Self-supervised Learning of Primitive-based Robotic Manipulation

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

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

genehmigte

Dissertation

von

Lars Berscheid

Tag der mündlichen Prüfung:	30. Oktober 2023
Erster Referent:	Prof. Torsten Kröger
Zweiter Referent:	Prof. Tamim Asfour

Abstract

Manipulation tasks such as grasping, bin picking, pick-and-place, or garment folding are challenging for robots, as they involve contact-rich actions, complex dynamics, unstructured environments, and high-dimensional visual data. In this work, we investigate an action-centric perspective that predicts and then optimizes an action's outcome. Using a data-driven approach, the robot *learns* to solve a task by interacting with the real world via trial and error. As learning quickly becomes costly in terms of time and resources, this work proposes:

First, we simplify the robot's action to *manipulation primitives*. By pre-defining a sequence of robot actions and motions, the controller only needs to decide where and which primitive is applied. Second, a sliding window on an orthographic image predicts an action's outcome as if it were applied at a large number of planar poses. In particular, we implement this as a *fully convolutional neural network* to efficiently infer actions in below 40 ms on our hardware. Third, we generate real-world data for learning in a *self-supervised* manner. This way, large-scale robot interaction in the range of hundreds of robot-hours is made possible, as the training requires minimal human intervention. In particular, collisions appear naturally during trial and error of manipulation tasks. We present an algorithm for online trajectory generation to ensure a safe and robust data collection.

We apply these methods to various real-world tasks: First, we investigate grasping, pre-grasping manipulation, and semantic (targeted) grasping in the context of bin picking. Grasp success rates of over 95 % in cluttered scenarios were achieved. In general, the action-centric approach allows for *generalization* to unknown objects and environments. For 6 DoF grasping, a grasp rate of 90 % was measured for unknown objects. Non-prehensile primitives are introduced for pre-grasping manipulation that increase the subsequent grasp probability and allow to empty a bin completely. By grasping multiple objects at once as well as inferring multiple actions from a single observation, over 700 picks per hour were achieved. Second, we extend grasping by targeted placing to an inter-dependent pick-and-place action. To keep the flexibility regarding unknown objects, the user-defined target pose is specified by a single visual demonstration at run-time. Trained on 25 000 actions, the system achieves a translational placement error of 2.7 mm. Third, a bimanual system learns garment folding from an initially crumpled state. A success rate of 93 % with an average fold duration of under 120 s was achieved. As for all tasks, the capability to generalize, e.g. to garments of unknown textile, shape, and color, was demonstrated.

Zusammenfassung

Manipulationsaufgaben wie das Greifen, der Griff in die Kiste, das gezielte Ablegen von Objekten oder das Falten von Kleidungsstücken sind für Roboter herausfordernd, da sie kontaktreiche Aktionen, eine komplexe Dynamik, unstrukturierte Umgebungen und hochdimensionale visuelle Daten erfordern. In dieser Arbeit wird ein *aktionsorientierter* Ansatz untersucht, der den Erfolg einer Aktion vorhersagt und darauf aufbauend optimiert. Der Roboter löst eine Aufgabe, indem er mit der realen Welt interagiert und durch Versuch und Irrtum *lernt*. Solche datengetriebenen Methoden sind jedoch zeit- und ressourcenintensiv. Um dennoch praktische Manipulationsaufgaben lösen zu können, schlägt diese Arbeit vor:

Erstens, werden Aktionen des Roboters zu *Manipulationsprimitiven* vereinfacht. Dabei wird eine Trajektorie und/oder eine Sequenz von Aktionen für eine spezifische Aufgabe definiert, sodass der Roboter nur entscheiden muss, wo und welches Primitiv ausgeführt wird. Zweitens, wird ein *Sliding Window* auf einem orthografischen Bild verwendet, um den Erfolg einer Aktion als skalare Größe vorherzusagen. Durch das Verschieben des Fensters kann der Erfolg für eine Vielzahl an unterschiedlichen Posen abgeschätzt werden, sodass dann die vielversprechendste Aktion ausgewählt werden kann. Insbesondere wird dies als *vollständig faltungsbasiertes Neuronales Netzwerk* (fully convolutional neural network) implementiert, welches auf der verwendeten Hardware effizient Aktionen in weniger als 40 ms berechnen kann. Drittens, wird ein Schwerpunkt auf eine umfangreiche und skalierbare Datengenerierung in der realen Welt gelegt. Ein *selbstüberwachtes* Training mit minimalem menschlichen Eingriff ermöglicht das Aufnehmen von mehreren 10 000 Aktionen in hunderten von Roboterstunden. Beim Erlernen von Manipulationsaufgaben treten unvermeidlich Kollisionen auf, die eine sichere und robuste Datenerfassung erschweren. Wir präsentieren *Ruckig*, einen Algorithmus zur Berechnung zeitoptimaler Trajektorien von Bewegungen zwischen beliebigen kinematischen Zuständen mit Geschwindigkeit-, Beschleunigungs- und Ruckgrenzen. Seine Echtzeitfähigkeit ermöglicht den sicheren Umgang mit Kollisionen.

Wir wenden diese Methoden auf verschiedene Aufgaben an: Erstens, wird das Greifen, die Manipulation vor dem Greifen und das semantische (gezielte) Greifen im Kontext des Griffs in die Kiste untersucht. Der Roboter probiert in einem typischen Training etwa 25 000 Griffe innerhalb von 100 h aus, und trainierte mit diesem Datensatz das Neuronale Netzwerk. In unübersichtlichen Szenarien wurden Greiferfolgsraten von über 95 % erzielt. Im Allgemeinen ermöglicht der aktionsorientierte Ansatz eine *Generalisierung* auf unbekannte Objekte und Umgebungen. Beim 6 DoF-Greifen wurden für unbekannte Objekte eine

Greifrate von 90 % gemessen. Zusätzliche Primitive für die Manipulation vor dem Greifen ermöglichen es, die nachfolgende Greifwahrscheinlichkeit zu erhöhen und damit eine Kiste vollständig zu leeren. Durch das gleichzeitige Greifen mehrerer Objekte und die Vorhersage mehrerer Aktionen aus einer einzigen Beobachtung wurden Geschwindigkeiten von über 700 Griffen pro Stunde (PPH) erreicht. Zweitens wird das Greifen durch gezieltes Ablegen zu einer voneinander abhängigen Pick-and-Place-Aktion erweitert. Um die Flexibilität in Bezug auf unbekannte Objekte zu erhalten, wird die Zielposition durch eine einzige visuelle Demonstration zur Laufzeit vorgegeben. Mit 25 000 Aktionen trainiert, erreicht das System einen translatorischen Fehler von 2.7 mm zur Zielposition. Drittens lernt ein zweiarmiges System das Falten von Kleidungsstücken aus einem anfänglich zerknitterten Zustand. Es wurde eine Erfolgsquote von 93 % mit einer durchschnittlichen Faltdauer von unter 120 s erreicht. Wie bei allen Aufgaben wurde die Fähigkeit zur Generalisierung, z.B. auf Kleidungsstücke unbekannter Textilien, Formen und Farben, nachgewiesen.

Contents

1 Introduction **1**

2 Related Work **5**
2.1 Classical Approaches . . . 6
2.2 Object-based Learning . . . 8
2.3 Robot Learning . . . 9
2.3.1 Influencing Works . . . 9
2.3.2 Data Source . . . 10
2.3.3 Observation Space . . . 12
2.3.4 Action Space . . . 13
2.4 Task-specific Work . . . 16
2.4.1 Planar Grasping . . . 16
2.4.2 6 DoF Grasping . . . 16
2.4.3 Pre-grasping manipulation . . . 17
2.4.4 Pick-and-place . . . 18
2.4.5 Garment Folding . . . 18

3 Learning for Manipulation **21**
3.1 Reinforcement Learning . . . 21
3.1.1 Markov Decision Process . . . 22
3.1.2 Finding a Policy . . . 22
3.1.3 Typical Challenges . . . 23
3.2 Observing the Scene . . . 24
3.3 Actions for Manipulation . . . 25
3.3.1 Robot Kinematics . . . 26
3.3.2 Coordinate Frames . . . 26
3.3.3 Manipulation Primitives . . . 28
3.4 Viewpoint Invariance . . . 28
3.5 Principle of Locality . . . 29
3.6 Reward Estimation . . . 32
3.6.1 Principle of Maximum Likelihood . . . 33
3.6.2 Gradient-Based Optimization . . . 34
3.6.3 Neural Networks . . . 34

3.6.4 Network Architecture . . . 36
3.6.5 Regularization . . . 37
3.7 Selection Policy . . . 38

4 Real-world Training **41**
4.1 Generating Training Data . . . 41
4.2 Online Trajectory Generation . . . 44
4.2.1 Problem Definition . . . 45
4.2.2 Ruckig Algorithm . . . 46
4.2.3 Experimental Results . . . 54
4.2.4 Discussion . . . 58
4.3 Hardware Setup . . . 60
4.4 Software Setup . . . 62

5 Grasping **65**
5.1 Planar Grasping . . . 65
5.1.1 Primitives . . . 66
5.1.2 Model Architecture . . . 67
5.1.3 Training Process . . . 69
5.1.4 Experimental Results . . . 70
5.2 Model-based Adaptive Primitives for 6 DoF Grasping . . . 76
5.2.1 Advantages and Challenges of 6 DoF Grasping . . . 76
5.2.2 Model-based Orientation . . . 77
5.2.3 Collision Avoidance . . . 79
5.2.4 Experimental Results . . . 80
5.3 Fully Convolutional Actor-Critic for 6 DoF Grasping . . . 84
5.3.1 Model Architecture . . . 84
5.3.2 Combined Self-supervised and Imitation Learning . . . 86
5.3.3 Experimental Results . . . 88
5.4 Pre-grasping Manipulation . . . 100
5.4.1 Learning for Shifting . . . 101
5.4.2 Combined Learning and Inference . . . 102
5.4.3 Experimental Results . . . 103
5.5 Semantic Grasping . . . 107
5.5.1 Object Correspondence Reward . . . 107
5.5.2 Embedding Learning . . . 108
5.5.3 Experimental Results . . . 111
5.6 Discussion . . . 114

6 Pick-and-place **117**
6.1 Challenges . . . 117
6.2 Pick-and-place Primitives . . . 118
6.3 State Space . . . 119

6.4 One-shot Imitation Learning . . . 120
6.5 Model Architecture . . . 122
6.6 Self-supervised Learning . . . 124
6.7 Experimental Results . . . 124
6.7.1 Data Collection and Training . . . 125
6.7.2 Object Placement Error . . . 125
6.7.3 Insertion Task . . . 127
6.7.4 Selection Error . . . 127
6.7.5 Multiple-step Tasks . . . 128
6.8 Discussion . . . 128

7 Transition Model for Composed Primitives 131
7.1 Model Architecture . . . 131
7.2 Uncertainty Prediction . . . 133
7.3 From Predicting to Planning . . . 134
7.4 Experimental Results . . . 135
7.4.1 Image Prediction . . . 135
7.4.2 Grasp Rate . . . 137
7.4.3 Benchmarking Picks Per Hour . . . 138
7.4.4 Planning . . . 138
7.4.5 Generalization . . . 139
7.5 Discussion . . . 140

8 Garment Folding 141
8.1 Problem Statement . . . 141
8.2 Primitives . . . 142
8.3 Model Architecture . . . 143
8.4 Reachability Calibration . . . 145
8.5 Training for Smoothing . . . 146
8.6 Folding Pipeline . . . 147
8.7 Experimental Results . . . 148
8.7.1 Experimental Setup . . . 150
8.7.2 Sufficiently Smoothed . . . 150
8.7.3 Folds per Hour . . . 152
8.7.4 Generalization to Unseen Garments . . . 152
8.7.5 System Limitations . . . 153
8.8 Discussion . . . 153

9 Conclusion 155

10 Outlook 159

.

Chapter 1

Introduction

Robotics research is driven by some of the wildest dreams of mankind: Replacing human labor and workforce with powerful and flexible machines, making use of automation to increase productivity and welfare, but also having intelligent machines that assist and ease our day-to-day life. However, despite all the progress made in recent years, the state of robotics is still far from those noble goals. There are various reasons and challenges, but of paramount importance is the *great complexity* that robots have to deal with in the real world. Truly helpful robots need to successfully operate in an unfamiliar, changing, and non-deterministic environment. Therefore, sensors are required to observe the robot's environment and allow it to re-plan its task execution according to the newest available information. The robot acts on the environment by controlling its motors to follow a desired trajectory. Mapping sensor data to motor torques is the core of all robotic software; equally for industrial robot arms, humanoid robots, or autonomous vehicles.

This mapping is challenging even for simple and well-confined applications: First, high-dimensional sensor input like visual data is oftentimes required to observe and provide a sufficient level of detail about the environment. For example, a camera image is a matrix of millions of values that need to be *understood* by the robot. Second, sensor data might be noisy, incomplete, and just a partial observation of the actual real-world state. Third, motor torques are an abstract and unintuitive measure for solving relevant tasks. The robot controls its environment only indirectly, and the effect of motor torques on the environment is challenging to predict. Fourth, the robot usually requires prior knowledge about the environment, itself, or general physics to solve a task. As integrating all these requirements into a truly universal robot controller is too far ahead, robots are constrained to specific tasks.

In this work, we focus on robot arms *manipulating* their environment. In particular, tasks like *grasping* or *pick-and-place* take an essential role as a building block for further, application-specific manipulation. The robot needs to combine visual information and prior knowledge in a way that allows a robust yet flexible task execution. To highlight these difficulties, this work focuses on the setting of *bin picking*. Here, a robot grasps objects

out of an unstructured environment like a randomly filled bin. In comparison to general robotic grasping, bin picking emphasizes challenges like partially hidden objects and an obstacle-rich environment.

Classical approaches have been object-centric and model-based. In industrial use cases, state-of-the-art techniques localize the target object via *pose estimation* and manipulate in a pre-defined manner relative to the object. This object representation greatly simplifies programming, for example, to allow a target pose for pick-and-place. However, extracting objects out of a scene remains challenging in general, and even more so for dense clutter or soft objects. Moreover, *unknown* objects do not work.

In research, model-based approaches without knowledge of object instances can be found. While these can be applied more flexibly, several issues have prevented widespread adoption: First, these approaches require high-quality and complete geometrical information about the scene. Practical solutions demand a single camera sensor that cannot observe back sides. Second, a model-based solution assumes knowledge about the object, the environment, and the robot that might not be available in reality. This often contradicts the desire for good generalization to unknown situations.

In recent years, much progress has been achieved in the area of deep learning, in particular regarding learning from images and other high-dimensional data. In this work, we make use of these innovations by developing a primarily *learned* and action-centric approach to robotic manipulation. In general, the proposed method skips the object representation and instead maps the sensor image directly to action success probabilities (or so-called *rewards*). A key idea of robot learning is that the robot performs actions within an environment, and tries to optimize for a scalar reward signal. With the cost of this training, data-driven methods have the intrinsic capability to generalize to a certain degree, e.g. to unknown objects. This way, learning allows to avoid the model building, contact dynamics, and parameter estimation of model-based manipulation.

While this action-centric approach is (theoretically) able to learn arbitrary complex manipulation tasks, the effort shifts to *data generation* for training. The robustness, task flexibility, and generalizability of learned approaches are practically limited by the available amount of data. To achieve capable manipulation with limited resources, the overall approach should (1) rely on learning only where necessary, (2) use data-efficient task formulations and learning algorithms, and (3) be able to scale the data generation. Respectively, this work deals with the following research questions: What aspects of manipulation should be learned, and what aspects should be modeled classically? How can robotic manipulation tasks be formulated so that the learning is data-efficient? What is needed to scale real-world data generation for manipulation tasks?

In the following, chapter 3 introduces *manipulation primitives* as a level between model-based and data-driven approaches. Briefly, a primitive is a pre-defined action and motion that is applied at a (partially) learned position and orientation (pose). This way, the robot

can apply its model-based kinematics and controllers. Moreover, it allows to utilize specific deep learning architectures to efficiently predict an action's outcome for a large number of different poses, and then optimize over this search space.

Chapter 4 presents techniques for *self-supervised* robot learning. To allow large-scale data collection, the robot should be able to generate data continuously, safely, and autonomously. Amongst others, we derive approaches for safe collision handling and their integration into our experimental setup.

Following, we explore different manipulation tasks in their respective chapter. Chapter 5 deals with the fundamental task of *grasping*, and is divided into the subtasks for planar (top-down) grasping (section 5.1), 6 DoF grasping (section 5.2 and 5.3), pre-grasping manipulation to better grasp objects afterwards (section 5.4), and semantic grasping of a target object type (section 5.5). In chapter 6, our approach for *pick-and-place* allows to place a grasped object at a precise position and orientation. Chapter 7 introduces a *transition model* to predict the environment's state after an applied manipulation primitive. This way, multiple grasping or pre-grasping actions can be calculated in advance and used for manipulation planning. Finally, chapter 8 presents our approach for the task of *garment folding*. In comparison to grasping or pick-and-place, folding requires dynamic manipulation primitives, even more challenging real-world data generation, and bimanual actions. This way, it demonstrates the effectiveness of primitive-based manipulation and self-supervised learning on a truly complex task.

Chapter 2

Related Work

Our contributions in the subsequent chapters are built upon and inspired by numerous works within and outside the robotics community. Robotic manipulation is a broad field, and as such it accumulated ten-thousands of papers in decades of cumulative research. In the following, we try to survey only the most important ideas relevant to our work. This chapter intends to structure the state-of-the-art to better contextualize the contributions of the following chapters.

As introduced in chapter 1, a primary differentiation for manipulation approaches is the usage of prior information about the *object* of interest. Usually, this implies geometric knowledge in the form of a 3D model like an object scan or a CAD file. While this might be sufficient for static tasks like grasping, other more dynamic manipulation tasks would greatly benefit i.a. from inertia or friction parameters. For soft objects, prior knowledge might also be an object deformation model. On the other hand, relying less on prior object information directly implies the need to know more about the manipulation action and the ability to predict its outcome. Of course, this also becomes easier with explicit object knowledge, but oftentimes implicit knowledge or manipulation planning under uncertainty can be used instead. Recently, data-driven methods allow to forgo an explicit object model,

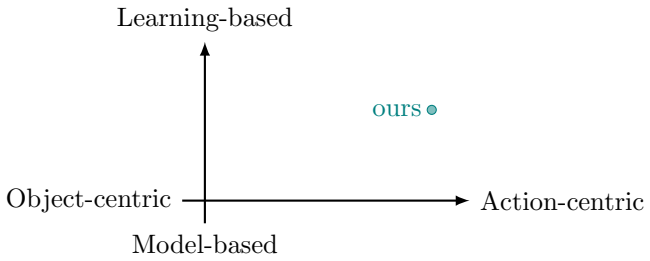


Figure 2.1: Two major criteria for structuring robotic manipulation are the usage of (1) prior knowledge about the object of interest (left-right) and (2) data-driven methods in comparison to user-derived models (top-down).

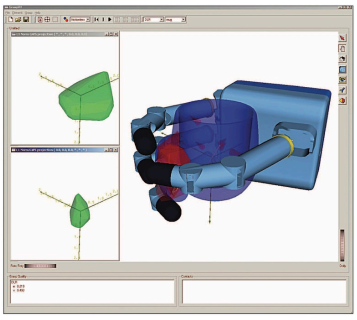
and therefore enable more flexible manipulation even with unknown objects. Moreover, approaches can specialize in object types to reduce the dependency on object information. Fig. 2.1 shows object vs. action-centric approaches as a first, horizontal axis.

