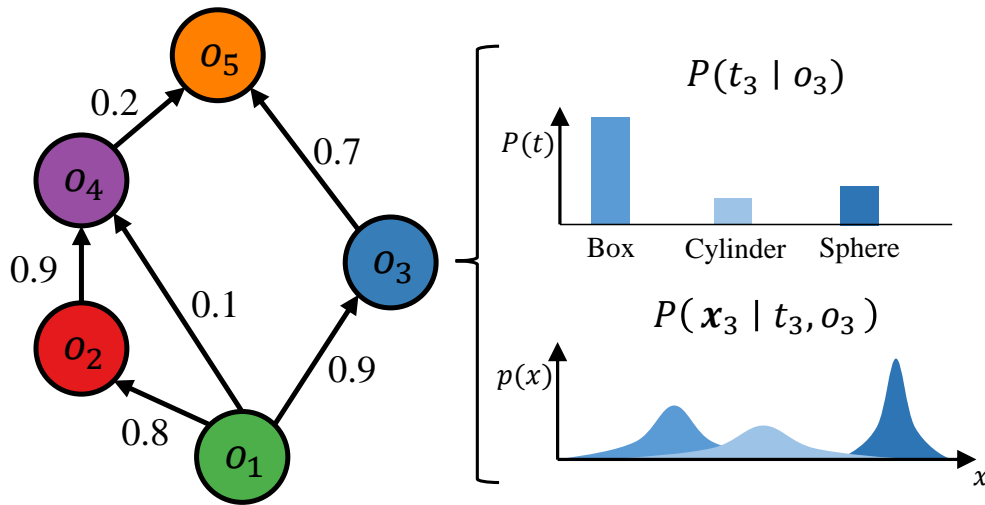


Figure 3.2.: A probabilistic support graph for the object set $\mathcal{O} = \{o_1, o_2, o_3, o_4, o_5\}$. Each vertex contains the parameters for distributions over primitive type and state vector. Each edge is annotated with the existence probability of a support relation between the corresponding objects.

The existence probability of a support relation $\text{SUPP}(A, B)$ depends not only on the objects A and B , but also on other objects in \mathcal{O} :

$$P(\text{SUPP}(A, B) \mid \mathcal{O}) = P((A, B) \in \text{SUPP} \mid \mathcal{O}), \quad A, B \in \mathcal{O}$$

We can now represent objects and the corresponding support relations as a directed graph $\mathcal{G} = (V, E)$, in which the vertices $V = \mathcal{O}$ represent objects and the edges $E = \text{SUPP}$ represent support relations. The graph \mathcal{G} is called a *deterministic* support graph. By considering the probability distributions over the scene geometry and support relations, we can define a *probabilistic* support graph $\mathcal{G}_{\text{prob}} = (V_{\text{prob}}, E_{\text{prob}})$ where each vertex $v_i \in V_{\text{prob}}$ includes the parameters of the joint probability distribution $P(t_i, \mathbf{x}_i \mid o_i)$ and each edge $e_i \in E_{\text{prob}}$ includes the existence probability of a support relation $P(e_i \in \text{SUPP} \mid \mathcal{O})$. Figure 3.2 shows an example of a probabilistic support graph.

3.2. Probabilistic Geometric Primitive Fitting

Here, we describe how the probabilistic object representation can be extracted from an input point cloud of a scene. First, we segment the input point cloud using the LCCP algorithm (Stein et al., 2014). Based on the segmented point cloud, we propose a method for extracting a probability distribution over possible object geometries using a modified Random Sample Consensus (RANSAC) algorithm. After describing how we use RANSAC for fitting geometric primitives like spheres, cylinders and boxes, a novel way for quantifying the uncertainty in the extracted geometry is proposed.

Problem Formulation

A colored point cloud C consists of m points

$$C = \{\mathbf{p}_j = (x_j, y_j, z_j, c_j) \mid j \in [1, m]\},$$

where each point contains position (x_j, y_j, z_j) and color c_j .

A segmentation \mathcal{S} separates a point cloud into n disjunct segments.

$$\mathcal{S} = \{S_i \subseteq C \mid i \in [1, n]\}, \quad \bigcup_{i \in [1, n]} S_i = C, \quad \forall i, j \text{ with } i \neq j : S_i \cap S_j = \emptyset$$

Given a segmentation \mathcal{S} , we want to determine the probability distribution over geometric primitives $P(\mathcal{P} \mid \mathcal{O})$. We assume that each segment S_i corresponds to an object o_i and, therefore, a geometric primitive p_i in the scene, i. e. $|\mathcal{O}| = |\mathcal{P}| = |\mathcal{S}| = n$. This problem can be further subdivided into finding the probability distribution over primitive type t_i and geometric parameters \mathbf{x}_i for each object o_i given its corresponding segment S_i :

$$P(t_i, \mathbf{x}_i \mid o_i) = \text{FitGeometricPrimitive}(S_i)$$

Geometric Primitive Fitting using RANSAC

Given a segment S , we want to determine the best fitting geometric parameters \mathbf{x} for all possible primitive types $t \in \{\text{Box}, \text{Cylinder}, \text{Sphere}\}$ as well as an error

Algorithm 1: Geometric Primitive Fitting using RANSAC

Input: S : Point cloud segment
 j_{\max} : Maximum number of iterations
 n_{MSS} : Number of points in the Minimum Sample Set
 δ : Inlier error threshold

Output: \mathbf{x}_{best} : Best fitting model parameters

$\mathbf{x}_{\text{best}} = \mathbf{0}$;
 $\varepsilon_{\text{best}} = \infty$;

for $j \in [1, j_{\max}]$ **do**
 $MSS = \text{selectRandomSubset}(S, n_{MSS})$;
 $\mathbf{x}_{\text{fit}} = \text{fitModel}(MSS)$;
 $I = \emptyset$;
 for $p_k \in S$ **do**
 $\varepsilon = \text{computePointError}(\mathbf{x}_{\text{fit}}, p_k)$;
 if $\varepsilon < \delta$ **then**
 $I = I \cup \{p_k\}$;
 end
 end
 $\varepsilon_{\text{inlier}} = \text{computeInlierError}(\mathbf{x}_{\text{fit}}, I)$;
 if $\varepsilon_{\text{inlier}} < \varepsilon_{\text{best}}$ **then**
 $\varepsilon_{\text{best}} = \varepsilon_{\text{inlier}}$;
 $\mathbf{x}_{\text{best}} = \mathbf{x}_{\text{fit}}$;
 end
end
return \mathbf{x}_{best} ;

measure. Since we only consider a single segment, we omit the object index i for easier notation.

RANSAC (Derpanis, 2010; Bolles and Fischler, 1981) is a method for model parameter estimation under the assumption that a given set of observed data contains outliers. It is a probabilistic and iterative algorithm, where the probability to find the best fitting parameters increases with the number of iterations. In our case, the model parameters \mathbf{x} describe the pose and shape of geomet-

ric primitives and the observed data are points $\mathbf{p}_j \in S$ in the segmented point cloud S .

An implementation of RANSAC for geometric primitive fitting is depicted in algorithm 1. Central part is the loop, which iterates until the limit j_{\max} is reached. During each iteration, a Minimum Sample Set (MSS) is drawn from all points in S . The MSS has a fixed size n_{MSS} , which is the minimum number of points necessary to estimate the model parameters of a geometric primitive. Note that this number varies between primitive types. The model parameters \mathbf{x}_{fit} for the selected MSS are computed. Then, the inlier set $I \subseteq S$ is determined. The inlier set contains all points $p_k \in S$, which have an error smaller than the threshold δ to the computed model. As a final step, the error over all inliers is computed. If this inlier error is smaller than the error of the current best model, we update the best model accordingly. We use mean quadratic distance of the inlier points to the surface of the fitted geometric primitive as the error function. The implementation of fitting the model for a given MSS is different for each geometric primitive type. In the following, we describe the procedure for the different primitive types.

- A sphere can be described by four parameters $\mathbf{x}_{\text{Sphere}} = (c_x, c_y, c_z, r)$ with the center point $\mathbf{c} = (c_x, c_y, c_z)$ and the radius r . Each point $\mathbf{p} \in \mathbb{R}^3$ on the surface of a sphere fulfills the following equation:

$$\|(\mathbf{p} - \mathbf{c})\|^2 = r^2 \quad (3.4)$$

Determining a sphere's parameters requires a MSS with four distinct points and solving equation 3.4 for \mathbf{c} and r .

- A cylinder is parametrized by a center point $\mathbf{c} = (c_x, c_y, c_z)$, a central axis direction $\mathbf{a} = (a_x, a_y, a_z)$ with $\|\mathbf{a}\| = 1$, a radius r , and a height h . By estimating the surface normals for each point in S , two points with normals can be used as a MSS for cylinders. First, the intersection of two lines along the normal vectors through the two points is calculated as the center point \mathbf{c} of the cylinder. In a next step, the vector perpendicular to the plane both points lie in, is calculated as the axis direction \mathbf{a} of the cylinder. The radius r is the distance of either point from the MSS to the central axis line $L : \mathbf{c} + t \cdot \mathbf{a}$. To determine the height h of the cylinder, we project the inliers

onto the line L and determine the distance between the furthest projected inliers.

- A box can be described by its central position $\mathbf{c} = (c_x, c_y, c_z)$, orientation $\mathbf{q} \in \mathbb{H}$ as a quaternion, and the extents $\mathbf{e} = (e_x, e_y, e_z)$. Garcia (2009) proposed an approach for extracting boxes from point clouds, that requires all six faces of a box to be visible. We modified this approach to only require two visible faces, making it viable to be used with point clouds captured from a single view point. The MSS for boxes contains five points $MSS = \{\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4, \mathbf{p}_5\}$. First, a plane P_1 through the first three points $\mathbf{p}_1, \mathbf{p}_2$ and \mathbf{p}_3 is constructed.

$$P_1 : \mathbf{p}_1 + s \cdot (\mathbf{p}_2 - \mathbf{p}_1) + t \cdot (\mathbf{p}_3 - \mathbf{p}_1) \quad s, t \in \mathbb{R}$$

Note that this requires $\mathbf{p}_1, \mathbf{p}_2$, and \mathbf{p}_3 to not be co-linear. Then a second plane P_2 is constructed from the remaining points \mathbf{p}_4 and \mathbf{p}_5 and the projection of \mathbf{p}_4 onto the plane P_1 .

$$P_2 : \mathbf{p}_4 + s \cdot (\mathbf{p}_5 - \mathbf{p}_4) + t \cdot (\text{projection}(\mathbf{p}_4, P_1) - \mathbf{p}_4) \quad s, t \in \mathbb{R}$$

This requires \mathbf{p}_4 and \mathbf{p}_5 to not lie on the plane P_1 .

We can now construct the three normals of the box with planes P_1 and P_2 .

$$\begin{aligned} \mathbf{n}_1 &= \frac{(\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_3 - \mathbf{p}_1)}{\|(\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_3 - \mathbf{p}_1)\|} \\ \mathbf{n}_2 &= \frac{(\mathbf{p}_5 - \mathbf{p}_4) \times (\text{projection}(\mathbf{p}_4, P_1) - \mathbf{p}_4)}{\|(\mathbf{p}_5 - \mathbf{p}_4) \times (\text{projection}(\mathbf{p}_4, P_1) - \mathbf{p}_4)\|} \\ \mathbf{n}_3 &= \mathbf{n}_1 \times \mathbf{n}_2 \end{aligned}$$

In order to define the plane equations for three sides of the box, we then determine the most frequent displacements d_1, d_2 and d_3 given the normals $\mathbf{n}_1, \mathbf{n}_2$ and \mathbf{n}_3 and the points in the segment S . We use a histogram with 128 bins for this task and select the displacement with the maximum bin count. Each normal \mathbf{n}_i and corresponding displacement d_i define a side plane of the box. The three normals determine the orientation $\mathbf{q} \in \mathbb{H}$ of the box. For each plane, the set of inlier points $I_i \subseteq S$ is determined, whose distance to the plane is below the threshold δ .

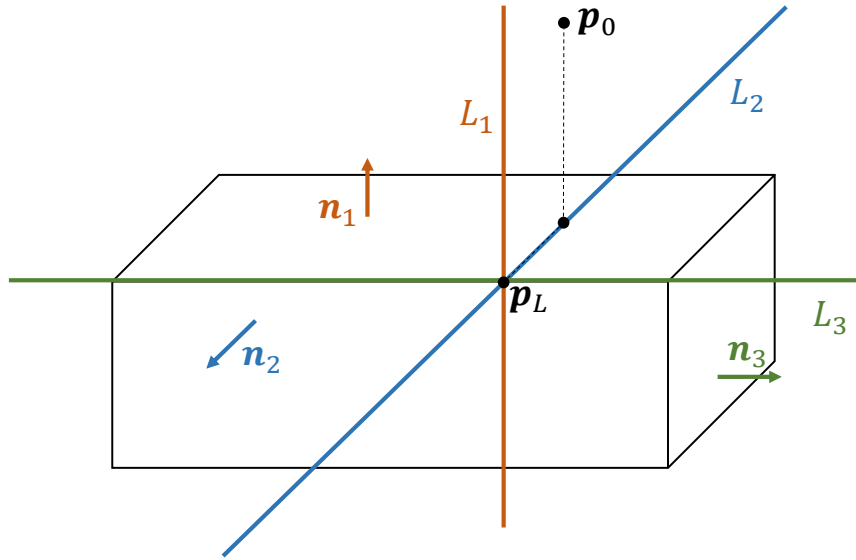


Figure 3.3.: By projecting an inlier point \mathbf{p}_0 onto planes P_1 and P_2 , we compute the line starting point \mathbf{p}_L . The three lines L_1 , L_2 and L_3 go through this point along their respective plane normals \mathbf{n}_1 , \mathbf{n}_2 and \mathbf{n}_3 .

As a final step, we compute the central position \mathbf{c} and the extents \mathbf{e} of the box, given the three planes (\mathbf{n}_i, d_i) and inlier sets I_i . First, we define three lines L_i along the respective normals \mathbf{n}_i that intersect at a single point \mathbf{p}_L that is the result of two consecutive projections of an inlier point \mathbf{p}_0 onto the first two planes (see Figure 3.3).

$$L_i : \mathbf{p}_L + t \cdot \mathbf{n}_i$$

For each line L_i , we project the inlier points I_i onto the line and compute their projection parameters $t \in T_i \subset \mathbb{R}$:

$$T_i = \{t \mid \mathbf{p} = \mathbf{p}_L + t \cdot \mathbf{n}_i, \mathbf{p} \in I_i\}$$

We could now compute the extents of the box as the difference of the maximum and minimum from these projection parameters. Since this can easily be biased by outliers, we use the first and the 99th percentile instead.

$$e_i = P_{99\%}(T_i) - P_{1\%}(T_i)$$

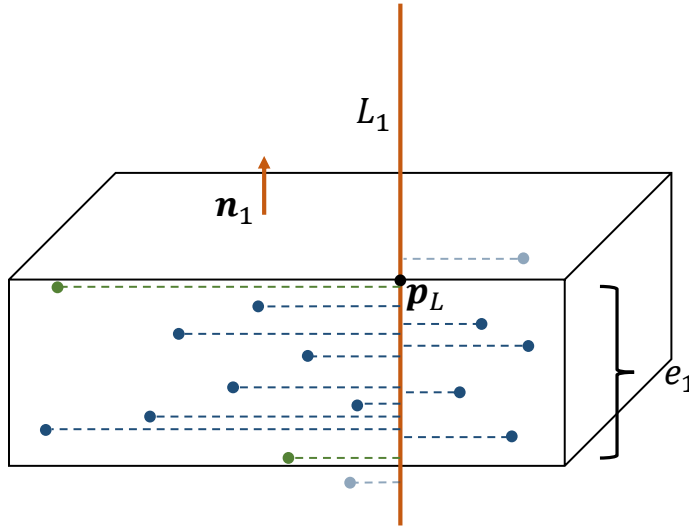


Figure 3.4.: Projection of the blue inlier points onto line L_1 , whose direction is along the plane normal \mathbf{n}_1 . Not all inlier points are drawn for illustration purposes. Points in gray are filtered by the percentile outlier detection. The green points mark the first and the 99th percentile, which are used to compute the size of the box e_1 along L_1 .

Figure 3.4 illustrates the projection for line L_1 . The center point of the box can now be computed starting at the middle point \mathbf{m}_3 of the edge along L_3 and going halfway along the normals:

$$\mathbf{c} = \mathbf{m}_3 - \frac{e_1}{2} \cdot \mathbf{n}_1 - \frac{e_2}{2} \cdot \mathbf{n}_2$$

Quantifying Uncertainty during Geometric Primitive Fitting

We now modify the RANSAC algorithm for geometric primitive fitting to estimate the joint probability $P(t_i, \mathbf{x}_i \mid o_i)$ over primitive type t_i and geometric parameters \mathbf{x}_i .

During the geometric primitive fitting, points are randomly drawn from the segment S_i . For each primitive type, the state vector is estimated based on the drawn points. Then, the number of inlier points and the sum of squared errors

to all points in the segment is computed. Instead of selecting the primitive parameters which produced the maximum number of inliers, we collect all fitted primitive types $t_{i,j}$, state vectors $\mathbf{x}_{i,j}$, and the corresponding sum of squared errors $\varepsilon_{i,j}$ where j indicates the RANSAC iteration count. Given the total number of RANSAC iterations j_{\max} , we can now estimate the discrete probability distribution over primitive types $P(t_i | o_i)$ by counting the occurrences of each concrete primitive type T :

$$P(t_i = T | o_i) = \frac{1}{j_{\max}} \cdot \sum_{j=1}^{j_{\max}} (t_{i,j} = T) \quad (3.5)$$

The error can be used to calculate the weighted mean and covariance for the state vectors per primitive type. For each object o_i and primitive type t_i in each RANSAC iteration j , we assign a normalized weight $w_{i,j}$ using a Boltzmann distribution over the error $\varepsilon_{i,j}$. The constant β is a parameter, which controls the impact of errors on the weight. The the sum of all weights for a particular object η serves as a normalization factor that ensures the sum of all weights equals 1.

$$w_{i,j} = \frac{1}{\eta} \cdot e^{-\varepsilon_{i,j}/\beta}, \quad \eta = \sum_{j=1}^{j_{\max}} e^{-\varepsilon_{i,j}/\beta}$$

Now the weighted means $\mu_{i,\text{Box}}$, $\mu_{i,\text{Cylinder}}$ and $\mu_{i,\text{Sphere}}$ are computed per primitive type T :

$$\begin{aligned} \mu_{i,T} &= \sum_{j=1}^{j_{\max}} w_{i,j} \cdot \mathbf{x}_{i,j} \\ \Sigma_{i,T} &= (\sigma_{j,r}), \quad \sigma_{j,r} = \frac{\sum_{j=1}^{j_{\max}} w_{i,j} \cdot (\mathbf{x}_{j,r} - \mu_{i,T}) \cdot (\mathbf{x}_{i,r} - \mu_{i,T})^T}{1 - \sum_{j=1}^{j_{\max}} w_{i,j}^2} \end{aligned}$$

This defines a multi-variate Gaussian distribution per primitive type as defined in equations 3.1 – 3.3.

$$P(\mathbf{x}_i | t_i = T, o_i) \sim \mathcal{N}(\mu_{i,T}, \Sigma_{i,T}) \quad (3.6)$$

Together with the discrete probability distribution over primitives types, we get the desired joint probability

$$P(t_i, \mathbf{x}_i | o_i) = P(t_i | o_i) \cdot P(\mathbf{x}_i | t_i, o_i). \quad (3.7)$$

In appendix C, we show that the samples generated by RANSAC actually follow a multivariate Gaussian distribution.

3.3. Extraction of Probabilistic Support Relations

Given a probability distribution over geometric primitives in the scene as determined by the previous section, this section proposes a method for extracting support relations and their existence probability. First, a method for extracting support relations from concrete geometric primitives is presented. Then, an extension to propagate the uncertainty from geometric primitives to support relations is proposed.

3.3.1. Deterministic Support Relation Extraction

We want to determine the binary support relation $\text{SUPP} \subseteq \mathcal{P} \times \mathcal{P}$ with physical grounding, given a concrete set of primitives $\mathcal{P} = \{p_1, p_2, \dots, p_n\}$ with $p_i = (t_i, \mathbf{x}_i)$. Figure 3.5 depicts an example scene as a point cloud, the extracted geometric primitives, and the derived support relations.

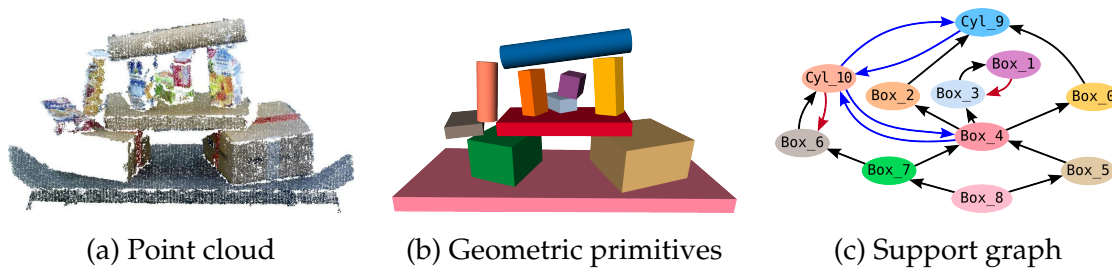


Figure 3.5.: Geometric primitives are extracted from a 3D point cloud captured by an RGB-D camera (a) and used as input. Given the geometric shape and pose of each object (b), we extract binary support relations and represent them as a directed graph (c). Black edges represent certain support, blue edges are unknown and red edges are potential top-down support relations.

Existing force analysis methods (Mojtahedzadeh et al., 2015; Zhang et al., 2019) for extracting support relations are based on Static Equilibrium Analysis (SEA). SEA requires all objects in the scene and the range of their physical properties to be known. Since objects are extracted from point clouds, our method needs to handle incomplete knowledge. To achieve this, we first extract the ACT relation, which describes the forces acting at a contact between two objects due to gravity. Then, we propose a support polygon analysis to capture potential top-down support relations, in which an object above supports another object below it. Finally, we search for suitable parameters of the proposed methods using selected validation scenes.

ACT Relation Analysis

In the first step of the support relation extraction, $ACT \in \mathcal{P} \times \mathcal{P}$ relations as proposed by Mojtahedzadeh et al. (2015) are computed. Let $A, B \in \mathcal{P}$ be two geometric primitives with given type, pose and shape. Motivated by Newton's third law of motion, $ACT(A, B)$ indicates that, due to gravity, A is exerting a force on B . In this case, A is called acting and B is called reacting. First, the contacts between the geometric primitives A and B are computed. To deal with perceptual

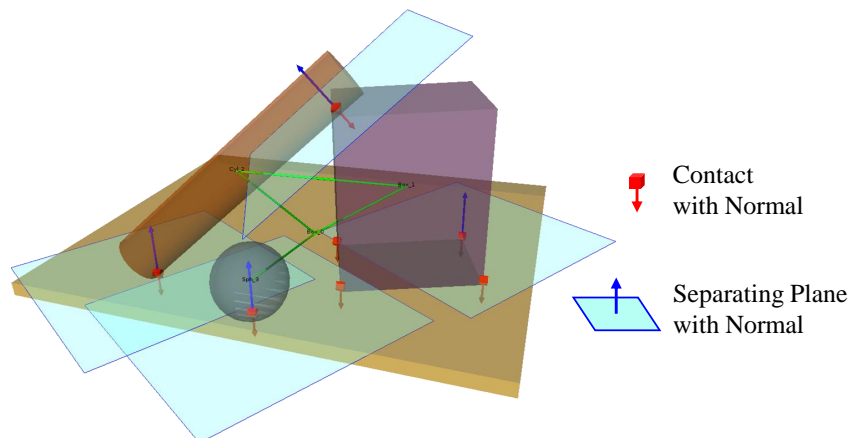


Figure 3.6.: Contact points (red) and separating planes (blue) with their respective normals (arrows) for an example scene. Green lines connect geometric primitives which are in an ACT relation.

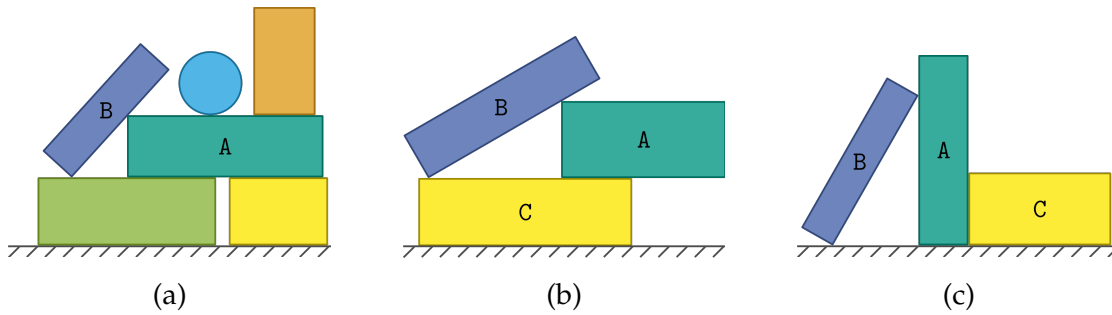


Figure 3.7.: (a) Scene with solely bottom-up support. All support relations are correctly determined by the act hypothesis. (b) Scene with possible top-down support from B to A. Depending on the mass distribution of A, it may fall or not when B is removed. (c) Scene with vertical separating planes. While A supports B, there is no support between A and C.

inaccuracies, we increase the size of the objects by a small margin $\delta_c \in \mathbb{R}_{\geq 0}$ for contact detection. Using the contact points and normals, the plane \mathcal{P}_{sep} separating A and B is constructed. If A is above \mathcal{P}_{sep} and B is below, we set $\text{ACT}(A, B)$, and vice versa if B is above \mathcal{P}_{sep} . Figure 3.6 illustrates contact points and separating planes for an example scene.

When objects are located horizontally next to each other as shown in Figure 3.7c, it might not be clear which object is acting, reacting, or whether a force is exerted at all. This is the case if the separating plane is almost or completely vertical. As an extension to Mojtahedzadeh et al. (2015), we introduce a threshold α_{max} . If the separating plane's rotation angle relative to the vector of gravity is below α_{max} , we define the ACT relations between the respective objects as *unknown*. Whether unknown ACT relations indicate support relations or not, will be investigated in the evaluation.

After computing the ACT relation for each pair of objects, we use it to generate our first support relation hypothesis by

$$\text{ACT}(B, A) \Rightarrow \text{SUPP}(A, B) .$$

In other words, we state that an object B is supported by another object A if B acts on A. This heuristic is valid in many scenes where objects are stacked on

top of each other (see Figure 3.7a), thereby offering a good basis for a support hypothesis. However, there are cases of support, which are not covered by ACT relations. For instance, consider the configuration shown in Figure 3.7b. Clearly, B acts on A, and thus $\text{SUPP}(A, B)$ according to the ACT heuristic. In addition, it seems likely that A falls if B is removed. As A does not act on B, this possible support is not found by the ACT heuristic. By vision alone, one cannot know whether B supports A. If most of A's mass is located at its left side, resting on C, it might not need support from B to be stable under gravity. Still, this uncertainty must be detected and taken into account when executing actions affecting a possible support between B and A. In the following, we propose method for detect such *top-down* support relations.

Support Polygon Analysis

Generally, objects do not only support other objects above them, but can also support objects underneath (see Figure 3.7b). Approaches addressing top-down support typically assume a uniform mass distribution (e. g. Mojtahedzadeh et al. (2015); Fromm and Birk (2016)). We present a purely geometric approach for detecting potential top-down support.

Let $A, B \in \mathcal{P}$ be two geometric primitives with $\text{ACT}(B, A)$, i. e. B acts on A implying an existing support relation $\text{SUPP}(A, B)$. In order to decide whether B may also support A, we consider how likely it is that A falls when B is removed. For example in Figure 3.7a, A is well supported by the objects below it and will not fall when removing B. In Figure 3.7b, however, A is badly supported by the object below it. Since A is at rest nonetheless, it seems likely that it is supported by B.