Second, we find the extent of *learning* used by the approach to be an important criterion. Without any learning, user-defined analytical or numeric models for geometric, kinematic, or dynamic predictions of the environment allow to plan and optimize manipulation actions. In contrast, learning-based methods allow to infer these models implicitly or explicitly from data. In the most extreme case, the complete robot controller, mapping raw sensor data to motor torques, can be learned end-to-end by corresponding data or experience. However, learning can be applied at various points for manipulation planning, and current research deals about where it should be used. In general, learning can be introduced gradually, e.g. to use it for hard-to-model parts of the manipulation planning only and keep analytical models e.g. for kinematic calculations. In this context, we classify approaches into model- or learning-based (Figure 2.1).

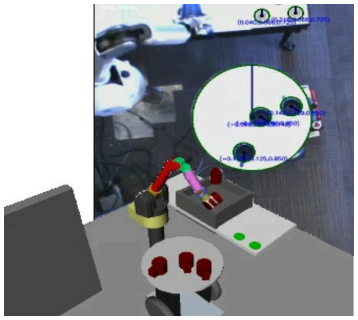
2.1 Classical Approaches

Both criteria of object-centric and model-based manipulation intersect naturally when manipulation is planned entirely based on an object model (lower left of Figure 2.1). Historically, these were the first approaches introduced for robotic manipulation more than 40 years ago. Even more than today, robotic grasping was of primary interest due to its foundational role in further manipulation. In this regard, most classical approaches were derived with grasping in mind and only extended slightly to other, more complex tasks.

In 2000, Bicchi and Kumar [13] surveyed the state-of-the-art for robotic grasping. Given a geometric model of the object and the robotic gripper, a grasp can be analyzed regarding its force closure on the object of interest. Commonly, contacts are assumed to be point contacts, either frictionless (only normal forces), with friction (additional tangential forces), or soft contacts (with torsional torques). A force closure is achieved when an object is in a static equilibrium despite a wrench acting on it. Salisbury [103] derived an analytical procedure to test for force closure. This is however only a necessary condition for a grasp, as it also requires *stability*, e.g. against model errors or external wrenches. Therefore, the forces of a quasi-static yet dynamic grasp model were commonly investigated, resulting in the requirement of the stiffness matrix describing the grasp being positive definite [103]. Several grasp performance measures were introduced to summarize a grasp candidate, e.g. by building upon force closure, controllability, dexterity, observability, or reachability. Commonly, these approaches were evaluated in simulation only, as object and gripper models can be flexibly created [13]. In real-world experiments, however, errors occur naturally and are inherent to a robotic system, primarily due to modeling simplifications or errors. Bicchi and Kumar [13] argued that approaches at that time were not sufficiently considering grasp robustness under imprecise finger placement.



(a) Grasp planning inside the *GraspIt!* simulator [86]



(b) Pick-and-place planning inside *OpenRAVE* [28]

(c) Transferring planned manipulation to a robot [28]

Figure 2.2: Examples of classical approaches of grasp generation.

With the introduction of *GraspIt!* [86] and other simulators in 2004, the focus shifted towards a larger-scale grasp synthesis. As a common differentiation, *grasp generation* refers to the generation of grasp hypotheses that are subsequently *ranked* by a grasp performance measure. According to Bohg et al. [16], approaches differed mostly in the grasp hypotheses generation. For ranking, the ϵ-metric by Ferrari and Canny [35] was widely popular. It extends the force closure analysis to the *grasp wrench space* by computing the convex hull over all wrenches that result in a stable grasp. The ϵ-metric corresponds to the radius of the maximum sphere that is still contained within the wrench space, and it thereby incorporates both the stability as well as the robustness of the grasp. However, transferring these approaches onto real-world robotic systems has still proven to be difficult. Then, the pose of an object needs to be estimated by sensor data, and the resulting pose error was found to have a great effect on the grasp stability [16, 123]. Moreover, all approaches presume an object model or high-quality scan, although a single camera sensor is only able to view parts of the object. *Shape completion* is an approach for constructing a full 3D model of an object or scene from partial observations [121]. Prior information based on familiar objects or general symmetries allows the system to use these classical approaches from sensor observation. However, unknown objects are always able to break the shape completion and the downstream manipulation, rendering them unreliable for the general case. These challenges motivate the usage of *learning* to let the robot explore and identify grasps from real-world experience.

Other manipulation tasks were studied much less frequently, even more so in a similar manner using analytical models. Task-oriented grasping extends grasp planning by integrating constraints of downstream manipulation tasks. Sahbani et al. [102] identifies the primary difficulty as the modeling of the task. In this regard, general measures like the manipulability of the robot or transmission of velocities or forces onto the object were introduced as optimization criteria during grasp planning. Representative for more complex

manipulation, Dogar and Srinivasa [29] evaluated pushing actions for pre-grasping manipulation. As tasks become more *dynamic* and require force analysis in motion, the analytical models become more complex to compute and optimize. First and foremost, however, model identification and subsequent transfer into the real world become the limiting challenge.

2.2 Object-based Learning

Most approaches presented so far require the pose of the object to be known. Moreover, pose estimation is particularly relevant for industrial tasks like bin picking, as a known pose allows to (1) grasp at pre-defined points and even (2) place the object precisely. While pose estimation was a major bottleneck for transferring classical grasp planning to real-world execution, data-driven methods have tackled some of those challenges in recent years [65]. These include dense clutter and occlusions, changing lighting conditions and sensor noise, as well as the required flexibility regarding the object type. While classical approaches like the *Iterative closest point* algorithm [24] exist, they usually require an object-specific and therefore expensive, manual tuning to achieve a satisfying task performance [65].

Modern approaches for 6D pose estimation are commonly based on a **CNN!** (**CNN!**), either by solving an underlying classification (within a spatial discretization) or a 6D regression task. Nearly all approaches rely on supervised learning utilizing a large dataset of color-, depth- or point cloud data with labels of the visible object poses. Xiang et al. [126] introduced *PoseCNN* for 6D pose estimation of different object classes from images. It splits the calculation into a first localization of the object's center within the image, a second estimation of its distance, and a third regression of a quaternion representation of the object's orientation. More recently, OP-Net by Kleeberger and Huber [64] achieved state-of-the-art results in terms of average precision and computation time. Working directly on the point cloud instead of intermediate image representations, the approach regresses the object's pose and confidence for a large number of subdivided volume elements.

All approaches share the need for large amounts of labeled data and rely on their quick generation for easy integration of novel objects. Therefore, training on synthetic data from simulation is dominating current approaches. Here, labeled data is abundant as the object pose annotations are perfectly known for all cases. However, transferring trained approaches into the real world creates the *sim-to-real* gap, as the real data distribution might differ from the simulated training data. The sim-to-real gap for pose estimation is smaller than for a complete manipulation simulation, as only the synthetic sensor data needs to match the real input data. Besides photorealistic image rendering and user-defined camera models for point clouds, data generation typically involves dropping objects into the scene from random configurations in simulation. Particularly for bin picking scenarios, this ensures a diverse and cluttered training set with expected object configurations.

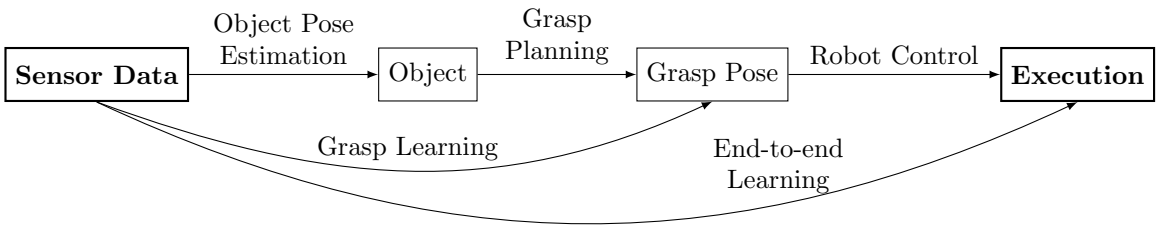


Figure 2.3: Learning enables skipping an intermediate object representation prevalent in prior manipulation approaches, opening the door for generalization to unknown objects.

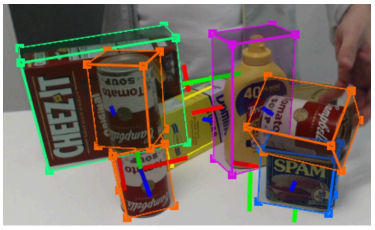
2.3 Robot Learning

So far, these approaches have been using an intermediate object representation and relied on analytical models or heuristics, written into code by a human programmer. *Robot learning* takes a fundamentally different approach. Here, the robot infers strategies for solving manipulation from *data*, corresponding to a mapping between two domains. Data-driven methods allow mappings that were too complex or tedious for a human to program, in particular mapping from raw visual sensor data to an output signal like motor torques, motor velocities, or manipulation success probabilities. As sketched in Figure 2.3, this flexibility enables robot learning to skip intermediate object representations and map directly to grasp poses or robot actions. Therefore, robot learning can be naturally used for *action-centric* approaches. This way, learning enables approaches that can generalize to unknown, never-seen objects.

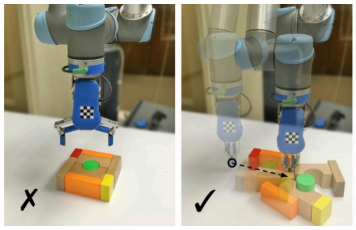
2.3.1 Influencing Works

Historically, the work of Saxena et al. [105] has influenced later approaches to robot learning significantly. Here, the grasp point was estimated by a logistic regression taking different edge and texture filters of a (depth) image as input. This way, the model does neither try nor require to use a 3D object representation or analytical formulation, but focuses solely on the object features that *cause* a good grasp. The regression model was trained on synthetic labeled data via supervised learning and was able to generate successful grasps for objects not seen in the training set. However, the manual creation of object features held back this approach until the widespread emergence of deep learning around 2014.

In 2016 and 2017, several influencing papers were published. Similar to [105], Pinto and Gupta [94] investigated the problem of planar grasping from depth images. They used a convolutional neural network (NN) to learn a mapping from a window patch, representing the xy-translation and single rotation of a grasp, to a grasp success probability. In particular, the NN learns directly from the sensor input without any manual feature selection; this however increases the demand for more data. For this reason, they scaled the data



(a) Pose estimation of household objects for grasping [126]



(b) Learning pushing and grasping primitives [128]

(c) Large-scale data collection for end-to-end learning [73]

Figure 2.4: Examples of robot manipulation with different extent of learning.

collection by 40 times over prior works, resulting in a dataset of 50 000 grasp attempts recorded in 700 robot hours. The work achieved a grasp success rate of 73 % for previously seen objects and 66 % for unknown objects, with good qualitative results even in clutter.

Levine et al. [73] advanced the real-world data collection even further to over 800 000 grasp attempts from multiple robots over months. While keeping planar grasps, the work differed in the used action space: In comparison to prior works focusing on the binary classification of a single grasp pose, the NN controls the Cartesian velocity of the end-effector from an over-the-shoulder RGB camera. Prominently, this approach also needs to learn the hand-eye-calibration that is else provided by the user. In this regard, the work introduced more general reinforcement learning (RL) algorithms to robotic manipulation tasks.

Mahler et al. [82] improved grasp success rates by combining a convolutional NN with a large yet synthetic grasp dataset. The dataset included 6.7 million grasps with a depth image patch that represents the planar position, angle, and depth of the grasp, and a labeled analytical grasp quality metric. The resulting NN can grasp known isolated objects with a success rate of 93 % and achieves state-of-the-art results on unknown objects. Two contributions were essential to the field: First, it showed that deep learning in combination with large-scale analytical grasp metrics can accelerate and improve grasp generation. Second, the transfer gap from real-world to synthetic depth images could be bridged successfully via deep learning.

In the following, we survey robot learning for manipulation systematically. Approaches differ, both from their theoretical background as well as their practical implementation, in the data source, the action space, and the observation space. In this regard, the fundamental questions for robot learning are: Where does the knowledge come from? What does the policy control? What does the policy sense in the environment?

2.3.2 Data Source

Sahbani et al. [102] classify approaches for learning robot manipulation by their data source into either *object-* or *human observation*-based. In accordance with [16], we extend this

classification by synthetic data sources from simulation or analytical models.

Real-world Direct *interaction* between the robot and the real world allows the best transfer to applications, as the training and inference environment are usually similar or even equivalent. However, this comes with a generally slow and costly data collection, not only because of expensive robot time, but also because of safety or robustness issues, or regular human intervention due to environment preparation or resets [32, 61, 72, 73, 90, 94]. Therefore, real-world data collection is generally desirable but difficult to scale. For this reason, overfitting the robot due to too few trial and error interactions remains problematic [72, 90].

Simulation By replicating the real-world environment virtually, simulation aims to predict the relevant physics of the manipulation task. This allows for a much more scalable data generation, as a simulation can be easily accelerated and parallelized. Moreover, it allows full control and knowledge over the environment which might be helpful during training. However, the learned policy needs to transfer to its real-world application. Two major challenges are: First, the implemented physics model might only *approximate* the reality due to analytical or numerical shortcomings. In general, current state-of-the-art simulators such as Bullet [25] or Mujoco [120] struggle with robust contact dynamics, mostly because of their underlying mathematical discontinuity. Second, modeling the real environment in simulation is a difficult task, as parameters such as inertia matrices or friction parameters are hard to measure. [16, 123] explain further challenges in simulating highly unstructured environments.

Nonetheless, learning in simulation has found widespread adoption due to the data consumption of deep learning [18, 107, 119, 124, 128]. Domain adaption is the task of bridging the *sim-to-real gap*, usually by making the policy robust to domain shifts by randomizing the simulation [18, 119]. Other techniques are an active field of research [66].

Analytical Models While simulation can be seen as a comprehensive end-to-end model, we differentiate simplified analytical models that focus on specific components of the problem. In particular, models developed for manipulation planning (section 2.1) are commonly used as a data source for learning [81, 82]. In comparison to simulation, analytical models are faster to compute, easier to implement, and better suited for customization.

Human Annotation or Demonstration Human input as a data source for learning allows to transfer prior knowledge from the human into the robotic system. This can either be done as data annotation (labeling pre-recorded sensor data, commonly in the form of mapping observations to robot actions [129]) or as human demonstration (recording human interactions within the environment, [84, 112, 113, 92]). While the latter is more challenging due to the transfer from the human to the robot, it is usually faster and more intuitive to perform. In general, human input is the least scalable way to generate data, although several ways to speed up annotations (e.g. via online outsourcing [113]) or demonstrations

(e.g. via custom tools [112]) have been proposed. Multiple data sources can be combined to be used in parallel or subsequently, e.g. human demonstrations are oftentimes used as an initial policy to guide further exploration with real-world data.

2.3.3 Observation Space

Kroemer et al. [66] gave a recent survey about learning for robot manipulation, focusing on skill policies and environment representations. This representation of the observation space is essential as the input for the robot's policy, however, it also influences the structure of the policy and other properties of the overall system. For example, a robot might use a combination of different sensors, multiple images from different perspectives, or an explicit temporal history by taking the last n measurements as input. Here, [66] differs between object-level, part-level, or point-level representation.

Object- or Part-level Object representations capture information about the complete or partial object of interest. Oftentimes, manipulation is about objects regardless of low-level features [27, 59, 64, 84]. The observation space might include the object's pose, geometry, mass, or other material properties. The representations can be extended by higher-level information such as semantic labels, affordances, or object relations [59]. Generalization to different tasks implies variation of these properties, e.g. to allow manipulation within novel situations or of unknown objects. In this regard, properties that are excluded from the observation space are implicitly learned and are therefore not able to generalize. Low-dimensional properties such as the object's pose are commonly able to generalize to variation, however, high-dimensional properties such as the object's geometry are usually implicitly learned. Instead, generalizable manipulation can be planned by finding correspondences between similar objects [27].

Part-level representations commonly focus on parts that are associated with manipulation actions and affordances. Practical examples can be a tool handle for grasping or a bin for placing objects [14]. Parts allow to represent an object in a more granular way, with improved geometrical approximation, relationships between parts, and action-related constraints. Moreover, correspondences at the part level allow for improved generalization across objects, as objects oftentimes share parts intended for interaction (such as handles).

Point-level High-dimensional input such as point cloud, pixels, or voxels can capture low-level features of the robot's environment [18, 40, 53, 61, 72, 73, 80, 118, 128, 130]. Sensors like RGB or depth cameras generate this input data with minor post-processing. We differentiate between point clouds as a list of 3D points, and pixels (images) or voxels as a grid-like structure for two or three dimensions respectively. In this regard, depth images can be understood as a structured point cloud along the image plane. Pixels, voxels, and points usually contain depth and/or color modalities, but can also include downstream semantic information such as object segmentations [106]. Convolutional layers are used

to predict higher-level features from pixel or voxel structures. In comparison, specialized architectures such as Point-Net [95] directly operate on point cloud data.

In general, object representations build upon point-level features. While 2D images allow for a compact representation in comparison to voxels or point clouds, they suffer from the pixel's uni-modality and e.g. resulting unknown back side of the object. In contrast, point clouds can be fused from multiple sensors to increase the resolution and the visibility of the environment. A pixel or voxel representation can easily be rendered from a point cloud.

Correspondences for generalization can also be found in point-level features. In particular, robots learn to identify the necessary low-level features for manipulation tasks and therefore can transfer to unknown objects, scenarios, or higher-level representations.

2.3.4 Action Space

The action space includes all possibilities of how the robot can *act* within and onto the environment, and therefore defines the output space of the policy. In the end, manipulation tasks are solved by *moving* the robot along calculated trajectories. On a low level, the robot controller runs a real-time control loop (commonly more than 100 Hz) that takes as input a desired joint position, joint velocity, or joint torque and executes the respective motor torques to achieve the desired configuration. The input trajectory needs to fulfill a range of constraints, e.g. the robot's kinematic position, velocity, or acceleration limits. Calculating a valid trajectory is non-trivial, and various algorithms exist to calculate a trajectory from an intermediate motion representation.

Commonly, these representations enable passing a desired motion to the robot's controller with a lower frequency than the real-time control loop. The algorithm will then interpolate the trajectory, e.g. by calculating a path, a time parametrization, or just by low-pass filtering. Typical representations are

- low-frequent (less than 1 Hz) position waypoints that are reached by the robot subsequently, either in joint space or transformed via the robot's inverse kinematics from Cartesian space [56],
- mid-frequent (around 10 Hz) velocities that are filtered for a smooth acceleration signal, either in joint space or transformed via the robot's Jacobian from Cartesian space [73],
- mid-frequent joint accelerations that are restricted in a way that the future trajectory will stay within the kinematic limits [62].

Cartesian waypoints allow to define the actions in the same space as the manipulation task or common 3D sensor data, and therefore circumvent learning additional transformations between spaces. Moreover, manipulation tasks are oftentimes about specific contacts and less about free-space motions. Defining manipulation via waypoints such as grasp poses,

place poses, or general contact poses allows for outsourcing the free-space motions to a motion planner or trajectory generator. These waypoints are also an understandable and natural representation of manipulation actions. By pre-defining arbitrary complex trajectories relative to a Cartesian position-waypoint, they can be applied to specific manipulation tasks and not only free-space motions. Overall, the action space is inherently connected to an intermediate representation of the robot trajectories.

Manipulation tasks are commonly solved by using the n dimensional joint space (e.g. with six or seven degrees of freedom (DoFs) for robot arms), six-dimensional Cartesian space, or a four-dimensional planar Cartesian sub-space. A planar action space sets two degrees of rotation fixed and is oftentimes used for top-down manipulation. For example, the end-effector might always point down (along the negative z-axis), but can freely translate and rotate around z. Reducing the dimensionality of the action space can simplify the learning problem drastically, and finding the smallest action space that can solve a task is oftentimes a challenging first step. In the following, we differentiate between continuous, discrete, and hybrid action spaces.

Continuous Actions Continuous actions are necessary to control a robot with arbitrary precision. To also allow for arbitrary complex motions, the action space can be directly mapped to the motor torque signals within the high-frequency control loop of the robot. In practice, manipulation policies are rarely able to be faster than 10 Hz due to the computational complexity of image- or point cloud processing.

Instead, the action space maps to a velocity-like control of the end-effector. In many recent works such as QT-Opt [61] for grasping, the action space controls the Cartesian velocity of the equilibrium pose of the downstream impedance controller [18, 37, 38, 57, 73, 97]. With low positional gains, the robot will stay within its limits, however at the expense of positional accuracy as well as low speeds. The impedance control is in particular beneficial when dealing with contact-rich tasks, as it is commonly found in learning for manipulation. If a contact or collision occurs, the impedance control will result in a smoothly increasing force in the direction of the equilibrium pose, instead of a sudden stop of the motion. In general, this is a very powerful and expressive approach, but it requires a lot of data due to its comparably time-dependent and high-dimensional action space. It extends easily to closed-loop control.

More recently, the action space covers the next Cartesian pose of the manipulation action, typically with low frequencies such as 1/2 Hz [56]. The trajectory from the current state to the next estimated Cartesian pose was either calculated by direct profiles or with the means of a complete motion planner including obstacle avoidance and time optimization.

Discrete Actions From a theoretical perspective, most algorithms for reinforcement learning (RL) have been developed on top of discrete or even tabular action spaces [115, 87]. While many extensions for continuous action spaces have been proposed [93, 76], discrete action spaces have several theoretical advantages. First, discrete actions simplify

the mapping from an action a to an action value Q or estimated total reward r. Second, all actions can the calculated and compared for maximum value. Third, the observation space is also oftentimes discretized in space, e.g. for pixel or point cloud inputs. Then, the same discretization of the Cartesian space can be used for the observation and action space.

In this regard, a common approach is to use a depth image in combination with an action space of planar Cartesian poses [81, 130]. The viewing direction and the z-axis of the frame are chosen to be parallel so that the image frame and planar plane correspond. Then, the depth image can be centered at and oriented according to a planar action pose. The transformed image - relative to the action pose - is oftentimes used to estimate the value of an action. For grasping, this might correspond to a grasp quality metric or, when normalized, to a grasp success probability. While images are spatially discrete, they can however be transformed continuously e.g. by interpolating pixels. Finding a maximum value becomes challenging as both the transformation as well as the value predictions might be computationally expensive. For this, one continuous approach for optimization is the cross-entropy method [81, 82]. Other approaches are based on image discretization and transform and evaluate the image at a fixed set of poses. This can be either done on a random set of poses [94], or at every possible pixel as an action pose [128].

When reducing the temporal resolution of actions, the most extreme simplification is single-step actions for manipulation. Then, a complete manipulation task is solved with a single action representation of arbitrary dimensions, computed entirely beforehand, and then executed in an open-loop manner. Following related work [34, 36], we adopt the term *manipulation primitives* for this approach. Oftentimes, the policy decides where to apply which pre-defined primitive, reducing the complete action space to a single Cartesian position and a discrete primitive selection.

In case a discretization exists that can solve a task, discrete actions have a range of advantages in practice. First, they are comparably more *data-efficient* due to the limited number of actions. Second, they are more *precise* as continuous actions require interpolation and smoothing of the policy or the value function. In this regard, discrete actions lend themselves for high-precision tasks, in particular in the context of open-loop primitives. While continuous actions, combined with a closed-loop approach, might also allow for high precision, the time dependency and slow execution create complexity in the learning process.

Hybrid Actions Continuous and discrete action spaces are not exclusive and can be combined into a hybrid action space. Slightly different tasks or subtasks might benefit from different action spaces. For example, task-specific grasping can be solved in a discrete space, while the downstream task can be parametrized continuously [130]. Still, both should be calculated simultaneously due to their interdependence. Residual learning is a branch of research about using classical model-based approaches and learning an error function on top. Here, hybrid action spaces are also used frequently [130].

2.4 Task-specific Work

We introduce related work that is highly relevant to the tasks investigated furthermore.

2.4.1 Planar Grasping

In recent years, research on robotic grasping in unsystematic environments gained a lot of traction. Bohg et al. [16] divided possible approaches into the grasping of known, familiar, or unknown objects. For known objects, grasping is mostly reduced to object recognition and pose estimation. More flexibly, this process can be extended by matching the similarity of known objects to new or familiar objects. In the case of unknown objects, the approaches can be split into analytical and empirical methods [102]. Analytical grasp synthesis relies on the definition of a grasp quality measure, such as the force closure or other geometrically motivated measures. However, these approaches usually need high-quality and flawless 3D maps of the scene. Data-driven methods intrinsically develop generalization and error robustness, but suffer from the demand for large datasets. First implementations sampled grasp candidates in a simulator and evaluated them based on an analytical measure [86]. In 2008, Saxena et al. [105] applied a probabilistic model to a mostly simulated dataset to identify grasping points in image patches. More recently, Mahler et al. [82] built a large synthetic grasping database with analytic metrics, used supervised learning to train a NN, and applied it to bin picking [80]. Reducing the gap between simulation and reality is of high priority in the robotic community [18].

Alternative to simulation, the system can be automated and trained in a large-scale self-supervised manner. Our work is closely related to the research of Pinto and Gupta [94]. They were able to retrain a NN within 700 h and achieved a grasping rate for seen objects of 73 %. Levine et al. [73] scaled up the data collection to 800 000 grasp attempts on 14 robots. They trained in an end-to-end manner, including camera calibration, spatial relationships, and time dependency. While deep RL showed impressive results in learning from visual input in simple simulated environments [101, 87], direct applications of RL for robot learning have proven to be more difficult. On real robots, continuous policy-based methods have been applied to visuomotor learning, either in an end-to-end fashion [72] or by using spatial autoencoders [38]. Quillen et al. [97] showed in a recent comparison of RL algorithms for simulated grasping that simple value-based approaches performed best.

2.4.2 6 DoF Grasping

For data-driven or hybrid 6 DoF grasping, simulation and analytical metrics have been explored so far. In this context, Gualtieri et al. [46] used a (mostly) model-based grasp synthesis and focused on the grasp ranking. Here, a convolutional NN maps a rendered depth image, corresponding to the viewpoint of the grasp from the approach vector, to a learned grasp quality score. ten Pas et al. [118] extended this approach to multiple viewpoints around the grasp pose to better consider spatial data of the environment.

To avoid the costly rendering of the depth images, Liang et al. [75] proposed GPD, an algorithm that learns analytical grasp metrics directly from point clouds. The grasp evaluation makes use of the PointNet architecture to learn a grasp measure from 3D spatial relations, and can include complex geometric structure of the contact area between the gripper and the object. Qin et al. [96] developed a single-shot grasp proposal network built upon the PointNet++ architecture. Still, the training data generation relies mostly on antipodal grasps evaluated by a classical grasp quality measure. Furthermore, NNs for point cloud processing are one to two orders of magnitude slower than convolutional NNs for depth images.

Mousavian et al. [88] introduced a grasp sample NN based on a variational autoencoder and simulated training data. Subsequently, an iterative grasp refinement step selects and improves grasps taking the point cloud as input. Gualtieri and Platt [45] use a multiple time-step and hierarchical approach that can learn a sequence of actions to focus attention on the task-relevant parts of the scene. They model the grasp as a RL problem and develop a set of constraints to find a suitable yet generally applicable subset of state and action representations. These methods however suffer from their slow execution and low picks per hour (PPH).

Recent works in shape completion have improved grasp success rates based on a single depth image [121]. Lundell et al. [78] render multiple viewpoints from discrete angles and use an fully convolutional neural network (FCNN) to predict the grasp quality at a large number of 6 DoF poses. To improve grasp rates in dense clutter, Murali et al. [89] extended this concept to a learned collision-checking module. In contrast, Schmidt et al. [107] chooses a single depth image as the state space. Here, a convolutional NN is trained to predict the 6 DoF grasp pose as a regression problem, without any grasp ranking. Due to its non-stochastic and single-modal nature, this approach works only for isolated objects.

2.4.3 Pre-grasping manipulation

As modeling grasps with classical approaches is already challenging, even more complex interactions like pre-grasping actions were studied less frequently. Within this scope, Dogar and Srinivasa [29] combined pushing and grasping into a single action, enabling them to grasp more cluttered objects from a table. Chang et al. [23] presented a method for rotating objects to find more robust grasps for transport tasks. In recent years, the progress of machine learning in computer vision enabled robot learning based on visual input [38], oftentimes learned from simulated data. However, as contact forces are difficult to simulate, training of more complex object interactions for pre-grasping manipulation remains challenging.

Alternatively, a robot can be trained in a self-supervised manner from real-world object manipulation. An end-to-end approach for general grasping strategies comes with the cost of sparse rewards and high data consumption. Kalashnikov et al. [61] trained an expensive multi-robot setup for 580 000 grasp attempts, resulting in an impressive grasp rate of 96 % for unknown objects. Furthermore, the robots implicitly learned pre-grasping manipulation

like singularization, pushing, and reactive grasping. In contrast, Zeng et al. [128] introduced a pushing motion primitive and learned grasping and pushing in synergy by rewarding the sparse grasp success. Using significantly less training data than [61], their robot was able to clear a table from tightly packed objects.

2.4.4 Pick-and-place

Task-based grasping investigates the interdependence between grasps and the subsequent task [74, 15, 91]. Regarding pick-and-place, Zeng et al. [129] used classical image matching to pick-and-place in the broader sense of semantic grasping, without further requirements for a *precise* place pose. While the robot was able to grasp objects out of clutter reliably, grasping was trained on human-annotated data. Gualtieri et al. [47] learned pick-and-place of 6 DoF in simulation and were able to transfer the results to real-world cluttered scenes. However, the robot is limited in generalizing to unknown object classes and novel place poses. Besides grasping, some work has focused on individual subproblems of pick-and-place. Jiang et al. [60] have focused on finding a stable place pose given a grasping configuration. Zhao et al. [132] calculated object displacements during the grasping action without an object model. Recently, Zeng et al. [130] have learned to pick and throw objects into target bins, extending prior work of primitive-based grasping. However, future toss actions do not influence prior grasps.

Pick-and-place is a popular task within *Imitation Learning* and RL. Finn et al. [39] used meta-learning on multiple pick-and-place tasks to achieve one-shot imitation learning for novel objects and scenarios. Their accuracy suffices for placing objects within a larger box or bowl, however, their grasping approach seems restricted to non-cluttered scenarios. Duan et al. [31] used one-shot imitation learning for the task of block stacking, including training in simulation, virtual reality, and a final sim-to-real transfer. Singh et al. [110] has learned robot manipulation in the context of Inverse RL. While their bookshelf scenario is quite simplified in the context of pick-and-place, their method allows for an easy and flexible definition of a goal state without an explicit reward definition.

2.4.5 Garment Folding

We use a bimanual robotic system for the task of garment smoothing and folding. In this regard, bimanual robotic manipulation has been studied extensively in fields from surgical robotics to industrial manipulation [111]. A dual-arm system extends the workspace, allows for increased payload and for more complex behaviors than a single arm system [124, 71, 52, 54], but comes at the cost of higher planning complexity due to the additional DoFs and self-collisions [33]. A promising line of research is to employ dual-arm systems for garment manipulation [42]. Garments are especially difficult to control and manipulate due to their large configuration space, self-occlusions, and complex dynamics [41]. Recent works have mainly focused on garment smoothing from arbitrary configurations [49], or garment folding, assuming the garment has been initially flattened [124].

As garment folding has many applications in hospitals, homes, and warehouses, interest from research has increased in the last decade. Early approaches rely heavily on heuristics and can achieve high success rates, but have long cycle times on the order of 10 min to 20 min per garment [30, 83, 6, 5, 117]. Recent methods have been focusing on learning goal-conditioned policies in simulation [124, 53, 109, 116] and directly on a physical robot [70].

Garment smoothing aims to transform the garment from an arbitrary crumpled configuration to a smooth configuration [108]. Prior works have focused on extracting and identifying specific features such as corners and wrinkles [30, 83, 114, 125]. Recent methods have used expert demonstrations to learn garment smoothing policies in simulation [41, 53, 108], however, these methods learn quasi-static pick-and-place actions that require a large number of interactions on initially crumpled garments. Ha and Song [49] introduced a novel 4 DoF dynamic fling action parameterization learned in simulation that can achieve $\sim 80\,\%$ garment coverage within three actions. However, this parameterization is (1) limited to fling actions, (2) fails to fully smooth garments, and (3) induces grasp failures in more than 25 % of actions.

Chapter 3

Learning for Manipulation

In classic literature, Mason [85] defines manipulation as the process of moving or rearranging objects in the environment. While this definition holds true within the scope of this work, it also illustrates an *object-centric* point of view still prevalent in robotics. Instead, we suggest changing the view on manipulation and defining it as a targeted spatial change of the environment. This *action-centric* view has two advantages: First, it gets along without an (even harder) object definition. For example, soft or loose objects do not have a fixed coordinate frame that moving might refer to. Second, it emphasizes the intention of the action, as manipulation is executed to solve a specific *task*. This different perspective allows to *correlate* a robotic action with a task outcome directly, without the need for intermediate representations like object or environmental models. Finding this correlation between action and task outcome might be *learned*, for example by trial and error.

In this chapter, we introduce our approach to manipulation from a general point of view. First, we present manipulation in the notation of reinforcement learning (RL) (3.1). After defining the visual state space (3.2) and the action space (3.3), we derive and motivate simplifications of our general approach (3.4-3.5). Section 3.6 and 3.7 describe the foundational algorithms for data-driven RL. In the chapters thereafter, we apply and adapt this approach to various real-world tasks.

3.1 Reinforcement Learning

Reinforcement Learning (RL) is a powerful framework to describe, and more importantly, solve general robotic tasks. It applies *machine learning* to solve a task by learning from data instead of explicit programming. Although we simplify the *general* RL setting significantly further (e.g. by reducing it to a single or very few time steps), we want to briefly introduce its formal definition.

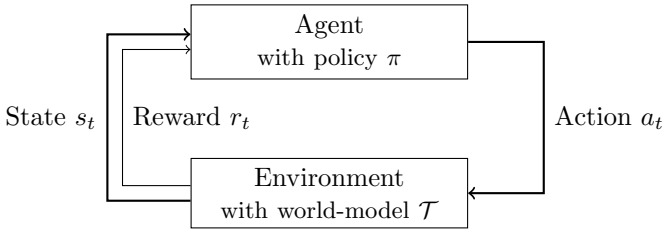


Figure 3.1: A summary of reinforcement learning (RL): An agent tries to maximize the collected reward $R = \sum r_t$ over multiple time steps t by observing and acting within an environment according to a learned policy π.

3.1.1 Markov Decision Process

A Markov decision process (MDP) is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, p_0, r)$ with a state $s \in \mathcal{S}$, an action $a \in \mathcal{A}$, a transition model $\mathcal{T}$, the initial configuration p_0 and a reward function r. The transition model $T : \mathcal{S} \times \mathcal{A} \mapsto p(\mathcal{S})$ maps an action in a given state s to a new state probability $p(s)$. The reward function $r : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{R}$ maps an action in a given state to a single scalar r. The goal of an agent is to maximize the discounted sum of collected rewards given by

$$R = \sum_{i}^{N} \gamma^i r(s_i, a_i) \tag{3.1}$$

with the (possibly infinite) episode length N. The discount factor $\gamma < 1$ weighs the expected rewards in the near future more than in the long term. This is motivated by the higher uncertainty of rewards in the distant future [115].

The solution of a RL problem is a *policy* π

$$\pi : \mathcal{S} \mapsto P(\mathcal{A}) \tag{3.2}$$

mapping a state s_t to a probability distribution of actions at every time step t. To find a final executable action a_t, a policy might be split into a probability distribution $\psi : S \times A \mapsto [0, 1]$ over the state-action space and a subsequent selection function σ. The first function ψ gives a probability for every possible action $a_t \in \mathcal{A}$ in the current state s_t. The selection function σ either maps directly to an action a_t or to another random variable conditioned on action a_t. The overall policy can then be written as the composition $\pi = \sigma \circ \psi$. Some literature keep the name *policy* for the probability distribution ψ alone.

3.1.2 Finding a Policy

Most approaches to RL build upon the state-value function $V_\pi(s)$. Let $\mathbb{E}(\hat{x})$ denote the expected value of a random variable $\hat{x}$. The state-value equals the expected reward starting

from state s and following a given policy π:

$$V_\pi(s) = \mathbb{E}_\pi\{R_t | s_t = s\}. \tag{3.3}$$

Similarly, the action-value function $Q_\pi(s, a)$ denotes the expected reward starting from state s, then performing action a at the next time step and afterwards following a given policy π:

$$Q_\pi(s, a) = \mathbb{E}_\pi\{R_t | s_t = s, a_t = a\}. \tag{3.4}$$

Presuming a known action-value function, a typical policy is the *greedy* action

$$a^* = \arg\max_a Q_\pi(s, a). \tag{3.5}$$

Based on these definitions, different approaches to the general RL problem were developed. First, state-value based solution methods estimate the expected reward for a set of possible states, e.g. via the Bellmann equation [115] update, and then apply a selection function σ to find an action. The set of states $\mathcal{S}$ can be estimated by applying a number of actions to a known model of the environment $\mathcal{T}$, leading to so-called model-based RL. Alternatively, composing the Q_π-function with a selection function σ, e.g. as in Equation 3.5, works without an explicit transition model and is called model-free RL. Here, the Q_π-function will *intrinsically* learn information about the transition model of the environment. The set of evaluated actions $\mathcal{A}$ is preferred to be discrete and finite, similar to the set of states in model-based RL.

Policy-based solution methods learn the policy $\pi : S \mapsto A$ directly as a model-free approach. In particular, they can be extended by an action-value function to *actor-critic* methods. First, a normal action-value function $Q_\pi(s, a)$ is learned to predict the expected reward after an action a. Subsequently, a policy π is learned by maximizing the action-value $Q_\pi(s, \pi(s))$. This way, actor-critic methods allow to learn a policy from a known action-value function, and in particular do so for continuous, high-dimensional, and uncountably infinite action sets in a straightforward manner. However, because the policy depends on the action-value, actor-critic methods are prone to learning and amplifying errors. [115]

3.1.3 Typical Challenges

RL problems suffer from typical challenges independent of the approach. The long-term goal of a policy is to maximize the expected reward. To do so, the agent needs to know and explore the environment and its reactions in the first hand. This leads to a discrepancy between exploration vs. exploitation which is often individual to each task. At $t = 0$ of an unknown MDP, the agent explores the state-action space by choosing a uniform random action. Often, a parameter $\varepsilon \in [0, 1]$ is defined as a probability of choosing a random action instead of a learned policy π. By decreasing ε over time, the exploration shifts to an exploitation phase.

Exploration is usually done by trial and error in RL. This implies that all learned information originates from trying out successful or unsuccessful actions. For many tasks, this results in a large number of required actions and low *data efficiency*. For manipulation tasks like grasping, a large number of possible grasps for even one object type exists that needs to be explored.