The key idea is to investigate how well A is supported by the other objects in the scene using a *support polygon analysis*. The approach is detailed in algorithm 2. First, we project A to the ground plane, resulting in a 2D polygon P_A . If an object has round faces or edges, it is approximated by a triangle mesh. Second, each object C supporting A is projected onto the ground plane as well, and its projection polygon P_C is intersected with P_A . Then, the intersecting areas of all supporting objects are combined into the set of polygons

$$\mathbb{P}_A = \{P_A \cap P_C \mid A, C \in \mathcal{P}, \text{SUPP}(C, A)\}$$

Algorithm 2: Support Polygon Analysis

Input: \mathcal{P} : Set of geometric primitives
 ACT: ACT relation on \mathcal{P}
 $r_{s,\min}$: Threshold for support area ratio
Output: SUPP: Support relation on \mathcal{P}
 SUPP = \emptyset ;
for $(B, A) \in \text{ACT}$ **do**
 SUPP = SUPP \cup $\{(A, B)\}$;
 $P_A = \text{projectToGroundPlane}(A)$;
 $\mathbb{P}_A = \emptyset$;
 for $C \in \mathcal{P}$ **do**
 if $(A, C) \in \text{ACT}$ **then**
 $P_C = \text{projectToGroundPlane}(C)$;
 $\mathbb{P}_A = \mathbb{P}_A \cup \{P_C\}$;
 end
 end
 $P_s = \text{Conv}(\mathbb{P}_A)$;
 $r_s = \text{area}(P_s) / \text{area}(P_A)$;
 if $r_s < r_{s,\min}$ **then**
 SUPP = SUPP \cup $\{(B, A)\}$;
 end
end
return SUPP;

representing the directly supported area of A. The support polygon P_s is the convex hull of the polygons in \mathbb{P}_A . Figure 3.8 visualizes the construction of the support polygon in an example scene.

Finally, the support area ratio r_s is computed as the proportion of the supported area of A to its total area

$$r_s = \frac{\text{area}(P_s)}{\text{area}(P_A)} .$$

Note that $r_s \in [0, 1]$, where $r_s = 1$ if A is fully supported, and $r_s = 0$ if A is not supported at all, i. e. A is floating. If r_s is below a threshold $r_{s,\min}$, A is consid-

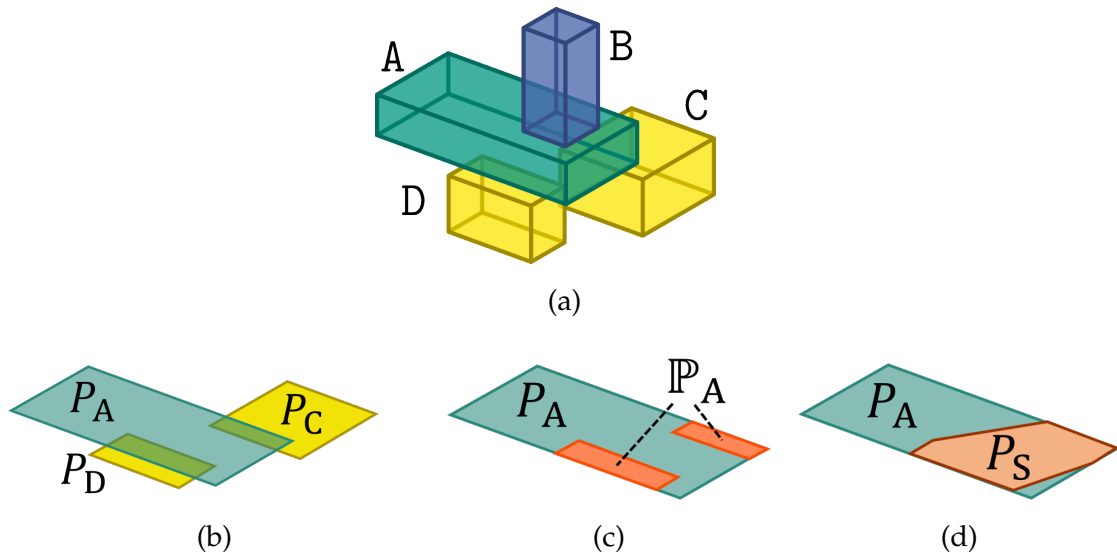


Figure 3.8.: Example of a support polygon construction for object A from a scene with three other objects B, C and D. (a) The object constellation. A supports B and is supported by C and D. (b) The objects A, C and D are projected onto the ground, creating polygons P_A , P_C and P_D , respectively. (c) P_C and P_D are intersected with P_A , constructing \mathbb{P}_A . (d) The support polygon P_s is the convex hull of the polygons in \mathbb{P}_A .

ered potentially unstable. In this case, a new support relation (B, A) labeled as top-down is added to SUPP. If r_s is above the threshold $r_{s,\min}$, A is likely well supported by the objects below it and will stay at rest when B is removed, so no support is detected. This reasoning is performed for all pairs of objects, adding a support relation where potential support is detected. The supported area ratio is a simple heuristic which does not require explicit assumptions about the mass distribution of objects. A more sophisticated model could use the supported volume ratio or estimate the object's center of gravity which requires more prior knowledge.

A support graph consisting only of ACT relations is an acyclic graph, since edges are added only in one direction depending on the relation between the separating plane and the gravity vector. Note that the edges added to the support graph by support polygon analysis create cycles. This might require additional care when deriving a safe manipulation order from the support graph.

3.3.2. Probabilistic Support Relation Extraction

Given a probability distribution $P(\mathcal{P} \mid \mathcal{O})$ over geometric primitives in the scene, we want to determine the existence probability $P(\text{SUPP}(A, B) \mid \mathcal{O})$ of a support relation between each object pair $A, B \in \mathcal{O}$. We achieve this through a *Monte-Carlo simulation*. First, we sample concrete scene geometries and evaluate support relations using the deterministic extraction method detailed above. Then, we estimate the existence probability of support relations over sampled scene geometries.

Sampling from the Geometric Scene Distribution: Algorithm 3 describes sampling from a probability distribution $P(\mathcal{P} \mid \mathcal{O})$ over geometric primitives in a scene. Drawing a sample from the joint distribution $P(t_i, \mathbf{x}_i \mid o_i)$ according to equation 3.7 produces a concrete geometric primitive $p_i^s = (t_i^s, \mathbf{x}_i^s)$. To implement this sampling, first, a primitive type t_i^s is sampled from the discrete probability distribution $P(t_i \mid o_i)$ (see equation 3.5). Then, a state vector \mathbf{x}_i^s is generated by sampling from the multivariate Gaussian distribution $\mathcal{N}(\mu_{i,t_i^s}, \Sigma_{i,t_i^s})$ (see equation 3.6). Note that the primitive type t_i^s is a concrete value and not a random variable. Since we assume independence between separate geometric primitives, drawing a sample from the distribution $P(\mathcal{P} \mid \mathcal{O})$ can be implemented by drawing samples from $P(t_i, \mathbf{x}_i \mid o_i)$ for each object $o_i \in \mathcal{O}$. This results in a set of sampled geometric primitives $\mathcal{P}_s = \{p_1^s, p_2^s, \dots, p_n^s\}$ with $p_i^s = (t_i^s, \mathbf{x}_i^s)$.

Algorithm 3: Sample from Geometric Scene Distribution

Input: $P(\mathcal{P} \mid \mathcal{O})$: Probability distribution over geometric scenes

Output: $\mathcal{P} = \{p_1, p_2, \dots, p_n\}$: Set of geometric primitives

$\mathcal{P} = \emptyset$;

for $i \in \{1, n\}$ **do**

$t \sim P(t_i \mid o_i)$;

$\mathbf{x} \sim P(\mathbf{x}_i \mid t_i, o_i)$;

$\mathcal{P} = \mathcal{P} \cup \{(t, \mathbf{x})\}$;

end

return \mathcal{P} ;

Algorithm 4: Estimate Support Relation Probability

Input: $P(\mathcal{P} \mid \mathcal{O})$: Probability distribution over geometric scenes
Output: $P(\text{SUPP})$: Probability distribution over support relations
 count = $\{((A, B), 0) \mid (A, B) \in \mathcal{O} \times \mathcal{O}\}$;
 $m = |\mathcal{O}|$;
for $k \in \{1, m\}$ **do**
 $\mathcal{P}_k \sim P(\mathcal{O})$;
 $\text{SUPP}_k = \text{extractSupportRelation}(\mathcal{P}_k)$;
 for $(A, B) \in \text{SUPP}_k$ **do**
 count $[(A, B)] = \text{count}[(A, B)] + 1$;
 end
end
for $(A, B) \in \mathcal{O} \times \mathcal{O}$ **do**
 $P(\text{SUPP}(A, B)) = \text{count}[(A, B)]/m$;
end
return $P(\text{SUPP})$;

Estimating Existence Probabilities: We approximate the existence probabilities of support relations via a Monte-Carlo simulation. First, we generate m sets of primitives $\{\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_m\}$ by sampling from the geometric scene distributions as described above. Then, we extract pairwise support relations between the sampled primitives using ACT relation and support polygon analysis presented in section 3.3.1. This way, we can estimate the existence probability for a support relation between two objects o_i and o_j by counting the occurrences of concrete support:

$$P(\text{SUPP}(o_i, o_j) \mid \mathcal{O}) \approx \frac{1}{m} \cdot \sum_{k=1}^m \text{SUPP}(p_i^k, p_j^k), \quad p_i^k, p_j^k \in \mathcal{P}_k$$

where $p_i^k, p_j^k \in \mathcal{P}_k$ are primitives sampled from their objects' geometric probability distributions. As $m \rightarrow \infty$, this estimate converges to the actual probability.

$$\lim_{m \rightarrow \infty} \left(\frac{1}{m} \cdot \sum_{k=1}^m \text{SUPP}(p_i^k, p_j^k) \right) = P(\text{SUPP}(o_i, o_j) \mid \mathcal{O})$$

Algorithm 4 describes the procedure as presented in this section. The variable count keeps track of the number of detected support relations between each object pair.

3.4. Evaluation

This section evaluates the proposed extraction of support relations from point clouds. First, the deterministic method based on ACT relation and support polygon analysis is evaluated. Then, the impact of the probabilistic extraction method is compared to the deterministic method.

3.4.1. Deterministic Support Relation Analysis

To evaluate the deterministic methods for support relation extraction based on ACT relations and support polygon analysis, the following questions will be addressed:

1. How to handle vertical separating planes between objects in contact?
2. Does support polygon analysis improve support relation detection?

To answer these questions, we evaluate our extraction method on multiple scenes containing piles of objects on a table.

This evaluation uses both real-world as well as simulated scenes. The point clouds of the real world scenes were recorded using an ASUS Xtion Pro. For the simulation, a depth camera was simulated. For each scene, we created a ground truth support relation $SUPP_{GT}$. In the simulation, we determined support relations of each object by registering what other objects move due to the removal of the inspected object. For the real scenes, we annotated the support relations by hand. We manually matched extracted objects \mathcal{O}_{Hyp} and ground truth objects \mathcal{O}_{GT} , since we are only interested in the extracted support relations.

Vertical Separating Planes: Separating planes are inserted between two objects in contact. The angle α between the plane's normal and the gravity vector

Strategy Scene	$\alpha 0$		$\alpha 10N$		$\alpha 10S$	
	Precision	Recall	Precision	Recall	Precision	Recall
S1	1.00	1.00	1.00	1.00	1.00	1.00
S2	0.71	1.00	0.83	1.00	0.63	1.00
S3	0.71	1.00	1.00	1.00	0.56	1.00
S4	0.80	0.89	0.89	0.89	0.73	0.89
R1	1.00	1.00	1.00	1.00	1.00	1.00
R2	0.78	0.78	1.00	0.78	0.73	0.89
R3	0.67	0.67	1.00	0.67	0.75	1.00
R4	0.86	0.86	1.00	0.86	0.93	0.93
R5	0.86	0.86	1.00	0.86	0.81	0.93
R6	1.00	1.00	1.00	1.00	1.00	1.00
R7	1.00	0.75	1.00	0.75	1.00	0.75
R8	0.75	0.75	0.94	0.75	0.58	0.75
R9	1.00	0.83	1.00	0.83	1.00	0.83
R10	0.86	0.75	1.00	0.75	0.89	0.89
Mean	0.86	0.87	0.98	0.87	0.83	0.92

Table 3.1.: Precision and recall of extracted support relations for all scenes and separating plane strategies. S1 – S4 are simulated and R1 – R10 are real scenes. The last row contains the mean over all scenes.

determines the ACT relation. If $\alpha < \alpha_{\max}$, a separating plane is labeled as vertical. We compare three strategies for handling vertical separating planes:

- $\alpha 0$: No separating plane angle threshold ($\alpha_{\max} = 0^\circ$)
- $\alpha 10N$: Within threshold, assume that no support exists ($\alpha_{\max} = 10^\circ$)
- $\alpha 10S$: Within threshold, assume support exists ($\alpha_{\max} = 10^\circ$)

Either we make a hard decision for every separating plane ($\alpha 0$), or we introduce a threshold for vertical separating planes ($\alpha 10N$ and $\alpha 10S$). If we use the threshold, we can either assume no support exists ($\alpha 10N$) or support exists ($\alpha 10S$) between objects with vertical separating planes.

We extract support relations SUPP_{Hyp} with every strategy using a contact margin $\delta_c = 10$ mm. Table 3.1 shows precision and recall for all scenes and each of the

four strategies calculated as

$$\text{Precision} = \frac{|\text{SUPP}_{\text{GT}} \cap \text{SUPP}_{\text{Hyp}}|}{|\text{SUPP}_{\text{Hyp}}|}, \quad \text{Recall} = \frac{|\text{SUPP}_{\text{GT}} \cap \text{SUPP}_{\text{Hyp}}|}{|\text{SUPP}_{\text{GT}}|}. \quad (3.8)$$

Selected evaluation scenes are presented in Table 3.3. The $\alpha 10\text{N}$ strategy is a strict improvement over $\alpha 0$, demonstrating the usefulness of the threshold α_{max} to detect vertical separating planes. Almost all support edges detected by $\alpha 10\text{N}$ are correct. However, it misses more edges than the other strategies resulting in a worse recall. $\alpha 10\text{S}$ produces a higher recall than $\alpha 10\text{N}$ due to adding unknown edges to the support hypothesis. Yet, precision is reduced considerably since not all of the added edges are correct.

Support Polygon Analysis: The proposed support polygon analysis adds potential top-down support relation. Here, we investigate the impact of these additionally detected support relations on precision and recall. This goal of this ablation study is to identify the benefits and drawbacks of adding support polygon analysis on top of ACT relations.

Table 3.2 shows precision and recall for the three strategies $\alpha 0$, $\alpha 10\text{N}$, and $\alpha 10\text{S}$ combined with support polygon analysis (SPA) on the evaluation scenes (see Table 3.3). All support relations were extracted using a contact margin $\delta_c = 10$ mm and a threshold for the support area ratio $r_{s,\text{min}} = 70\%$.

Adding support polygon improves recall without negatively affecting precision in scenes containing top-down support (e. g. S4, R2, R7). As expected, $\alpha 10\text{S}+\text{SPA}$ adds the most edges out of all strategies and therefore achieves the best recall. It also adds the most false positives, resulting in the worst precision. Overall, $\alpha 10\text{S}$ is too conservative and adds too many edges. $\alpha 10\text{N}+\text{SPA}$ achieves high precision and recall values by adding only potential top-down support edges. It offers the best compromise between correctness and completeness of the support hypothesis. Therefore, we conclude that the $\alpha 10\text{N}+\text{SPA}$ strategy is most suitable for extracting support relations in cluttered environments.

Strategy Scene	$\alpha 0$ +SPA		$\alpha 10N$ +SPA		$\alpha 10S$ +SPA	
	Precision	Recall	Precision	Recall	Precision	Recall
S1	1.00 _(+0.00)	1.00 _(+0.00)	1.00 _(+0.00)	1.00 _(+0.00)	1.00 _(+0.00)	1.00 _(+0.00)
S2	0.71 _(+0.00)	1.00 _(+0.00)	0.83 _(+0.00)	1.00 _(+0.00)	0.63 _(+0.00)	1.00 _(+0.00)
S3	0.71 _(+0.00)	1.00 _(+0.00)	1.00 _(+0.00)	1.00 _(+0.00)	0.56 _(+0.00)	1.00 _(+0.00)
S4	0.75 _(-0.05)	1.00 _(+0.11)	0.90 _(+0.01)	1.00 _(+0.11)	0.75 _(+0.02)	1.00 _(+0.11)
R1	1.00 _(+0.00)	1.00 _(+0.00)	1.00 _(+0.00)	1.00 _(+0.00)	1.00 _(+0.00)	1.00 _(+0.00)
R2	0.80 _(+0.02)	0.89 _(+0.11)	1.00 _(+0.00)	0.89 _(+0.11)	0.75 _(+0.02)	1.00 _(+0.11)
R3	0.67 _(+0.00)	0.67 _(+0.00)	1.00 _(+0.00)	0.67 _(+0.00)	0.75 _(+0.00)	1.00 _(+0.00)
R4	0.81 _(-0.05)	0.93 _(+0.07)	0.76 _(-0.24)	0.93 _(+0.07)	0.74 _(-0.19)	1.00 _(+0.07)
R5	0.82 _(-0.04)	1.00 _(+0.14)	0.93 _(-0.07)	0.93 _(+0.07)	0.78 _(-0.03)	1.00 _(+0.07)
R6	0.75 _(-0.25)	1.00 _(+0.00)	0.75 _(-0.25)	1.00 _(+0.00)	0.75 _(-0.25)	1.00 _(+0.00)
R7	1.00 _(+0.00)	1.00 _(+0.25)	1.00 _(+0.00)	1.00 _(+0.25)	1.00 _(+0.00)	1.00 _(+0.25)
R8	0.67 _(-0.08)	0.80 _(+0.05)	0.80 _(-0.14)	0.80 _(+0.05)	0.53 _(-0.05)	0.80 _(+0.05)
R9	1.00 _(+0.00)	1.00 _(+0.17)	1.00 _(+0.00)	1.00 _(+0.17)	1.00 _(+0.00)	1.00 _(+0.17)
R10	0.89 _(+0.03)	0.89 _(+0.14)	1.00 _(+0.00)	0.89 _(+0.14)	0.89 _(+0.00)	1.00 _(+0.11)
Mean	0.83 _(-0.03)	0.94 _(+0.07)	0.93 _(-0.05)	0.93 _(+0.07)	0.79 _(-0.04)	0.99 _(+0.07)

Table 3.2.: Precision and recall of extracted support relations for all scenes and separating plane strategies combined with support polygon analysis (SPA). S1 – S4 are simulated and R1 – R10 are real scenes. The last row contains the mean over all scenes. Changes compared to Table 3.1 without SPA are marked in parentheses.

3.4.2. Probabilistic Support Relation Extraction

We present experimental results for the extraction of support relations from point cloud data. First, we introduce the two datasets and the evaluation procedure. Then, we present and discuss the results. The code and a reproduction guide for the results presented in this section are available online¹.

¹<https://gitlab.com/h2t/interactive-scene-exploration>

Scene	Point Cloud	Objects	Hypothesis	Ground Truth
S3				
S4				
R2				
R3				
R5				
R7				
R8				
R9				
R10				

Table 3.3.: Selected evaluation scenes. We show the recorded point cloud, the extracted objects, our support hypotheses and the ground truth support relation. Support hypotheses for all four strategies are visualized by color coding differing edges. Black edges were generated by the α_{10N} strategy and are also part of the three other strategies. Blue edges are part of the graph if we use α_{10S} , while red edges are added by the support polygon analysis. The scenes not shown in this table are slight variations of their neighbors, e. g. R1 is a simpler version of R2 without the unknown and uncertain edges.

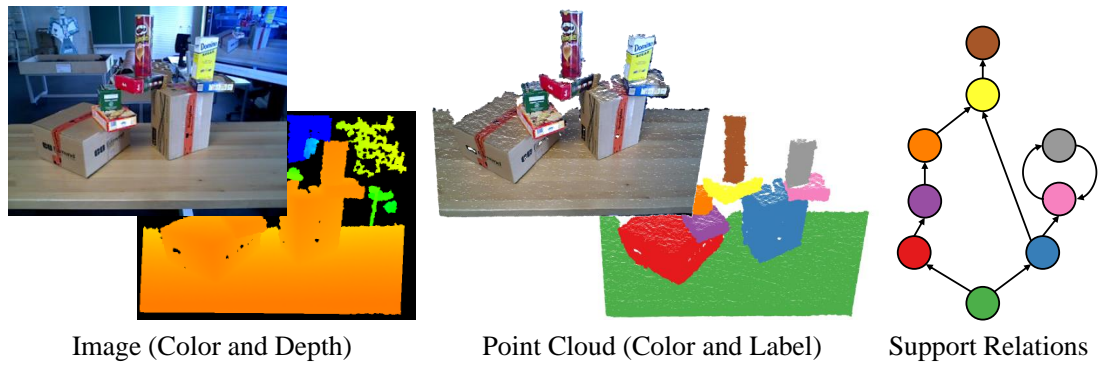


Figure 3.9.: A single entry in the KIT Support Relation (KIT-SR) dataset consists of color and depth images of the scene, a colored and labeled point cloud as well as manually annotated support relations.

Datasets: We present the KIT Support Relation (KIT-SR) dataset, which contains table-top scenes with a varying number of objects and support relations. We provide full pixel-wise segmentation and support relation annotations for each of the 60 scenes. The dataset is publicly available (see appendix A for details). Figure 3.9 shows a scene from the dataset and the corresponding available data.

Additionally, we use the Object Segmentation Dataset (OSD) as used in Richtsfeld et al. (2012) to compare our probabilistic support relation extraction with deterministic methods. The OSD contains table-top scenarios with varying object geometry and scene complexity, ranging from simple scenes with two objects to cluttered scenes with up to 16 objects. Since the dataset focuses on object segmentation, we added ground truth annotations for support relations to all 111 scenes by hand.

Metrics: The evaluation compares the proposed probabilistic support extraction with two deterministic approaches. The first deterministic approach is based on ACT relations and assumes no support for vertical separating planes ($\alpha 10N$) and serves as a baseline for the comparison. The second deterministic approach extends ACT relations with support polygon analysis ($\alpha 10N+SPA$) to detect top-down support relations, i. e. an object is supported by an object that is geometrically above it.

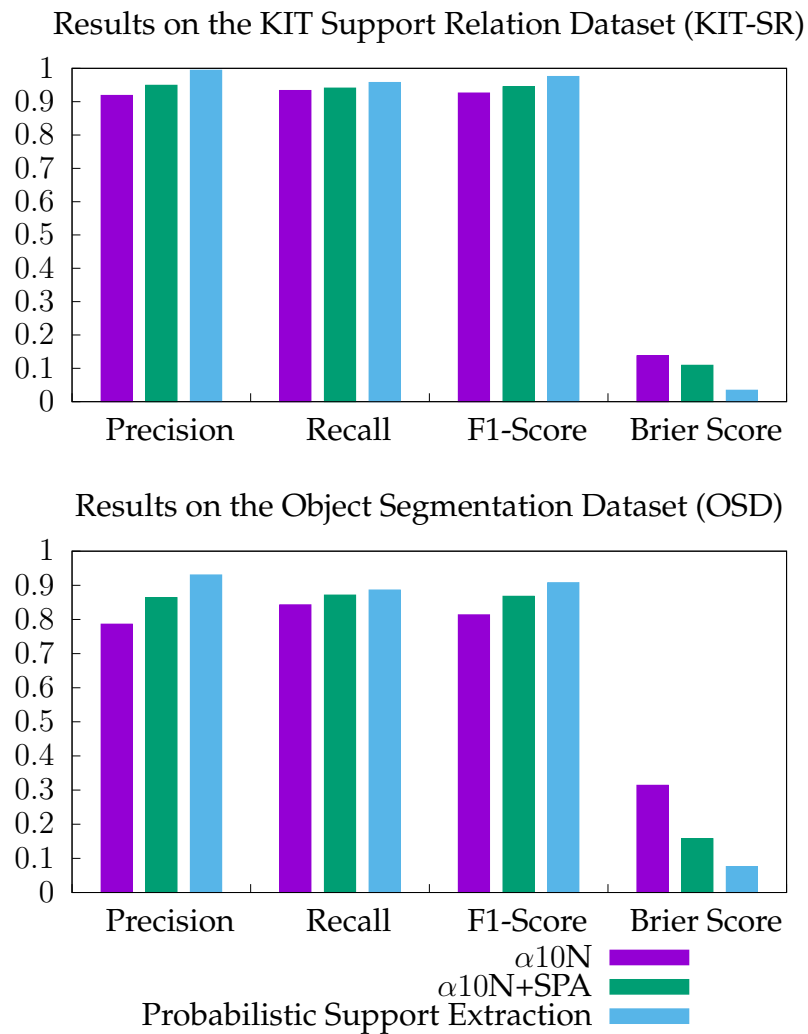


Figure 3.10.: Evaluation of the three approaches using the evaluation metrics precision, recall, F1-Score, and Brier score on the two datasets KIT-SR and OSD. Higher values for precision, recall, and F1-Score indicate better results, while lower values for the Brier score indicate better results.

We evaluate the three approaches on the datasets by computing the metrics precision, recall, F1-Score, and Brier score. Given the ground truth support relation SUPP_{GT} and a hypothesis SUPP_{Hyp} generated by one of the extraction methods, we can calculate precision and recall as follows:

$$\text{Precision} = \frac{|\text{SUPP}_{\text{GT}} \cap \text{SUPP}_{\text{Hyp}}|}{|\text{SUPP}_{\text{Hyp}}|}, \quad \text{Recall} = \frac{|\text{SUPP}_{\text{GT}} \cap \text{SUPP}_{\text{Hyp}}|}{|\text{SUPP}_{\text{GT}}|}.$$

The F1-Score is the harmonic mean of precision and recall:

$$\text{F1 - Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

When working with binary classification metrics and a probabilistic support representation, we first need to binarize the probabilities, i. e. decide whether support exists or not. We define support between A and B to exist if the existence probability is above the threshold 0.5: $P(\text{SUPP}(A, B)) > 0.5$

As a proper score function, the Brier score can be directly used with probabilities. It measures the accuracy of our probabilistic support relation extraction. The Brier score gets smaller when the hypothesis gets more accurate compared to the ground truth. The probabilities $P_{\text{Det}}(\text{SUPP}(A, B))$ for the deterministic approaches are set to 1 if support exists between objects A and B, 0 otherwise.

$$\text{Brier Score} = \frac{1}{|\mathcal{O} \times \mathcal{O}|} \sum_{A, B \in \mathcal{O} \times \mathcal{O}} (P_{\text{Hyp}}(\text{SUPP}(A, B)) - P_{\text{GT}}(\text{SUPP}(A, B)))^2$$

Results: Figure 3.10 and table 3.4 show the evaluation results on the KIT-SR dataset and the OSD. We can see a significant improvement in precision and a moderate increase in recall when comparing the probabilistic with the deterministic approaches. The Brier score is also significantly improved. Furthermore, the addition of support polygon analysis ($\alpha 10\text{N} + \text{SPA}$) yields better results than the method based on solely ACT relations ($\alpha 10\text{N}$) but remains behind the probabilistic approach. The false positives and false negatives of the deterministic approaches are mostly caused by uncertainties about object geometry due to the dataset only containing single viewpoint clouds. The probabilistic approach captures these uncertainties and avoids making hard deterministic decisions early.

Table 3.4.: Evaluation results on the two datasets.

Dataset	Method	Precision	Recall	F1	Brier Score
		(higher is better)			(lower is better)
KIT-SR	α 10N	0.919	0.934	0.926	0.138
	α 10N+SPA	0.949	0.941	0.945	0.109
	Prob. Supp. Extr.	0.994	0.957	0.976	0.034
OSD	α 10N	0.786	0.842	0.813	0.314
	α 10N+SPA	0.864	0.872	0.868	0.158
	Prob. Supp. Extr.	0.931	0.886	0.908	0.076

3.5. Summary and Review

This chapter introduced methods for representing and extracting support relations from point clouds. Three questions were addressed:

- How to represent objects and their support relations while capturing the uncertainty in object shapes, poses and relation existence?
- How to fit probabilistic geometric shape models to a point cloud of a scene captured by the robot?
- How to extract probabilistic support relations from a probability distribution over geometric shape models?