For many real-world RL problems, the reward function r_t is sparse with $r_t = 0$ for most t. Those reward functions correspond to problems that can only solved by a good *sequence* of actions. Then, if a high reward is given, it cannot be assigned to a single action. As an example, a successful garment fold might be rewarded by a single high reward afterwards. However, the folding might have required a large number of smoothing, turning, folding, or stretching actions to work out at the end. Finding out which actions are relevant to the final reward is difficult with a single scalar. Additionally, the importance of actions for a reward might be unrelated to the temporal proximity to the reward. This challenge is known as a time-delayed reward. [115]

3.2 Observing the Scene

The agent observes and estimates its environment in a state $s \in \mathcal{S}$. The state corresponds to all information that is available to the robot for planning or reacting. For manipulation tasks, visual sensors like a color (RGB), depth (D), or combined (RGBD) camera are commonly used as primary input. Interoceptive sensors like force sensors or motor encoders are, at least indirectly, used for lower-level motion control. For completeness, there exists work for manipulation with other sensor types like tactile sensors, electromagnetic sensors (e.g. RFID), LIDAR sensing, or microphones for audio signals.

In this work, the robot calculates manipulation actions based on a depth-sensing camera. Stereo cameras, usually with an infrared projector to improve depth estimation, are available at a reasonable price. Typically, the raw camera output is given as a *point cloud* defined by a list of points i with position $(x, y, z)_i$. Some cameras include color information $(r, g, b)_i$ as a textured point cloud. The number of points is in the order of 1×10^5 or more. Usually, a recorded point cloud is projected into the image space. This is done for the following reasons: (1) an image grid gives a spatial context for a point cloud that (2) reduces the data size of the state significantly, as an image is a much denser representation and (3) is used by a lot of downstream algorithms in computer vision. A point cloud is projected to an image plane according to a linear mapping

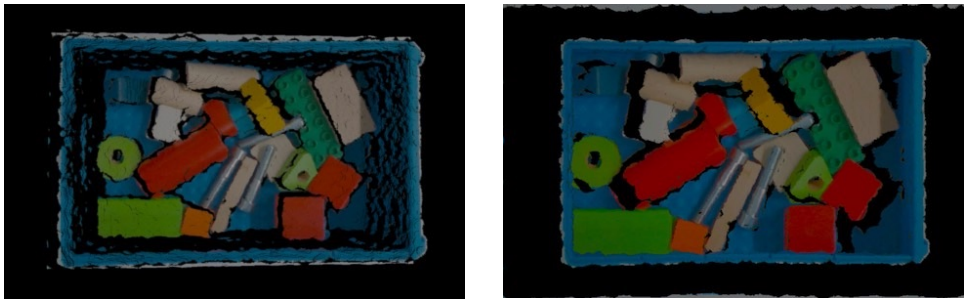
$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = I\,P\,R\,T \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{3.6}$$

chaining the camera (extrinsic) transformation RT, the projection matrix P, and the camera intrinsics I. The intrinsic and extrinsic transformation needs to be calibrated for each

experimental setup. A point cloud can be projected into an image via different projections: First, the *perspective projection* P_P maps points to a picture plane, projecting distant objects as smaller than nearer counterparts, resulting in lines parallel in world coordinates vanishing at a single point. This projection approximates cameras and the human eye.

$$P_O = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad P_P = \begin{pmatrix} \frac{2n}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2n}{t-b} & 0 & 0 \\ \frac{r+l}{r-l} & \frac{t+b}{t-b} & -\frac{f+n}{f-n} & -1 \\ 0 & 0 & -\frac{2fn}{t-b} & 0 \end{pmatrix} \tag{3.7}$$

Second, the *orthographic projection* P_O has a direction of projection orthogonal to the projection plane. This way, every plane of the scene appears in affine transformation on the image, and the distance of an object does not change the position within the image plane. Equation (3.7) shows the mapping as used furthermore, mapping points from inside the so-called viewing *frustum*, a pyramid with dimensions to the *left* l, *right* r, *top* t, *bottom* b, *near* n, and *far* f side. Figure 3.2 shows example images of a bin in orthographic and perspective projection.



(a) Perspective Projection **(b)** Orthographic Projection

Figure 3.2: Example of a perspective and orthographic image projection on the same point cloud data. While the perspective projection shows more distant objects smaller, the orthographic projection preserves distances between points.

Naturally, the observation of the environment is limited by the field of view of the camera. In particular, a single camera is only able to record images from a single perspective, leading to viewing "shadows" and undercuts in the depth estimation. These are prominent when dealing with high objects or dense clutter within the observed scene (Figure 3.2).

3.3 Actions for Manipulation

The agent interacts with its environment by executing actions $a \in \mathcal{A}$. This action space is a fundamental distinction of approaches for learning manipulation. In the following, we will introduce a *primitive-based* action space.

3.3.1 Robot Kinematics

Let us first recap the kinematics of a robot. In this work, a robot arm refers to a serial kinematic chain with known relative frames between each link. Commonly, the Denavit-Hartenberg convention is used to parametrize the kinematic chain. Then, the transformation T_n^{n-1} between the reference frames n and $n-1$ is given by

$$T_n^{n-1} = \begin{pmatrix} \cos\theta_n & -\sin\theta_n & 0 & a_{n-1} \\ \sin\theta_n \cos\alpha_{n-1} & \cos\theta_n \cos\alpha_{n-1} & -\sin\alpha_{n-1} & -d_n \sin\alpha_{n-1} \\ \sin\theta_n \sin\alpha_{n-1} & \cos\theta_n \sin\alpha_{n-1} & \cos\alpha_{n-1} & d_n \cos\alpha_{n-1} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.8}$$

with the two angles θ and α, as well as two distances a and d. The end of the last chain is known as the tool center point (TCP) or end-effector of the robot. The pose of the TCP is then given by

$$T_N = \prod_{i=1}^{N} T_i^{i-1} \tag{3.9}$$

The number of actuators N in between the links is usually equal to the DoF of the robot. Common robot arms have 6 or 7 DoF to reach a target position with a given orientation. The vector of all actuator positions $\vec{q} = \{\theta_1, \cdots, \theta_N\}$ is referred to as the joint position $\vec{q}$ of the robot. Let p denote a possible Cartesian end-effector pose, combining a translation $\vec{t}$ and a rotation R.

The forward kinematics $f_k(\vec{q}) \mapsto p$ maps a joint position to an end-effector pose according to Equation 3.9 for a kinematic chain. Let $J = \nabla_q p(\vec{q})$ be the Jacobian of the robot's pose in respect to its joint positions. The inverse kinematics $f_k^{-1}(p) \mapsto q$ maps an end-effector pose to a joint position. The inverse kinematics is much more complex, as joint positions are not unique or might not exist for a target pose p. In case of more than 6 DoFs, the robot usually has infinite solutions, the so-called *null-space* of the robot. For example, a robot with 7 DoFs resembling a human arm can move its elbow while keeping the end-effector pose fixed. There exist *analytical* and *numerical* approaches for finding the inverse kinematics. While analytical methods are usually preferred due to their faster and more exact calculation, they in general exist only for robots with less than 7 DoFs. Otherwise, numeric approaches allow to solve for joint positions with flexible secondary goals. Typically, the inverse Jacobian is used to iteratively refine the current end-effector pose $\vec{q}' = \vec{q} - J^{-1}(\vec{q})\Delta p$ to its target value.

3.3.2 Coordinate Frames

The start of the kinematic chain T_0 relative to a world frame is called the robot's base frame. The end-effector pose $p = T_N$ is naturally given in this coordinate frame. A robot commonly manipulates the environment with a *gripper* as its end-effector. Here, the TCP

or gripper frame is usually defined as the fingertip of the (closed) gripper. Point clouds and images are taken in the camera frame. The camera can be mounted at the flange of the robot (the so-called eye-in-hand configuration) or externally with a fixed transformation to the robot's base frame. While both configurations enable different approaches regarding manipulation, this difference is irrelevant throughout this work from an algorithmic point of view. From a more practical perspective, they require different calibration techniques and robotic motions. Additionally, we define a *scene frame* in the center of the currently relevant scene. For the task of bin picking, this refers to the center of the current bin and allows to easily switch between bins.

Besides the full homogeneous transformation that corresponds to a pose, its 6 DoF parametrization is of interest furthermore. A pose p can also be expressed by

$$p \in \mathrm{SE}(3), \quad p = \{x, y, z, \alpha, \beta, \gamma\} \tag{3.10}$$

via the translation (x, y, z) and the Euler-angles (α, β, γ) of the rotation. Figure 3.3 gives an example for the TCP relative to the scene (bin) frame. Note that we use an unusual Euler-angle convention: An extrinsic rotation α around z is followed by two intrinsic rotations β and γ around x' and y' respectively. First, the pose p is then a simple extension of a *planar pose*

$$\tilde{p} \in \mathrm{SE}(2), \quad \tilde{p} = \{x, y, \alpha\} \tag{3.11}$$

corresponding to a 3 DoF parametrization (or sometimes 4 DoF with the height z). More-

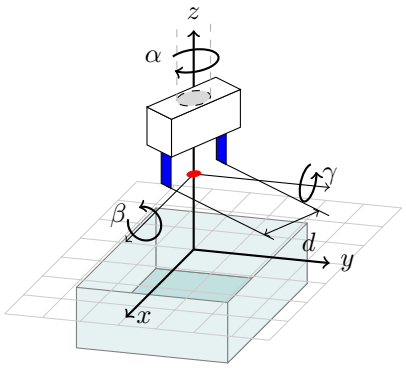


Figure 3.3: The pose of the gripper in the scene frame. The rotation is defined by an extrinsic rotation α around z, following intrinsic rotations β and γ around x and y' respectively.

over, decoupling the first rotation α from the intrinsic β and γ angles is helpful for pose definitions relative to the camera frame (as shown in Figure 3.4). It allows keeping α aligned to the rotation of the image plane, and *both* β and γ fixed to the prior transformation α. In particular, rotating the camera around its viewing direction does not alter the angles β and γ. This work makes extensive use of the TCP's pose relative to the camera frame. If not otherwise stated, the camera observes the bin in a top-down view, with only a fixed offset along the z-axis between the *camera* and *scene frame*.

3.3.3 Manipulation Primitives

Let $\mathcal{M}$ denote a set of *manipulation primitives*. The key idea is to define a *primitive* as a sequence of motions, actions, or arbitrary robot control signals that are parametrizable by very few to zero parameters. The parameters allow to make the task execution of the robot flexible and adaptable at run-time. At the same time, the high-level motion of the primitive allows to integrate task-specific knowledge into the robot's behavior. A primitive should have as many parameters as necessary to solve a task under all circumstances. On the other hand, it should have as few parameters as possible to simplify the run-time adaption.

In general, a primitive m is defined by a parametrizable trajectory

$$\vec{q} = m_{\vec{u}}(t) \quad \text{for } t < T \tag{3.12}$$

mapping a time t and the parameters $\vec{u}$ to the robot's joint position. Here, the joint position $\vec{q}$ may also include the gripper position d. Let T be the duration of the manipulation primitive. Two exemplary primitives are: First, a common use-case is to have a trajectory in Cartesian space $\vec{r}(t)$ that is parametrized only by the *start pose* $\vec{u} = p$. The Cartesian trajectory is then executed relative to p, and the robot's inverse kinematics calculates the final joint trajectory $\vec{q}$. Given a set $\mathcal{M}$ of these primitives, the robot needs (only) to decide *where* to execute *which* primitive. Formally, this corresponds to a policy

$$\pi : S \mapsto \mathcal{M} \times \mathrm{SE}(3) \tag{3.13}$$

mapping a state s to a pre-defined manipulation primitive m parametrized by its starting pose p. For the task of grasping, a simple primitive might follow a straight-line path parallel to the robot's fingers (in positive z-direction) until the TCP reaches the desired grasp point. The position of the grasp point is specified by the pose p, and the orientation defines the approach direction.

Second, a dynamic fling motion is another example of a manipulation primitive. Similarly, the grasp pose p is a first parameter. Afterwards, the robot follows a trajectory that is, besides a smooth transition in between, independent from the initial grasp point. The height h of the dynamic fling trajectory needs to be adapted for a successful task execution. Then, the overall manipulation primitive has seven parameters, six for the grasp pose and one for the height.

3.4 Viewpoint Invariance

A robot manipulating its environment is a physical system that (neglecting some marginal cases) can be understood by classical mechanics. As such, object-centric manipulation has focused on geometric and dynamic models of the robot and its environment. In particular, mathematical models were derived to make *predictions* regarding the task success. Although this work avoids these models intentionally, we still use specific characteristics of the underlying physics to simplify our approach.

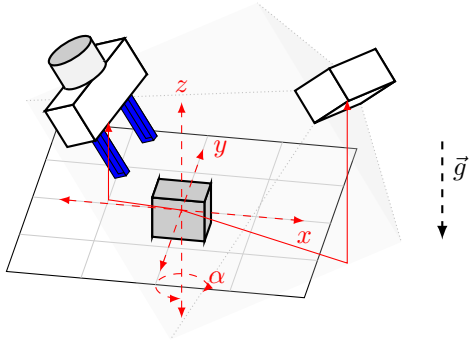


Figure 3.4: Given a setup with a gripper (left), a camera (right), and an object of interest (center), the gravitational vector $\vec{g}$ influences the manipulation and a (e.g. dynamics) model used for prediction. If the model does not know about $\vec{g}$, the model remains invariant under translations of x, y, z and rotations α around $\vec{g}$. We call this the *viewpoint invariance*.

First, we investigate transformations under which manipulation tasks are invariant, intending to uncover possible simplifications. Figure 3.4 sketches a scene including a robotic gripper, an object of interest, and a camera. The physics of manipulation does not change under translation or rotation of the *entire scene* around the vector of gravity $\vec{g}$. For $\vec{g} = -\vec{e}_z$, this corresponds to the parameters x, y, z, and α of the scene frame pose (Equation 3.10). This invariance does not hold for the remaining rotations β or γ due to the fixed gravitational vector. For example, grasping an object from the side requires a different policy than grasping from the top, as the object might slide or tip over. Moreover, moving the camera freely does, of course, not influence the manipulation itself but only the observation thereof. *Modeling* the manipulation, e.g. for predicting the manipulation outcome from an observation, requires explicit knowledge about the gravitational vector $\vec{g}$. For example, a complete model would require the dynamics of the object, which depends on the gravitational force. If the observer (e.g. the camera) however does not know about $\vec{g}$ explicitly, it might assume a fixed vector (e.g. from an implicit learned model). Then, likewise to the manipulation invariance above, the model remains *only* correct as long as the camera is translated or rotated around the gravitational vector. We refer to this as the *viewpoint invariance* of a manipulation model furthermore.

3.5 Principle of Locality

Second, classical mechanics gives clues about robot-world interactions relevant to manipulation. In particular, mechanics obeys the principle of *locality*. In other words, each action of the robot affects only a bounded proximity, and vice-versa only a bounded proximity of the environment influences an action outcome. Here, proximity refers to a spatial closeness between the robot and the parts of the environment of interest.

We investigate the locality that needs to be observed to predict the outcome of a ma-

nipulation action to a *certain degree*. First, we approximate manipulation as a slow action, neglecting any dynamic effects. For these *quasi-static* actions, the mechanical equilibrium holds throughout. Second, the accuracy of the approximation depends on the considered proximity. We differentiate between:

Gripper locality The mechanical equilibrium between the gripper and an environment's object is determined by the *contact* forces and torques. The contact is primarily influenced by the geometry of the gripper's and object's contact surfaces and their friction parameters. With full knowledge of the contact, manipulation actions can be described and predicted for a wide range of situations. For example, a firm frictional grasp depends only on this locality, and further collision-free motions can be fully predicted. The gripper locality includes the gripper itself as well as the inside contact surfaces.

Object locality For some actions, this is not sufficient to predict a manipulation action outcome, as knowledge about an object's mass or density distribution might be required. For example, an object might slowly rotate within the gripper as a frictional grasp might not be firm enough. Here, the object's center of mass needs to be known to predict the resulting movement. Similarly, objects oftentimes move while the gripper closes its fingers. To predict this displacement, the density distribution as well as the contact surface with the table beneath needs to be known. The object locality includes the object of interest and its contact surfaces. Usually, the object size is limited - either by a known object set or by gripper or robot hardware constraints. Then, the object locality can be approximated by the maximal object size.

Robot locality Knowledge about the robot might be required as soon as the robot gets in contact with an arbitrary object of the environment. For bin picking, this might be caused by a collision between the robot arm and the bin itself. Here, the robot locality considers the robot and all adjoined contact surfaces.

Environment locality In general, arbitrary objects might influence the manipulation, making knowledge about the complete environment necessary. For example, the object of interest might get jammed by other objects during bin picking. Or, an air draft might move a garment just before closing the gripper. In practical manipulation tasks, the environment is bounded by the robot work cell.

Figure 3.5 illustrates the mentioned localities and their catchment areas as an overlapping set. As a locality limits the available knowledge, a distant locality (e.g. containing the robot or the entire environment) is able to better predict an action outcome. However, we argue that contact forces are the main drivers of manipulation in practice, and therefore localities closer to the *contact points* of the manipulation are more relevant. For this reason, we motivate using a smaller locality to simplify the manipulation planning, usually between the *object* and *robot* locality furthermore.

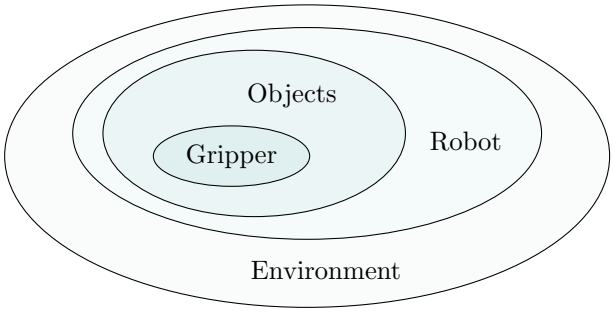


Figure 3.5: The considered locality around the gripper is essential to predict an action outcome. We motivate our *local* approach by the stronger influence of objects closer to the contact points of the manipulation action.

To implement a locality in our *image* state space, we use a cropped window with side length l centered around the position (x, y) and rotation α. Let s be the complete image of the scene, then we denote $\hat{s}$ as the cropped image window. Moreover, we *slide* this window around the image to evaluate the algorithm at different positions and rotations (Figure 3.6) in a so-called **sliding-window** approach. This way, we make use of the viewpoint invariance

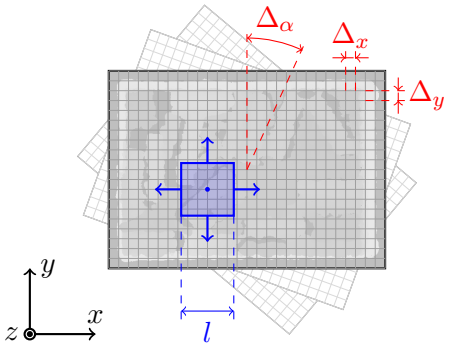


Figure 3.6: The translational invariance and bounded locality of the manipulation problem allow to solve it via a sliding-window approach: An algorithm evaluates a local proximity (blue) at a grid of planar poses with stride (Δ_x, Δ_y) for translation and Δ_α for rotation (red). The cropped window $\hat{s}$ is a subset of the scene image s.

of a manipulation model introduced in section 3.4. However, the invariance requires the z-axis of the camera frame and the gravitational vector $\vec{g}$ to be parallel. Usually, the planar pose (x, y, α) correspond to the pose of the robot's TCP, resulting in an image crop around the gripper and the *desired* contact points of manipulation. Then, the locality can be easily adapted by the window size l.

3.6 Reward Estimation

Most manipulation primitives work in a single-step and open-loop manner and thereby reduce the underlying MDP to a single time step $t = 0$. The action-value function $Q(s, a) = r$ is then given directly by a reward r that can be interpreted as a success metric of the primitive. Based on the recent developments in computer vision, we apply neural networks (NNs) for reward estimation. NNs with many layers are called *deep*, giving the name to the field of *deep learning*. We want to briefly introduce the most important terminology and methods.

In general, deep learning is a small subset within the broader field of *machine learning*. In the most general way, a machine learning algorithm learns a task T when a performance measure P improves with experience E. We explain these three terms furthermore.

Task A task T is fully defined by a function f mapping an input x to an output y. Out of many more types, we focus on the *classification* and *regression* task. For classification with k categories, an algorithm maps an input x to a category $f : \mathbb{R}^n \mapsto \{1, \dots, k\}$. The probability distribution $f : \mathbb{R}^n \mapsto [0, 1]^k$ is often used instead. Classification was of great importance for deep learning historically, as handwritten character recognition was one of the early successful tasks [69]. Two special cases are the binary classification for $k = 2$, e.g. to learn a success metric of a primitive, or the multi-class classification. The latter maps to $[0, 1]^k$ without being a normalized probability distribution, allowing an input to correspond to multiple classes simultaneously.

For regression, a function predicts numerical values from an input $f : \mathbb{R}^n \mapsto \mathbb{R}^m$ from n to m dimensions. Many relevant parameters of primitives such as its pose or other control signals are physically numeric, making regression the second fundamental task in this work, but also for broader research in robot learning.

Performance Measure The performance measure P of a specific task T is a scalar value that allows the evaluation and comparison of different algorithms. The *accuracy* is a common performance measure for classification tasks. It is defined as the fraction of correctly classified examples over the total number of examples. A binary classifier predicts positive or negative labels with four possible outcomes. The true positive (TP), false positive (FP), false negative (FN), and true negative (TN) are summarized in the confusion matrix. For the regression task, the performance measure is usually the mean squared error (MSE).

Experience Assuming the experience is savable, we refer to it as a *data set*. *Supervised learning* is the most common type of experience with a data set mapping inputs to known outputs. The creation of labeled data is very costly, as oftentimes manual work is required for thousands to millions of data samples. In contrast, *unsupervised learning* works only on the input data itself, for example in tasks like clustering or compression.

The data exploration found in RL is another type of experience (section 3.1). Commonly, an agent interacts with an environment in an *online* manner, getting direct reward feedback after choosing an action in a given state. *Offline* RL however uses a fixed data set of states, actions, and rewards to optimize for a policy. This way, the latter is conceptually related to supervised learning.

In *active learning*, an agent can sample examples from a data source to collect a data set. However, as this sampling process might be costly, the focus lies on selecting the optimal samples to learn quickly and robustly. Active learning is related to RL, and even more so to single-step RL, as this sampling problem corresponds to the exploration strategy.

In the context of robot learning, *self-supervised* learning refers to a robot collecting data (mostly) autonomously. The appeal of this method is to use robots for what they do best: Working on tedious tasks like data acquisition or data labeling without human operators. This *can* reduce the manual effort dramatically. In contrast in the general machine learning field, self-supervised learning denotes techniques using auxiliary tasks on a larger unsupervised data set to boost the performance measure on an original supervised data set. We keep the robotics meaning of *self-supervised* furthermore.

3.6.1 Principle of Maximum Likelihood

Let $\mathbb{X} = \{x^{(1)}, \dots, x^{(m)}\}$ be a data set of m examples, given by a training set as a part of an experience. In statistics, a function g that calculates a quantity of interest θ based on given data $\mathbb{X}$ is called an estimator. A point estimator is a function

$$\hat{\theta} = g(x^{(1)}, \dots, x^{(m)}), \tag{3.14}$$

where $\hat{\theta}$ denotes an estimation of the true value θ. The difference between is called the $bias(\hat{\theta}) = \mathbb{E}(\hat{\theta}) - \theta$. It is oftentimes desired for an estimator to have $bias(g) = 0$. However, it is challenging to find a sufficient estimator for arbitrary data directly. Instead, the principle of maximum likelihood provides a systematic derivation of estimators [43].

In the following, let θ denote the parameters θ of the task function $f(x)$, written as $f(x;\theta)$. An estimator could theoretically predict a function $\hat{f}$ directly, however, nearly all machine learning algorithms use a parametrized function $f(x;\theta)$ for simplicity. Let the data set $\mathbb{X}$ be drawn independently from an unknown data-generating distribution $p_{data}(x)$. The task function f predicts the probability $p_{model}(x;\theta)$ of a data point x. The principle of maximum likelihood defines an estimator for θ as

$$\theta_{ML} = \arg\max_{\theta} p_{model}(\mathbb{X};\theta) = \arg\max_{\theta} \prod_{i=1}^{m} p_{model}\left(x^{(i)};\theta\right) \tag{3.15}$$

that maximizes the probability of the given data using the model parameters θ. For computational and numerical reasons, the logarithm is maximized instead, as it does not change

the arg max operator:

$$\theta_{ML} = \arg\max_{\theta} \sum_{i} i = 1^{m} \log p_{model}\left(x^{(i);\theta}\right) = \arg\max_{\theta} \mathbb{E}_{x\sim p_{data}} \left[\log p_{model}(x;\theta)\ \right]. \quad (3.16)$$

In other words, the parameters θ are chosen to maximize the expected value of the log-model-distribution. This simplifies the problem of finding the best parameters θ for a function f solving a given task to a common optimization problem.

3.6.2 Gradient-Based Optimization

Optimization deals with finding the minimum $\arg\min_x f(x)$ of a known loss function $f(x)$. This process is also referred to as *training* a machine learning algorithm. The first-order Taylor series of the loss function is given by

$$f(x+\epsilon) \approx f(x) + \epsilon \nabla f(x) \quad (3.17)$$

with a small $\epsilon > 0$ and the gradient $\nabla f(x)$. In general, the gradient points into the direction of steepest accent. Therefore,

$$f(x - \epsilon \nabla f(x)) < f(x) \quad (3.18)$$

holds true for a sufficiently small ϵ called the *learning rate*. Repeating this update step $x' = x - \epsilon \nabla f(x)$, $f(x')$ will eventually converge to a local minimum. This method is referred to as *gradient descent*. In addition, stochastic gradient descent (SGD) averages the gradient over a batch of elements $x_1, x_2, \cdots, x_N$. Momentum is a technique that improves convergence behavior by averaging the last and current update change as a low-pass velocity filter. The Adam optimizer builds upon these techniques for a robust and fast optimization algorithm [63].

3.6.3 Neural Networks

Neural Networks (NNs) are parametrizable functions built of simple connected neurons showing an overall emergent behavior. NNs that try to approximate a function $f^*(x) = y$ are called feedforward, and information flows to the output without any feedback. A feed-forward NN defines a function $f(x;\theta)$; the goal of learning is to optimize the parameters θ so that $f(x;\theta)$ approximates $f^*(x)$ as closely as possible. It can be shown that NNs are able to approximate arbitrary functions with θ being of large enough dimension. In many practical use-cases, NNs are able to inter- and extrapolate comparatively well, which is known as *generalization*. [43]

The building blocks of NNs are simple matrix multiplications such as

$$z = Wx + b \quad (3.19)$$

with a weight matrix W and a bias vector b. The parameter set $\theta = \{W, b\}$ includes all weights and biases of the NN. Feedforward NNs are constructed as follows:

Output Units The output of the NN must resemble the output of $f^*(x)$, in particular regarding its shape and numeric range. If $f^*(x) \mapsto \mathbb{R}^m$, the most simple NN consists of the matrix multiplication (Equation 3.19) called a single-layer perceptron, with the parameters being the weights of the matrix. If $f^*(x) \mapsto [0, 1]$ or another bounded interval, the output can be limited by the logistic sigmoid function σ

$$x = \sigma(z) = \frac{1}{1 + \exp(-z)}. \tag{3.20}$$

If $f^*(x)$ maps to a discrete probability distribution, the softmax function

$$y = \text{softmax}(z) = \frac{\exp(z)}{\sum_j \exp(z_j)} \tag{3.21}$$

is commonly applied. The output of the softmax function is normalized to 1 by calculating the sum of all outputs z_j, e.g. for a classification task. One advantage of the sigmoid and softmax function over the MSE is that the exponential cancels out with the logarithm to a simple sum of the maximum likelihood. This way, Equation 3.16 can be simplified to

$$\theta_{ML} = \arg\max_\theta \sum_{i=1}^{m} [y_i \log(\sigma(z_i)) + (1 - y_i) \log(1 - \sigma(z_i))] \tag{3.22}$$

$$= \arg\min_\theta \sum_{i=1}^{m} [y_i - y_i z_i + \log(1 + \exp(-y_i))] \tag{3.23}$$

with the so-called logits z_i of the output layer and the label y_i.

Hidden Units Besides the output function, the NN based on Equation 3.19 is a *linear* model. Hidden units and activation functions are introduced to approximate general nonlinear functions $f^*(x)$. Hidden units h calculate a matrix multiplication, followed by a nonlinear activation function g

$$h = g(Wx + b). \tag{3.24}$$

Hidden units are stacked consecutively, using the output $h(i - 1)$ of the previous hidden units as its own input. The first hidden unit $h(1)$ uses x as input, the output unit uses the last hidden unit $h(n)$. This way, the units are organized as layers in a feedforward NN. The number of layers $n + 1$ is called the depth of the network. The term deep learning implies that the NNs have large depth, usually $n \geq 3$ [43].

Following Equation 3.24, the activation function g is essential for the entire network. A common example is the rectified linear unit (ReLU), defined by $g(z) = \max\{0, z\}$ as a computationally simple nonlinearity. Various alternatives exist, and one generalization is the leaky ReLU

$$g(z) = \max\{0, z\} + \alpha \min\{0, z\} \tag{3.25}$$

with the nonzero slope $\alpha > 0$ for negative $z < 0$. A key advantage of leaky ReLUs over standard ReLU is that gradients do not vanish for $z < 0$.

After constructing the NN, all its parameters θ are optimized based on the principle of maximum likelihood. It allows to derive a so-called *loss* function H that is finally minimized with a gradient-based optimization. The gradient ∇H is calculated, in particular for multiple layers, by the backpropagation algorithm. Without going into detail, it is an efficient algorithm for applying the chain rule of calculus to large NNs [43].

3.6.4 Network Architecture

Different types of layers can be derived from Equation 3.24. For applications such as computer vision, the vector x and h can be imagined to be a matrix, either by stacking the vector or by generalizing the matrix multiplication to a higher-dimensional *tensor*. Typically, data (and in particular images) are represented as a three-dimensional tensor of width W, height H, and number of channels C. Research in machine learning has come up with various specialized layer types. The normal matrix multiplication (Equation 3.24) is called a dense layer. In this work, 2D *convolutional layers* acting on matrix inputs play an essential role.

Convolution A convolution is a mathematical operation defined by an input matrix and a *kernel* (filter) matrix of smaller size. At a specific position (i, j), the output is the sum of the element-wise multiplication between the kernel and the input. The kernel is then shifted over the input with a step size called *stride*. Due to the kernel's extent, not all elements of the input matrix are mapped to an output. The relation between the input and output size for a single dimension is given by

$$output = \frac{input - kernel + 2 \cdot padding}{stride} + 1. \tag{3.26}$$

Figure 3.7 illustrates an example of a convolutional operation. In general, a *convolutional layer* maps an input of multiple channels $C > 1$ to an output of multiple channels $C' > 1$. Then, the layer is composed of $C \times C'$ kernels for each combination. The parameters θ of all kernels are learned independently. Commonly, padding with zero or constant values is added around the input matrix, allowing the convolution to have equal input and output size.

Importantly, convolutional layers can operate on inputs with *variable* sizes. In fact, the size can change *flexibly* during training or run-time. The output then scales with the input according to Equation 3.26, keeping a fixed offset due to the kernel's size. In contrast, dense layers operate on fixed inputs only.

Fully Convolutional Neural Network A NN that only consists of convolutional layers is called a fully convolutional neural network (FCNN) [77]. While a deep FCNN allows

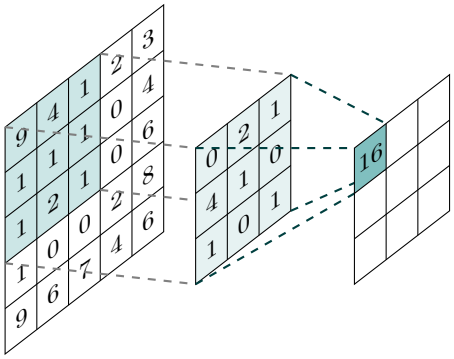


Figure 3.7: A single multiplication step of a convolution operation. A 3×3 kernel matrix (mid) is shifted over an 5×5 input matrix (left), resulting in a 3×3 output (right).

to approximate more complex functions than a single layer, it allows keeping the desired properties of the convolution: First, the *entire* FCNN can adapt to the input size and predict at each position in the width and height dimension. In the context of computer vision, different positions correspond to pixels in an image. FCNNs allow to predict tasks for each pixel individually, e.g. for *dense* pixel-wise classification or semantic segmentation. Second, FCNN keep an *effective* kernel size determined by the overlay of all kernels. This way, an FCNN only incorporates *local* information around its output position. By rescaling the kernel size, the stride, and the number of layers the *locality* of the FCNN can be adjusted.

In particular, FCNNs allow different image sizes during training and inference. As the effective kernel size remains fixed, the FCNN will then only predict on image patches corresponding to the training size. Third, an FCNN is translational invariant, as the kernel remains constant for all positions on the input. In particular, this allows for very *efficient* implementations, as each layer might reuse multiplications from neighboring positions as well as due to parallelization. This can speed up computations by multiple orders of magnitude over simple calculations of each position separately.

We will make extensive use of FCNNs to integrate these desired properties (from section 3.4 and 3.5) into manipulation planning. An FCNN implements a sliding-window approach for the translational DoFs (Figure 3.6). The effective kernel size corresponds to the window. However, convolutions restrict the sliding window approach significantly in terms of mathematical operations and expressiveness. In return, this allows for their efficient implementation.

3.6.5 Regularization

The *capacity* of a NN denotes a measure of the ability to approximate arbitrary functions f. The deeper the NN and the wider its layers, the higher its capacity. While there are also quantitative definitions of capacity, a qualitative understanding suffices furthermore.

Regularization refers to techniques improving the performance measure on the test set, even if that decreases the performance on the training set. In this regard, *overfitting* is a fundamental problem of machine learning algorithms. Here, the NN learns features of the limited samples in the training set which cannot be generalized to the underlying data distribution. Overfitting arises in particular for small training sets or NNs with high capacity. On the other hand, a particular capacity is necessary for learning the *true* features of the task. A variety of techniques were developed to reduce overfitting [43]:

Early Stopping Before training, NNs are initialized with random weights θ. The optimization stops as soon as the performance measure on the *test* set cannot be improved further.

Batch Normalization The idea is to normalize the batched input of a layer to zero mean and standard deviation of one. During training, this operation is applied for every batch and collects the average mean and covariate shift in the background. These values are then accounted for during inference.

Dropout During dropout, a fraction of neurons in a layer are randomly neglected for the calculation of the following layer. Instead, the activation of the remaining neurons is scaled to keep the mean activation. Dropout reduces the effective capacity of the NN, resulting in random and different subnetworks for each training step. For inference, the dropout rate is set to zero, which can be interpreted as averaging over all trained subnetworks. In this regard, dropout can be understood as an *ensemble* technique.

Norm Penalties A norm penalty of the weights θ or some activations $h(i)$ is added to the loss function. A penalty with L1-norm $|\theta|$ leads to minimal, sparse weights or activations. In comparison, the Euclidean L2 norm $\|\theta\|^2$, reduces large weight values. This induces smoother functions f regarding a smaller mean gradient $\nabla_x f$.

Data Augmentation Data augmentation is the technique of artificially increasing the training set size. In most cases, this exploits symmetries or invariances of the data. For example in image classification, the object classification stays fixed under image rotation or flipping. Then, the training pipeline randomly rotates or flips the image to create artificial training data. For learning manipulation based on orthographic images, these transformations (flipping or rotating) however might change the meaning of the label. Data augmentation is therefore discussed in the task-specific sections (chapter 5 and following).

3.7 Selection Policy

Value-based methods commonly estimate the action-value $Q(s, a)$ for a discrete set of actions $\mathcal{A}' \subset \mathcal{A}$. The RL agent then selects a final action $a^\star$ according to a *selection method*

σ, resulting in a policy $\pi = \sigma \circ Q(s, a)$ composed of a value-estimation and a selection step. The *greedy* approach with

$$a^\star = \arg\max_a Q(s, a) \tag{3.27}$$

maximizes the expected reward according to Equation 3.1. However, this only holds true when the estimated action-value Q corresponds to the *true* value. As the value estimation is learned, data needs to be collected first by an *inexperienced* agent. This corresponds to the *exploration* phase. When no prior knowledge is available, a uniform random selection strategy can be used to bypass the value estimation. A ε-greedy policy selects

$$\pi(a^\star|a) = \begin{cases} 1 - \varepsilon \text{ if } a^\star = \arg\max_a Q(s, a) \\ \varepsilon/(|A| - 1) \text{ otherwise} \end{cases} \tag{3.28}$$

probabilistically with $\varepsilon \in [0, 1]$. Here, the greedy action is taken in $1 - \varepsilon$ of all cases, otherwise, a uniform randomly chosen action is selected. It corresponds to a greedy approach for $\varepsilon = 0$ and a purely random approach for $\varepsilon = 1$.

For learning manipulation in the context of RL, we introduce two specific selection methods. First, the **Top(N)** method returns one of the N maximal actions uniformly randomly. Let $\max_a^N$ be the set of N actions with the highest following argument.

$$\pi(a^\star|a) = \begin{cases} 1/N \text{ if } a \in \max_a^N Q(s, a) \\ 0 \text{ otherwise} \end{cases} \tag{3.29}$$

This is a stochastic variant of a purely greedy approach. It is used for exploiting and maximizing the expected reward, however with a small stochastic component to avoid repeating the same action a when the state s does not change. It corresponds to the default greedy approach for $N = 1$ and transitions into a random approach for $N = |\mathcal{A}|$.

Second, the **Prob(n)** method selects an action a stochastically with probability $P(a)$ depending on the action-value $Q(s, a)$. In particular, the scalar n exponentiates the normalized action-value according to

$$\pi(a^\star|a) = P(a) \sim \left| \frac{Q(s, a)}{\max_a Q(s, a)} \right|^n, \quad n \in \mathbb{N}_0 \tag{3.30}$$

We make use of this selection method when the reward r is positive and upper bounded. Then, this method corresponds to the greedy approach for $n \to \infty$ and to the random approach for $n = 0$. Both selection methods correspond to the greedy and uniform random approach in their respective limits. In between, both allow a smooth transition for a non-uniform sampling of actions for exploration.