First, a probabilistic representation for geometric primitives and support relations among them was proposed. Second, a method for fitting geometric primitives to point clouds using the proposed probabilistic representation was implemented. Third, a method for extracting support relations from the probabilistic object representation was developed. We evaluated the proposed methods on different datasets containing scenes with varying degree of complexity and showed their effectiveness.

Probabilistic Object and Support Relation Representation: The core idea of the proposed representation is to use probability distributions for both object poses, shapes and their relations. To approximate objects in a scene, we consider parametrizable geometric primitives, i. e. boxes, cylinders and spheres. A

joint probability distribution over primitive types and geometric parameters including pose and shape was specified. The distribution over primitive types is modeled as a discrete distribution, while the geometric parameters are modeled as a multi-variate Gaussian distribution. Then support relations and their existence probability between object pairs were defined.

Probabilistic Geometric Primitive Fitting: The geometric fitting method is based on RANSAC. For each segment in the input point cloud, geometric parameters are estimated using a minimum sampling set and evaluated based on the number of inlier points. Methods for estimating the geometric parameters of spheres, cylinders and boxes are presented. For boxes, a novel method, which only requires two visible faces is developed. A modified RANSAC algorithm was proposed and implemented, which generates a joint probability distribution over primitive type and geometric parameters taking into account the uncertainty about those properties.

Extraction of Probabilistic Support Relations: A physically grounded extraction method for probabilistic support relations was proposed. The main idea is to use a deterministic approach for support relation extraction and a Monte-Carlo simulation to estimate existence probabilities of support relations. First, a deterministic extraction approach based on ACT relations was implemented and extended by a support polygon analysis to capture top-down support relations. In order to apply this deterministic extraction method, we sample from the geometric scene distribution and approximate the existence probability of support relations by counting support occurrences in sampled scenes.

Evaluation: We show the effectiveness of the different components of the proposed support relation extraction in separate ablation studies. First, the benefits of handling vertical separating planes and applying the support polygon analysis on top of existing ACT relations are demonstrated. Both lead to a higher recall while maintaining a high precision. Then, the advantages of a probabilistic representation and extraction method are evaluated. Here, all metrics indicate an improvement over deterministic approaches.

4. Graph-based Prediction of Action Effects

When planning and executing actions, robots need the ability to predict the effects of their actions. Relevant actions might be pushing, grasping or lifting an object. For the pushing case, action effects include the position and orientation of the object after push execution, but might also include other objects in the scene interacting with the pushed object. Even for simple cases involving only a single object, hard-coding action effects into the prior knowledge of a robot by human experts is still a challenging task. However, when multiple objects interact, methods for learning prediction models for complex action effects are needed.

Predicting how things might be, not only now, but also in the future are a fundamental part of human cognitive abilities which allow us to choose how to act (Berthoz, 2000). A cognitive system, which uses an internal model to reason about action consequences in the world and learns from experience, should also be able to explain why and what it is doing (Brachman, 2002). To this end, simulations have been used in combination with logic-based reasoning to acquire common sense knowledge (Johnston and Williams, 2008). Extending this idea, physics-based simulations enable action parameter optimization in many applications ranging from table-top pushing tasks (Weitnauer et al., 2010) to preparing a pancake (Kunze and Beetz, 2017). Understanding the effects of actions is essential for planning and executing robot tasks. By imagining possible action consequences, a robot can choose specific action parameters to achieve desired goal states.

In this chapter, we propose an object-centric, graph-based representation for scenes both as input and output of action effect prediction models. The robot

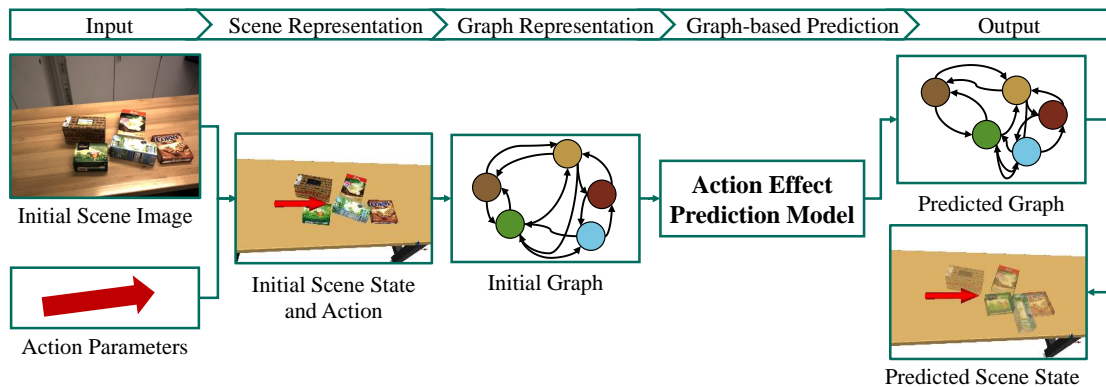


Figure 4.1.: Overview of the graph-based action effect prediction using pushing into multiple interacting objects as an action.

extracts the initial scene state from a perceived scene by localizing known objects using Azad et al. (2009) and Pauwels and Kragic (2015) and approximating unknown objects using geometric primitive fitting as described in chapter 3. Given the initial scene state and parameters of an action to execute, e.g. start and end point of a push, we create a graph that encodes object properties in its vertices, relations in its edges and action parameters in its global attributes. The graph is the input of the learned action effect prediction model, a graph neural network that takes graphs as input and produces graphs as output (see section 2.3). Finally, from the output graph the predicted scene state is reconstructed. Figure 4.1 illustrates the proposed approach.

The graph-based action effect prediction is implemented and evaluated for two different scenarios:

- **Pushing Multiple Rigid Objects:** In a table-top scenario, multiple rigid objects on a flat surface interact during the execution of pushing actions.
- **Interacting with a Deformable Object:** A deformable, cloth-like bag interacts with multiple rigid spheres while different actions are executed. The actions include pushing an object towards the bag, opening the bag, lifting the bag, and moving a handle of the bag along a trajectory.

In this chapter the following research questions are addressed:

- How can action effect prediction models for scenes with a varying number of interacting objects be learned?

- How effective are models learned from simulation to predict effects on real-world data?
- How can graph-based scene representations containing both deformable and rigid objects be built?

The chapter is structured as follows. Section 4.1 develops the methods for graph-based action effect prediction for the scenario of pushing multiple rigid objects. Central aspects, like the scene graph representation and the formulation of action effect prediction as a graph learning problem are introduced. The approach is evaluated on a large-scale synthetic dataset generated in simulation and on recorded data from real robot experiments. Section 4.2 extends the developed methods to handle deformable objects and more actions. To this end, a sparse keypoint representation for deformable objects as well as a two-stage prediction model are proposed. This extension is evaluated on a novel dataset for interactions between a deformable bag and multiple rigid spheres. Finally, the chapter is summarized in section 4.3. Parts of this chapter have been published in Paus et al. (2020) and Weng et al. (2021).

4.1. Action Effect Prediction for Rigid Objects

We represent perceived scenes as object-centric graphs and learn an internal model, which predicts object pose changes due to pushing actions. We train this internal model on a large synthetic data set, which was generated in simulation, and record a smaller data set on a real robot for evaluation.

A robot interacting with the world needs the ability to reason about the effects of its actions (Krüger et al., 2011). Humans can predict the effects of actions based on low-level motor control as well as high-level reasoning (Moore and Haggard, 2008; Sato, 2009). When comparing physics engines with intuitive physics models based on probabilistic simulations of interactions between objects and their relations, humans tend to give results more consistent with the latter (Battaglia et al., 2013). Taking inspiration from this idea, we want to enable a robot to predict action effects based on objects and the spatial relations between them. This results in two challenges. First, a scene can contain an arbitrary number

of objects. Second, training a prediction model requires many samples, which are hard to obtain on a real robot. We address the first challenge by learning a model which is invariant to the number of objects and their order. By using a physics simulation, we address the second challenge.

We start by presenting a graph-based action-centric scene representation in section 4.1.1. Each scene is represented as an attributed directed graph, in which vertices store object properties and edges contain relative spatial information between object pairs. In section 4.1.2, we explain the data generation in a physics simulation and on a real robot. The training data for learning push effects is generated by executing pushes in randomized scenes using the physics simulation MuJoCo (Todorov et al., 2012). To formulate an optimization problem on these scene graphs, we use graph neural networks (as discussed in section 2.3), which provide machine learning building blocks with graphs as input and output. Section 4.1.3 defines action effect prediction as a graph learning problem and trains a graph neural network using the generated data. The evaluation in section 4.1.4 shows a high prediction accuracy of our internal model in simulation and on real data. It verifies the ability of proposed approach to inherently handle a varying number and order of objects.

The contribution is a data-driven method for push effect prediction with high prediction accuracy while being faster than executing a physics simulation. In contrast to state-of-the-art approaches, our method is neither limited to planar object movement nor a fixed number of objects.

4.1.1. Scene Representation

Given a scene as a set of n objects \mathcal{O} , the goal is to predict pose changes for these objects caused by a pushing action.

$$\mathcal{O} = \{o_i = (\mathbf{t}_i, R_i, \mathbf{s}_i) \mid i \in [1, n]\}$$

We represent an object $o_i = (\mathbf{t}_i, R_i, \mathbf{s}_i) \in \mathcal{O}$ as a tuple consisting of global position $\mathbf{t}_i \in \mathbb{R}^3$, global orientation $R_i \in SO(3)$, and oriented bounding box size $\mathbf{s}_i \in \mathbb{R}^3$. A pushing action $a = (\mathbf{d}, \mathbf{e}) \in \mathcal{A}$ is represented as a direction $\mathbf{d} \in \mathbb{R}^3$ and an endpoint $\mathbf{e} \in \mathbb{R}^3$ which specifies where the end-effector stops after executing

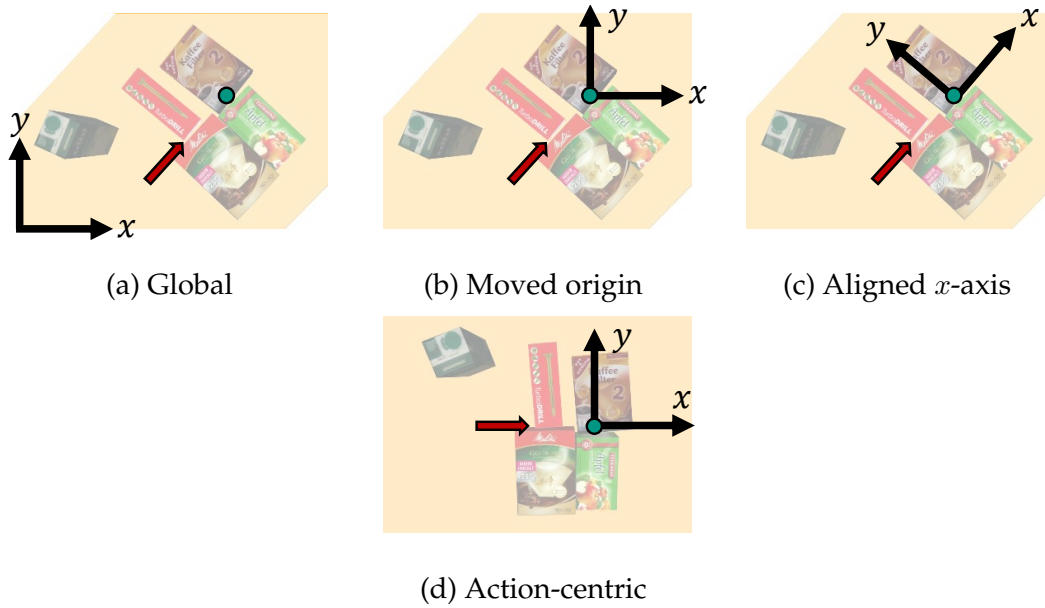


Figure 4.2.: Example of a transformation from the global coordinate system (a) to the action-centric coordinate system (d). The x -axis (in red) and the y -axis (in green) of the coordinate system are shown as arrows. The pushing direction and the end point of the push are illustrated as arrow and circle. First, the origin is moved to the end point of the push (b). Then, the x -axis is aligned with the pushing direction (c).

the action. We want to learn a prediction model $M : \text{Pow}(\mathcal{O}) \times \mathcal{A} \rightarrow \text{Pow}(\mathcal{O})$ which, given an initial scene $\mathcal{O}_{\text{before}}$ and a push action $a \in \mathcal{A}$, outputs the scene $\mathcal{O}_{\text{after}}$ after executing the push.

$$\mathcal{O}_{\text{after}} = M(\mathcal{O}_{\text{before}}, a)$$

We want to find an scene representation, which encodes object properties, relations and actions into a graph and is invariant to translation and rotation. To this end, we first transform the scene into an action-centric representation, before building a graph from this local representation.

Action-centric Representation: Since object poses and push directions depend on an arbitrarily chosen global coordinate system, we transform the scene to a local representation. First, the push endpoint $\mathbf{e} \in \mathbb{R}^3$ becomes the origin

of the new local coordinate frame. Second, we rotate the scene so that the push direction $\mathbf{d} \in \mathbb{R}^3$ is aligned with the positive x -axis of the local coordinate system while keeping the z -axis pointing upwards (aligned with the global z -axis). The y -axis is chosen to create a right-handed coordinate system. Since the local origin is the push endpoint and the local x -axis is the push direction, the action parameters are fully encoded in this action-centric coordinate system and no explicit push parameters are needed. Figure 4.2 illustrates this transformation.

Graph Representation: To use a graph neural network, the input and output data need to be represented as a directed attributed graph $G = (V, E, \mathbf{u})$, where V is a set of vertices, E a set of edges, and \mathbf{u} a global feature vector. For each object, we create a corresponding vertex in the graph, i. e. $|V| = |\mathcal{O}|$. After transforming the scene to the action-centric representation, we encode the features of each object o_i as a stacked vector of position $\mathbf{t}_i \in \mathbb{R}^3$, orientation $R_i \in SO(3)$, and size $\mathbf{s}_i \in \mathbb{R}^3$, resulting in a vertex feature vector $\mathbf{v}_i \in \mathbb{R}^{15}$. We found that representing the orientation as a 3×3 matrix gives better results during training compared to quaternions or Euler angles. Additionally, we create edges between each vertex pair $(\mathbf{v}_i, \mathbf{v}_j)$ using the relative position difference as the feature vector

$$\mathbf{f}_{i,j} = \mathbf{t}_j - \mathbf{t}_i \in \mathbb{R}^3 .$$

A global feature vector \mathbf{u} is not needed, as the action parameters are already encoded in the action-centric representation. Figure 4.3 shows the different steps to create the described graph representation.

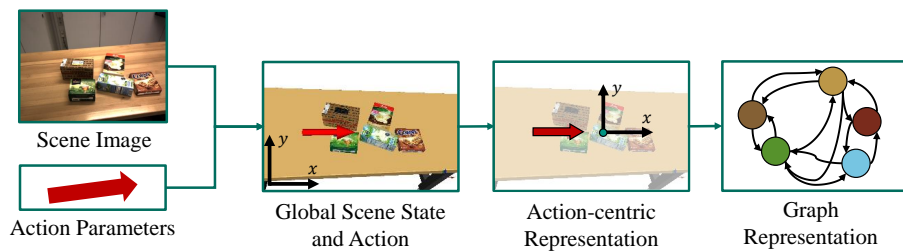


Figure 4.3.: The scene image and action parameters are represented as a global scene state and action before they are transformed into an action-centric representation. The graph representation is built from the action-centric representation.

4.1.2. Data Generation

To learn a model that can predict the effects of pushing actions, we need to generate data, with which the model can be trained and evaluated. The data needs to contain the initial scene state $\mathcal{O}_{\text{before}}$ before action execution, the parameters of the push action a , as well as the scene state $\mathcal{O}_{\text{after}}$ after action execution. To this end, we generate a large synthetic dataset using a physics simulation for training and collect a smaller dataset on a humanoid robot for evaluation.

Physics Simulation: We use the MuJoCo (Multi-Joint dynamics with Contact, Todorov et al. (2012)) physics simulator to execute pushes in simulation. MuJoCo is designed for dynamic, contact-rich interactions, which can occur if multiple objects interact while pushing an object. We generate randomized scenes and execute random pushes in simulation. Figure 4.4 shows examples of generated scenes and pushes. The following parameters have been randomized during scene generation using a uniform distribution over all possible values:

- Object number: Scenes contain one to five objects
- Object position: The center point of each object is chosen inside a rectangular region with size $1\text{m} \times 1\text{m}$.
- Object rotation: Only the rotation around the z -axis is randomized to ensure that the objects can stand upright.
- Object size: Width, height, and depth of the boxes can be in the interval $[0.05\text{m}, 0.20\text{m}]$.

Push parameters were chosen by first selecting a target object. Then, a local offset is sampled around the target’s center point, taking into account the size of the object. The endpoint of the push is chosen to be the target’s center point shifted by the local offset. This way, we ensure that most pushes are executed close to objects where relevant interactions are happening. Then, the push direction is sampled, allowing only pushes parallel to the ground.

During data generation, we use the model of the hand of ARMAR-6 (Asfour et al., 2019) as end-effector. Then, we generate a random scene as described above. In this scene, we execute 200 random pushes, after which we proceed

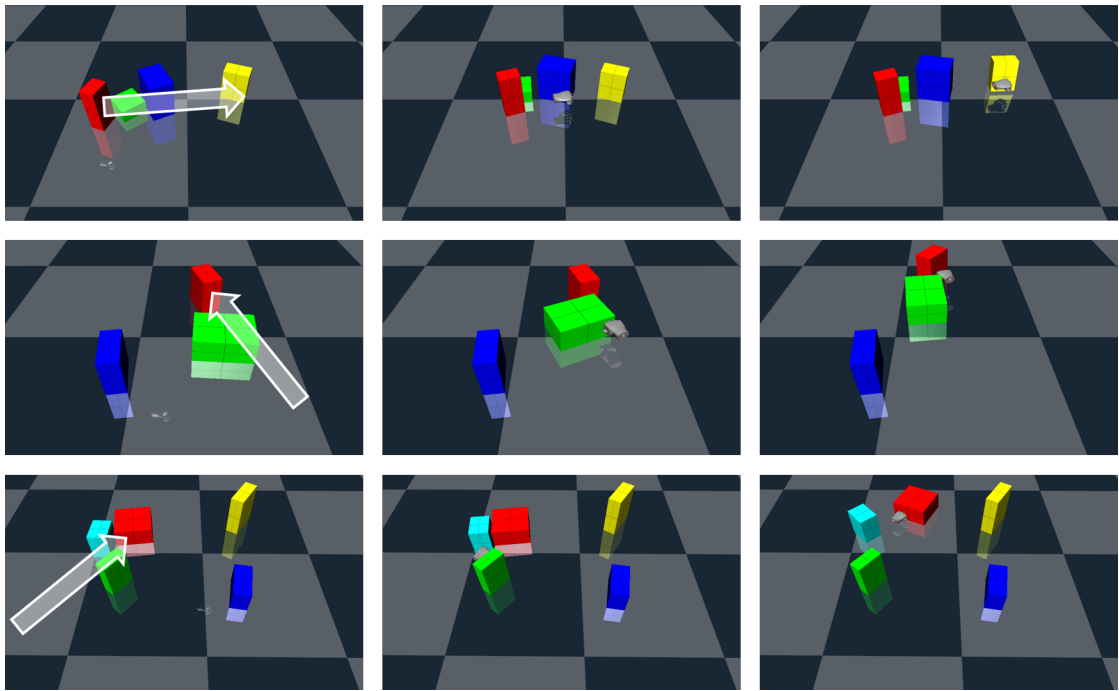


Figure 4.4.: Examples of randomized scenes in simulation. Each row represents a push executed in a generated scene. The first column shows the initial scene state and the sampled push direction as a white arrow. The second column shows the scene during the execution of the push. In the third column, the final scene state after the push is shown.

with the next scene. The object poses before and after the push, as well as the push parameters, are saved to train and evaluate the prediction model.

Humanoid Robot: We use a humanoid robot to generate data in the real world. On ARMAR-6, we execute 185 pushes in 20 different scenes. The involved objects were taken from the KIT and YCB object sets (Kasper et al., 2012; Calli et al., 2015). For ground truth data generation, we localize the involved objects (Azad et al., 2009; Pauwels and Kragic, 2015) before and after execution and store their poses and the push parameters accordingly. The recorded scenes contain between one and five objects. Figure 4.5 shows camera images from the recorded data set.

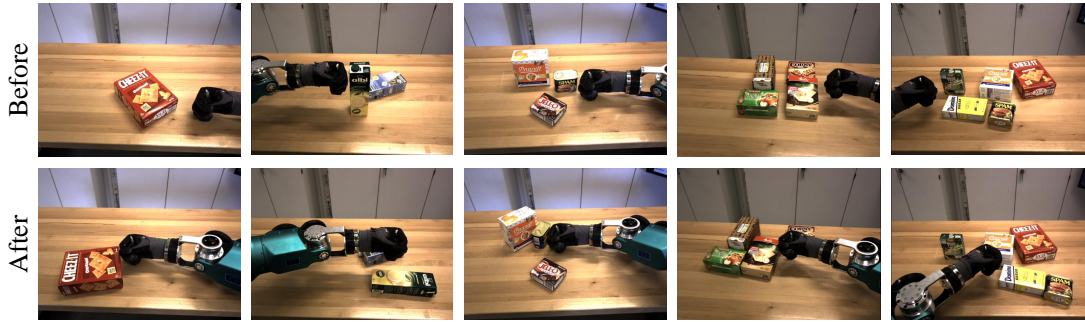


Figure 4.5.: Scenes from the data set recorded on the real robot. The top row contains images from the robot’s cameras right before the push action begins. The bottom row shows images after the push has been executed. Each row shows a scene with an increasing number of objects.

4.1.3. Graph Learning Problem

The generated data contains scenes before and after action execution as well as push parameters. We can convert these tuples $(\mathcal{O}_{\text{before}}, a, \mathcal{O}_{\text{after}})$ into the graph representation as introduced in section 4.1.1. This produces an input graph G_{in} representing the scene before action execution, and an output graph G_{out} representing the scene after action execution. Both graphs use action a as reference for the transformation into the action-centric representation, which simplifies the learning problem, since the model does not need to learn translation and rotation invariance. We now define and train a graph neural network GNN_{θ} parametrized by θ that takes G_{in} as input and produces G_{out} as output:

$$G_{\text{out}} = GNN_{\theta}(G_{\text{in}})$$

Network Architecture: We chose an Encode-Process-Decode architecture for the graph network as discussed in section 2.3 (see Figure 4.6). First, the input graph G_{in} is encoded where vertices, edges, and globals are expanded into a latent representation. The process step consists of a full graph network block which processes the latent representation 10 times. The process block uses a

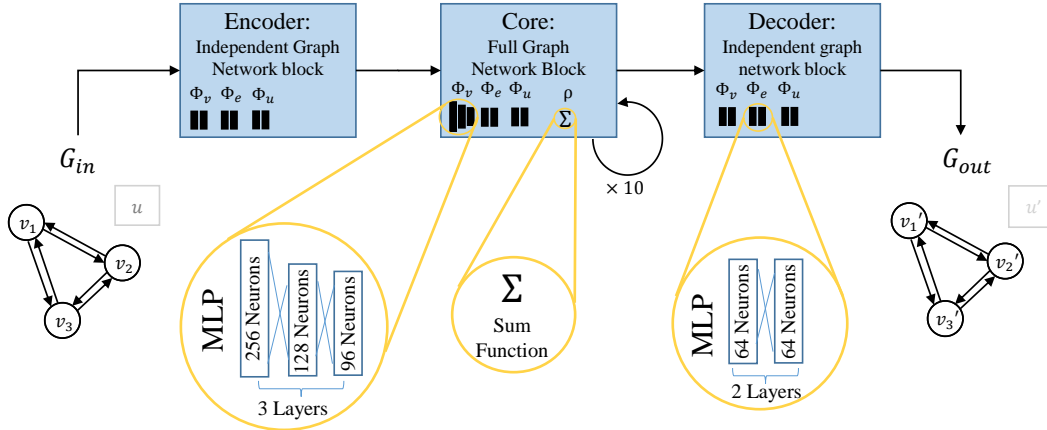


Figure 4.6.: The architecture of the prediction model is an Encode-Process-Decode graph network. The input graph G_{in} is encoded into a latent space using a graph independent block, i. e. , vertices, edges and global attributes are transformed individually. We execute a full graph network block ten times as the core processing step of our model. The latent representation is then transformed into the output graph G_{out} using a graph independent block as a decoder.

Multi-layer Perceptron (MLP) with batch normalization for each update function:

- Vertex update Φ_v : 3-layer MLP with layer sizes [256, 128, 96]
- Edge update Φ_e : 2-layer MLP with layer sizes [128, 96]
- Global update Φ_u : 2-layer MLP with layer sizes [64, 64]

We choose element-wise sum for all aggregation functions $\rho_{e \rightarrow v}$, $\rho_{v \rightarrow u}$, and $\rho_{e \rightarrow u}$. The encoder and decoder both use a graph independent block with two-layer MLPs with layer sizes [64, 64] for all update functions. All the MLPs contain parameters, which need to be optimized during training. We summarize them into the model parameter vector θ to define the parametrized prediction model GNN_θ . As the loss function \mathcal{L} , we use the mean squared error over the vertex attributes $\mathbf{v} \in \mathbb{R}^{15}$ from the predicted and ground truth scene, excluding the bounding box size \mathbf{s} , since it cannot change during a push. Here, we assume that the pushed objects are rigid.

Now, we want to find a model parameter vector θ which minimizes the loss over all input and ground-truth output graphs, G_{in} and G_{GT} :

$$\arg \min_{\theta} \mathcal{L}(GNN_{\theta}(G_{\text{in}}), G_{\text{GT}})$$

The dataset of 2,000,000 simulated pushes is split into training (70%), validation (20%), and test set (10%). During training, the model is optimized based on the training set using a batch size of 256. We use the Adam optimizer with a learning rate of 0.001. The layer sizes have been determined by a hyperparameter search using the validation set.

Output Normalization: The network predicts position as a vector $\mathbf{t} \in \mathbb{R}^3$ and orientation as a matrix $R \in \mathbb{R}^{3 \times 3}$. The predicted matrix does not necessarily satisfy the requirements for a rotation matrix: $R \in SO(3) : R \cdot R^T = I, \det(R) = 1$. Therefore, the predicted matrix needs to be orthogonalized and normalized to ensure it represents a proper rotation matrix.

Consider the Singular Value Decomposition (SVD) of the predicted matrix R : $R = U \cdot \Sigma \cdot V^T$, where $U, V \in \mathbb{R}^{3 \times 3}$ are orthogonal matrices and $\Sigma \in \mathbb{R}^{3 \times 3}$ is a diagonal matrix, which contains the singular values σ_i of R on its diagonal. Since the product of the singular values determine the absolute value of the determinant

$$\|\det(R)\| = \prod_1^3 \sigma_i,$$

we replace Σ with the identity matrix I . Furthermore, the sign of the determinant needs to be accounted for, since we only fixed the absolute value of the determinant. Thus, the orthogonalized and normalized matrix $R^* \in SO(3)$ is computed as follows:

$$R^* = \begin{cases} U \cdot I \cdot V^T & \text{if } \det(U \cdot I \cdot V^T) > 0 \\ -U \cdot I \cdot V^T & \text{if } \det(U \cdot I \cdot V^T) < 0 \end{cases}$$

This is called symmetric orthogonalization and minimizes the Frobenius norm (Aiken et al., 1980):

$$\|R^* - R\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n \|r_{ij}^* - r_{ij}\|^2}$$

4.1.4. Evaluation

First, we evaluate the performance of our approach quantitatively by looking at the position and orientation prediction errors on the simulated and real data generated in section 4.1.2. Then, we investigate the ability for combinatorial generalization, i. e. how the model handles an unseen number of objects. Finally, we discuss the runtime for model evaluation.