Most manipulation primitives simplify the RL problem to a single action step, after which a reward corresponding to the *task success* is given. The exploration for RL then resembles the task of *active learning*. Here, an agent needs to choose a limited number of samples from a large dataset for learning. Briefly, the objective is to find the next sample that will improve the performance metric of the learning task the most. In active learning, *uncertainty* is an important metric: If an agent such as a NN has a high uncertainty about an action-value estimation, then learning about the exact outcome will improve the task. The introduced and furthermore used *Top(n)* and *Prob(n)* strategies fall short of this introspecting behavior. As uncertainty estimation with NNs is still a challenging problem and ongoing research, we found more intelligent exploration strategies hard to implement and tune in practice. Primarily, the cycle time for evaluating an exploration strategy corresponds to the training time of the policy itself.

Chapter 4

Real-world Training

Learning manipulation in a *self-supervised* manner is an essential aspect of this work. In robotics, self-supervised refers to an autonomous training without constant human supervision and only very limited intervention. As the term only makes sense for *real-world* robotic training, it emphasizes its practical complexity as well as the usual manual effort for human-guided training such as learning from demonstration. In this chapter, we present and compare the difficulties of real-world data generation. Particularly, the training process needs to be able to cope with unforeseen events and be able to recover to an initial training state. For robotic manipulation, *collisions* and resulting force violations are a major challenge for safe and large-scale training. We present an algorithm for online trajectory generation (OTG) that allows to instantaneously react to collisions and other sensor events.

4.1 Generating Training Data

Data-driven approaches shift the main challenge of solving a robotic task to the *data generation*. In comparison, classical approaches focus on analytical model building and optimizing actions therewith. Here, the primary challenges lie in (1) deriving a suitable model from the robot and its environment (that approximates the behavior of interest well while being conceptionally and computationally simple), (2) measuring the corresponding parameter (e.g. masses, inertia matrix, or friction coefficients), and (3) tuning an optimization algorithm to find the best possible action. Most of the first and all of the second step are replaced with data generation instead.

Large-scale Data Generation We want to focus on primitive-based grasping in a bin picking environment. Here *data* generally refers to recorded tuples (s, a, r) of the visual state s, an action a given by pose and primitive type, and a single reward r as a success metric. As learned approaches improve with the amount of data (corresponding to the number of tuples), the method of generating training data should *scale* as easily as possible. In this work, the robot commonly attempts to measure 10 000 to 100 000 cycles. On one hand, robots are perfectly designed for automating repetitive work. On the other hand, this is

trivial only for a fully known and non-changing environment. Machine learning however requires the data distribution of the training and the later application to match. Therefore, the training process needs to cope with the same partially unknown and stochastic environment that is expected during run-time.

State Space First, the state space and the reward need to be *measurable*. Visual observations are used throughout the work and can be easily measured by a camera. By mounting the camera at the robot's flange, the robot can position the camera flexibly before taking images. In comparison to a fixed overhead camera, this comes at the cost of more robot movements and slower data generation. Moreover, an overhead camera would allow to measure videos while the robot is interacting with the environment. This is however not necessarily required by primitive-based manipulation.

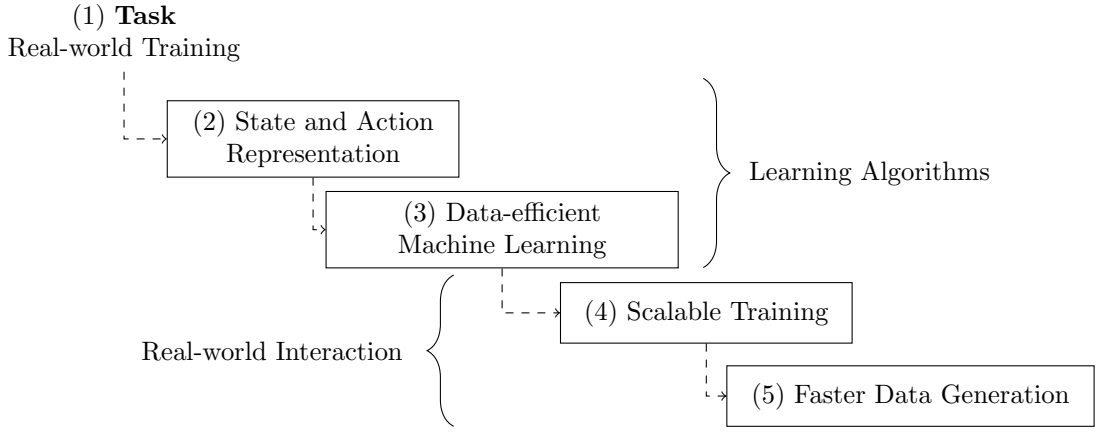


Figure 4.1: To speed-up the real-world training of a given task (1), various decisions or improvements can be made both within the learning algorithm as well as the real-world interaction. First, the state and action representation (2) as well as the underlying machine learning algorithm (3) have great effect on the data-efficiency. On top, the real-world interaction should allow to scale easily for minimal human supervision (4) and high experimental throughput (5).

Reward Measuring the reward is generally more challenging and might have a great effect on the data generation and possible learning algorithms. For the task of bin picking, the robot should learn the *best possible* grasp according to a given metric. However, it can only measure a single grasp for one observation, as it is generally not possible to reset the objects exactly to try out multiple grasps in a fixed scene. Instead, only a binary grasp success can be measured easily, e.g. by distance or force sensors in the gripper's fingers. This directly causes the general approach of *ranking* different actions in this work. For the task of smoothing a garment, the reward might be more complicated. Here, one possible reward consists of the coverage, measured using computer vision algorithms on a segmented overhead image, and an itself learned classifier.

This is in strong contrast to learning in simulation. As all information about the environment is accessible, expressive rewards are easier to define. For grasping, e.g. the torques at the fingertips might indicate grasp points that are far away from the object's center of mass; and could be discounted in the reward definition. For smoothing, a metric could be defined based on the *fully known* geometric state of the garment. While the reward does not need to be measured during real-world application, a simulation-only reward prohibits further fine-tuning.

Scalability Real-world data generation needs to be *repeatable* and *scalable*. First, the environment needs to be resettable to initial states of the task's distribution. This is often combined (and only possible) with randomization to learn from a diverse set of states. For grasping, a selection of objects is placed randomly within a bin. When the robot interacts with the bin, it might unintentionally shift objects or randomly place grasped objects. This results in a novel randomized state for the next cycle. Resetting an environment might be a separate task (e.g. for bringing a smooth garment into a crumpled state), or part of the training process (e.g. when switching between multiple bins for learning to grasp out of differently filled bins).

Second, the data generation needs to be *robust*. In practice, various problems and difficulties might make human interaction necessary; the required scale of the data generation determines *how much* should work in a pure self-supervision manner. We discuss three common challenges related to learning manipulation:

- The robot learns to interact with the environment by *trial and error*. As such, it makes use of its complete action space, although some actions might be undesired and possibly unsafe. While the field of Safe RL deals with this problem conceptionally, oftentimes the actions are restricted manually in practice. For grasping, the robot should grasp within and also near the edges of the bin. However, a binary mask, corresponding to a blacklist of possible grasp poses, might be required to strictly prevent the robot from grasping the bin itself, although the bin might look like a perfectly graspable object.

- During interaction, objects might get dropped outside of the robot's action space. In general, *bins* are used commonly to prevent this to some degree. Still, large objects might fall out of the bin during placement or get dropped while moving between two bins. In this sense, learning grasping is ideal as multiple objects will still be available in the bin. Some tasks like smoothing however cannot use bins and therefore rely on additional functions to reset from out-of-distribution states.

- Collisions occur naturally when interacting with a complex environment, and are oftentimes even required or desirable for the manipulation task. However, a bad collision might not only damage objects or the environment, but also the robot itself and might contribute to its wear and tear. Oftentimes the robot's controller internally monitors collisions and safety-stops the robot until a human unlocks the robot

again. We introduce a motion control algorithm that is able to avoid or weaken these collisions in section 4.2.

Speeding up Data Generation Figure 4.1 shows key aspects for *speeding up* the overall training. Given a task that should be solved with real-world learning, the state and action representations, as well as a data-efficient learning algorithm need to be chosen first. Chapter 3 introduced key ideas to simplify learning for manipulation. In general, the state and action spaces should be *precise* and only include relevant information. In practice, there is a tradeoff for the boundaries of the action space: On one hand, the action space should be kept small for fast exploration as well as simplified safety and robustness. On the other hand, however, it should allow for all required actions to solve the task. Moreover, the exploration strategy of the RL algorithm might be crucial for data efficiency.

For speeding up the data generation itself, one might i.a. refer to the number of robot hours, the invested man hours, or the overall project time as the crucial metric. In this work, we focus primarily on reducing the man hours, all while keeping the robot hours within a fixed time budget. As only a single robot was used in this work at a time, the project time scales equally with the robot hours in practice. For this reason, the training should be with as little human intervention and supervision as possible, e.g. to allow training *overnight*. The need for *self-supervised* learning is mostly founded in a scalable data generation.

Lastly, the data generation can be sped up by increasing the throughput of the manipulation experiments. Usually, the majority of the time is spent on robot motions. Other steps like the manipulation planning, gripper actions, or the manipulation success measurement are relatively fast. Taking the task of bin picking as an example, a typical grasping cycle is around 15 s in our experiments. Here, the longer motions, e.g. between the grasping and a filing bin (and return), as well as slower motions, e.g. to approach a grasp point, are the bottleneck with a duration of over 12 s. To reduce the time spent in these motions, the experiment should first be designed for the shortest paths, e.g. by using relatively compact setups or bins with short distances in between. Second, the robot should move as fast as possible. Usually, the speed is limited by safety issues, in particular regarding collisions with the unstructured environment the robot operates in. To allow for fast yet safe motions, the robot should be able to detect a collision and stop (or replan) as fast as possible to minimize damage at a given speed level. In the next section, we present an algorithm to calculate trajectories that can instantaneously react i.a. to force sensor events. Moreover, it is - in some cases - able to compute time-optimal motions given the kinematic capabilities of the robot and therefore enable fast motions for high experimental throughput.

4.2 Online Trajectory Generation

In robotic manipulation, collision handling is an important aspect of robust training. We use online trajectory generation (OTG) to react instantaneously to collisions and other sensor events, and are therefore able to prevent force violations and safety-stops. This is also an essential problem within the broader field of robotics and automation, as robots

are supposed to manipulate their unknown and non-deterministic environments on-the-fly. Then, a new trajectory needs to be generated in a real-time manner, allowing the robot to adapt the task execution within the scope of its dynamical resources. The trajectory representation is particularly important, as it serves as an interface between the (more abstract) task planning and (lower level) motion planning. A common representation is waypoint-based: The task execution is given as a single or list of waypoints with defined kinematic state. Then, OTG will calculate a trajectory to the new waypoint target considering the robot's constraints. Commonly, second-order (namely velocity and acceleration) constraints take the dynamical resources into account. However, third-order constraints (an additional jerk limit) are desirable to reduce mechanical stress, wear and tear, and the robot's overall cost over lifetime. In fact, we find that modern industrial robots (like the used Franka Emika robot) monitor the jerk in the internal controller and terminate in case of acceleration discontinuities.

Furthermore, we propose a novel algorithm for OTG named *Ruckig* [7] that is more powerful yet significantly simpler than prior approaches. While guaranteeing a solution, Ruckig enables three contributions: First, target waypoints can be defined not only by their position and velocity, but by their complete kinematic state including acceleration. This improves the practical usability of OTG in dynamic tasks. Ruckig is the first algorithm to allow arbitrary target states for jerk-limited trajectory with multiple DoFs. Second, Ruckig is the first freely available time-optimal OTG implementation with constrained jerk. Third, this work introduces directional velocity and acceleration limits. This makes it easier to exploit the full dynamic capabilities of the robot. In the following, we formalize the problem of waypoint-based OTG, derive the proposed algorithm, and share details about the implementation. We evaluate the robustness and real-time performance on a set of randomly generated trajectories. Finally, we show real-world applications highlighting the proposed contributions.

4.2.1 Problem Definition

Let $\vec{x}$ be the state of a kinematic particle with N DoFs. The state $x_i(t)$ of DoF $i \in \{1, \dots, N\}$ at time t is defined by the position p_i and its partial derivatives of up to *third order*

$$x_i = \left(p_i,\, v_i := \frac{\partial p_i}{\partial t},\, a_i := \frac{\partial^2 p_i}{\partial t^2},\, j_i := \frac{\partial^3 p_i}{\partial t^3} \right)$$

named velocity v_i, acceleration a_i, and jerk j_i. We consider the kinematic time-optimality, otherwise the total instead of partial derivatives would be required. Given an initial state $\vec{x}_0$ and a target (final) state $\vec{x}_f$, we seek the time-optimal trajectory $\vec{x}^*(t)$ defined by

$$\vec{x}^*(t) = \arg\min_{\vec{x}(t)} T_f, \quad \vec{x}(0) = \vec{x}_0, \quad \vec{x}(T_f) = \vec{x}_f$$

satisfying the velocity, acceleration and jerk constraints

$$\begin{aligned} v_{i,min} &\leq v_i(t) \leq v_{i,max} \\ a_{i,min} &\leq a_i(t) \leq a_{i,max} \\ j_{i,min} &\leq j_i(t) \leq j_{i,max} \end{aligned}$$

for all times $t \in [0, T_f]$ and DoFs i. T_f is called the trajectory duration. In case no vector notation is given furthermore, we calculate each DoF independently. Note that we consider both the initial as well as target state to be *complete* with possibly all derivatives non-zero. For simplicity, we assume $j_{min} = -j_{max}$ furthermore, but keep the directional acceleration and velocity limits. Moreover, not every kinematic target state is physically possible. We define an upper bound of the allowed target acceleration by

$$a_f \leq \sqrt{2 j_{max} \max\left(|v_{max} - v_f|, |v_{min} - v_f|\right)}, \tag{4.1}$$

because an acceleration target requires a minimum velocity interval to be reached with constrained jerk.

4.2.2 Ruckig Algorithm

We divide the OTG problem into six subsequent steps. Following a short overview, each step is explained in detail in its own subsection.

A. An optional **brake pre-trajectory** is calculated if the initial state x_0 exceeds or will inevitably exceed the kinematic limits v_{min}, v_{max}, a_{max}, or a_{min}. In this case, recovering to a safe kinematic state is the most urgent task.

B. In **Step 1: Extremal times**, all possible profiles that utilize the full dynamic resources of the robot are calculated for each DoF i independently. We call this the set of *valid extremal profiles*. The duration of the fastest profile is called $T_{i,min}$.

C. The target $\vec{x}_f$ should be reached at the same time point T_f by each DoF i. Therefore, some DoFs might need to slow down. In general, not every trajectory duration $T > \max_i(T_{i,min})$ is possible, as there might be a limited number of **blocked intervals** for the duration. We derive these intervals based on the set of valid extremal profiles.

D. The **minimum trajectory duration** T_f is the fastest duration that is not blocked by any DoF. This duration corresponds to a limiting profile as well as a limiting DoF and is included in the set of valid profiles.

E. In **Step 2: Time synchronization**, we calculate a profile for every DoF that reaches its target x_f at the given trajectory duration T_f.

F. Finally, the **new state** at a given time t on the trajectory can be calculated.

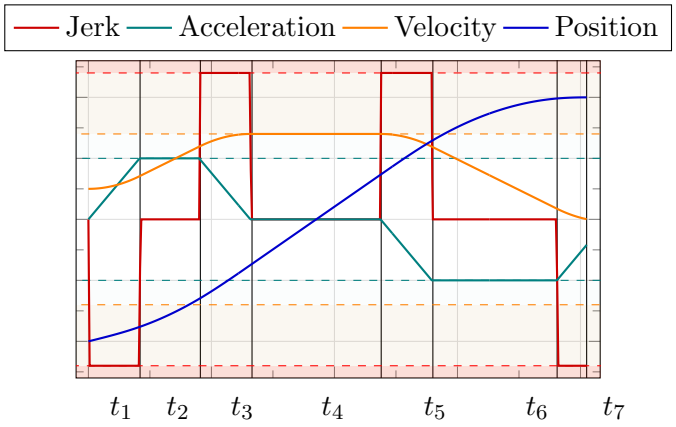


Figure 4.2: A time-optimal profile of a single DoF with initial velocity $v_0 \neq 0$ and target acceleration $a_f \neq 0$. The proposed algorithm is able to generate a time-synchronized trajectory for multiple DoFs with given velocity, acceleration and jerk constraints (dashed) in a limited number of operations (real-time capability).

A time-optimal trajectory will be limited by a single DoF l which uses its entire dynamical resources at all times t. Therefore, this DoF will use a *bang-bang*-like jerk profile with $j_l(t) \in \{-j_{max}, 0, j_{max}\}$. We call such profiles *extremal*. In particular, the total duration T of a profile can be changed by its underlying jerk profile in infinitesimal steps. If and only if the profile is extremal, the duration is bounded on one side. More generally, we formulate each trajectory as a sequence of constant jerk values $j_k = s_k j_f$ with corresponding non-negative time steps $t_k \geq 0$. Let $s_k \in \{-1, 0, 1\}$ be the jerk sign and $j_f > 0$ the jerk value constant throughout the profile. An extremal profile with third-order constraints results in a linear acceleration, quadratic velocity, and in a cubic polynomial for the position. Figure 4.2 shows an illustrative example of an extremal profile with seven steps k.

Brake Pre-trajectory

If the initial state x_0 exceeds or will exceed the acceleration or velocity limits, a so-called pre-trajectory is introduced to brake the system below its respective limits. Due to limited jerk, cases exist that will inevitably brake the velocity constraints at a later point in time, while being below the velocity and acceleration limit right now. Let T_{ib} be the duration of the brake pre-trajectory or zero if none is required. In contrast to the remaining algorithm, this step works in the velocity domain ignoring any position values. We introduce a decision tree (Figure 4.3) depending on a_0, v_0, a_{max}, a_{min}, v_{max}, and v_{min} that calculates the fastest profile to reach the limits. It can be seen by distinction of cases that a resulting profile includes up to two time-steps t_{ib0} and t_{ib1} with corresponding jerk $j_{ib0} \in \{-j_{max}, j_{max}\}$ and $j_{ib1} = 0$. The second step with zero jerk might be necessary, as no *new* constraints should get broken.

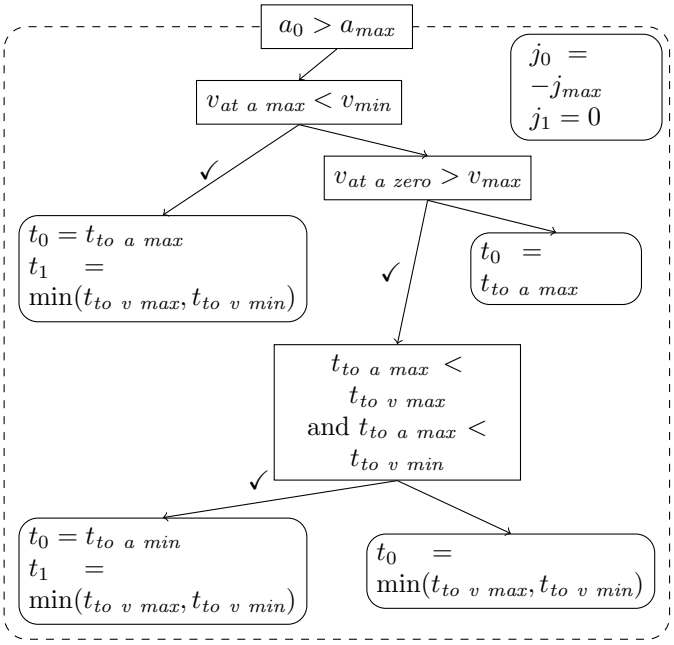


Figure 4.3: A part of the decision tree (given $a_0 > a_{max}$) for calculating an optional brake pre-trajectory. If required, we calculate the time-optimal profile to transfer the system to a safe kinematic state. The pre-trajectory is determined by up to two steps of constant jerk j_0 and j_1 and their respective duration t_0 and t_1.

Step 1: Extremal Times

We want to find all extremal profiles that reach the target state x_f for all DoFs independently.

Lemma 4.2.1. *A velocity limit might only be reached once in an extremal profile.*

At a velocity limit, the profile has zero acceleration and zero jerk. The profile can always be decelerated by reducing the velocity plateau. We show by contradiction: If two velocity limits would be in the same direction, the profile could be accelerated by removing the intermediate deceleration and extending the maximal velocity. If the velocity limits were in opposite directions, the profile could be accelerated by removing the distance traveled from the shorter direction from the other one. As the duration can be shortened and extended, it cannot be an extremal profile.

Lemma 4.2.2. *There are only up to two acceleration limits in an extremal profile.*

Otherwise, a third acceleration peak exists resulting in one direction being reached at least two times. Then, the profile could be sped up by shifting the acceleration from the later-reached peak to the prior one. The profile could be decelerated with the inverse approach.

Therefore, only up to three limits can be reached in total (as shown in Figure 4.2): First, an acceleration limit called `ACC0`, second a velocity limit called `VEL`, and third an acceleration limit called `ACC1`. Introducing optional steps of constant jerk before, after, and between the limits leads to a maximal number of seven steps k with jerk $j_k = s_k j_f$ and corresponding duration t_k (Table 4.1). We denote the sign of the non-zero jerk s_k as either $\uparrow = +1$ or $\downarrow = -1$. A redundant step is encoded with zero duration $t_k = 0$.

Four non-zero jerk steps result in 16 possible combinations. However, only four unique profiles meet the kinematic constraints and are non-redundant: $\uparrow\downarrow\uparrow$, $\uparrow\downarrow\uparrow\downarrow$, $\downarrow\uparrow\uparrow\downarrow$, and $\downarrow\uparrow\downarrow\uparrow$. As the overall problem is invariant to a sign change in j_{max} and exchanging $v_{min} \leftrightarrow v_{max}$ and $a_{min} \leftrightarrow a_{max}$, the set of distinct jerk profiles can be simplified further to $\uparrow\downarrow\uparrow$ and $\uparrow\downarrow\uparrow\downarrow$ profiles. Then, the first jerk sign corresponds to either the `UP` or `DOWN` direction.

The final list of profile types includes every combination of the above three limits and the final two jerk profile types. Table 4.2 lists all 16 distinct profile types for a single direction. In step 1, only 12 profiles are possible as a $\uparrow\downarrow\uparrow\downarrow$ profile with a positive acceleration after the velocity limit is not valid.

Mathematically, each profile type maps the initial state x_0, the target state x_f, and the given limits

$$\begin{aligned}\mathcal{S}1 : &(p_0, p_f, v_0, v_f, a_0, a_f, v_{max}, a_{max}, a_{min}, j_{max}) \\ &\mapsto (t_1, t_2, t_3, t_4, t_5, t_6, t_7)\end{aligned}$$

to corresponding times t_1 to t_7. Given 3 equations for position, velocity and acceleration and 7 variables, 4 additional conditions need to be introduced. Three conditions are set by their limits or a zero step duration. The final condition is either set to $a_3 = 0$ for constant velocity, $t_5 = 0$ for fusing the centering steps $\uparrow\downarrow\uparrow$ if $t_4 = 0$, or $t_7 = 0$ to reach time-optimality for the $\uparrow\downarrow\uparrow\downarrow$ profile. Note that profiles might have multiple solutions. Most profiles are analytically solvable. For some profiles, however, roots of up to sixth-order polynomials need to be found. Here, we make use of a safe Newton root-finding algorithm: Given an isolated root of a polynomial in an interval, a Newton method ensures quadratic convergence on average. Using a bisection method as a fallback strategy, an upper bound of the number of iterations for a given tolerance can be specified. This is required to ensure real-time capability. The initial interval is found by solving the second derivative of

Table 4.1: Steps of Constant Jerk of an Extremal Profile.

Step k	Jerk Sign s_k	Limit
t_1	$\uparrow$ *or* $\downarrow$	-
t_2	0	`ACC0`
t_3	$\uparrow$ *or* $\downarrow$	-
t_4	0	`VEL`
t_5	$\uparrow$ *or* $\downarrow$	-
t_6	0	`ACC1`
t_7	$\uparrow$ *or* $\downarrow$	-

Table 4.2: Time-Optimal Profile Types for the `UP` Direction.

Step	Jerk Profile	Limits	Condition I	Condition II	Condition III	Condition IV
Step 1 + 2	↑↓↓↑	`ACC0 VEL ACC1`	$a_1 = a_{max}$	$v_3 = v_{max}$	$a_5 = a_{min}$	$a_3 = 0$
		`ACC0 VEL`	$a_1 = a_{max}$	$v_3 = v_{max}$	$t_6 = 0$	$a_3 = 0$
		`VEL ACC1`	$t_2 = 0$	$v_3 = v_{max}$	$a_5 = a_{min}$	$a_3 = 0$
		`VEL`	$t_2 = 0$	$v_3 = v_{max}$	$t_6 = 0$	$a_3 = 0$
		`ACC0 ACC1`	$a_1 = a_{max}$	$t_4 = 0$	$a_5 = a_{min}$	$t_5 = 0$
		`ACC0`	$a_1 = a_{max}$	$t_4 = 0$	$t_6 = 0$	$t_5 = 0$
		`ACC1`	$t_2 = 0$	$t_4 = 0$	$a_5 = a_{min}$	$t_5 = 0$
		`NONE`	$t_2 = 0$	$t_4 = 0$	$t_6 = 0$	$t_5 = 0$
	↑↓↑↓	`ACC0 ACC1`	$a_1 = a_{max}$	$t_4 = 0$	$a_5 = a_{max}$	$t_7 = 0$
		`ACC0`	$a_1 = a_{max}$	$t_4 = 0$	$t_6 = 0$	$t_7 = 0$
		`ACC1`	$t_2 = 0$	$t_4 = 0$	$a_5 = a_{max}$	$t_7 = 0$
		`NONE`	$t_2 = 0$	$t_4 = 0$	$t_6 = 0$	$t_7 = 0$
Step 2	↑↓↑↓	`ACC0 VEL ACC1`	$a_1 = a_{max}$	$v_3 = v_{max}$	$a_5 = a_{max}$	$t_7 = 0$
		`ACC0 VEL`	$a_1 = a_{max}$	$v_3 = v_{max}$	$t_6 = 0$	$t_7 = 0$
		`VEL ACC1`	$t_2 = 0$	$v_3 = v_{max}$	$a_5 = a_{max}$	$t_7 = 0$
		`VEL`	$t_2 = 0$	$v_3 = v_{max}$	$t_6 = 0$	$t_7 = 0$

the sixth-order polynomial analytically and checking if a root exists between two extrema. This step is repeated for the first derivative, leading to intervals with isolated roots for the polynomial itself.

We calculate the numeric times t_k for all 24 possible profile types, as given in Table 4.2 per direction. However, not all solutions are physically reasonable or within the kinematic limits of the system. First, we check that every time step $1 \leq k \leq 7$ is non-negative

$$t_k \ \geq \ 0.$$

Then, we integrate position, velocity, and acceleration

$$a_{k+1} = a_k + s_k j_k t_k, \tag{4.2}$$

$$v_{k+1} = v_k + a_k t_k + \frac{s_k j_k}{2} t_k^2, \tag{4.3}$$

$$p_{k+1} = p_k + v_k t_k + \frac{a_k}{2} t_k^2 + \frac{s_k j_k}{6} t_k^3 \tag{4.4}$$

for each time step. We check the acceleration limits via

$$a_{min} < \{\, a_1, a_3, a_5 \,\} < a_{max}$$

and the velocity limits via

$$\begin{aligned} v_{min} < v_k - \frac{a_k^2}{2 s_k j_k} < v_{max} \quad &\text{if} \quad a_k \cdot a_{k+1} \leq 0, \\ v_{min} < v_k < v_{max} \quad &\text{if} \quad a_k = 0. \end{aligned}$$

If the profile passes all checks, it is added to the set of *valid extremal profiles*. The total duration is given by

$$T = \sum_{k=1}^{7} t_k. \tag{4.5}$$

We find the fastest profile and its duration $T_{i,min}$ for each DoF i easily by comparing.

Blocked Duration Intervals

Given the set of valid extremal profiles, we want to find all possible duration $T_i > T_{i,min}$. In general, a number of *blocked intervals* $(T_{1\alpha,start}, T_{1\alpha})$ might exist, in which a DoF cannot reach the target with a duration within the interval. Figure 4.4 illustrates an example of a single blocked interval.

Lemma 4.2.3. *For a third-order target state, up to two blocked intervals might exist.*

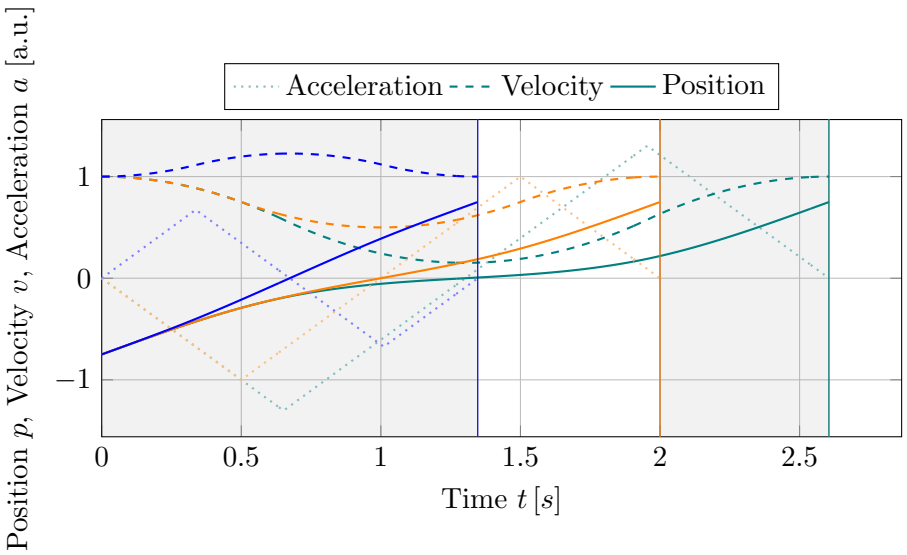


Figure 4.4: Example of a single blocked interval: Given $p_0 = -0.75$, $p_f = 0.75$, and $v_0 = v_f = 1.0$. No trajectory is physically possible with a duration below $t_{min} = 1.35$ *and* between $t \in (2.0, 2.6)$.

Velocity	Acceleration	Max. Number of Blocked Intervals
$v_f = 0$	$a_f = 0$	0
$v_f \neq 0$	$a_f = 0$	1
-	$a_f \neq 0$	**2**

Here, we refer to the work of Kröger and Wahl [68]. In particular, the maximal number of blocked intervals depends on the target velocity and acceleration being non-zero. In our case, we denote the two possible blocked intervals as α and β. Furthermore, we clarify the relationship between blocked intervals and extremal profiles.

Lemma 4.2.4. *A blocked interval is between two extremal profiles and each valid extremal profile corresponds to a blocked interval boundary.*

For all but extremal profiles, the duration can be adapted infinitesimally by changing the jerk j_f or introducing a velocity or acceleration plateau. As the duration is constrained (to one side) for a boundary profile, it must be extremal and vice versa.

Lemma 4.2.5. *A blocked interval can only exist between two neighboring profiles (regarding their duration).*

Otherwise, there would be a valid profile within a blocked interval.

Given up to two intervals, the set of valid extremal profiles must include *exactly* 1, 3, or 5 profiles. Given a sorted list of the profile duration, the profiles are mapped to blocked intervals as follows:

One profile results in no blocked intervals.

Three profiles lead to a single blocked interval α. The interval is between the second and third profiles.

Five profiles correspond to two blocked intervals α and β. The first interval is between the second and third profiles, the second one between the fourth and fifth profiles.

Else a failure of the algorithm would have occurred. In particular, the implementation has to deal with edge cases where different profile types merge.

Minimum Duration

Given the blocked duration intervals for each DoF, we want to find the minimum duration that is possible for *all* DoFs. Figure 4.5 shows an exemplary illustration of this problem.

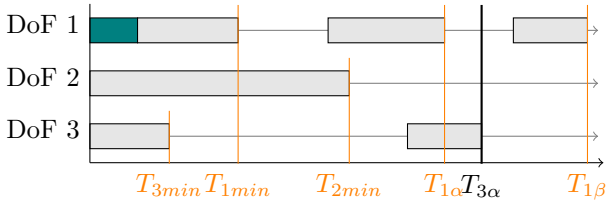


Figure 4.5: Example of finding the minimum non-blocked duration of multiple DoFs. Shown are the blocked intervals (gray), possible minimum duration (orange), possible braking pre-trajectories (blue), and the final minimum duration (black).

The minimum duration T_f needs to be either time-optimal for a single DoF or correspond to the right boundary of a blocked interval:

$$\begin{aligned} T_f \ \in \{\ & \forall i \in \{0, \dots, N\} : \\ & T_{ib} + T_{i,min}, \\ & T_{ib} + T_{i\alpha} \quad \text{if } \alpha \text{ exists }, \\ & T_{ib} + T_{i\beta} \quad \text{if } \beta \text{ exists } \} \end{aligned}$$

The duration of a possible braking pre-trajectory needs to be added. Then, up to $3N$ possible durations are sorted and evaluated in ascending order. The first duration that is not blocked in any DoF is the *final* trajectory duration T_f. The DoF l that corresponds to the resulting duration is called the *limiting* DoF.

Step 2: Time Synchronization

Given the trajectory duration T_f, let $T_{ip} = T_f - T_{ib}$ be the duration of the profile without possible braking. Except for the limiting DoF l (where we can reuse the calculated profile

from the prior step), we need to find trajectories of corresponding duration T_{ip}. Therefore, step 2 maps the duration T_f, the initial state x_0, the target state x_f and the given limits

$$\begin{aligned}\mathcal{S}2 : (T_p, p_0, p_f, v_0, v_f, a_0, a_f, v_{max}, a_{max}, a_{min}, j_{max}) \\ \mapsto (t_1, t_2, t_3, t_4, t_5, t_6, t_7, j_f)\end{aligned}$$

to corresponding times t_k and the final jerk constant j_f. In comparison the extremal profiles, we adapt the duration by changing the velocity plateau `VEL` to v_{plat} with $v_{min} < v_{plat} < v_{max}$ or by reducing the jerk $|j_f| < j_{max}$. With a velocity plateau below its limit, the ↑↓↑↓ profile gets possible for all profile types with `VEL` limit, leading to the full 16 possible profiles for each direction (Table 4.2). Here, we check all 32 profiles similarly to step 1, but return after the first valid profile is found.

New State

So far, *Ruckig* has calculated the step duration t_k and corresponding jerk signs s_k, the jerk value j_f, and a possible two-step brake pre-trajectory t_{ib} and j_b for each DoF i. For each step $k \leq 7$, we integrate the acceleration a_{k+1}, velocity v_{k+1} and position p_{k+1} of the final profile according to Equation 4.2, Equation 4.3, and Equation 4.4. Then, we can calculate the state at a given time t by finding the last index s that fulfills

$$\sum_{i=1}^{s} t_i \leq t$$

and integrating from index s for time $\Delta t = t - t_s$ starting from the kinematic state p_s, v_s, and a_s. If a brake trajectory exists, we apply the same principle to this pre-trajectory. Usually, only this final integration of the new state will be repeated every control cycle of the robotic system. If and only if the input parameters change, the whole trajectory needs to be recalculated.

4.2.3 Experimental Results

Ruckig is implemented as a C++17 library without further dependencies and open-sourced under the permissive MIT license[1]. Symbolic equations were solved ahead of time using *Wolfram Mathematica*; the corresponding notebooks are included in the repository. The generated equations were exported as C/C++. To keep the implementation simple, we preferred to export a polynomial whose roots correspond to the profile solution. This form is then solved by our own C++ polynomial root solver. A non-real-time Python wrapper using *pybind11* is available. Moreover, we've implemented a velocity-control interface that calculates time-optimal trajectories ignoring the current position, the target position, and velocity limits. Due to its simplicity, we focus only on the complete position-control interface in this paper.

[1] Available at `https://github.com/pantor/ruckig(accessedonDecember9th,2022)` (accessed on December 9th, 2022)

Robustness

To evaluate the robustness and numerical stability of the proposed algorithm, we generated a test suite of over 1 000 000 000 random trajectories with up to 7 DoFs each. The input parameters are drawn from

$$\begin{aligned} p_0,\, p_f &\sim \mathcal{N}(\sigma = 4.0) \\ v_0,\, v_f,\, a_0,\, a_f &\sim \mathcal{N}(\sigma = 0.8) \\ v_{max},\, a_{max},\, j_{max} &\sim \mathcal{G}(2.0, 2.0) + 0.05 \end{aligned}$$

with the Normal distribution $\mathcal{N}$, the gamma distribution $\mathcal{G}$ and a minimum limit of 5×10^{-2}. We skip cases that violate the target acceleration requirement (Equation 4.1). Then, we define a successful calculation if the maximal deviations Δ between the result and the target state of

$$|\Delta_p| < 10^{-8}, \quad |\Delta_v| < 10^{-8}, \quad |\Delta_a| < 10^{-12}$$

are met. Here, we achieve a robustness of 100 %. However, Ruckig is quite sensitive to long trajectory duration. During integration, the numerical error will propagate with $\Delta_p = T^2 \Delta_a$. With the precision of a double type for a and the required position accuracy of p, this results in a maximal trajectory duration of

$$T = \sqrt{\frac{\Delta_p}{\Delta_a}} \approx \sqrt{\frac{1 \times 10^{-8}}{2 \times 10^{-16}}} = 7.1 \times 10^3$$

that fulfills the above numerical error. In SI units, this corresponds to an upper limit of around 16 min. If cases above this duration threshold are ignored, Ruckig achieves a robustness of 100 % even without any minimum limit value. Note that the input parameters are invariant to the unit of distance, so the input can also be scaled without loss of generality.

We compare the duration of trajectories generated by Ruckig and Reflexxes Type IV for the above input distribution with $a_f = 0$. We find that both durations of every trajectory within our test suite are within a numeric deviation of $|\Delta_t| < 10^{-6}$, supporting the claim of time-optimality of each other.

Calculation Duration

As an online trajectory generator, Ruckig is real-time critical and *must* output the next state within one control cycle of the robot. Typical control cycles range between 0.5 ms and 5 ms. The following measurements were done on an Intel i7-8700K CPU 3.70GHz 6-core CPU using a single thread with PREEMPT-Linux. The test suite for benchmarking reuses the above input distribution.

Table 4.3 shows the mean and worst calculation performance for a robotic system with 7 DoFs. Figure 4.6 shows the calculation duration depending on the number of DoFs.

Table 4.3: Calculation Performance for 7 DoFs.