Position and Orientation Error: We evaluate the accuracy of our learned model by comparing position and orientation errors in the training, validation, and test set. For each executed action $a \in A$, we have the input scene graph $G_{\text{in}}^a = (V_{\text{in}}^a, E_{\text{in}}^a, \mathbf{u}_{\text{in}}^a)$ and the ground truth graph $G_{\text{GT}}^a = (V_{\text{GT}}^a, E_{\text{GT}}^a, \mathbf{u}_{\text{GT}}^a)$ after action execution. The learned model produces the predicted output graph

$$G_{\text{out}}^a = (V_{\text{out}}^a, E_{\text{out}}^a, \mathbf{u}_{\text{out}}^a) = \text{GNN}_{\theta}(G_{\text{in}}^a) .$$

For different actions, the number of vertices in a scene graph varies. We extract the position $\mathbf{t} \in \mathbb{R}^3$ and orientation $R \in SO(3)$ from each vertex. The position and orientation error is calculated for each vertex and averaged over all vertices and actions:

$$\Delta t_{\text{mean}} = \frac{1}{\|A\|} \sum_{a \in A} \frac{1}{\|V_{\text{out},a}\|} \sum_{i=0}^{\|V_{\text{out},a}\|} \|\mathbf{t}_{\text{out},i}^a - \mathbf{t}_{\text{GT},i}^a\|$$

$$\Delta \alpha_{\text{mean}} = \frac{1}{\|A\|} \sum_{a \in A} \frac{1}{\|V_{\text{out},a}\|} \sum_{i=0}^{\|V_{\text{out},a}\|} \|\text{angle}((R_{\text{GT},i}^a)^{-1} \cdot R_{\text{out},i}^a)\|$$

The angle θ of the difference matrix $R_{\text{diff}} = (R_{\text{GT},i}^a)^{-1} \cdot R_{\text{out},i}^a$ can be computed via the trace, i. e. sum of the diagonal elements:

$$\text{tr}(R_{\text{diff}}) = 1 + 2 \cdot \cos(\theta)$$

$$\text{angle}(R_{\text{diff}}) = \theta = \text{acos}\left(\frac{\text{tr}(R_{\text{diff}}) - 1}{2}\right)$$

The mean and standard deviation of both errors are calculated separately over the different data subsets. The training, validation and test set contain generated data from simulation, while the real set contains data collected on the real robot.

Table 4.1.: Position and orientation error of the learned model

	Position Error in [cm]		Orientation Error in [°]	
	Mean	Stddev.	Mean	Stddev.
Training	0.83	0.57	4.21	6.54
Validation	0.86	0.61	5.35	7.77
Test	0.87	0.60	5.24	7.60
Real	1.84	2.66	14.82	23.83

The model was only trained on the synthetic training data. Table 4.1 shows the mean value and the standard deviation for the position and the orientation error as defined above.

The proposed model achieves an average position error of less than one centimeter and an orientation error of around 5° on the training, validation and test set, which were generated in simulation. On the real dataset the position error is below two centimeters and the orientation error is around 15° . The difference between the results for simulated and real data can be explained by the fact, that the model was trained only on simulation data. In order to assess the quality of these error, we can compare the prediction with the average motion of objects in the dataset as shown in Table 4.2. Especially the position is predicted very well, compared to an average movement of around 8 cm. For the orientation, the results on simulated data are good, but on the real data the prediction error is closer to the average motion in the dataset (15° vs 19°). However, the variance in the prediction is much lower. This underperformance of the model for orientation changes on real data is caused by different friction and inertial parameters of real objects compared to the simulation environment.

Combinatorial Generalization: Next, we investigated the ability of our model to cope with an unseen number of objects, i. e. , combinatorial generalization. We trained a model only on scenes containing 2 – 3 objects and tested the prediction errors in scenes with 1, 4 and 5 objects. In Table 4.3, we can see that scenes with fewer objects are no problem. Also, scenes with 4 or 5 objects only

Table 4.2.: Position and orientation changes for the data sets

	Position Change in [<i>cm</i>]		Orientation Change in [$^{\circ}$]	
	Mean	Stddev.	Mean	Stddev.
Simulation	8.45	15.92	18.61	34.69
Real	7.90	11.67	19.18	31.30

Table 4.3.: Performance on an unseen number of objects

Number of Objects	Position Error in [<i>cm</i>]		Orientation Error in [$^{\circ}$]	
	Mean	Stddev.	Mean	Stddev.
2 – 3	0.79	0.48	4.17	5.90
1	0.76	0.45	4.02	5.87
4	0.91	0.75	7.03	8.05
5	0.98	0.81	7.61	8.46

incur a minor increase in position and orientation error. The whole data set was used for this evaluation. This supports the claim that graph networks can be effectively used to learn models which are independent of the number of objects and even generalize to an unseen number.

Runtime: Regarding the runtime, we compare our approximate model with the full physics simulation. Executing a single push in MuJoCo takes on average 2.27s (with rendering disabled). Processing a batch of 1000 scenes with our prediction model takes around 1.16s. We conclude that our model is much faster than using the full physics simulation. All measurements were done on an Intel Core i7-5820K CPU with 3.30GHz. This runtime efficiency enables us to test hundreds of pushes in under a second and find push parameters, which achieve a certain manipulation goal efficiently.

4.2. Action Effect Prediction for Deformable Objects

Now, we present the extension of the graph-based prediction model to interactions between rigid and deformable objects. In particular, we investigate different actions involving a deformable bag and rigid spheres over longer time periods. Figure 4.7 shows examples of such interactions. Predicting action effects for deformable objects requires a more sophisticated representation that encodes the configuration, not only the object pose. Furthermore, predicting over longer time periods is a challenging problem, since prediction errors can accumulate fast if not handled explicitly.

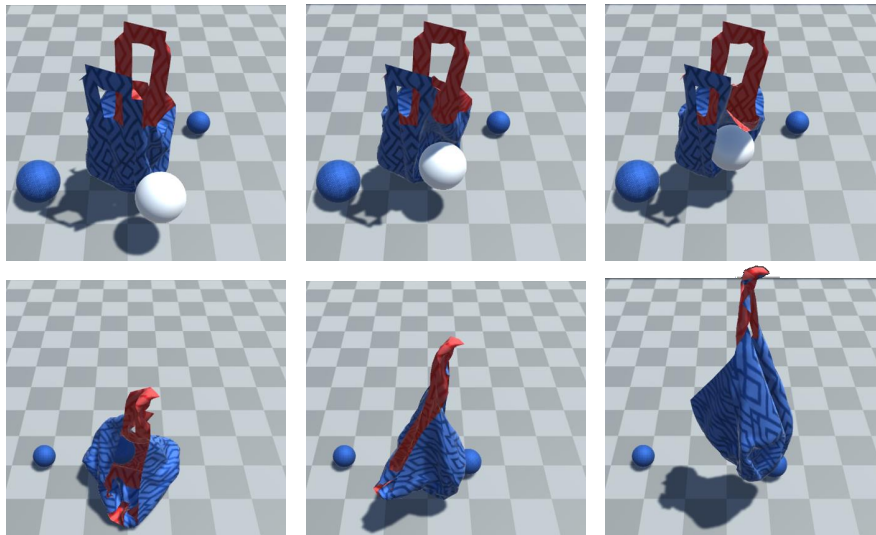


Figure 4.7.: Two example interactions with a deformable bag. The first row shows a white rigid sphere being pushed towards the bag, which is held in place at both handles. The second row shows the bag being lifted from the ground using one handle, while the other handle is left loose.

We start by introducing the term task in section 4.2.1. A task description contains the executed action, the objects in the scenes, and their initial configuration. In section 4.2.2, we present a publicly available dataset for interactions between a deformable bag and rigid objects. The data generation process is de-

tailed before the contents of the dataset are described. The action-centric graph representation for scenes is extended in section 4.2.3 to handle deformable objects. To this end, we propose a sparse keypoint representation that is able to encode different topologies. Then, section 4.2.4 proposes and implements a two-stage prediction model, which first classifies active vertices likely to move during the action and only predicts motion for those active vertices. By combining prediction models with different time steps, we further propose a mixed-horizon model to mitigate accumulation of errors over multiple time steps. Finally, the proposed models are evaluated on the dataset in section 4.2.5. Multiple ablation studies show the benefits of both the two-stage and the mixed-horizon aspects.

The main contributions can be summarized as follows. First, we build a novel publicly available dataset for task-specific action effect prediction by leveraging a deformable simulation engine. Scenes contain interactions between a cloth-like deformable object and multiple rigid objects. Second, we propose a method for predicting complex interactions between deformable and rigid objects by representing the scenes as graphs and building a two-stage prediction model, which first classifies which parts of the scene move at all and then applies position updates selectively in a second stage. By combining two-stage models with different prediction horizons, our method outperforms baseline approaches with only one prediction horizon. Parts of this section have been published in Weng et al. (2021).

4.2.1. Task Description

We consider tasks like opening a bag, pushing an object into a bag and moving a handle of the bag along a specific trajectory with constant speed. A task consists of a parameterized action, the objects in the scene, and further task parameters like how stiff the bag is (*Bag Stiffness*), whether the bag is empty or a rigid object is inside (*Bag Content*), and whether the handles are fixed in place, loose or moved along a trajectory (*Handle State*). Each scene contains a deformable bag, some number of rigid spheres, and a table. The bag can interact with rigid objects and the table.

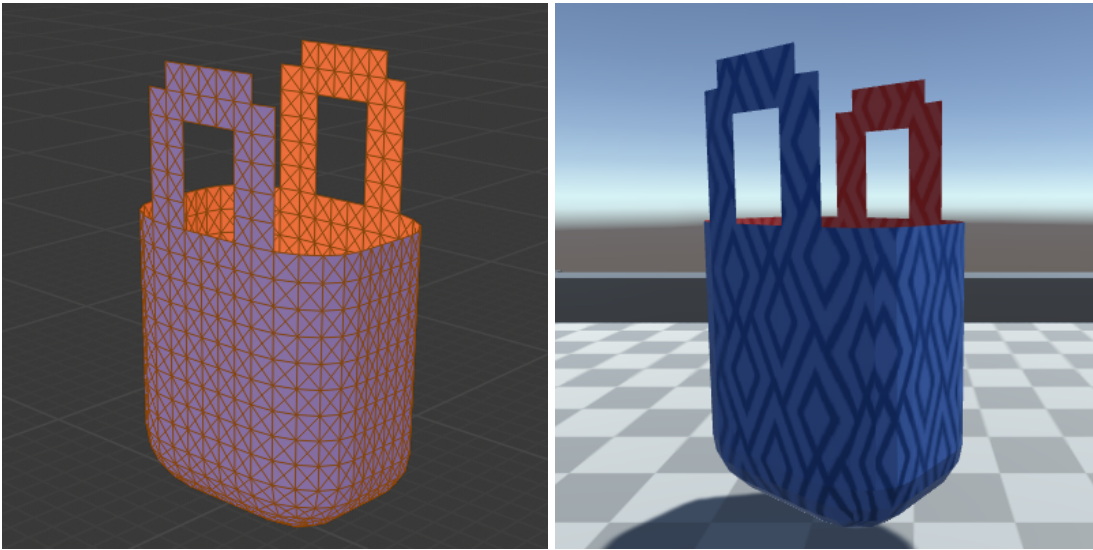


Figure 4.8.: The deformable bag in its initial pose. The left figure shows the bag template in Blender. The right figure shows the bag in the Unity simulation environment.

For the deformable bag, we model the mesh in Blender as shown in Figure 4.8. Compared to the cloth-like objects in previous works, our model has a more complicated structure. The whole mesh consists of 1277 particles and 4326 edges. For the actions, we investigate pushing an object towards the deformable bag, moving a handle of the bag along a circular trajectory, opening the bag, and lifting the bag. The handle motions are achieved by grasping the top part of a handle and moving it along a trajectory. We consider the following actions:

- *Pushing an object towards the bag:* We sample a position to create a sphere with a random radius around an existing object. A push trajectory is generated by sampling a planar motion direction pointing to either the bag or one of the other rigid objects. By applying this strategy, we ensure that most of the actions lead to meaningful object interactions.
- *Handle motion along circular trajectory:* We move one handle along a circular trajectory as shown in Figure 4.9 (left). The trajectory is placed in one of the coordinate planes. The radius and direction are randomized.
- *Opening the bag:* As shown in Figure 4.9 (right), we move one handle away from the other fixed handle in order to stretch the bag horizontally. Before

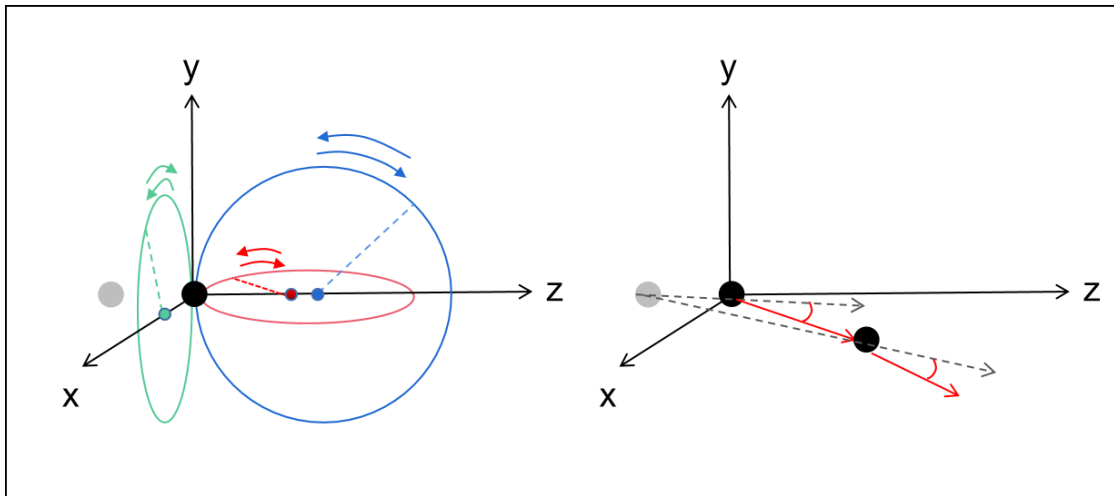


Figure 4.9.: Handle actions. The black point is the manipulated handle and the gray point is the non-targeted handle. The left figure shows examples of circular handle movement in three different coordinate planes. The right figure shows examples of opening actions.

performing the manipulation, we randomly choose a small horizontal deflection angle. During the manipulation, we calculate a base directional vector depending on the handle position differences and rotate it by using the deflection angle to construct the final moving vector.

- *Lifting the bag*: Before performing this action, the bag is dropped on the table. Then, one handle performs an upward motion, which lifts the bag from the table. The other handle is left loose.

4.2.2. Data Generation

We generate interaction data between the deformable bag and rigid spheres in simulation. In section 2.5, we analyzed different simulators for deformable objects, and identified Unity¹ with the Obi Cloth² extension as a suitable option for our use case. The Obi physics solver is optimized for real-time cloth simulation and supports particle-level manipulation, rich types of interactions, and

¹<https://unity.com>

²<http://obi.virtualmethodstudio.com>

editable physical constraints (e. g. distance constraints, bending constraints, and aerodynamics).

The simulated scenes include a deformable bag, a table, and multiple rigid spheres with random radii for each task. Further task parameters are generated as follows. By adjusting the bending constraints in the solver, we vary the stiffness of the bag material (*Bag Stiffness*). We either initialize the bag in an empty state or with a rigid sphere inside (*Bag Content*). The left and right bag handles are either left loose or grasped (*Handle State*). If a handle is grasped, it either is fixed in place or moves along a given trajectory (opening, lifting, or circular).

During action execution, we record the complete scene state 10 times per second. For every recorded time step, the scene state consists of the vertex positions of the deformable bag, the positions and radii of all rigid objects including the pushed sphere, and the grasped vertices on the left and right handle. Our goal is to learn task-specific models, therefore the dataset is grouped by task. For each task, we simulate 1000 trajectories, which results in 60000 recorded time steps. The simulated task data is split into training (80%), validation (10%), and test set (10%). We vary actions and task parameters to create data for 20 different tasks. For each row, we collect data for both bag stiffness values (soft and stiff). Figure 4.10 shows examples for simulated tasks. A detailed description of the dataset content can be found in appendix B.

4.2.3. Scene Representation

Based on the generated dataset, we want to learn task-specific prediction models for actions involving the deformable bag and other rigid objects. Given a scene state as a set of rigid and deformable objects O_t , the goal is to learn a dynamics model M to predict the future scene state O_{t+1} after performing action \mathbf{a}_t at time step t .

$$O_{t+1} = M(O_t, \mathbf{a}_t)$$

The set of rigid objects consists of a variable number of spheres whose state can be represented by their position and radius. The state of the deformable bag consists of the position of all vertices and their topology. The action \mathbf{a}_t is parameterized by the start position $\mathbf{p}_{start} \in \mathbb{R}^3$, the end position $\mathbf{p}_{end} \in \mathbb{R}^3$ and

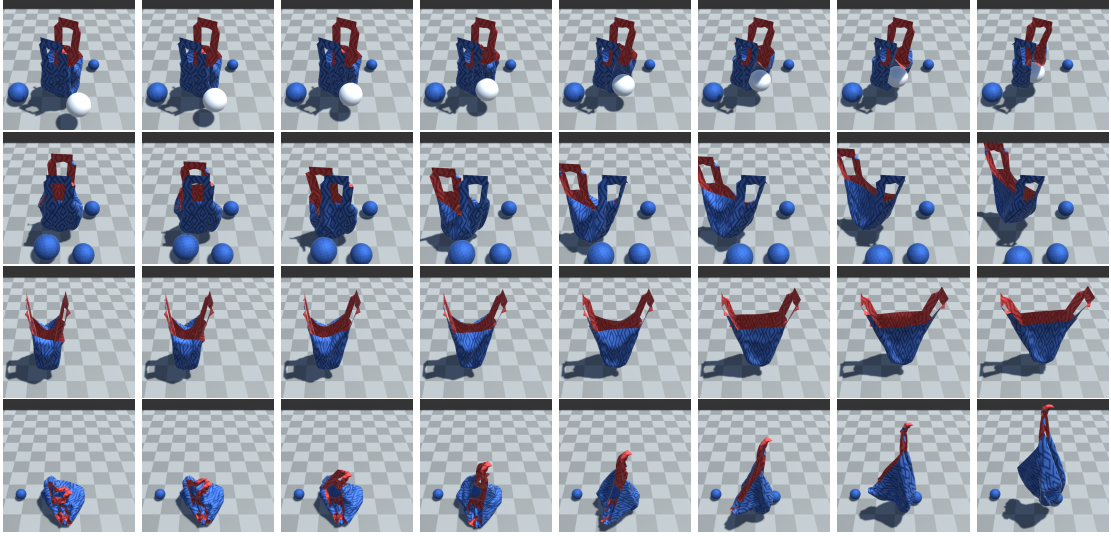


Figure 4.10.: Example trajectories of four actions in the dataset. Each row shows different time steps of an action. From top to bottom: pushing an object towards the bag, handle motion along circular trajectory, opening the bag, and lifting the bag.

the radius $r_a \in \mathbb{R}$ of the manipulated target, which can be either a rigid object or one of the bag's handles.

$$\mathbf{a}_t = (\mathbf{p}_{start}, \mathbf{p}_{end}, r_a)$$

We define a graph representation that captures the state of the rigid objects and approximates the state of the deformable bag using a set of sparse keypoints. We want to represent the state of the scene objects O_t and the action a_t at time step t as a graph $G_t = (V, E, \mathbf{u})$ with vertices V , edges E , and a global feature vector \mathbf{u} . The set of vertices V encodes position information about the rigid and deformable objects in the scene, see Figure 4.11.

The vertex feature vector $\mathbf{v} = (\mathbf{t}, r, f) \in \mathbb{R}^5$ encodes position $\mathbf{t} \in \mathbb{R}^3$, radius $r \in \mathbb{R}$, and a binary feature indicating whether the vertex is fixed in place $f \in \{0, 1\}$. Each rigid object is represented as a vertex with the feature vector $\mathbf{v} = (\mathbf{t}, r, 0) \in \mathbb{R}^5$. The fixed flag is never set for rigid, free-moving objects. For the deformable bag, we use a sparse keypoint representation, where task-relevant vertices are chosen from the bag's mesh. Each keypoint is represented as a vertex with a

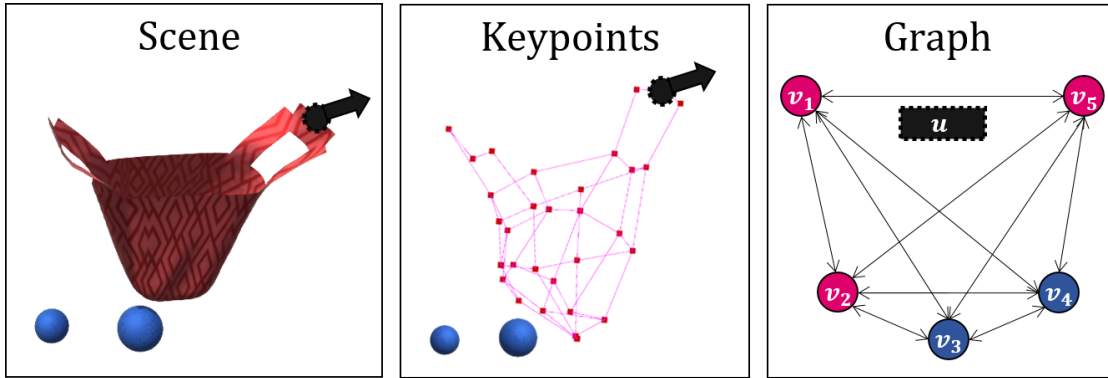


Figure 4.11.: We transform a scene consisting of deformable and rigid objects into a sparse keypoint representation. Based on the keypoints, we build a fully connected graph, whose vertices represent keypoints and whose edges encode the connectivity between them. The motion of the handle along the black arrow is encoded in the global graph feature \mathbf{u} .

feature vector $\mathbf{v} = (\mathbf{t}, 10^{-5}, f)$, where the radius r is set to a small constant value and f indicates whether it can freely move ($f = 0$) or is grasped, i. e. fixed in place ($f = 1$). Since the choice of a global coordinate system is arbitrary, we transform the positions to an action-centric, coordinate system, whose origin is the starting position \mathbf{p}_{start} of the manipulated object.

The edges E build a fully connected bidirectional graph between the vertices V . We use an edge feature vector $\mathbf{e} = (\mathbf{d}, c) \in \mathbb{R}^4$ consisting of the pairwise position differences $\mathbf{d} \in \mathbb{R}^3$ between vertices and the physical connection flag $c \in \{0, 1\}$. The edges connecting neighboring vertices from the deformable bag have their physical connection flag set to $c = 1$. All other edges have no direct physical connection ($c = 0$).

The global feature vector $\mathbf{u} = (\mathbf{p}_{end} - \mathbf{p}_{start}, r_a) \in \mathbb{R}^4$ encodes the position change $\mathbf{p}_{end} - \mathbf{p}_{start} \in \mathbb{R}^3$ of the manipulated target and the radius of the manipulated object $r_a \in \mathbb{R}$.

4.2.4. Prediction Models

Using the proposed scene representation, we formulate a two-stage graph learning problem to facilitate fixed time step predictions. Then, we combine multiple prediction models with different time step horizons to enable predictions of up to 60 time steps into the future.

Two-stage Graph Prediction Model: The purpose of the two-stage graph prediction model is, given the current scene state G_t , to predict the scene graph G_{t+h} after h time steps where h is constant. We call h the time step horizon of the prediction model. In this work, we address single time step horizons ($h = 1$) and longer time step horizons ($h = 5$). For each time step horizon h , we learn a dynamics model which consists of two separate modules: the Active Prediction Module (APM) and the Position Prediction Module (PPM). APM is a binary classifier predicting whether rigid objects or keypoints of the deformable bag will move in the next time step. The classification is done for every vertex in the scene state G_t . The ground-truth active labels are generated during training based on the position difference between the time steps t and $t + h$. PPM is a regression module that directly predicts the next scene state, i.e. the expected positions of all vertices at time step $t + 1$. Both APM and PPM are implemented as graph neural networks with an Encode-Process-Decode architecture as described in section 2.3.1.

The APM outputs a binary classification mask through a final softmax activation layer for the vertex features. The classification stage uses the following cross-entropy loss

$$L^{\text{APM}} = -\frac{1}{N} \sum_{i=1}^N \text{CrossEntropy}(y_i^{\text{gt}}, y_i^{\text{pred}}),$$

where N denotes the number of vertices in the scene graph, $y_i^{\text{gt}} \in \{0, 1\}$ is the ground-truth active flag and $y_i^{\text{pred}} \in [0, 1]$ is the predicted flag. The ground-truth active flag is set to 1 if the position difference is above a pre-set threshold.

PPM is a regression model to predict the scene graph after action execution using a final linear activation layer for the vertex features. The regression stage

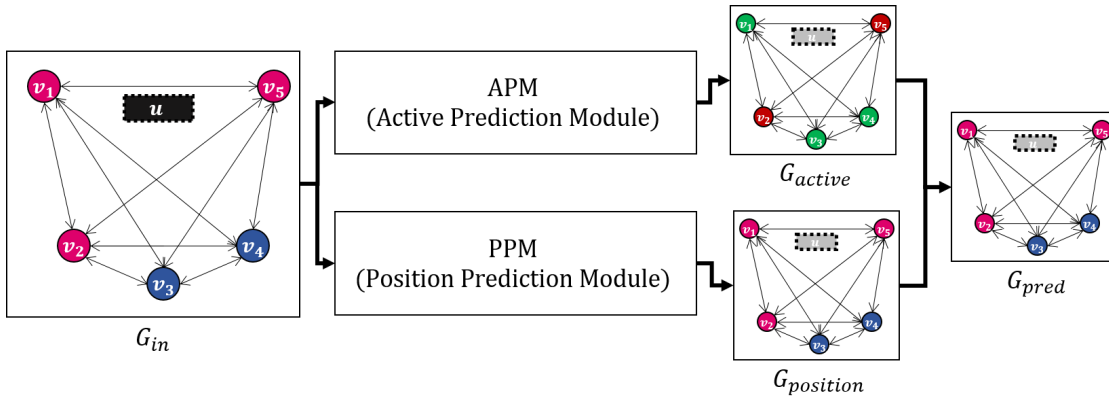


Figure 4.12.: The two-stage model takes as input the scene state as a graph G_{in} at a certain time step. This graph is fed into both the APM and PPM. The APM classifies which vertices are active, i. e. will move in the next time step. In the graph G_{active} , the green vertices have been classified as active and the red ones as inactive. The PPM predicts the positions of vertices in the next time step as a graph $G_{position}$. In a final step, only the position updates, whose corresponding vertices have been classified as active in G_{active} , are applied to the prediction result G_{pred} .

uses a mean square error loss L^{PPM} between the ground-truth and predicted positions:

$$L^{PPM} = \frac{1}{N} \sum_{i=1}^N (\mathbf{t}_i^{gt} - \mathbf{t}_i^{pred})^2$$

We train both models separately on the tasks in the generated dataset. By only applying the regression update to those vertices which have been classified as active, we prevent spurious motion of vertices that are not involved in the interaction between objects in the current time step. Figure 4.12 shows how the APM and PPM are combined to transform an input graph G_{in} into a predicted graph G_{pred} .

We define M_h^{PPM} and M_h^{APM} as the PPM and APM for a specific time step horizon h . We call the combination of APM and PPM the *two-stage* model (APM+PPM), whereas the regression stage alone is called *one-stage* model (PPM):

$$\begin{aligned} M_h^{\text{one-stage}}(G_i) &= M_h^{\text{PPM}}(G_i) \\ M_h^{\text{two-stage}}(G_i) &= M_h^{\text{APM}}(G_i) \odot M_h^{\text{PPM}}(G_i) \end{aligned}$$

Here the operator \odot only applies the position updates from the PPM if the vertices have been classified as active by the APM.

Long Horizon Prediction Model: The one-stage and two-stage prediction models only predict the scene for a fixed prediction horizon h . The longer horizon model M_5 is trained with a prediction horizon $h = 5$, and the single time step model M_1 is trained with a horizon $h = 1$. By chaining these models recursively together, we can make predictions for any time step t .

If we only use the single time step model M_1 , we can predict the scene state G_t after t time steps given the initial scene state G_0 :

$$G_t = \underbrace{(M_1 \circ M_1 \circ \dots \circ M_1)}_{t \text{ times}}(G_0)$$

Here, we can either use the one-stage or the two-stage model. However, this causes the prediction error to accumulate fast. We can alleviate this problem, by also incorporating the longer horizon model M_5 . First, we run M_5 recursively for $\lfloor t/5 \rfloor$ steps. Then, M_1 is run for the remaining time steps $t \bmod 5$:

$$G_t = \underbrace{(M_1 \circ M_1 \circ \dots \circ M_1)}_{(t \bmod 5) \text{ times}} \circ \underbrace{(M_5 \circ M_5 \circ \dots \circ M_5)}_{\lfloor t/5 \rfloor \text{ times}}(G_0)$$

We call this combination of a multi-step prediction and a single-step prediction the *mixed-horizon* prediction model (see Figure 4.13 for an example).

4.2.5. Evaluation

The evaluation investigates the benefits and drawbacks of the proposed prediction models by answering the following questions:

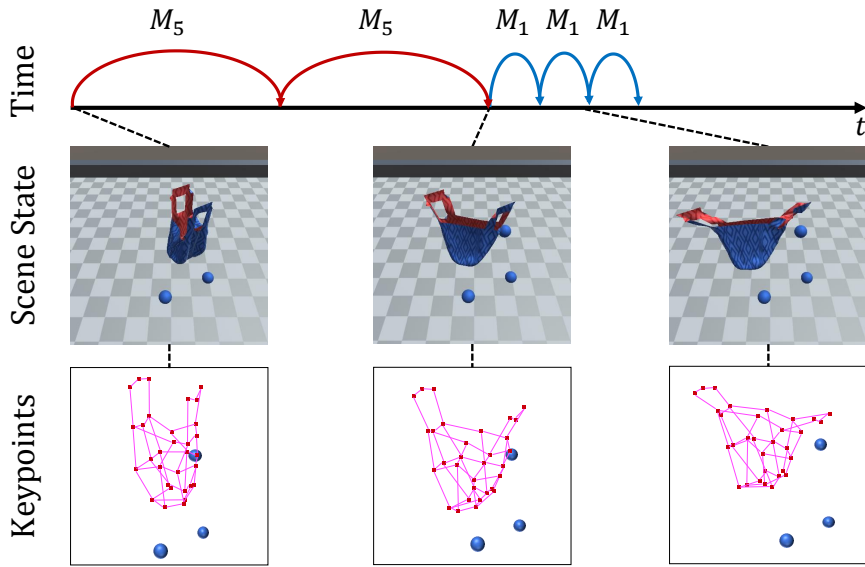


Figure 4.13.: The mixed-horizon model combines prediction models with different time steps. M_5 and M_1 are models with time step horizons of 5 and 1 respectively. The first row shows the time line and the prediction steps of M_5 and M_1 . The second row shows the scene state including the deformable bag and rigid spheres at selected time steps. The third row shows the sparse keypoint representation of the scene state.

1. Does the inclusion of the APM in the *two-stage* model improve the prediction results over the *one-stage* model with the PPM alone?
2. How does the material stiffness of the deformable bag influence the prediction accuracy?
3. Does the mixed-horizon model improve long-term prediction results compared to an iterative application of one-stage or two-stage models?

Metrics: As evaluation metrics, we consider the mean position error between ground-truth positions and predicted positions. These include both the positions of rigid bodies as well as the positions of vertices on the deformable bag. The mean position error metric is used to compare the overall performance of the investigated models.

In addition, we want to analyze the inter-task variance of the results. We calculate the mean position error per task. We calculate the standard error via the standard deviation of the task-specific mean position errors. In the following figures, the standard error is either shown as whiskers for bar plots or as colored areas for line plots.

Results: To answer the first question, we conduct an ablation study to quantify the benefits of the two-stage model compared to the one-stage model during single time step predictions. Both models are evaluated with the time step horizon $h = 1$. Fig. 4.14 shows that the two-stage model decreases the mean position errors while also lowering the inter-task variance. This effect can be equally seen in the training, validation and test set. Therefore, we conclude that the APM improves single time step predictions when compared to the PPM alone.

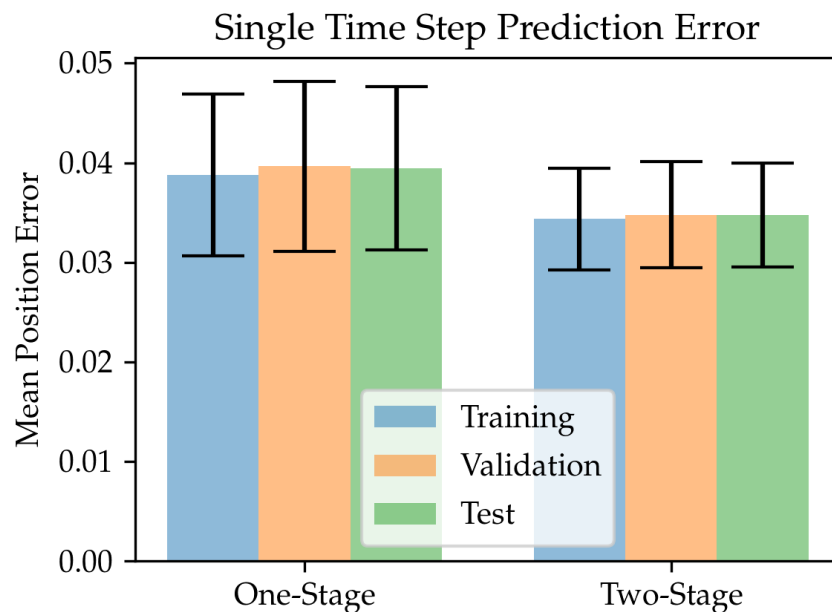


Figure 4.14.: Single time step prediction errors over all tasks for training, validation, and test set. The mean position error is shown as the bar height and the whiskers show the standard error over all tasks.

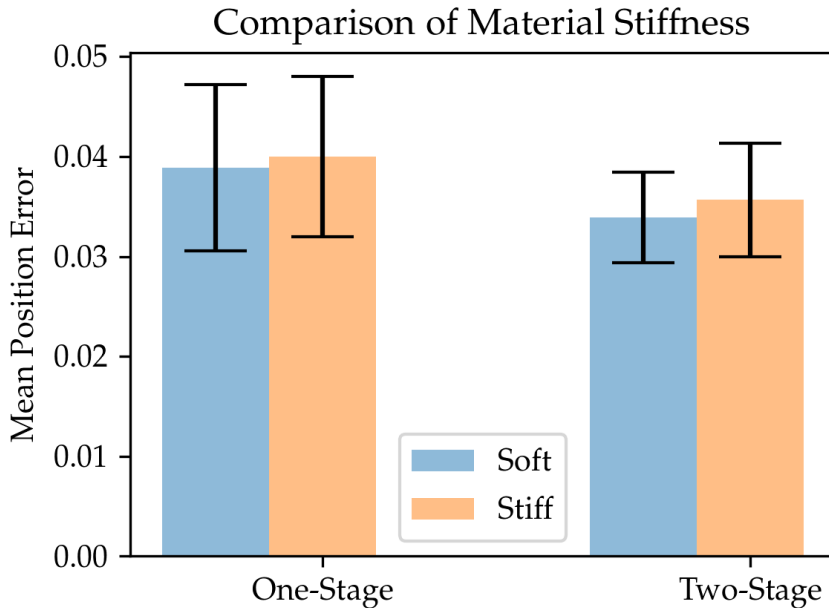


Figure 4.15.: Single time step prediction errors over all tasks grouped by material stiffness. The mean position error is shown as the bar height and the whiskers show the standard error over all tasks.

To address the second question, we compare the single time step prediction results for soft bag material with results for stiff bag material. Fig. 4.15 shows the mean position error and the standard deviation for both materials as well as the one-stage and two-stage prediction models. As can be seen, the tasks with soft bag material have a smaller prediction error. However, the difference is lower than the inter-task variance, indicating that both models are able to handle tasks independent of material stiffness.

For the third question, we compare the long horizon prediction results for the recursive one-stage, two-stage and mixed-horizon models on the test set. We initialize each model with the scene state G_0 at time step $t = 0$ and apply the prediction recursively as described in the section 4.2.4. Figure 4.16 shows the mean position errors and standard error for the four actions *Pushing an Object towards the Bag*, *Handle Motion along Circular Trajectory*, *Opening the Bag*, and *Lifting the Bag*. As can be seen in the different shapes of the graphs, the long horizon prediction performance variance between actions is quite high. However,

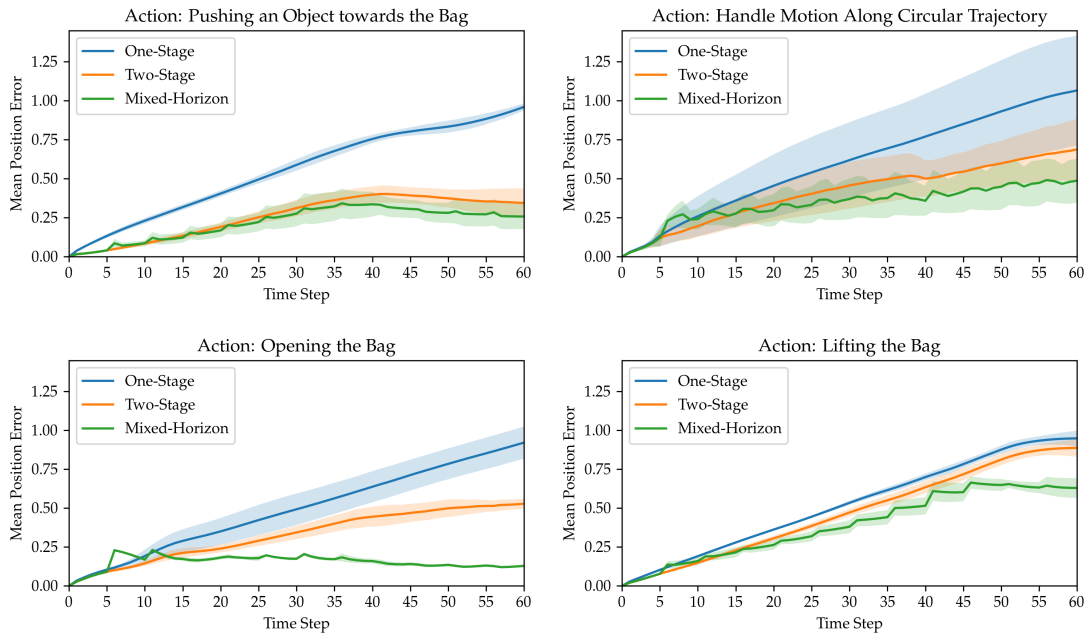


Figure 4.16.: Long horizon prediction errors per action for the one-stage, two-stage, and mixed-horizon models. The solid lines show the mean position error while the colored area around the line indicate the standard deviation.

a trend can be identified if the data is grouped by action. We can see that the two-stage model outperforms the one-stage model consistently, independent of the action. The difference between the models in the lifting action is quite small, since almost all parts of the bag move during this action. Therefore, the first movement classification stage is not as helpful as in the other actions. Furthermore, the mixed-horizon model outperforms the two-stage model for longer term predictions, while sometimes producing worse results for short term predictions. Depending on the action, the mixed-horizon model produces much better predictions than the two-stage model (e.g. opening the bag), while for other actions the improvement is marginal (e.g. pushing an object). For some actions, i. e. pushing, opening and to some degree lifting, the error of the mixed-horizon model actually decreases for later time steps when compared to earlier time steps. For example, the graph for the pushing action reaches the maximum error at around $t = 35$ and then decreases. Counter-intuitively, the later time

steps are sometimes easier to predict than the earlier. This can be explained by the fact, that the earlier time steps often include dynamic interactions between the deformable object and the other objects in the scene, which are hard to predict. The final state in the later time steps is more stable and is therefore easier to predict. Overall, the mixed-horizon model is better suited for predictions over a longer time periods than the one-stage and two-stage models.

4.3. Summary and Review

This chapter proposed, implemented and evaluated graph-based representations and methods for action effect prediction for both rigid and deformable objects. To this end, the following research questions have been addressed:

- How can action effect prediction models for scenes with a varying number of interacting objects be learned?
- How effective are models learned from simulation to predict effects on real-world data?
- How can graph representations for scenes containing both deformable and rigid objects be built?

First, a data-driven method for predicting pushing effects for multiple interacting rigid objects was proposed. This method relies on a graph-based scene representation and graph neural networks as a model. Training data was generated in simulation and evaluated on both simulated and real data. Second, the proposed method was extended to handle deformable objects and a wider range of actions, including lifting, opening and interacting with a cloth-like bag.

Action Effect Prediction for Rigid Objects: The proposed method is based on a scene graph representation, where the vertices encode information about the rigid objects and the edges encode information about spatial relations between objects. A graph neural network is used as a prediction model and trained on data, which was generated in a physics simulator. Additional evaluation data was collected during real-world robot experiments. In the evaluation, we show that the proposed model achieves high prediction accuracy in scenes with

a varying number of objects and, in contrast to state-of-the-art approaches, is able to generalize to scenes with more objects than seen during training. The goal was to build a prediction method based on intuitive physics, which does not need to know a physics parameters a-priori. Although there are some cases, in which the prediction model is slightly inaccurate, it is still able to predict plausible outcomes of the push actions. Thus, we will demonstrate that this push prediction system can be used on a robot to estimate action effects in the next chapter.

Action Effect Prediction for Deformable Objects: An extension to the prediction method for push effects is proposed and evaluated, which also handles deformable objects and considers additional actions besides pushing. We present a novel dataset for action effect prediction on scenes containing both rigid and cloth-like deformable objects. Then, we define a graph representation for the scene state, where the vertices are keypoints representing objects and their parts. Our predictive model can generalize to different numbers of vertices in the graph, allowing us to extend the method to different deformable objects in the future. Furthermore, we propose a two-stage prediction model to capture the dynamics based on graph neural networks. Additionally, we implement a mixed-horizon model on top of the learned modules to predict the future scene state and show the effectiveness of our methods in different tasks. In the evaluation, we demonstrate the benefits of the two-stage prediction compared to a one-stage model for single time step predictions. The mixed-horizon model outperforms both the one-stage and the two-stage model significantly for long time horizons. In summary, this demonstrates the effectiveness of the sparse keypoint representation for deformable objects and the mixed-horizon model for action effect prediction over multiple time steps.

5. Application to Robot Manipulation Tasks

The two previous chapters introduced methods for understanding support relations between objects and predicting action effects and evaluated these methods based on simulated and real data collected on the robot. This chapter will apply the proposed methods to robot manipulation tasks.

By extracting the support relations in a scene with multiple objects as described in chapter 3, the robot can infer a manipulation order by first picking the object, which has the highest probability of not supporting other objects, i. e. which is a leaf of the support graph. However, due to the uncertainty in perception, which is reflected in the probabilistic representation of the scene and the support relations, the robot cannot be sure whether this manipulation order does not lead to other objects potentially falling. Furthermore, due to the support polygon analysis introduced in section 3.3, the support graph can contain cycles representing potential support from objects above. To equip a humanoid robot with the ability to detect and handle these cases appropriately, we propose a manipulation strategy based on support relations and describe it in section 5.1.

The action effect prediction in chapter 4 gives a robot the ability to estimate scene states after it executes an action. Action sequences to achieve a particular manipulation goal are often generated by symbolic planners. They take into account the symbolic preconditions and postconditions of executable actions, e. g. an object can only be moved to a free, non-occupied place (precondition) and after moving the object the place is occupied (postcondition). However, the concrete parameters of, for example, a pushing action to achieve the desired postconditions still need to be chosen. These parameters are subsymbolic, e. g. in which direction to push and how far to push, and cannot be chosen by a sym-

bolic planner. Given a sequence of actions with their pre- and postconditions in symbolic form, we propose to leverage action effect prediction and describe it in section 5.2.

This chapter addresses the following research questions:

- How can a manipulation order be derived from support relation, maximizing the probability of a successful execution?
- How can cycles in the support graph be handled during manipulation?
- How can actions be parametrized to achieve desired postconditions?

Parts of this chapter have been published in Kartmann et al. (2018), Paus et al. (2020), and Paus and Asfour (2021).

5.1. Support Relations for Manipulation Tasks

We first introduce a method to derive a safe manipulation order from probabilistic support relations extracted using the ACT relation and support polygon analysis as described in chapter 3. Then, a bimanual manipulation strategy for a humanoid robot to support potentially falling objects is proposed and implemented.

5.1.1. Deriving Manipulation Order from Support Relations

When confronted with stack of objects in a scene, we want to derive a manipulation sequence for the objects in the scene, which does not cause other objects to fall or start moving during the individual manipulation actions. In particular, we consider picking actions, where the robot grasps and lifts an object from the scene. We aim at deriving a manipulation order as seen in Figure 5.1. To achieve this, we determine the object in the scene, which is most likely to not support other objects, remove this object from the scene, and repeat this process. To estimate how likely an object does not support other objects, we rely on the probabilistic support relations as extracted in section 3.3.

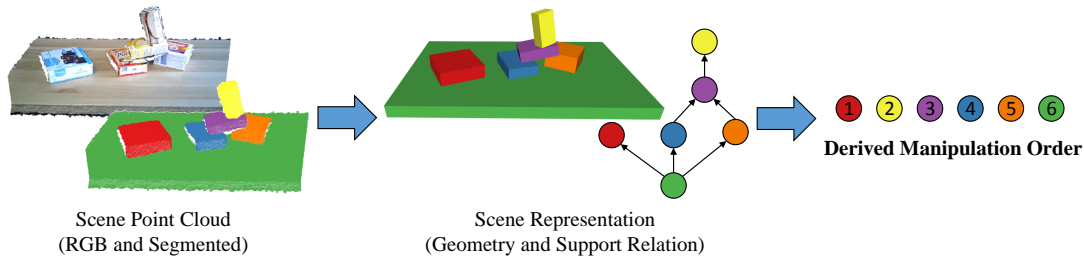


Figure 5.1.: From the point cloud of the scene, we derive a probabilistic scene representation as described in chapter 3. From this scene representation, we want to derive a manipulation order, i. e. sequence of objects to manipulate.

Problem Definition: Given a scene consisting of objects \mathcal{O} and a probability distribution over support relations $P(\text{SUPP}(A, B) \mid \mathcal{O})$, $A, B \in \mathcal{O}$, we want to determine an ordered sequence of the objects in \mathcal{O} , which is most likely to not cause undesired side effects, e. g. other objects falling as a consequence of an action execution.

Approach: First, we calculate the probability that any object $A \in \mathcal{O}$ is safe to manipulate, i. e. not supporting other objects, by multiplying the probabilities of no support to all other objects $B \in \mathcal{O} \setminus \{A\}$:

$$P(\text{SAFE}(A)) = \prod_{B \in \mathcal{O} \setminus \{A\}} P(\neg \text{SUPP}(A, B)) = \prod_{B \in \mathcal{O} \setminus \{A\}} (1 - P(\text{SUPP}(A, B)))$$

Now, we can determine the first object in the manipulation order $S \in \mathcal{O}$, which has the highest probability of being safe to manipulate:

$$S = \arg \max_{A \in \mathcal{O}} (P(\text{SAFE}(A)))$$

To determine the sequence of objects that define the manipulation order, we first remove the object S from the object set \mathcal{O} , and repeat the procedure of determining the object with the highest probability of being safe until all objects have been removed from \mathcal{O} .

5.1.2. Bimanual Manipulation Strategy

The manipulation order as derived in section 5.1.1 only considers cases, where single objects in the scene can be safely manipulated. However, there are cases, where this assumption is too limiting. Figure 5.2 illustrates the case, where an object supports another from above, i. e. removing the coffee filters box would cause the yellow container beneath it to fall. With the support polygon analysis (section 3.3.1), these support relations can be detected. Here, we propose to handle these cases with a bimanual manipulation strategy, where one hand of the humanoid robot supports the potentially falling object, while the other hand executes the desired manipulation, e. g. lifting or pushing an object.

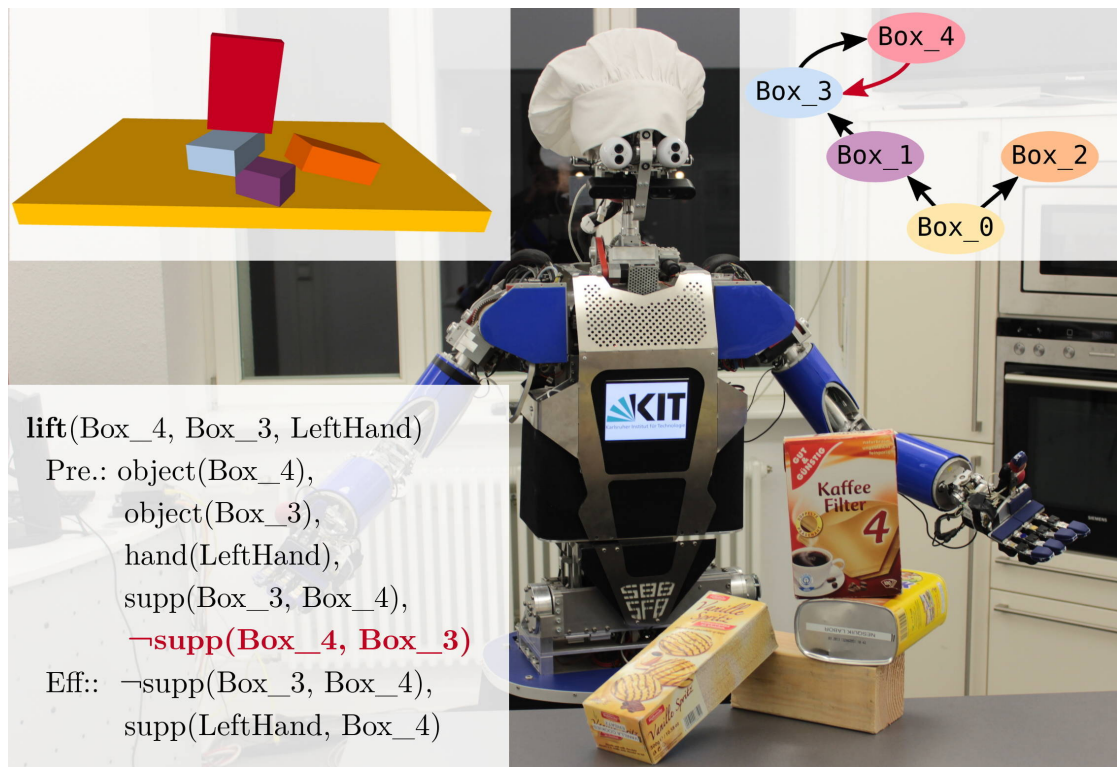


Figure 5.2.: The humanoid robot ARMAR-III segmented the scene based on RGB-D images. The support relations are visualized in a graph, in which uncertain edges are marked red. In order to lift the coffee filters on top of the pile, a precondition requires no top-down support.

Consider the scene in Figure 5.2. The robot’s task is to lift the coffee filters box on top of an object stack. The robot extracted a probabilistic scene representation from the point clouds captured by its depth camera. Based on scene representation, physically plausible but not necessarily accurate support relations are extracted and represented as a directed graph, in which objects are the vertices and support relations are the edges. The edge from the top-most object to the box underneath is marked red, indicating that such support relation creates a cycle, requiring special attention during manipulation. Lifting the coffee filters (`Box_4`) might cause falling of the underlying box (`Box_3`). We can predict these action effects by observing the preconditions of the lift action, which require the non-existence of a support relation between the lifted object and the box underneath (highlighted in red in Figure 5.2). Instead of not executing the action, our approach employs a bimanual manipulation strategy by which one hand executes the action, and the second hand secures potentially falling objects. Further, we use force measurements obtained from force-torque sensors in the robot’s wrists to detect whether an object fell into the second hand and update the support relations accordingly. To this end, we propose bimanual manipulation strategies to cope with the inherent uncertainty about the consequences of action executions.

Detection of the Need for Bimanual Manipulation: If the robot has chosen the next object A to manipulate, either by the manipulation order defined above or by some task given to it, we want to determine whether it is safe to manipulate the object with a single hand or whether support with a second hand is needed. A second hand is needed if A still supports other objects from above as computed by the support polygon analysis. We can compute this joint probability for every other object $B \in \mathcal{O}$:

$$P(\text{SUPP}(B, A) \wedge \text{SUPP}(A, B)) = P(\text{SUPP}(B, A)) \cdot P(\text{SUPP}(A, B) \mid \text{SUPP}(B, A))$$

Note that the conditional probability $P(\text{SUPP}(A, B) \mid \text{SUPP}(B, A))$ models the case, where we have support from above from A to B , which is only possible once support from from B to A through the ACT relation exists. If this probability exceeds a chosen threshold for any $B \in \mathcal{O}$, the need for bimanual manipulation is detected.

We consider two actions `push` and `lift`, which the robot can execute in order to manipulate a pile of objects. Listing 1 specifies actions, their preconditions, and effects. We use the predicate `OBJECT(A)` to denote that `A` is an object with which the agent can interact. The predicate `HAND(H)` identifies the end-effector `H` used to execute the action, and `SUPP(A, B)` requires a support relation between objects `A` and `B`.

Uncertainties in the extraction of support relations or the existence of support from above might lead to the execution of actions, which violate the defined preconditions or have additional undesired effects on the world state. We focus on the undesired effects on the support graph, i. e. pushing or lifting an object causes other objects to change their support relation. If the action contains a precondition that relies on the non-existence of a support relation involving the object to be manipulated, we can still execute the action using a bimanual manipulation strategy. Consider the case of lifting the coffee filters (`Box_4`) in Figure 5.2, which is supported by (`Box_3`). Support polygon analysis adds a support relation `SUPP(Box_4, Box_3)`, which will be detected. Using the second hand of the humanoid robot, we can prevent these undesired changes in the scene structure.

Securing Potentially Falling Objects: Given an object set \mathcal{O} , a support relations between them, an action a on a target object $T \in \mathcal{O}$ and an object $S \in \mathcal{O}$, which are in a potential support relation `SUPP(T, S)`, we want to execute action a , prevent any undesired effects caused by the existence of `SUPP(T, S)` and detect whether `SUPP(T, S)` was true in the initial scene.

We solve this problem by using one hand H_T to execute the primary action on the target object `T`, and the other hand H_S to secure the supporting object `S`. If `S` would fall after or during the action execution, instead of falling unpredictably, a new support edge `SUPP(H_S, S)` from the securing hand H_S to the supporting object `S` is added. Using force-torque sensors in the hand of the robot we can decide whether `SUPP(H_S, S)` needs to be added to the graph.

Algorithm 5 shows the implementation of the safe manipulation strategy. First, we decide which hand executes the action and which hand secures the supporting object. The action hand is chosen by a simple heuristic. If the target object's

```
(define (domain robot-actions)
  (:predicates (object ?o) (hand ?h) (supp ?s ?o))

  ; Push object ?a on object ?b using hand ?h
  (:action push
   :parameters (?a ?b ?h)
   :precondition (and (object ?a)
                      (object ?b)
                      (hand ?h)
                      (supp ?b ?a)
                      (not (supp ?a ?b)))
   )

  ; Lift object ?a from object ?b using hand ?a
  (:action lift
   :parameters (?a ?b ?h)
   :precondition (and (object ?a)
                      (object ?b)
                      (hand ?h)
                      (supp ?b ?a)
                      (not (supp ?a ?b)))
   :effects (and (not (supp ?b ?a))
                 (supp ?h ?a))
   )
)
```

Listing 1: Definition of operators, i.e. actions which the robot can execute, including their preconditions and effects using STRIPS notation (Fikes and Nilsson, 1971).

position is to the right of the robot's base, we choose the right hand and vice versa. Then, we can calculate the secure and target poses T_S and T_T needed to execute the action safely. Before executing the action, the robot should move its end-effector to a suitable pre-pose $T_{\text{pre},T}$. We solve the Inverse Kinematics (IK) for the kinematic chain consisting of both arms, to allow bimanual manipulation. Using the resulting joint values q_{Body} , the robot moves its end-effectors to the secure pose and action pre-pose. Before executing the desired action, the force/torque measurements from the force/torque sensor in the wrist of the supporting hand are used for change detection. The force measurements are filtered using a median derivative filter to detect changes more easily. If the filtered force measurements in the global z -direction (direction of gravity) exceed the predefined force threshold F_{max} , the support relation existed and undesired side effects were prevented by adding $\text{SUPP}(H_S, S)$. Otherwise, the edge $\text{SUPP}(T, S)$ did not exist in the initial scene and can be removed from the corrected graph.

Determining Action and Support Hand Poses: The hand poses $T_T, T_{\text{pre},T}, T_S \in SE(3)$ need to be determined before the bimanual IK can be calculated. The action hand poses T_T and $T_{\text{pre},T}$ are chosen depending on the action, which should be executed. For a lifting action, the object needs to be grasped. Therefore, a grasp is generated using a skeleton-based grasp planner (Vahrenkamp et al., 2018) that uses the geometry information of the target object T . For the pushing action, the center point of the object is chosen as the translation, while the orientation is predefined. The pre-pose $T_{\text{pre},T}$ is a translational offset away from the target pose along the grasp approach or push direction.

The supporting hand pose T_S is determined by a polygon analysis and collision checks between the hand model and the geometric models of the scene.

Detecting Fallen Objects using the Force/Torque Information: In order to compensate for noise and drift in the force/torque measurements of our robot, we use a median derivative filter to detect the fall of an object onto the securing hand. Let $F_w(k)$ be the latest $w \in \mathbb{N}$ force sensor values at time step $k \in \mathbb{N}$:

$$F_w(k) = \{f(k), f(k-1), \dots, f(k-w+1)\},$$

Algorithm 5: Safe Manipulation Strategy

Input: H_R, H_L : Right and left hand/end-effector
 T, S : Target and supporting object
 a : Action to be executed
 F_{\max} : Force threshold for fall detection
 $\mathcal{G}_s = (\mathcal{O}, E)$: Estimated support graph
Output: \mathcal{G}_{new} : Resulting support graph
 $\mathcal{G}_{\text{corr}}$: Corrected initial support graph
 $H_T, H_S = \text{ChooseHands}(H_R, H_L, T, S);$
 $T_S = \text{CalculateSecurePose}(H_S, S);$
 $T_T = \text{CalculateTargetPose}(a, H_T, T);$
 $T_{\text{pre},T} = \text{CalculatePrePose}(a, H_T, T_T);$
 $\mathbf{q}_{\text{Body}} = \text{SolveBimanualIK}(H_T, T_{\text{pre},T}, H_S, T_S);$
 $\text{MoveBody}(\mathbf{q}_{\text{Body}});$
 $FT = \text{CreateFilteredForceTorqueSensor}(H_S);$
 $\text{ExecuteAction}(a, \mathbf{p}_T);$
 $E_{\text{new}} = E_{\text{corr}} = E;$
if $FT.\text{MaxForceValue}.z > F_{\max}$ **then**
 $E_{\text{new}} = E \cup \{(H_S, S)\};$
else
 $E_{\text{corr}} = E \setminus \{(T, S)\};$
end
return $((\mathcal{O}, E_{\text{new}}), (\mathcal{O}, E_{\text{corr}}));$

where $f(k) \in \mathbb{R}$ is the reported force value along the direction of gravity at time step k . Let $\text{median}(F)$ return the median of a finite set $F \subset \mathbb{R}$. Then, the filtered value $\hat{f}(k) \in \mathbb{R}$ at time step $k \in \mathbb{N}$ is determined as follows:

$$\hat{f}(k) = \text{median}(F_v(k)) - \text{median}(F_w(k))$$

where $v, w \in \mathbb{N}$ are window sizes with $w \gg v$. The filtered value will be close to zero if the applied force does not change. However, a caught object will result in a significant peak (for an example see Figure 5.3).

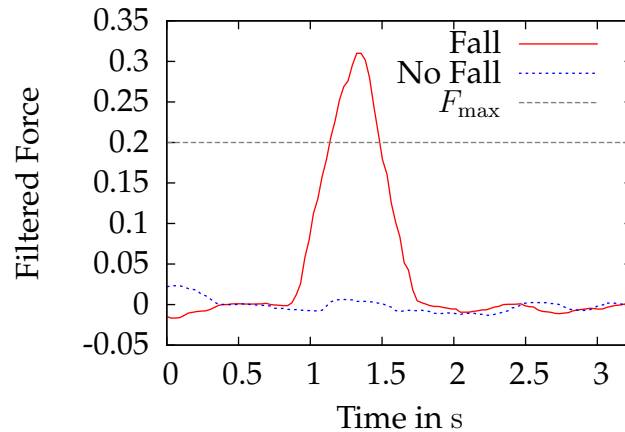


Figure 5.3.: Filtered force values during a push action. A fallen object produces a clear peak, while the filtered force values stay around zero when the object remains on the pile.

5.1.3. Validation Experiments

The proposed bimanual manipulation strategy was implemented and tested on the humanoid robots ARMAR-III (Asfour et al., 2006) and ARMAR-6 (Asfour et al., 2019). Both robots have two arms equipped with under-actuated five-fingered hands and force/torque sensors in the wrists.

Experiments with ARMAR-III: In order to validate the bimanual manipulation strategy, we conducted experiments on the humanoid robot ARMAR-III. We purposefully created scenes that contained uncertain top-down support relations to trigger the safe manipulation strategy. The robot was given the task of either lifting or pushing an object which potentially supports another object underneath it. We chose the same parameters for the support extraction as in section 3.4, configured the force-torque filter with $v = 11$, $w = 101$ (corresponding to time spans of about 300 ms and 3000 ms in our setup) and set the force detection threshold to $F_{max} = 0.2$ N.

In the first scenario, the task was to lift the coffee filters on top of the yellow container (see Figure 5.4). We shifted weights inside the container to alter the mass distribution creating both cases of existing and non-existing top-down support.

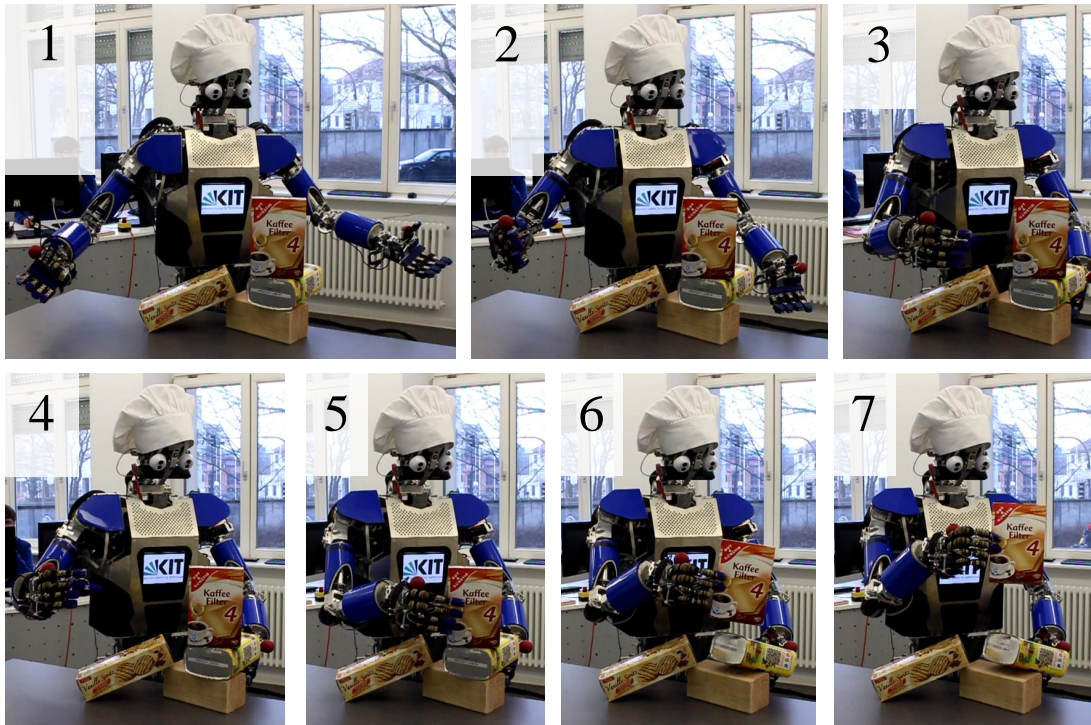


Figure 5.4.: ARMAR-III perceives the scene and extracts support relations (1). Then, the action hand is moved to the pre-pose for grasping and the supporting hand is moved to secure the potentially falling object (2 and 3). First, the coffee filters box is grasped (4 and 5) and then lifted (6 and 7). During the lifting, the yellow box falls onto the supporting hand.

The second scenario involved a push action where we changed the positioning of the supporting blue cereal box to provoke both support cases. The robot was able to prevent the fall of the bottom object in the case of real top-down support and detect the fall of the object using its force-torque sensors (see Figure 5.3). In case of no top-down support, we detected that the object did not fall on the robot's hand.

Experiments with ARMAR-6: In addition, experiments with ARMAR-6 were conducted to show that the methods are transferable to different humanoid robots. Figure 5.5 shows such a scenario, where the humanoid robot ARMAR-6

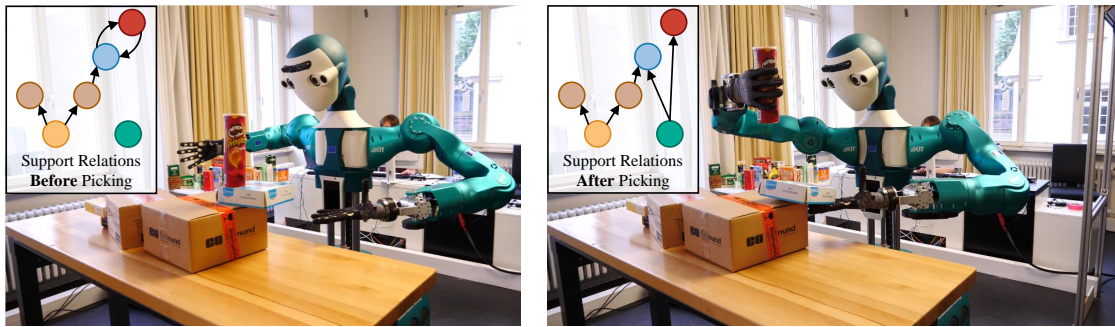


Figure 5.5.: ARMAR-6 picks a chips' can with the right hand and supports another object with the left hand to prevent the scene from collapsing.

tries to pick the chips can from a stack of objects. Using the probabilistic support relations, the robot is able to infer that the box below the chips can is likely to fall, and prevents this by supporting the box with the left hand.

5.2. Parametrizing Actions using Prediction

Now, we present an approach for parametrizing pushing actions based on a learned prediction model as described in chapter 4. These pushing actions must fulfill constraints given by a high-level planner, e. g., after executing a push action the brown box must be to the right of the orange box.

Figure 5.6 shows ARMAR-6 in front of a table with multiple objects. The robot has the task of bringing the large brown screw box to the left of the small orange screw box. The image shows the robot executing one of the pushes required to reach the goal state. For a given scene and goal state, the robot generates a set of possible pushing action candidates by sampling the parameter space and then evaluating the candidates by internal simulation, i. e. by comparing the predicted effect based on the internal model with the desired effect provided by the high-level planner.

This section is structured as follows. First, we determine action parameters to achieve desired postconditions based on sampling the action parameter space

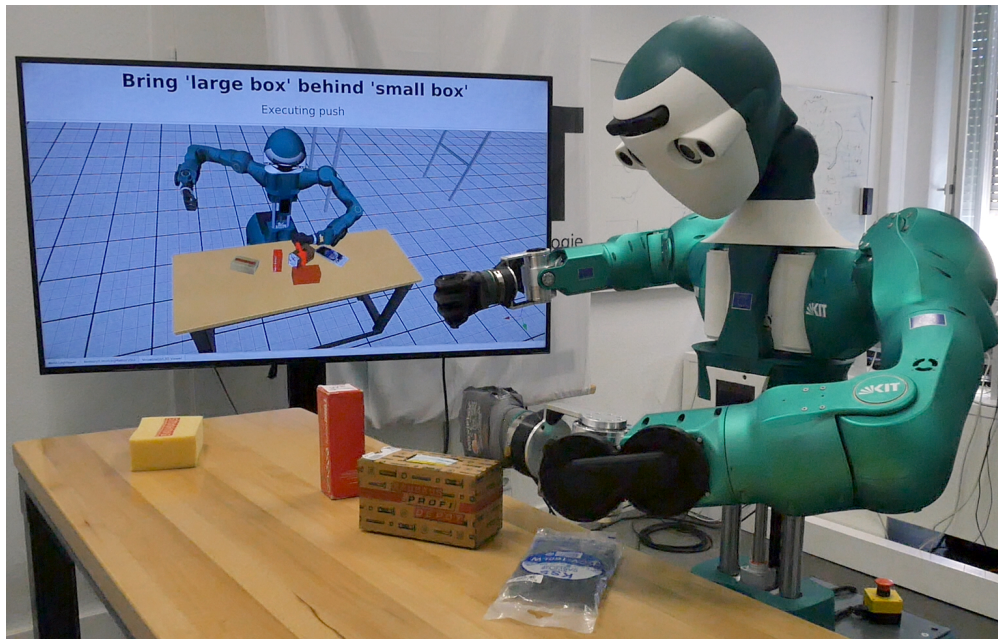


Figure 5.6.: ARMAR-6 has the goal of moving the large brown screw box behind the small orange box. The robot samples the parameter space of possible pushes and evaluates them according to the learned internal prediction model. We choose push parameters, which are predicted to produce the desired outcome. On the monitor, the red arrow indicates the chosen push to be executed and the red box shows the predicted pose of the brown box after the push.

and predicting action effects. Afterwards, this method is implemented and integrated into the robot software architecture. After explaining the details for executing pushing actions on ARMAR-6, the section concludes with validation experiments.

5.2.1. Determining Action Parameters based on Prediction

Given a set of symbolic actions defined with pre- and postconditions, we want to determine an action sequence that transforms the current state to a specified goal state. A symbolic planner can generate an action sequence given the current scene state and the actions defined in Listing 2.

```
(define (domain push-planning)
  (:predicates (object ?o) (side ?s) (free ?o ?s)
               (on-side ?t ?o ?s)
  )

  ; Push the target object ?tgt to the side ?s
  ; of object ?obj
  (:action push-to-side
   :parameters (?tgt ?obj ?s)
   :precondition (and (object ?tgt)
                      (object ?obj)
                      (side ?s)
                      (free ?obj ?s)
                      (not (on-side ?tgt ?obj ?s)))
   )
  :effect (and (not (free ?obj ?s))
               (on-side ?tgt ?obj ?s))
  )

  ; Make the side ?s of object ?obj free
  (:action make-side-free
   :parameters (?obj ?s)
   :precondition (and (object ?obj)
                      (side ?s)
                      (not (free ?obj ?s)))
   )
  :effect (free ?obj ?s)
  )
)
```

Listing 2: Definition of operators, i. e. actions which the robot can execute, including their preconditions and effects using STRIPS notation.

Algorithm 6: Determine Parameters for Action Sequence

Input: S : Planned action sequence (symbolic)
 \mathcal{O} : Current scene state as set of objects
 n : Number of samples for action effect prediction

Output: P : Parametrized action sequence

```

 $P = [ ]$ ;
 $\mathcal{O}_{\text{current}} = \mathcal{O}$ ;
 $\mathcal{O}_{\text{next}} = \mathcal{O}$ ;
foreach  $s \in S$  do
   $\mathbf{a}_p = \mathbf{0}$ ;
  foreach  $i \in [1, n]$  do
     $\mathbf{a}_s = \text{sampleActionParameters}(s)$ ;
     $\mathcal{O}_{\text{pred}} = \text{predictActionEffect}(\mathcal{O}_{\text{current}}, \mathbf{a}_s)$ ;
    if  $\text{satisfiesExpectedEffects}(\mathcal{O}_{\text{pred}}, s)$  then
       $\mathbf{a}_p = \mathbf{a}_s$ ;
       $\mathcal{O}_{\text{next}} = \mathcal{O}_{\text{pred}}$ ;
    end
  end
   $P.\text{append}(\mathbf{a}_p)$ ;
   $\mathcal{O}_{\text{current}} = \mathcal{O}_{\text{next}}$ ;
end
return  $P$ ;

```

In order to execute the planned symbolic action sequence, we need to parametrize push actions to achieve the expected symbolic effects of the individual actions in the plan. Symbolic effects include spatial relations, i. e. objects being on a specific side of other objects. Valid values for a side include *on the left*, *on the right*, *in front of*, and *behind*. Another symbolic effect is, that a side of an object is free, i. e. no other object is on this side.

We propose to use learned action effect prediction models to internally simulate the effects of different action parameters on the scene state. Algorithm 6 describes the steps to parametrize actions in a planned symbolic action sequence S producing a parametrized action sequence P . First, we initialize P to an empty

sequence. Then, we iterate over all actions in S and parametrize them via sampling and predicting action effects. To this end, we sample n push parameters \mathbf{a}_s by generating push endpoints in the vicinity of the target object's center. The direction is chosen as a random rotation around the global z -axis. We evaluate for which of the pushes the action effect prediction model outputs the scene state $\mathcal{O}_{\text{pred}}$ that satisfies the symbolic effects of the planned action $s \in S$. If multiple pushes produce such an outcome, we chose the push that produces the least amount of pose changes in not involved objects. After the push parameters \mathbf{a}_p have been chosen, we append them to sequence of parametrized actions P . We keep track of the current scene state $\mathcal{O}_{\text{current}}$ while iterating over the planned action sequence and updating it according to the chosen action parameters and the predicted effect $\mathcal{O}_{\text{pred}}$.

5.2.2. Integration into the Robot Software Architecture

The action effect prediction requires action parameters for the push as well as object poses and shapes in the global coordinate system (CS) as input. Furthermore, the robot needs to be able to store and load the learned action effect prediction model. The action effect prediction as presented in chapter 4 needs to be integrated into ArmarX (Vahrenkamp et al., 2015) and especially the memory architecture.

Figure 5.7 gives an overview of the involved components and the data flow. The action effect prediction model needs object poses and bounding boxes. Therefore, we localize objects in a scene based on 6D pose estimation for known objects (Azad et al., 2009; Pauwels and Kragic, 2015) or fit geometric primitives to unknown objects as described in chapter 3. Object poses along with the robot state are tracked in the working memory of the robot. The learned model is stored in and loaded from the prior knowledge of the robot. We assume that the model is trained offline and does not need to be updated during execution.

To execute the action effect prediction on objects in a global CS, some transformations need to be applied. In the first step, the method transforms the input objects into a local action-centric CS and builds a graph representing the objects and their relations (section 4.2.3). The next step uses the learned model to pre-

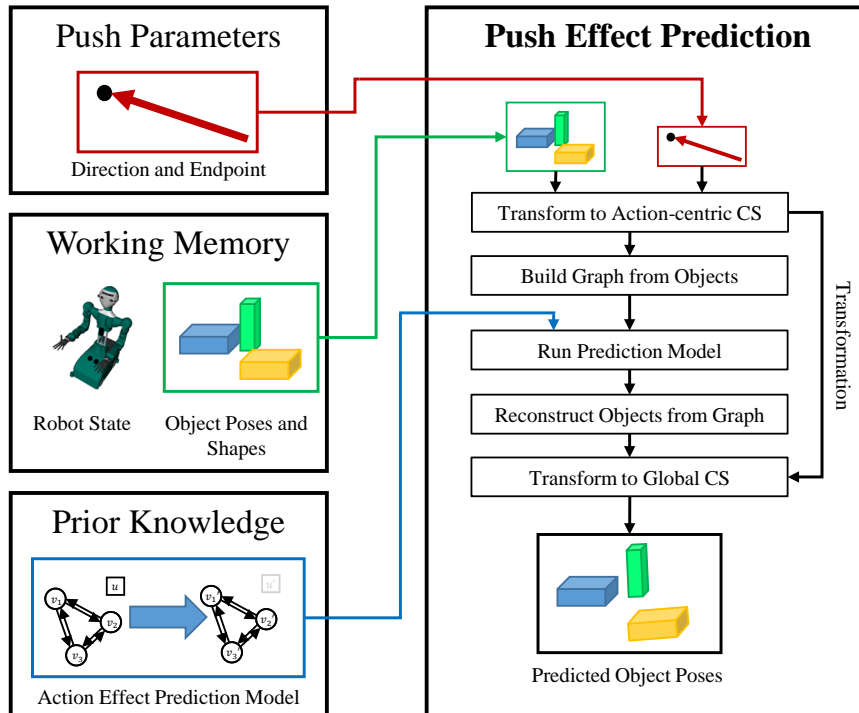


Figure 5.7.: The push effect prediction model is integrated into the robot's software architecture. Push parameters and object poses are used as input to predict object poses after executing the push. The global input poses are transformed into an action-centric CS before they are transformed into the input graph. From the predicted output graph, object poses are reconstructed and transformed back into the global CS.

dict an output graph, from which the local object poses after the push can be reconstructed. To get a prediction for object poses in the global CS, the initial local transformation is inverted and applied to all objects.

5.2.3. Push Execution

The humanoid robot ARMAR-6 has two 8-DoF arms, which we use for push execution. We will use a cartesian controller to control the 6D pose of the hand. This controller utilizes the two-dimensional nullspace to avoid joint limits.

Given the push parameters as endpoint $\mathbf{e} \in \mathbb{R}^3$ and direction $\mathbf{d} \in \mathbb{R}^3$, we want to select which arm to use and calculate a cartesian push trajectory. The trajectory begins at a defined initial pose, goes through a start point 10 cm before the object, pushes along the direction \mathbf{d} until the endpoint \mathbf{e} is reached. Then, the trajectory is reversed until it arrives at the initial pose again.

Since not all trajectories are executable with arbitrary hand orientation, we generate trajectories with hand rotations around the z -axis in the interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$. Now, we can evaluate the minimal distance Δq_{\min} to the joint limits for each generated trajectory $t \in T$ and each arm $arm \in \{\text{left}, \text{right}\}$:

$$\Delta q_{\min}(t, arm) = \text{minDistanceToLimits}(t, arm).$$

We choose the trajectory t^* and the arm arm^* which maximize Δq_{\min} :

$$t^*, arm^* = \arg \max_{t \in T, arm \in \{\text{left}, \text{right}\}} (\Delta q_{\min}(t, side))$$

This combination of trajectory and arm tries to avoid joint limits if possible, which prevents the loss of degrees of freedom in the cartesian controller.

Finally, the chosen trajectory t^* will be executed by the cartesian controller for the arm arm^* . Figure 5.8 shows an example optimization, where a trajectory with the left arm and a rotation of around $-1.396 \text{ rad} \approx -80^\circ$ will be selected.

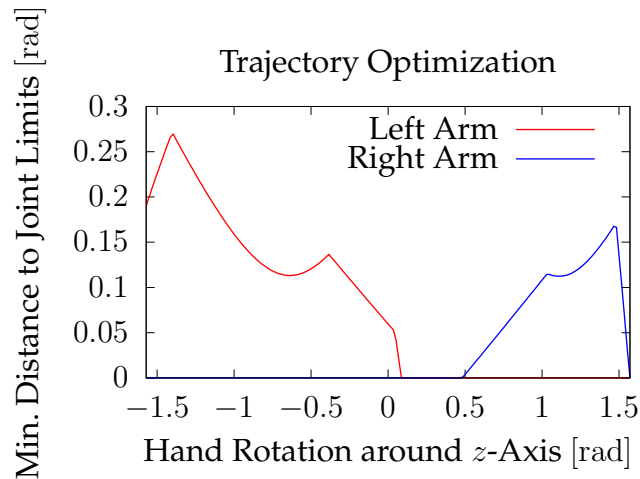


Figure 5.8.: Trajectory optimization for both arms showing the minimal distance to joint limits for varying hand orientations.

5.2.4. Validation Experiments

In experiments on ARMAR-6, we validate the transfer from simulation and show that the action effect prediction model can be used to manipulate scenes into desired states effectively. We demonstrate that the learned model enables goal-oriented manipulation for table-top scenes.

Figure 5.9 shows the initial and the goal scene configuration. The goal of the robot is to bring the large brown box to the right of the small red box. A symbolic planner has generated the following actions sequence:

1. Push the sponge away from the small red box:
(make-side-free(?sponge ?small-red-box ?right)).
2. Push the large box behind the small red box:
(push-to-side(?large-box ?small-red-box ?behind)).
3. Push the large box to the right of the small red box:
(push-to-side(?large-box ?small-red-box ?right)).

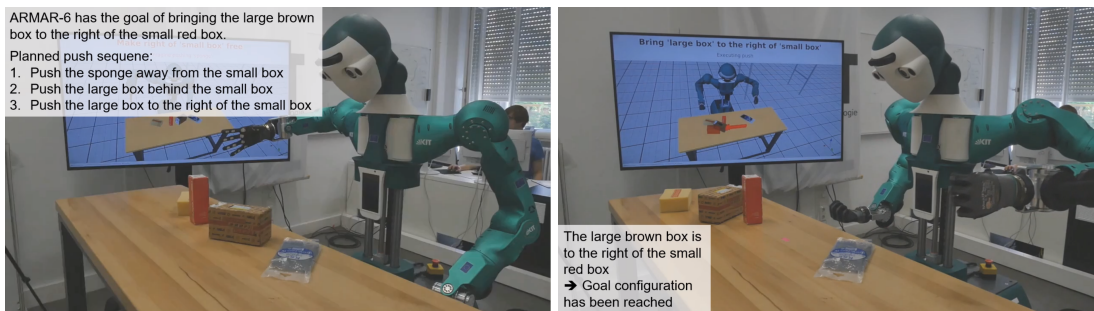


Figure 5.9.: The left image shows the initial scene state and the symbolic plan to manipulate the scene into the desired goal state. The right image shows the scene after executing the planned and parametrized actions.

Now, these actions need to be parametrized and executed sequentially. First, the sponge is pushed away to make the right side of the small red box free. Figure 5.10 shows parameter sampling and action execution for pushing the sponge away from the small red box.

During the next two actions, the large brown box is pushed behind and then to the right of the small red box. The sampled and executed actions are shown in

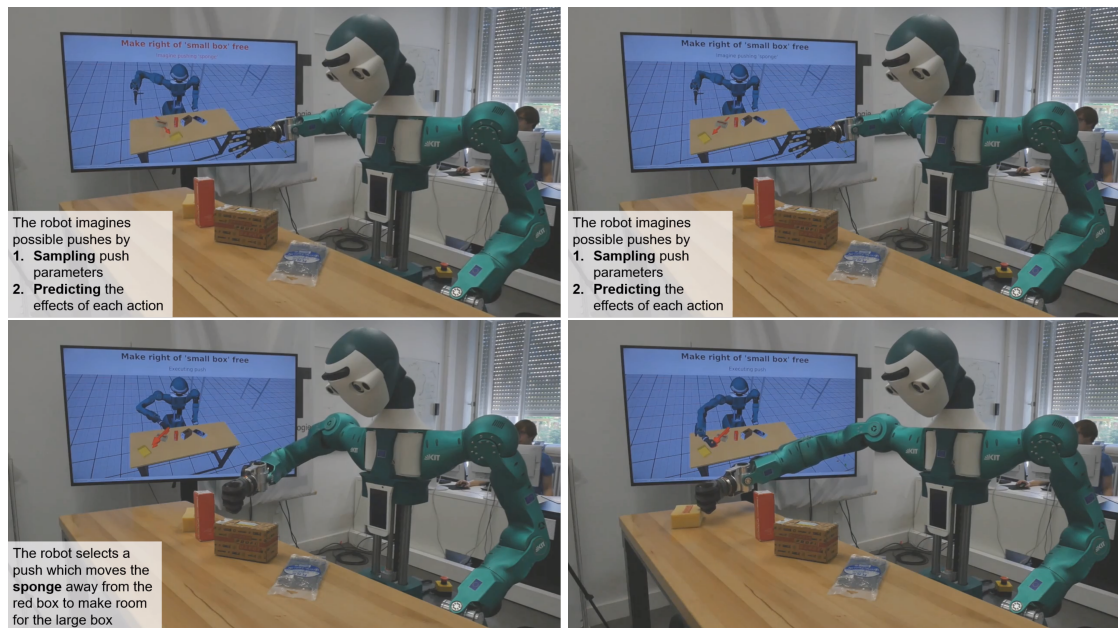


Figure 5.10.: The top row shows sampling of parameters for pushing the sponge away from the right side of the small red box. After a push end point and direction are chosen, the robot executes the push as seen in the bottom row.

Figure 5.11. The predicted pose of the pushed object is shown as a red box. As can be seen, the pose for the first push is accurate. During the push to the right side, the object rotates more than was predicted. However, the goal configuration could still be reached.

5.3. Summary and Review

This chapter implemented applications for support relations and action effect prediction in the context of humanoid robot manipulation tasks. First, we investigated how a humanoid robot can leverage extracted probabilistic support relations during the manipulation of stacked objects. Then, we presented a way to parametrize and execute pushing actions based on action effect prediction methods.

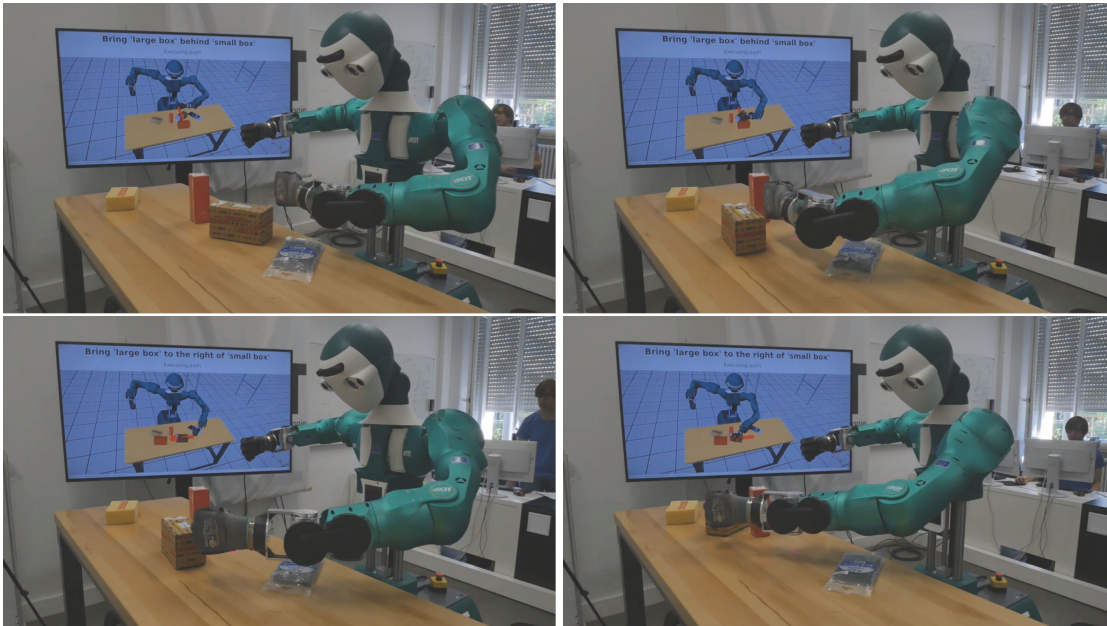


Figure 5.11.: The top row shows the predicted action effects and the executed pushing action for moving the large brown box behind the small box. In the bottom row contains the last pushing action, to move the large brown box to the final position.

Support Relations for Manipulation Tasks: Based on the probabilistic representation and extraction of support relations, a method for deriving a safe manipulation order was proposed. By choosing the next object to manipulate, which has the highest probability to not support other objects, and repeating this process, unintended motion of other objects can be avoided. For situations, in which an object supports another from above, a bimanual manipulation was developed. The humanoid robot uses its one hand to secure potentially falling objects while executing the manipulation task with its other hand. Experiments with the humanoid robots ARMAR-III and ARMAR-6 validated the proposed approaches in lifting and pushing tasks.

Parametrizing Actions using Prediction: Using learned action effect prediction models, we parametrized action sequences. First, the action parameter space is sampled and the samples are evaluated based on the predicted effect on

the scene. If the predicted effects matches the expected effect from the symbolic action description, the action parameters are chosen accordingly. Additionally, the action effect prediction method is integrated into the robot software and memory architecture. Object poses are retrieved from the working memory and the learned prediction model is loaded from the prior knowledge. During prediction, the global object poses are transformed to an action-centric CS before they are converted to the input scene graph. The prediction model produces an output graph, from which object poses are reconstructed and transformed to the global CS again. Another relevant aspect is the execution of push trajectories on a humanoid robot. We implemented a trajectory optimization that avoids joint limits by choosing an appropriate arm and hand orientation. Experiments with the humanoid robot ARMAR-6 demonstrated this approach in goal-oriented manipulation task.

6. Conclusion

The goal of this thesis was to show that interpreting a scene as objects and their relations enables a robot to predict action effects and facilitates manipulation tasks. Towards this goal, this thesis first addressed how a robot can semantically understand a scene and decompose it into objects and their support relations. Here, a probabilistic representation for both object geometry as well as support relations was proposed, and an extraction method from point clouds was implemented. Second, graph-based action effect prediction methods were proposed, which are able to learn and predict action effects for multiple interacting rigid and deformable objects. Third, we applied both the support relation extraction and action effect prediction models to humanoid robot manipulation tasks, including grasping, lifting, and pushing objects.

6.1. Scientific Contributions

This section summarizes the three main chapters of this thesis and discusses the scientific contributions.

Probabilistic Representation and Extraction of Support Relations: In chapter 3, a novel probabilistic representation and extraction method for support relations was proposed, implemented, and evaluated. The probabilistic representation includes object poses, shapes, and support relations. It encodes the geometry of the scene as parametrizable primitives, e. g. boxes, cylinders, and spheres. A joint distribution over primitive type and shape parameters captures the uncertainty in perception. A method to extract this representation from perceived point clouds was developed. First, it computes the joint distribution over

the scene's geometry via a modified RANSAC algorithm. Then, a Monte-Carlo simulation estimates the probability of support relation existence using ACT relation and support polygon analysis. The proposed support polygon is able to detect potential support from above, which other state-of-the-art methods based on ACT relations cannot detect.

The proposed extraction method was evaluated on two datasets: the KIT-SR dataset collected in this thesis and the OSD with additional annotations for support relations. The results show that both the support polygon analysis and the probabilistic representation improve detection rates significantly compared to current state-of-the-art approaches. The probabilistic representation and extraction method was published in Paus and Asfour (2021), which extended the previous work that introduced support polygon analysis in Kartmann et al. (2018).

Graph-based Prediction for Action Effects: Chapter 4 introduced action effect prediction methods based on graph neural networks for interactions between multiple rigid and deformable objects. First, a method for predicting the effects of pushing actions on multiple rigid objects was developed. The scene state was represented as a graph, in which vertices encode object properties and edges encode spatial relations between objects. Training data was generated by simulating pushes in randomized scenes. Then, a graph neural network was trained, whose input and output are graphs with an arbitrary number of vertices and edges. To extend the method for deformable object interactions, a sparse keypoint representation for a scene containing both a deformable bag and other rigid objects was proposed. Furthermore, a two-stage prediction method was implemented, which first classifies moving vertices and then conditionally updates them according to a regression model. Finally, a mixed-horizon model, which combines prediction models with different time step horizons, was proposed and implemented to handle predictions over longer time horizons.

The evaluation of the push effect prediction on both simulated and real data has shown that the method is able to accurately and efficiently predict the interactions of multiple rigid objects. In contrast to other state-of-the-art data-

driven methods, the approach is independent of the number of objects involved. For deformable object interactions, ablation studies have demonstrated that the two-stage model reduces spurious movement due to the classifier eliminating non-involved vertices. Furthermore, the mixed-horizon model improved prediction accuracy over longer time horizons. The action effect prediction method for rigid objects has been published in Paus et al. (2020) and extended for deformable objects in Weng et al. (2021).

Application to Humanoid Robot Manipulation Tasks: In chapter 5, the methods proposed in this thesis are applied to humanoid robot manipulation tasks and validated in experiments. First, based on probabilistic support relations, a method for deriving sequence of manipulation actions that maximizes the probability of successfully executing the actions, i. e. not causing other objects to move or fall, is proposed. Then, a bimanual manipulation strategy is implemented, which uses the second hand of the robot to secure potentially falling objects. Furthermore, action effect prediction methods are used to parametrize symbolic action sequences generated by a symbolic planner. By sampling the action parameter space and evaluating them based on the predicted effect, a set of action parameters can be identified, which satisfies the specified symbolic effects.

Finally, experiments were conducted on the humanoid robots ARMAR-III and ARMAR-6 to validate the proposed methods. The methods for leveraging support relations for manipulation tasks have been published in Paus and Asfour (2021) and Kartmann et al. (2018). Parametrizing actions using prediction models has been published in Paus et al. (2020).

6.2. Discussion and Future Work

This thesis introduced a novel representation and extraction method for support relations as well as action effect prediction for rigid and deformable objects. These methods have been applied to and validated in humanoid robot manipulation tasks. However, this thesis is the first step in this direction, and

further developments and extensions can be envisioned. This section presents and explores possible directions for future research.

Explainability of Decisions through Prediction: In this thesis, the robot has used both support relations and action effect prediction models to estimate what happens during or after it executes an action. This knowledge about action effects has helped choose a manipulation strategy and parametrize actions. The action effects and the choices based on them are essential to understanding why an autonomous robot takes a certain decision. This aims at explainable Artificial Intelligence (AI) or a robot system with the ability to explain its actions to a human. Towards this goal, the robot's perception, intended actions, predicted action effects, and its decisions need to be recorded in the robot's memory. Open questions in this area are how to represent this data, how to link decisions, action effects and perceptions, and how to store and retrieve this data.

A related research area is success and failure detection during and after action execution. If a robot is equipped with the ability to predict the effect of its actions, it can compare predicted effects with actual effects of its action on the environment. This can enable the robot to detect whether the action was executed successfully or whether something went wrong.

Natural Language Commands to Alter Scene States: This thesis assumed that the manipulation goal was predefined, either extracted from speech or generated by a symbolic planner. However, understanding natural language descriptions of scenes containing complex and often not well-defined object relations is a challenging task. Consider the seemingly simple command, "Put the cup to the left of the milk." In this sentence, a human bias and prior knowledge are required to understand what the relation "to the left of" actually means (Kartmann et al., 2020, 2021). To enable robots to understand these commands and alter scene states accordingly, a mapping between the symbolic description and the subsymbolic positions and orientations of objects needs to be established. Open research questions in this area include how to represent this mapping, learn it from human demonstration, and how to parametrize actions according to a given command.

A similar research topic is the verbalization of scene states, i. e. the robot describes a perceived scene in natural language to a human (Bärmann et al., 2021; Doğan et al., 2019; Zhu et al., 2017). This requires a high-level understanding of both spatial and support relations. Since relations can exist between each object pair, the robot needs to decide which relations are relevant to the human.

Model Predictive Control for Manipulation of Deformable Objects: In this thesis, methods for predicting action effects in scenes with deformable and rigid objects were proposed and implemented. A possible application of the prediction model would be to bring a deformable object in a specified goal state. A promising avenue to achieve this is model predictive control that uses the learned prediction model. Another problem is to estimate the state of a deformable object from vision. Based on the sparse keypoint representation proposed in this thesis, a possible solution could extract those keypoints from a partial view of the deformable object utilizing learned priors and symmetry. Both for control and perception, the differences between simulated and real data need to be considered. This might require techniques for domain randomization and transfer learning.

Appendix

A. KIT Semantic Relation Dataset

The KIT Support Relation dataset (KIT-SR) contains table-top scenes recorded with an RGB-D camera. The scenes have been annotated with a point-wise segmentation and ground-truth support relations. Additionally, the robot and camera poses are provided, with which the scenes have been recorded. The dataset is publicly available¹.

Structure of the Dataset

The dataset consists of 60 recorded table-top scenes. Each scene is stored in a separate numbered folder. The folder names are zero-padded to a length of 2 characters.

The scenes are grouped by complexity:

- 00 - 09: Free-standing objects on a table. No contacts between objects on the table.
- 10 - 19: Contact between two objects in the table.
- 20 - 29: Two support situations on the table (involving three to four objects).
- 30 - 39: 3 or more support situations on the table.
- 40 - 49: Cluttered scenes with small household objects.
- 50 - 59: Cluttered scenes including big boxes.

¹<https://gitlab.com/h2t/software/interactive-scene-exploration/-/wikis/KIT-SR>

Contents of an Entry

Each entry in the dataset consists of the contents in the corresponding folder:

Filename	File Type	Description
rgb.bmp	Windows Bitmap	Color image of the scene
depth.bmp	Windows Bitmap	Depth image of the scene
original.pcd	Point Cloud Data	Original point cloud in the camera coordinate system (Format: XYZRGBA)
global.pcd	Point Cloud Data	Global point cloud in the global coordinate system (Format: XYZRGBA)
labeled.pcd	Point Cloud Data	Labeled point cloud in the global coordinate system (Format: XYZRGBL)
support.csv	CSV	Annotated support relations
robot-state.json	JSON	Robot state and camera pose at the time of recording

Color Image (rgb.bmp)

The color image uses the Windows Bitmap format to store the recorded camera image of the scene. The image has the size 640x480 pixel and a bit-width of 24 (8-bit per color channel).

Example for entry 57:



Depth Image (depth.bmp)

The depth image uses the Windows Bitmap format to store the recorded depth information about the scene. The image has the size 640x480 pixel and a bit-width of 24. Each pixel stores the depth in millimeters in each pixel's 24 bit.

Example for entry 57:



Original Point Cloud (original.pcd)

The original point cloud was recorded in the camera coordinate system. The camera coordinate system is not aligned with the global coordinate system. The camera pose can be found in the robot state (robot-state.json). The file is stored in the binary PCD format. Each point consists of a coordinate (XYZ) and a color value (RGBA).

Example for entry 57:



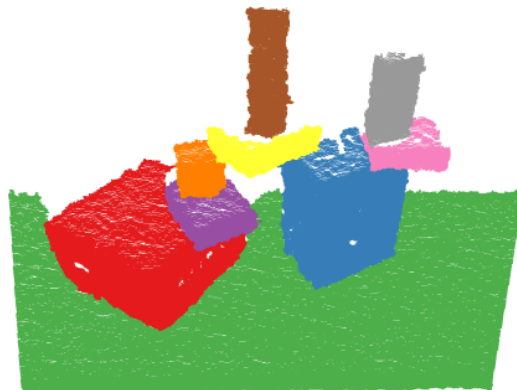
Global Point Cloud (global.pcd)

The global point cloud is equivalent to the original point cloud transformed to the global coordinate system using the camera pose from the robot state (robot-state.json). The file is stored in the binary PCD format. Each point consists of a coordinate (XYZ) and a color value (RGBA).

Labeled Point Cloud (labeled.pcd)

The labeled point cloud is a point-wise labeled version of the global point cloud. Each object is assigned a unique label that is annotated to each point. The file is stored in the binary PCD format. Each point consists of a coordinate (XYZ), a color value (RGB) and a label (L).

Example for entry 57:



Ground-truth Support Relations (support.csv)

The ground-truth support relations between objects have been annotated in a CSV file. The file contains two named columns, source and target. A row defines that a support relation exists from the label in column source to the label in column target. The labels are integers defined in the labeled point cloud (labeled.pcd).

Example for entry 57:

source	target
0	1
0	2
1	3
2	5
2	7
3	4
4	5
5	6
7	8
8	7

Robot State (robot-state.json)

The robot state at the time of recording the scene has been stored in a JSON file. The most interesting part is the global camera pose. However, the complete robot state, including the global platform pose and the joint configuration is provided. The dataset has been recorded with the humanoid robot ARMAR-6. The content has the following hierarchy:

- camera
 - frame_name
 - global_pose
- robot
 - global_pose
 - joint_configuration

B. Deformable Bag Interaction Dataset

The purpose of this dataset is to learn action effects in scenes with rich interactions between a deformable bag and multiple rigid objects.

This dataset contains 20 different tasks for four actions:

- Pushing an object towards the bag
- Handle motion along a circular trajectory
- Opening the bag
- Lifting the bag

For each task, we simulate 1,000 6-sec trajectories, and record the scene state 10 times per second, which results in 60,000 recorded time steps. The simulated task data is split into training (80%), validation (10%), and test set (10%). We vary actions and task parameters to create data for 20 different tasks.

The dataset is publicly available².

Tasks

The following parameters differentiate the tasks:

- Bag Stiffness: Is the bag's material stiff or soft?
- Bag Content: Is the bag empty or is a rigid object inside?
- Left/Right Handle State: Is the handle fixed in place, released or moved along a trajectory?
- Controlled Object: Which object is actively manipulated during the action?
- Action: Which action is executed?

The table on the following page contains all 20 tasks and their respective parameters.

²<https://kth.box.com/s/q4m2z0retfygq5kbai5innfeh55p9mst>

Task ID	Bag Stiffness	Bag Content	Left Handle State	Right Handle State	Controlled Object	Action
1	Soft	Object Inside	Fixed	Fixed	Sphere	Pushing an Object
2	Soft	Empty	Fixed	Fixed	Sphere	Pushing an Object
3	Soft	Object Inside	Moving	Fixed	Left Hand	Circular Handle Motion
4	Soft	Empty	Moving	Fixed	Left Hand	Circular Handle Motion
5	Soft	Object Inside	Moving	Released	Left Hand	Circular Handle Motion
6	Soft	Empty	Moving	Released	Left Hand	Circular Handle Motion
7	Soft	Object Inside	Moving	Fixed	Left Hand	Opening the Bag
8	Soft	Empty	Moving	Fixed	Left Hand	Opening the Bag
9	Soft	Object Inside	Moving	Released	Left Hand	Lifting the Bag
10	Soft	Empty	Moving	Released	Left Hand	Lifting the Bag
11	Stiff	Object Inside	Fixed	Fixed	Sphere	Pushing an Object
12	Stiff	Empty	Fixed	Fixed	Sphere	Pushing an Object
13	Stiff	Object Inside	Moving	Fixed	Left Hand	Circular Handle Motion
14	Stiff	Empty	Moving	Fixed	Left Hand	Circular Handle Motion
15	Stiff	Object Inside	Moving	Released	Left Hand	Circular Handle Motion
16	Stiff	Empty	Moving	Released	Left Hand	Circular Handle Motion
17	Stiff	Object Inside	Moving	Fixed	Left Hand	Opening the Bag
18	Stiff	Empty	Moving	Fixed	Left Hand	Opening the Bag
19	Stiff	Object Inside	Moving	Released	Left Hand	Lifting the Bag
20	Stiff	Empty	Moving	Released	Left Hand	Lifting the Bag

Contents of a Task Entry

The recordings for each task are contained in a HDF5³ file. The file format supports structured data storage and retrieval in table format. Column keys, data shape and description are detailed below.

Column Key	Data Shape	Description
randomname	(1000)	Random name for every trajectory.
clothid	(1000)	ID of the bag object. We provide one bag with a fixed index "20".
clothmaterial_bend	(1000)	Obi Cloth bending parameter for the bag material: 0.1 for soft, 0.01 for stiff material.
clothmaterial_dist	(1000)	Obi Cloth distance parameter for the bag material.
ballinside	(1000)	Is a sphere inside the bag? 1: Yes, 0: No.
numRigid	(1000)	Number of rigid objects in the scene, without counting the sphere effector. Defines the valid entries of the position and velocity of rigid objects.
posCloth	(1000, 61, 1277, 3)	Position of bag mesh vertices for each time step.
veloCloth	(1000, 61, 1277, 3)	Velocity of bag vertices in Unity.
posRigid	(1000, 61, 10, 4)	Position and radius of the free sphere. Only the first "numRigid" entries are valid.
veloRigid	(1000, 61, 10, 3)	Velocity of rigid sphere in Unity. Only the first "numRigid" entries are valid.
posEffector	(1000, 61, 1, 4)	Position and radius of the sphere effector. Only valid in the pushing action task.

³<https://www.hdfgroup.org/solutions/hdf5/>

B. Deformable Bag Interaction Dataset

graspnum_l	(1000, 61)	Number of grasped points for the first handle. Only valid when the left handle is not released.
graspnum_r	(1000, 61)	Number of grasped points for the first handle. Only valid when the right handle is not released.
graspnum_l	(1000, 61, 20)	The indices of grasped vertices of the left handle. Only valid if left handle is not released. Only first "graspnum_l" indices are available.
graspnum_r	(1000, 61, 20)	The indices of grasped vertices of the right handle. Only valid if right handle is not released. Only first "graspnum_r" indices are available.
circletype	(1000)	Circular type of the motion. We drawing circle in different coordinate planes. 0: Follows a circle in x-z coordinate plane, 1: Follows a circle in x-y coordinate plane, 2: Draw the circle in z-y coordinate plane.
initSpeedEffector	(1000)	Speed of the pushed sphere. Only valid for the pushing action task.
initPosEffector	(1000, 1, 4)	The initial position and radius of the pushed sphere. Only valid for the pushing action task.
sampleflag	(1000)	How the initial position of the sphere was chosen. Only valid in the pushing action task. 0: The sphere's position is generated near another rigid sphere, 1: The sphere's position is generated near the deformable bag.
towardsflag	(1000)	The target of effector pushing. Only valid in the pushing action task. 0: The sphere is pushed towards another rigid sphere, 1: The sphere is pushed towards the deformable bag.

C. Normality Test for Geometric Primitive Distributions

In section 3.1, we defined a multi-variate Gaussian distribution $P(\mathbf{x}_i | t_i, o_i)$ to represent the geometric pose and shape of objects:

$$\begin{aligned} P(\mathbf{x}_i | t_i = \text{Box}, o_i) &\sim \mathcal{N}(\mu_{i,\text{Box}}, \Sigma_{i,\text{Box}}) \\ P(\mathbf{x}_i | t_i = \text{Cylinder}, o_i) &\sim \mathcal{N}(\mu_{i,\text{Cylinder}}, \Sigma_{i,\text{Cylinder}}) \\ P(\mathbf{x}_i | t_i = \text{Sphere}, o_i) &\sim \mathcal{N}(\mu_{i,\text{Sphere}}, \Sigma_{i,\text{Sphere}}) \end{aligned}$$

These distributions should capture the samples generated during the geometric primitive fitting via RANSAC. To show that this assumption holds, we perform a multivariate normality test based on the Henze-Zirkler test (Henze and Zirkler, 1990; Baringhaus and Henze, 1988).

Our null hypothesis H_0 assumes that the samples are drawn from a multivariate Gaussian distribution. We set $\alpha = 0.05$, i. e. we discard the null hypothesis if the probability for encountering the actual samples by drawing from a Gaussian distribution is below 5%. Discarding the null hypothesis means that we assume the alternative hypothesis H_1 is true, which means that the samples do not follow a normal distribution.

The Henze-Zirkler test calculates a statistic HZ_β over all samples. If HZ_β is large, the tests rejects the null hypothesis. The parameter β can be be optimally chosen depending on the dimension d of the n samples:

$$\beta = 2^{-0.5} \left(\frac{(2d + 1) \cdot n}{4} \right)^{1/(d+4)}$$

The P value is calculated based on HZ_β and is used to either confirm or discard the null hypothesis. If $P < \alpha$, the null hypothesis is rejected.

We calculated the HZ_β as well as the P value for the samples generated by RANSAC for all scenes in the two evaluation datasets (OSD and KIT-SR) separately. The following table shows the mean and standard deviation of both values over all scenes.

Table 6.1.: Mean and standard deviation of HZ_β and P values

	Mean	Standard Deviation
HZ_β	0.9567	0.0510
P value	0.4442	0.2935

With an average value of $0.4442 > \alpha = 0.05$, the null hypothesis H_0 is not rejected. This implies that the assumption that the samples drawn using RANSAC follow a multivariate Gaussian distribution still holds.

Acronyms

APM Active Prediction Module

CS coordinate system

DoF Degrees-of-Freedom

FEM Finite Element Method

GN Graph Network

IK Inverse Kinematics

LCCP Locally Convex Connected Patches

MLP Multi-layer Perceptron

MSS Minimum Sample Set

MuJoCo Multi-Joint dynamics with Contact

OSD Object Segmentation Dataset

PPM Position Prediction Module

RANSAC Random Sample Consensus

RL Reinforcement Learning

SEA Static Equilibrium Analysis

SOFA Simulation Open Framework Architecture

SVD Singular Value Decomposition

List of Figures

1.1. Manipulation of a complex scene	2
2.1. Example scenes of object supporting each other	8
2.2. Categorization of spatial relation representations	9
2.3. Categorization of support relation representations	11
2.4. Support relations for indoor scenes from Silberman et al. (2012) .	15
2.5. Extracted support relations from Panda et al. (2016)	16
2.6. Intuitive physics engine by Battaglia et al. (2013)	18
2.7. Separating plane and ACT relation from Mojtahedzadeh et al. (2013)	19
2.8. Challenges of traditional machine learning	23
2.9. Update step of a graph neural network	25
2.10. Architectures for graph neural networks	26
2.11. Prediction model by Janner et al. (2019)	28
2.12. Action effect prediction concept	29
2.13. SE3-Net prediction model form Byravan and Fox (2017)	32
2.14. Examples of deformable objects in Obi Cloth	38
3.1. Pipeline for extracting probabilistic support relations	44
3.2. Probabilistic support graph representation	47
3.3. Line construction during box fitting	52
3.4. Inlier projection during box fitting	53
3.5. From point cloud to support relation	55
3.6. ACT relation computation	56
3.7. Different kinds of support relations	57
3.8. Example for a support polygon analysis	60
3.9. Entry of the KIT-SR dataset	68
3.10. Evaluation of support relation extraction	69

List of Figures

4.1. Overview of graph-based action effect prediction	74
4.2. Transformation to an action-centric coordinate system	77
4.3. Graph-based scene representation	78
4.4. Simulation of pushing actions	80
4.5. Pushing actions executed on ARMAR-6	81
4.6. Graph neural network architecture for push effect prediction . . .	82
4.7. Example interactions with a deformable bag	87
4.8. Deformable bag in Blender and Unity	89
4.9. Handle actions on a deformable bag	90
4.10. Trajectories from the dataset for deformable object interactions . .	92
4.11. Sparse keypoint representation for deformable and rigid objects .	93
4.12. Architecture of the two-stage prediction model	95
4.13. Mixed-horizon model for prediction of object interactions	97
4.14. Evaluation of single time step prediction errors	98
4.15. Comparison of soft and stiff bag material	99
4.16. Comparison of long horizon prediction errors	100
5.1. Concept for deriving manipulation order	105
5.2. ARMAR-III extracts support relations from a scene	106
5.3. Filtered force values during a pushing action	112
5.4. Bimanual manipulation strategy with ARMAR-III	113
5.5. ARMAR-6 grasps and lifts a chips can	114
5.6. ARMAR-6 executes a pushing action	115
5.7. Integrating action effect prediction into the software architecture	119
5.8. Trajectory optimization for pushes	120
5.9. Symbolic plan and goal configuration	121
5.10. Parametrizing pushing the sponge away	122
5.11. Parametrizing pushing the sponge away	123

List of Tables

2.1. Comparison of Support Relation Representations	13
2.2. Comparison of Methods for Extracting Support Relations	21
2.3. Comparison of Action Effect Prediction Approaches	35
3.1. Evaluation of Strategies for Vertical Separating Planes	64
3.2. Evaluation of Strategies for Vertical Separating Planes	66
3.3. Evaluation scenes for support relation extraction	67
3.4. Evaluation results on the two datasets.	71
4.1. Position and orientation error of the learned model	85
4.2. Position and orientation changes for the data sets	86
4.3. Performance on an unseen number of objects	86
6.1. Mean and standard deviation of HZ_β and P values	141

List of Algorithms

1.	Geometric Primitive Fitting using RANSAC	49
2.	Support Polygon Analysis	59
3.	Sample from Geometric Scene Distribution	61
4.	Estimate Support Relation Probability	62
5.	Safe Manipulation Strategy	111
6.	Determine Parameters for Action Sequence	117

Bibliography

- Agrawal, P., Nair, A. V., Abbeel, P., Malik, J., and Levine, S. (2016). Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in Neural Information Processing Systems 29*, pages 5074–5082. MIT Press. Cited on pages 32 and 35.
- Ahmadi, S. S. and Khotanlou, H. (2017). Enhance support relation extraction accuracy using improvement of segmentation in RGB-D images. In *2017 3rd International Conference on Pattern Recognition and Image Analysis (IPRIA)*, pages 166–169. IEEE. Cited on page 21.
- Ahuja, S., Admoni, H., and Steinfeld, A. (2020). Learning vision-based physics intuition models for non-disruptive object extraction. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8161–8168. IEEE. Cited on page 21.
- Aiken, J. G., Erdos, J. A., and Goldstein, J. A. (1980). On Löwdin orthogonalization. *International Journal of Quantum Chemistry*, 18(4):1101–1108. Cited on page 83.
- Aksoy, E. E., Abramov, A., Dörr, J., Ning, K., Dellen, B., and Wörgötter, F. (2011). Learning the semantics of object–action relations by observation. *The International Journal of Robotics Research*, 30(10):1229–1249. Cited on page 9.
- Aksoy, E. E., Tamosiunaite, M., and Wörgötter, F. (2015). Model-free incremental learning of the semantics of manipulation actions. *Robotics and Autonomous Systems*, 71:118–133. Cited on page 9.
- Allard, J., Cotin, S., Faure, F., Bensoussan, P.-J., Poyer, F., Duriez, C., Delingette, H., and Grisoni, L. (2007). SOFA - an open source framework for medical

- simulation. In *MMVR 15-Medicine Meets Virtual Reality*, volume 125, pages 13–18. IOP Press. *Cited on page 37.*
- Asfour, T., Regenstein, K., Azad, P., Schröder, J., Vahrenkamp, N., and Dillmann, R. (2006). ARMAR-III: An integrated humanoid platform for sensory-motor control. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 169–175. *Cited on pages 5 and 112.*
- Asfour, T., Wächter, M., Kaul, L., Rader, S., Weiner, P., Ottenhaus, S., Grimm, R., Zhou, Y., Grotz, M., and Paus, F. (2019). ARMAR-6: A high-performance humanoid for human-robot collaboration in real world scenarios. *IEEE Robotics & Automation Magazine*, 26(4):108–121. *Cited on pages 5, 79, and 112.*
- Azad, P., Asfour, T., and Dillmann, R. (2009). Accurate shape-based 6-DoF pose estimation of single-colored objects. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2690–2695. *Cited on pages 74, 80, and 118.*
- Baillargeon, R. (2002). The acquisition of physical knowledge in infancy: A summary in eight lessons. *Blackwell handbook of childhood cognitive development*, 1(46-83). *Cited on page 1.*
- Baringhaus, L. and Henze, N. (1988). A consistent test for multivariate normality based on the empirical characteristic function. *Metrika*, 35(1):339–348. *Cited on page 140.*
- Bärmann, L., Peller-Konrad, F., Constantin, S., Asfour, T., and Waibel, A. (2021). Deep episodic memory for verbalization of robot experience. *IEEE Robotics and Automation Letters*, 6(3):5808–5815. *Cited on page 129.*
- Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. (2018). Relational inductive biases, deep learning, and graph networks. *arXiv:1806.01261*. *Cited on pages 24, 26, and 27.*
- Battaglia, P. W., Hamrick, J. B., and Tenenbaum, J. B. (2013). Simulation as an engine of physical scene understanding. *Proceedings of the National Academy of Sciences*, 110(45):18327–18332. *Cited on pages 1, 8, 10, 13, 17, 18, 21, 75, and 145.*

- Battaglia, P. W., Pascanu, R., Lai, M., Rezende, D., and Kavukcuoglu, K. (2016). Interaction networks for learning about objects, relations and physics. In *Advances in Neural Information Processing Systems*, pages 4502–4510. Cited on pages 27 and 39.
- Berthoz, A. (2000). *The brain's sense of movement*, volume 10. Harvard University Press. Cited on page 73.
- Bolles, R. C. and Fischler, M. A. (1981). A RANSAC-based approach to model fitting and its application to finding cylinders in range data. In *IJCAI*, volume 1981, pages 637–643. Citeseer. Cited on page 49.
- Brachman, R. J. (2002). Systems that know what they're doing. *IEEE Intelligent Systems*, 17(6):67–71. Cited on page 73.
- Byravan, A. and Fox, D. (2017). SE3-nets: Learning rigid body motion using deep neural networks. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 173–180. Cited on pages 31, 32, and 145.
- Byravan, A., Leeb, F., Meier, F., and Fox, D. (2017). SE3-Pose-Nets: Structured deep dynamics models for visuomotor planning and control. *CoRR*, abs/1710.00489. Cited on pages 31 and 35.
- Calli, B., Singh, A., Walsman, A., Srinivasa, S., Abbeel, P., and Dollar, A. M. (2015). The YCB object and model set: Towards common benchmarks for manipulation research. In *IEEE International Conference on Advanced Robotics (ICAR)*, pages 510–517. IEEE. Cited on page 80.
- Chen, Y., Pan, D., Pan, Y., Liu, S., Gu, A., and Wang, M. (2015). Indoor scene understanding via monocular RGB-D images. *Information Sciences*, 320:361–371. Cited on page 21.
- Deng, Z., Todorovic, S., and Jan Latecki, L. (2015). Semantic segmentation of rgb-d images with mutex constraints. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1733–1741. Cited on page 21.
- Derpanis, K. G. (2010). Overview of the RANSAC algorithm. *Image Rochester NY*, 4(1):2–3. Cited on page 49.

- Desingh, K., Jenkins, O. C., Reveret, L., and Sui, Z. (2016). Physically plausible scene estimation for manipulation in clutter. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 1073–1080. Cited on pages 18 and 21.
- Doğan, F. I., Kalkan, S., and Leite, I. (2019). Learning to generate unambiguous spatial referring expressions for real-world environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4992–4999. Cited on page 129.
- Eitel, A., Hauff, N., and Burgard, W. (2017). Learning to singulate objects using a push proposal network. In *International Symposium on Robotics Research (ISRR)*, pages 1–15, Puerto Varas, Chile. Cited on pages 32 and 35.
- Elliott, S., Valente, M., and Cakmak, M. (2016). Making objects graspable in confined environment through push and pull manipulation with a tool. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4851–4858. Cited on pages 31 and 35.
- Faure, F., Duriez, C., Delingette, H., Allard, J., Gilles, B., Marchesseau, S., Talbot, H., Courtecuisse, H., Bousquet, G., Peterlik, I., et al. (2012). SOFA: A multi-model framework for interactive physical simulation. In *Soft tissue biomechanical modeling for computer assisted surgery*, pages 283–321. Springer. Cited on page 37.
- Fikes, R. E. and Nilsson, N. J. (1971). Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208. Cited on page 109.
- Finn, C., Goodfellow, I., and Levine, S. (2016). Unsupervised learning for physical interaction through video prediction. In *Advances in neural information processing systems*, pages 64–72. Cited on pages 31 and 35.
- Fromm, T. and Birk, A. (2016). Physics-based damage-aware manipulation strategy planning using scene dynamics anticipation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 915–922. IEEE. Cited on pages 30, 35, and 58.

- Garcia, S. (2009). Fitting primitive shapes to point clouds for robotic grasping. *Master of Science Thesis. School of Computer Science and Communication, Royal Institute of Technology, Stockholm, Sweden. Cited on page 51.*
- Gupta, A., Efros, A. A., and Hebert, M. (2010). Blocks world revisited: Image understanding using qualitative geometry and mechanics. In *European Conference on Computer Vision*, pages 482–496. Springer. *Cited on pages 9, 14, and 21.*
- Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. (2019). Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pages 2555–2565. PMLR. *Cited on page 39.*
- Hamrick, J. B., Allen, K. R., Bapst, V., Zhu, T., McKee, K. R., Tenenbaum, J. B., and Battaglia, P. W. (2018). Relational inductive bias for physical construction in humans and machines. In *Proceedings of the 40th Annual Conference of the Cognitive Science Society. Cited on page 26.*
- Hartford, J., Graham, D., Leyton-Brown, K., and Ravanbakhsh, S. (2018). Deep models of interactions across sets. In *International Conference on Machine Learning*, pages 1909–1918. PMLR. *Cited on page 24.*
- Hayes, P. J. (1978). *The naive physics manifesto*. Université de Genève, Institut pour les études sémantiques et cognitives. *Cited on page 43.*
- Henze, N. and Zirkler, B. (1990). A class of invariant consistent tests for multivariate normality. *Communications in statistics-Theory and Methods*, 19(10):3595–3617. *Cited on page 140.*
- Hogan, F. R. and Rodriguez, A. (2020). Feedback control of the pusher-slider system: A story of hybrid and underactuated contact dynamics. In *Algorithmic Foundations of Robotics XII*, pages 800–815. Springer. *Cited on pages 30 and 35.*
- Hoque, R., Seita, D., Balakrishna, A., Ganapathi, A., Tanwani, A. K., Jamali, N., Yamane, K., Iba, S., and Goldberg, K. (2020). Visuospatial foresight for multi-step, multi-task fabric manipulation. In *Robotics: Science and Systems. Cited on page 40.*

- Hou, Y. C., Sahari, K. S. M., and How, D. N. T. (2019). A review on modeling of flexible deformable object for dexterous robotic manipulation. *International Journal of Advanced Robotic Systems*, 16(3):1729881419848894. Cited on page 38.
- Huang, S.-S., Fu, H., Wei, L.-Y., and Hu, S.-M. (2015). Support substructures: Support-induced part-level structural representation. *IEEE transactions on visualization and computer graphics*, 22(8):2024–2036. Cited on page 21.
- Humboldt, W. (1999). On language: On the diversity of human language construction and its influence on the mental development of the human species. *Cambridge University Press*. Cited on page 7.
- Janner, M., Levine, S., Freeman, W. T., Tenenbaum, J. B., Finn, C., and Wu, J. (2019). Reasoning about physical interactions with object-centric models. In *International Conference on Learning Representations*, pages 1–12. Cited on pages 27, 28, 33, 35, and 145.
- Jia, Z., Gallagher, A. C., Saxena, A., and Chen, T. (2014). 3d reasoning from blocks to stability. *IEEE transactions on pattern analysis and machine intelligence*, 37(5):905–918. Cited on pages 12, 13, 15, and 21.
- Johnston, B. and Williams, M.-A. (2008). Comirit: Commonsense reasoning by integrating simulation and logic. *Frontiers in Artificial Intelligence and Applications*, 171:200. Cited on page 73.
- Jonathan, F., Paxton, C., and Hager, G. D. (2017). Temporal and physical reasoning for perception-based robotic manipulation. *arXiv:1710.03948*. Cited on page 21.
- Kartmann, R., Liu, D., and Asfour, T. (2021). Semantic scene manipulation based on 3d spatial object relations and language instructions. In *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, pages 306–313. Cited on page 128.
- Kartmann, R., Paus, F., Grotz, M., and Asfour, T. (2018). Extraction of physically plausible support relations to predict and validate manipulation action effects. *IEEE Robotics and Automation Letters*, 3(4):3991–3998. Cited on pages 44, 104, 126, and 127.

- Kartmann, R., Zhou, Y., Liu, D., Paus, F., and Asfour, T. (2020). Representing spatial object relations as parametric polar distribution for scene manipulation based on verbal commands. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8373–8380. Cited on page 128.
- Kasper, A., Jäkel, R., and Dillmann, R. (2011). Using spatial relations of objects in real world scenes for scene structuring and scene understanding. In *IEEE International Conference on Advanced Robotics (ICAR)*, pages 421–426. IEEE. Cited on page 9.
- Kasper, A., Xue, Z., and Dillmann, R. (2012). The KIT object models database: An object model database for object recognition, localization and manipulation in service robotics. *The International Journal of Robotics Research*, 31(8):927–934. Cited on page 80.
- Kemp, C. and Tenenbaum, J. B. (2008). Structured models of semantic cognition. *Behavioral and Brain Sciences*, 31(6):717–718. Cited on pages 1 and 7.
- Kemp, C., Tenenbaum, J. B., Griffiths, T. L., Yamada, T., and Ueda, N. (2006). Learning systems of concepts with an infinite relational model. In *AAAI*, volume 3, page 5. Cited on page 1.
- Kloss, A., Schaal, S., and Bohg, J. (2017). Combining learned and analytical models for predicting action effects. *CoRR*, abs/1710.04102. Cited on page 31.
- Kloss, A., Schaal, S., and Bohg, J. (2020). Combining learned and analytical models for predicting action effects from sensory data. *International Journal of Robotics Research*. Cited on pages 31 and 35.
- Kopicki, M., Zurek, S., Stolkin, R., Mörwald, T., and Wyatt, J. (2011). Learning to predict how rigid objects behave under simple manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5722–5729. Cited on pages 31 and 35.
- Krüger, N., Geib, C., Piater, J., Petrick, R., Steedman, M., Wörgötter, F., Ude, A., Asfour, T., Kraft, D., Omrčen, D., Agostini, A., and Dillmann, R. (2011). Object-action complexes: Grounded abstractions of sensorimotor processes. *Robotics and Autonomous Systems*, 59:740–757. Cited on page 75.

- Kunze, L. and Beetz, M. (2017). Envisioning the qualitative effects of robot manipulation actions using simulation-based projections. *Artificial Intelligence*, 247:352–380. Cited on pages 30, 35, and 73.
- Lagneau, R., Krupa, A., and Marchal, M. (2020). Active deformation through visual servoing of soft objects. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8978–8984. Cited on page 37.
- Lee, T. M., Oh, Y. J., and Lee, I.-K. (2019). Efficient cloth simulation using miniature cloth and upscaling deep neural networks. *arXiv:1907.03953*. Cited on page 39.
- Li, J., Sun Lee, W., and Hsu, D. (2018). Push-net: Deep planar pushing for objects with unknown physical properties. In *Robotics: Science and Systems XIV*, pages 1–9. Robotics: Science and Systems Foundation. Cited on pages 32 and 35.
- Lin, X., Wang, Y., Olkin, J., and Held, D. (2020). SoftGym: Benchmarking deep reinforcement learning for deformable object manipulation. In *Conference on Robot Learning (CoRL)*. Cited on pages 36 and 39.
- Liu, Z., Chen, D., Wurm, K. M., and von Wichert, G. (2015). Table-top scene analysis using knowledge-supervised MCMC. *Robotics and Computer-Integrated Manufacturing*, 33:110 – 123. Special Issue on Knowledge Driven Robotics and Manufacturing. Cited on pages 10, 13, 16, and 21.
- Luiblé, C. and Magnenat-Thalmann, N. (2008). The simulation of cloth using accurate physical parameters. In *CGIM'08 Proceeding of the tenth IASTED International Conference on Computer Graphics and Imaging*, pages 123–128. ACTA Press Anaheim CA. Cited on page 39.
- Lynch, K. M., Maekawa, H., and Tanie, K. (1992). Manipulation and active sensing by pushing using tactile feedback. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 1. Cited on pages 30 and 35.
- Macklin, M., Müller, M., Chentanez, N., and Kim, T.-Y. (2014). Unified particle physics for real-time applications. *ACM Transactions on Graphics (TOG)*, 33(4):1–12. Cited on page 36.

- Mason, M. T. (1986). Mechanics and planning of manipulator pushing operations. *The International Journal of Robotics Research*, 5(3):53–71. Cited on pages 30 and 35.
- Meißner, P., Reckling, R., Jäkel, R., Schmidt-Rohr, S. R., and Dillmann, R. (2013). Recognizing scenes with hierarchical implicit shape models based on spatial object relations for programming by demonstration. In *IEEE International Conference on Advanced Robotics (ICAR)*, pages 1–6. IEEE. Cited on page 10.
- Mojtahedzadeh, R., Bouguerra, A., and Lilienthal, A. J. (2013). Automatic relational scene representation for safe robotic manipulation tasks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1335–1340. IEEE. Cited on pages 12, 13, 19, 21, and 145.
- Mojtahedzadeh, R., Bouguerra, A., Schaffernicht, E., and Lilienthal, A. J. (2015). Support relation analysis and decision making for safe robotic manipulation tasks. *Robotics and Autonomous Systems*, 71:99–117. Cited on pages 10, 21, 46, 56, 57, and 58.
- Moore, J. and Haggard, P. (2008). Awareness of action: Inference and prediction. *Consciousness and Cognition*, 17(1):136 – 144. Cited on page 75.
- Mösenlechner, L. and Beetz, M. (2011). Parameterizing actions to have the appropriate effects. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4141–4147. IEEE. Cited on pages 30 and 35.
- Mottaghi, R., Rastegari, M., Gupta, A., and Farhadi, A. (2016). “What happens if...” learning to predict the effect of forces in images. In *European conference on computer vision*, pages 269–285. Springer. Cited on pages 32 and 35.
- Müller, M., Heidelberger, B., Hennix, M., and Ratcliff, J. (2007). Position based dynamics. *Journal of Visual Communication and Image Representation*, 18(2):109–118. Cited on page 36.
- Nematollahi, I., Mees, O., Hermann, L., and Burgard, W. (2020). Hindsight for foresight: Unsupervised structured dynamics models from physical interaction. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 0–0. Cited on pages 32 and 35.

- Oh, Y. J., Lee, T. M., and Lee, I.-K. (2018). Hierarchical cloth simulation using deep neural networks. In *Proceedings of Computer Graphics International*, pages 139–146. Cited on page 39.
- Omrčen, D., Böge, C., Asfour, T., Ude, A., and Dillmann, R. (2009). Autonomous acquisition of pushing actions to support object grasping with a humanoid robot. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 277–283, Paris, France. Cited on pages 31 and 35.
- Panda, S., Hafez, A. A., and Jawahar, C. (2013). Learning support order for manipulation in clutter. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 809–815. IEEE. Cited on pages 12, 13, 16, and 21.
- Panda, S., Hafez, A. A., and Jawahar, C. (2016). Single and multiple view support order prediction in clutter for manipulation. *Journal of Intelligent & Robotic Systems*, 83(2):179–203. Cited on pages 12, 13, 16, 21, and 145.
- Paus, F. and Asfour, T. (2021). Probabilistic representation of objects and their support relations. In *International Symposium on Experimental Robotics (ISER)*, pages 510–519. Cited on pages 44, 104, 126, and 127.
- Paus, F., Huang, T., and Asfour, T. (2020). Predicting pushing action effects on spatial object relations by learning internal prediction models. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 0–0. Cited on pages 75, 104, and 127.
- Pauwels, K. and Kragic, D. (2015). SimTrack: A simulation-based framework for scalable real-time object pose detection and tracking. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1300–1307. IEEE. Cited on pages 74, 80, and 118.
- Paxton, C., Xie, C., Hermans, T., and Fox, D. (2022). Predicting stable configurations for semantic placement of novel objects. In *Conference on Robot Learning*, pages 806–815. PMLR. Cited on pages 10 and 13.
- Petit, A., Lippiello, V., and Siciliano, B. (2015). Tracking fractures of deformable objects in real-time with an RGB-D sensor. In *International Conference on 3D Vision*, pages 632–639. IEEE. Cited on page 37.

- Ren, J., Guo, H., Wu, Z., Long, X., and Wu, X. (2018). Scene understanding with support relation inference for exoskeleton robot. In *2018 IEEE International Conference on Cyborg and Bionic Systems (CBS)*, pages 144–150. IEEE. Cited on pages 17 and 21.
- Richtsfeld, A., Mörwald, T., Prankl, J., Zillich, M., and Vincze, M. (2012). Segmentation of unknown objects in indoor environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4791–4796. IEEE. Cited on page 68.
- Rosman, B. and Ramamoorthy, S. (2011). Learning spatial relationships between objects. *International Journal of Robotics Research*, 30(11):1328–1342. Cited on page 9.
- Sallami, Y., Lemaignan, S., Clodic, A., and Alami, R. (2019). Simulation-based physics reasoning for consistent scene estimation in an HRI context. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7834–7841. IEEE. Cited on pages 10, 13, 18, and 21.
- Sanchez, J., Corrales, J.-A., Bouzgarrou, B.-C., and Mezouar, Y. (2018). Robotic manipulation and sensing of deformable objects in domestic and industrial applications: a survey. *The International Journal of Robotics Research*, 37(7):688–716. Cited on page 38.
- Sanchez-Gonzalez, A., Heess, N., Springenberg, J. T., Merel, J., Riedmiller, M., Hadsell, R., and Battaglia, P. (2018). Graph networks as learnable physics engines for inference and control. In *International Conference on Machine Learning*, pages 4470–4479. PMLR. Cited on page 26.
- Sato, A. (2009). Both motor prediction and conceptual congruency between preview and action-effect contribute to explicit judgment of agency. *Cognition*, 110(1):74 – 83. Cited on page 75.
- Silberman, N., Hoiem, D., Kohli, P., and Fergus, R. (2012). Indoor segmentation and support inference from RGBD images. In *European conference on computer vision*, pages 746–760. Springer. Cited on pages 11, 12, 13, 15, 16, 21, and 145.

- Stein, S. C., Schoeler, M., Papon, J., and Worgotter, F. (2014). Object partitioning using local convexity. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 304–311. Cited on pages 44 and 48.
- Sui, Z., Xiang, L., Jenkins, O. C., and Desingh, K. (2017). Goal-directed robot manipulation through axiomatic scene estimation. *The International Journal of Robotics Research*, 36(1):86–104. Cited on page 9.
- Suárez-Hernández, A., Gaugry, T., Segovia-Aguas, J., Bernardin, A., Torras, C., Marchal, M., and Alenyà, G. (2020). Leveraging multiple environments for learning and decision making: a dismantling use case. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6902–6908. Cited on page 37.
- Tekden, A. E., Erdem, A., Erdem, E., Imre, M., Seker, M. Y., and Ugur, E. (2020). Belief regulated dual propagation nets for learning action effects on articulated multi-part objects. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 0–0. Cited on pages 33 and 35.
- Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5033. Cited on pages 37, 76, and 79.
- Vahrenkamp, N., Koch, E., Wächter, M., and Asfour, T. (2018). Planning high-quality grasps using mean curvature object skeletons. *IEEE Robotics and Automation Letters*, 3(2):911–918. Cited on page 110.
- Vahrenkamp, N., Wächter, M., Kröhnert, M., Welke, K., and Asfour, T. (2015). The robot software framework ArmarX. *it-Information Technology*, 57(2):99–111. Cited on page 118.
- Vosniadou, S. (2002). On the nature of naive physics. *Reconsidering conceptual change: Issues in theory and practice*, pages 61–76. Cited on page 43.
- Watters, N., Zoran, D., Weber, T., Battaglia, P., Pascanu, R., and Tacchetti, A. (2017). Visual interaction networks: Learning a physics simulator from video. In *Advances in neural information processing systems*, pages 4539–4547. Cited on page 39.

- Weitnauer, E., Haschke, R., and Ritter, H. (2010). Evaluating a physics engine as an ingredient for physical reasoning. In *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, pages 144–155. Springer. Cited on pages 30, 35, and 73.
- Weng, Z., Paus, F., Varava, A., Yin, H., Asfour, T., and Kragic, D. (2021). Graph-based task-specific prediction models for interactions between deformable and rigid objects. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5453–5460. Cited on pages 75, 88, and 127.
- Wu, J., Lu, E., Kohli, P., Freeman, B., and Tenenbaum, J. (2017). Learning to see physics via visual de-animation. In *Advances in Neural Information Processing Systems*, pages 153–164. Cited on pages 31 and 35.
- Wu, P., Chen, W., Liu, H., Duan, Y., Lin, N., and Chen, X. (2019). Predicting grasping order in clutter environment by using both color image and points cloud. In *2019 WRC Symposium on Advanced Robotics and Automation (WRC SARA)*, pages 197–202. IEEE. Cited on pages 16 and 21.
- Wu, Y., Yan, W., Kurutach, T., Pinto, L., and Abbeel, P. (2020a). Learning to manipulate deformable objects without demonstrations. In *Robotics Science and Systems (RSS)*. Cited on page 37.
- Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Philip, S. Y. (2020b). A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*. Cited on page 23.
- Xue, F., Xu, S., He, C., Wang, M., and Hong, R. (2015). Towards efficient support relation extraction from RGBD images. *Information Sciences*, 320:320–332. Cited on pages 12, 13, 16, and 21.
- Yan, M., Zhu, Y., Jin, N., and Bohg, J. (2020). Self-supervised learning of state estimation for manipulating deformable linear objects. *IEEE Robotics and Automation Letters*, 5(2):2372–2379. Cited on page 39.
- Yang, C., Lan, X., Zhang, H., Zhou, X., and Zheng, N. (2018). Visual manipulation relationship detection with fully connected CRFs for autonomous robotic

- grasp. In *2018 IEEE International Conference on Robotics and Biomimetics (RO-BIO)*, pages 393–400. IEEE. Cited on pages 17 and 21.
- Yang, M. Y., Liao, W., Ackermann, H., and Rosenhahn, B. (2017). On support relations and semantic scene graphs. *ISPRS journal of photogrammetry and remote sensing*, 131:15–25. Cited on pages 12, 13, 16, and 21.
- Yin, H., Varava, A., and Kragic, D. (2021). Modeling, learning, perception, and control methods for deformable object manipulation. *Science Robotics*, 6(54). Cited on pages 37 and 38.
- Yu, K.-T., Bauza, M., Fazeli, N., and Rodriguez, A. (2016). More than a million ways to be pushed. a high-fidelity experimental dataset of planar pushing. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 30–37. IEEE. Cited on page 30.
- Zaheer, M., Kottur, S., Ravanbakhsh, S., Póczos, B., Salakhutdinov, R. R., and Smola, A. J. (2017). Deep sets. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc. Cited on page 23.
- Zeng, A., Song, S., Welker, S., Lee, J., Rodriguez, A., and Funkhouser, T. (2018). Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4238–4245. Cited on pages 16, 32, and 35.
- Zeng, Z., Zhou, Z., Sui, Z., and Jenkins, O. C. (2018). Semantic robot programming for goal-directed manipulation in cluttered scenes. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 7462–7469. Cited on pages 13 and 21.
- Zhang, H., Lan, X., Zhou, X., Tian, Z., Zhang, Y., and Zheng, N. (2018). Visual manipulation relationship network for autonomous robotics. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 118–125. IEEE. Cited on pages 17 and 21.

- Zhang, P., Ge, X., and Renz, J. (2019). Support relation analysis for objects in multiple view RGB-D images. *arXiv:1905.04084*. Cited on pages 12, 13, 19, 21, and 56.
- Zheng, B., Zhao, Y., Yu, J., Ikeuchi, K., and Zhu, S.-C. (2015). Scene understanding by reasoning stability and safety. *International Journal of Computer Vision*, 112(2):221–238. Cited on pages 12, 13, 19, and 21.
- Zheng, B., Zhao, Y., Yu, J. C., Ikeuchi, K., and Zhu, S.-C. (2013). Beyond point clouds: Scene understanding by reasoning geometry and physics. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3127–3134. Cited on pages 12, 13, 18, and 21.
- Zhou, J., Mason, M. T., Paolini, R., and Bagnell, D. (2018). A convex polynomial model for planar sliding mechanics: theory, application, and experimental validation. *The International Journal of Robotics Research*, 37(2-3):249–265. Cited on pages 30, 31, and 35.
- Zhu, Q., Perera, V., Wächter, M., Asfour, T., and Veloso, M. (2017). Autonomous narration of humanoid robot kitchen task experience. In *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, pages 390–397. Cited on page 129.
- Ziaeetabar, F., Aksoy, E. E., Wörgötter, F., and Tamosiunaite, M. (2017). Semantic analysis of manipulation actions using spatial relations. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4612–4619. IEEE. Cited on page 10.
- Ziaeetabar, F., Pomp, J., Pfeiffer, S., El-Sourani, N., Schubotz, R. I., Tamosiunaite, M., and Wörgötter, F. (2020). Human and machine action prediction independent of object information. *arXiv:2004.10518*. Cited on page 10.