		Mean [μs]	Worst [μs]
Ruckig (ours)	($a_f \neq 0$)	19.8 ± 0.2	123 ± 13
Reflexxes Type IV	($a_f = 0$)	38.4 ± 0.4	155 ± 35
opt_control	($a_f \neq 0$)	727 ± 7	3203 ± 504

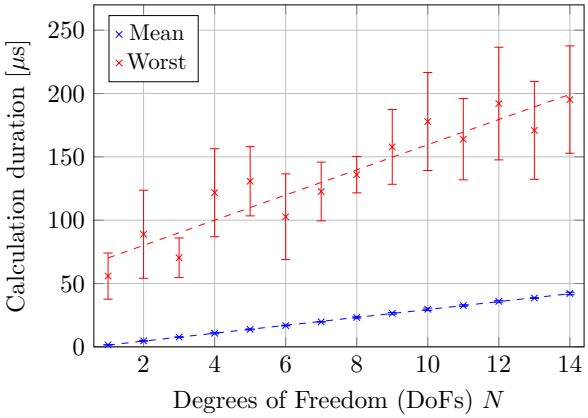


Figure 4.6: Calculation performance depending on the number of degrees of freedom (DoFs).

As expected, we find a near linear relationship $O(N)$ between the average performance and the number of DoFs. Furthermore, we find that *Ruckig* is well suited for control cycles as low as half a millisecond. In fact, $N = 82$ is the smallest number of DoFs that misses the control cycle of 1 ms in the worst case on our hardware.

Real-world Experiments

We have integrated *Ruckig* into our controller library *frankx* for the Franka Emika robot arm with 7 DoFs.[2]. As the Franka robot checks for acceleration discontinuities in its real-time control, a constrained jerk is a hard requirement for *frankx*. The robot has a control cycle time of 1 ms.

Ruckig is agnostic towards the used parametrization; commonly either the joint space or the Cartesian (task) space formulations are used. Furthermore, we use Cartesian space control with three translational, three rotational DoFs, and a single elbow parameter. Furthermore, we highlight two possible applications.

Online Reaction to Sensor Input A robot requires OTG to react to unforeseen sensor input. Figure 4.7 shows a concrete example within the field of HRC. Here, the position

[2]Available under the LGPL-license at `https://github.com/pantor/frankx(accessedonDecember9th, 2022)` (accessed on 2022/12/07) and allows for high-level motion generation (see section 4.4).

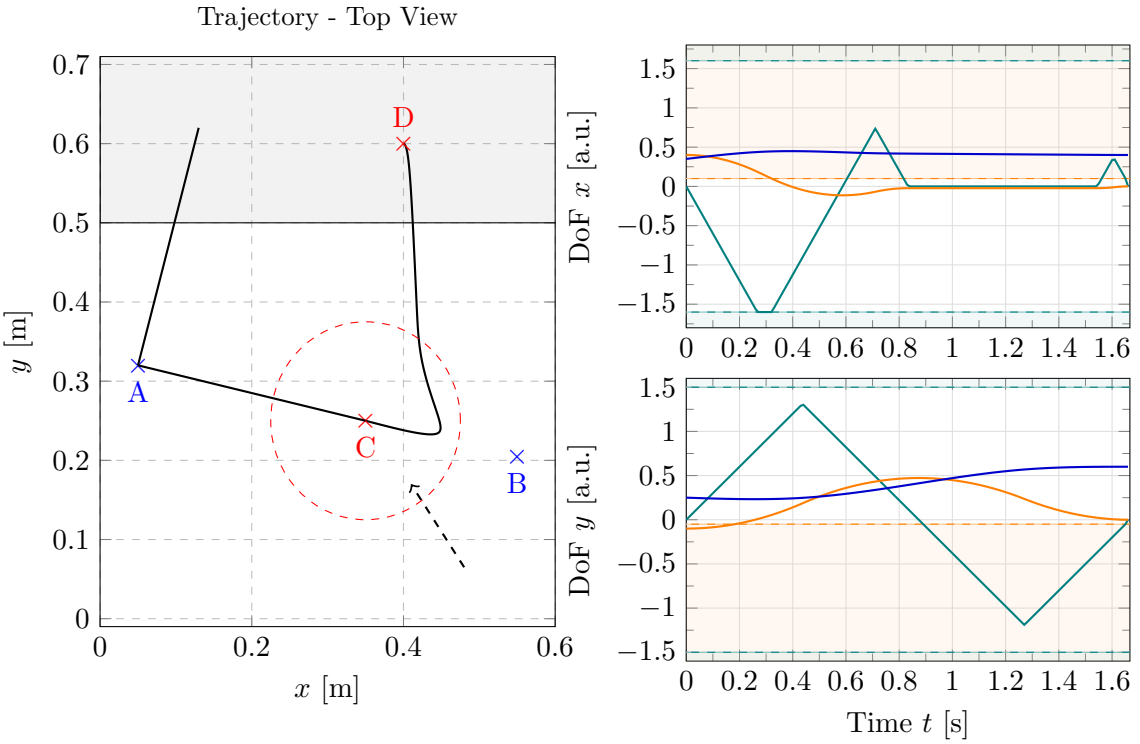


Figure 4.7: An example application in the field of human robot collaboration (HRC): The robot pick-and-places an object from (A) to (B). However, a worker accidentally enters the range of the robot (dashed) and triggers a safety violation at (C). Ruckig calculates a time-optimal trajectory within one control cycle to a pre-defined state (D) within a safe zone (gray). Notably, the velocity limits towards the human are near zero (due to safety) and much larger in the opposed direction (limited by the robot dynamics). This results in the desired behavior that the robot first brakes the velocity as fast as possible and then moves to the safe zone.

of a human worker is monitored. After a detected safety violation, the velocity towards the human should be reduced to a minimum to weaken possible collisions. Moreover, the robot should move away from the human as fast as possible to avoid bruises or potentially dangerous contact. *Ruckig* offers a high-level interface for this scenario: The robot moves as fast as possible to a safe target position with zero velocity, limited by a near-zero velocity v_{max} towards the human, and a robot-limited velocity v_{min} in the opposite direction.

Offline Trajectory Planning Ruckig allows for trajectory planning by following a list of successive waypoints. In particular, waypoints with zero target velocity and acceleration correspond the a piecewise path without blending. Given the waypoints in Table 4.4, we highlight the use of a non-zero acceleration target in a dynamic task: An object (without any jerk constraint) should be accelerated as fast as possible by a jerk-limited robot. Therefore, no impact between the robot and the object should occur, as this would lead to

Table 4.4: Example waypoints for offline trajectory planning.

	Position $[m]$	Velocity $[m/s]$	Acceleration $[m/s^2]$
0	0	0	0
1	0	0	1.2
2	0.68	1.0	0
3	0	0	0

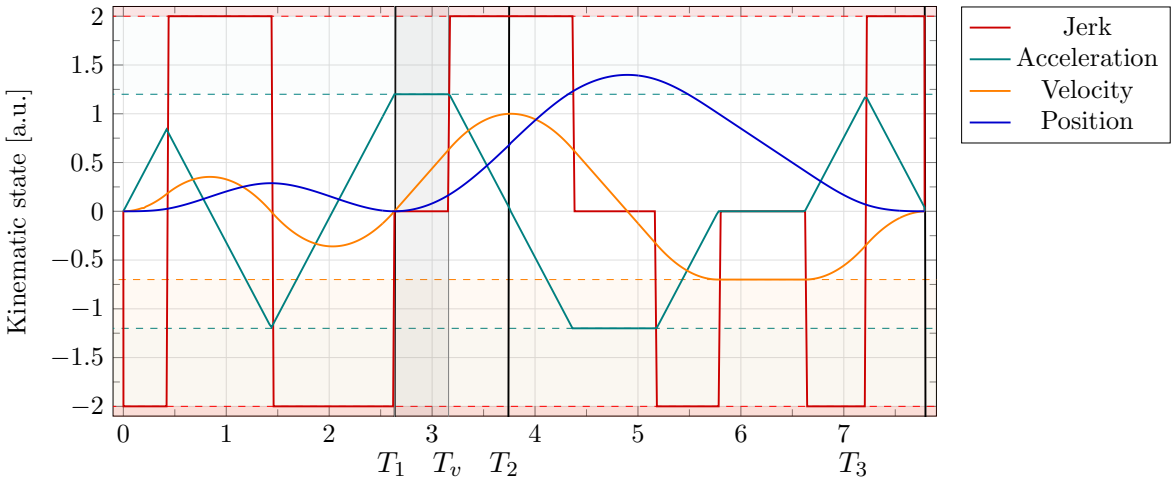


Figure 4.8: An example of offline planning of the following task: An object placed at $p = 0$ should be accelerated to a target velocity (without jerk constraints) in the shortest time possible by a robot (with jerk constraint). The robot starts from rest (T_0). As no impact should occur (otherwise leading to an acceleration violation), the contact between object and robot (gray) should happen at zero velocity but with maximum acceleration (T_1). After the object reaches its velocity (T_v), the robot brakes and reaches its maximum velocity (T_2). Then, the robot moves back to its origin (T_3) with reduced velocity.

an acceleration violation. Then, the robot should only be in contact with the object at its maximum acceleration. After reaching the target velocity of the object, the robot smoothly decelerates and moves back to its initial position. Figure 4.8 illustrates the resulting trajectory.

4.2.4 Discussion

In comparison to related work, *Ruckig* expands the capabilities of the proprietary Reflexxes Type IV library [67]. Reflexxes uses *decision trees* to find matching profiles as well as the blocked intervals, and calculates the numerical profile afterwards. With over 4760 unique nodes, these decision trees cause great complexity [68]. In contrast, Ruckig calculates all valid extremal profiles first and derives matching profiles and blocked intervals afterwards. The proposed algorithm is significantly simpler without decision trees: The relevant code-

base of Ruckig has around 2800 lines of code in comparison to 23 000 lines for Reflexxes Type IV. To our surprise, we find that the mean calculation performance of Ruckig is around twice as fast as Reflexxes. This is probably due to implementational details and better optimizations. In comparison to opt_control [12], Ruckig is able to handle blocked intervals for time synchronization and therefore *guarantees* a solution for arbitrary input states. Moreover, Ruckig is an order of magnitude faster and real-time capable. In contrast to other related work within the field of OTG, Ruckig supports a complete initial and target state [20, 51], time-optimality [3, 122], and multiple DoFs [79].

4.3 Hardware Setup

We briefly present the robotic hardware used throughout this work (except in chapter 8). The general setup consists of: A robot arm, a gripper, a depth camera, two bins with a variety of objects, and a computer system for processing and control.

Robot

A Franka Emika robot was mounted onto a table (Figure 4.9). The robot arm has 7 DoFs, a load of 3 kg, and a maximum reach of 85 cm. A key feature of the Franka robot is its integrated torque sensors that can be used i.a. for (1) changing the robot configuration by hand, (2) impedance or other force control, and (3) force supervision regarding safety regulations. The internal controller makes use of the force control in two ways: First, the controller appends an impedance control behind the regular robot control strategy, and second, brakes the robot when a fixed force threshold is violated. This way, the robot lessens the impact of possible collisions. This allows for a hands-on approach to real-world manipulation training, as the internal controller takes care of the *overall* safety of the robot and its environment. However, it requires a manual safety stop release afterwards.

Figure 4.9: The overall setup of the Franka Emika Panda robot arm (1), with an eye-in-hand mounted depth or RGBD camera (2), the default two-finger gripper (3), custom 3d-printed gripper fingers (4), and two bins with a variety of objects (5).

The default Franka hand is an electric two-finger gripper (Figure 4.9-(3)). It has a stroke between 0 and 87 mm, and includes a current sensor to estimate the gripping force. This way, a variable gripping force of up to 140 N can be applied. The bounding box of the gripper is 20 cm along the gripper opening axis and 5 cm orthogonal thereto. We

designed and 3D-printed custom gripper fingers[3] that allow to reach the ground of the bin. The fingers have a length of 10 cm, leading to a high torque at the attachment point. Additionally, the 3D-printed plastic material experiences wear and tear over large-scale real-world training. Therefore, the finger was robustly designed to withstand 100 000 of manipulation attempts including possible collisions. The fingertip is sloped so that they push objects away while moving down. This improves manipulation in very dense clutter.

Depth Camera

We used three different cameras in this work. While usually only a single camera was used at a time, sometimes two cameras were combined particularly for improved RGBD sensing. All three cameras are stereo cameras and measure depth via triangulation. They support pattern projection in an infrared wavelength which is invisible to the images.

Ensenso N10 The Ensenso is a depth-only camera primarily used in the industry. While the image resolution is only 752×480, it allows a depth resolution of below 1 mm in the working distance of interest.

RealSense D435 The RealSense is an RGBD camera. In comparison to the Ensenso, it produces images with more artifacts and of less quality. While the depth resolution achieves 3 mm after calibration, the images include noise and distortions that need to be accounted for. It is used commonly in robotics research due to its low price point.

Framos D435e The Framos shares the same sensor module as the RealSense D435. The key differences are a more robust housing and an Ethernet instead of a USB connection. We found this connection to be much more reliable and robust during training. Furthermore, we will not differentiate between Framos D435e and the RealSense D435.

Figure 4.10 shows exemplary images of the different cameras. The depth cameras were mounted eye-in-hand at the robot's flange to allow flexible viewpoints.

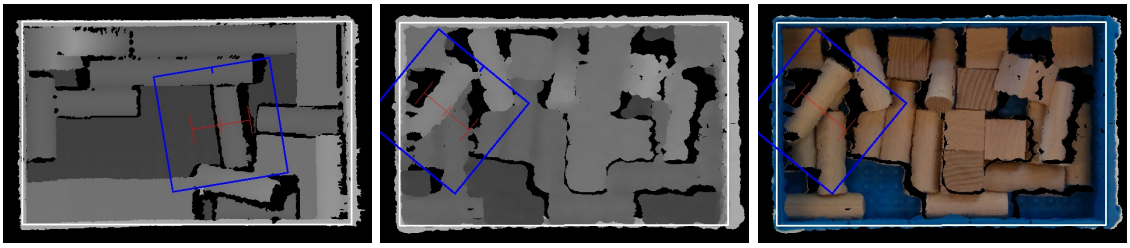
Bin and Objects

The used bins have a size of 15 cm width, 25 cm depth, and 9 cm height. They are either blue or red, and we inlay colored papers for further randomization. The used object sets differ for each task. Details can be found in the task-specific sections from chapter 5 to 8.

Computer System

Calculations were done on a computer with an Intel i7 7800-K @3.7 GHz CPU, 32 GB RAM, and two NVIDIA GTX 1070 Ti. However, both GPUs were used in single-mode for parallel inference (for real-world training with non-random exploration) as well as NN training.

[3]The CAD model of the default gripper finger is available at `https://github.com/pantor/learning-shifting-for-grasping` (accessed on December 9th, 2022)



(a) Ensenso N10 Depth **(b)** Realsense D435 Depth **(c)** Realsense D435 Color

Figure 4.10: Example images of the bin taken by the respective cameras, including the border of the bin (white), a possible planar grasp (red), and the used image window around the gripper (blue).

4.4 Software Setup

The algorithms and their implementations for learning different manipulation tasks presented in chapter 5 and following share the following common dependencies. Our software is either written in C++ (for low-level e.g. real-time capable) parts or Python (for higher-level process control), and are connected via the *pybind11* library. We use *OpenCV* [19] for classical computer vision algorithms and image handling. Our deep learning, NN optimization as well as inference, and data loading are based on *TensorFlow* [2] with its high-level *Keras* interface. *OpenGL* and *EGL* are used for rendering orthographic images from point clouds or projective transformation. Pose and frame definitions are implemented as isometric transformations of the *Eigen* library.

We developed two custom software packages for robot learning: First, a library for interfacing the robot called *Frankx*, and second a database solution tailored towards primitive-based manipulation.

Robot Control

The *libfranka* C++ library is provided by Franka Emika for controlling the robot in a low-level manner. The real-time control loop is exposed for different control strategies such as position, impedance, or torque control. To generate trajectories, an interface to the presented *Ruckig* library was written. This *Frankx* library[4] replaces necessary real-time programming with higher-level motion commands, and allows to react to collisions and other unforeseen events by utilizing OTG. We primarily used the included Python wrapper for controlling the robot via waypoints and reactions.

[4] Available under the LGPL-license at https://github.com/pantor/frankx (accessed on December 9th, 2022)

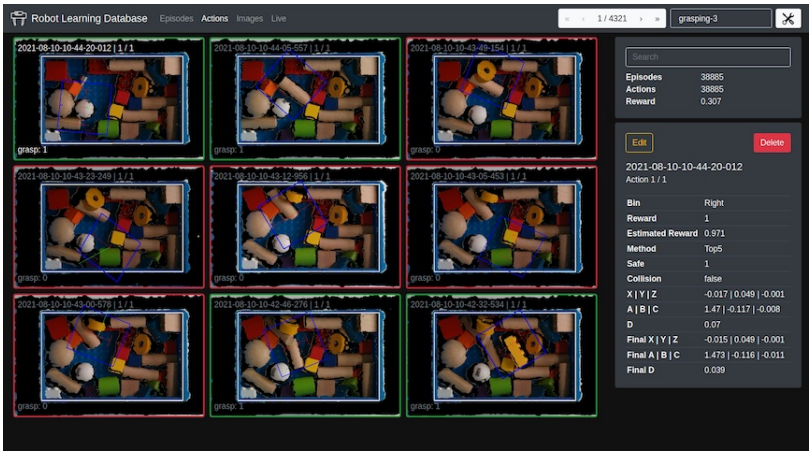


Figure 4.11: Screenshot of the database viewer, a custom web application for data management for primitive-based manipulation.

Data Management

Managing the collected data is an essential task for real-world robot learning. In our terms, each manipulation primitive corresponds to an *action*. Multiple images are recorded in relation to each action, e.g. an image before or after an action with one or multiple cameras. To combine multiple primitives that require a temporal correspondence (e.g. a pick and place action), multiple actions might be combined into an *episode*. Each experiment includes a large number of episodes; they are saved as collections in a *MongoDB* database.

The database is connected to a web server that manages in particular episode insertions. This would allow - although not used - to train on multiple robots in parallel. Moreover, the web server includes a frontend called *Database Viewer* for viewing and editing the database (Figure 4.11). In its default setting, it shows a grid of the most recent actions or episodes with detailed information about the selection action. It supports live viewing of the most recent measurements, as well as statistics about the current collection. The database viewer allows to delete episodes; typically required after an accidental human interaction with the system or after the lights were switched off during nightly training by mistake.

Chapter 5

Grasping

Chapter 3 and 4 introduced robot learning algorithms and data generation techniques for real-world manipulation. In the following chapters, we apply these approaches to practical robotic tasks. *Grasping* is arguably the most essential manipulation task in robotics. Industrial use cases like machine loading, large-scale warehouse automation, or order fulfillment rely on grasping to solve bin picking. Grasping is not only an important task by itself, but also serves as a starting point for further interaction with the environment. For example, many service robotics tasks like opening kitchen drawers, general tool use, as well as industrial tasks like peg-in-hole or cable fixation require the *primary* grasping of objects.

As motivated in chapter 4, bin picking eases learning for grasping in a self-supervised manner. This chapter is divided into multiple sub-tasks related to bin picking. At first, planar grasping with a top-down orientation of the gripper is learned as a starting point (section 5.1). On top, we introduce two methods for 6 DoF grasping. First, by combining a model-based method for the remaining DoFs (section 5.2). Second, by leveraging imitation learning to use the full power of learning methods for all DoFs (section 5.3). Then, *pre-grasping manipulation* is used to improve the expected success of a subsequent grasp. Finally, we extend primitive-based manipulation to *semantic grasping* to grasp a specific object type out of a scene with multiple object types (section 5.5). All introduced algorithms are evaluated in real-world experiments.

5.1 Planar Grasping

Chapter 3 introduced a technique for using FCNNs as a sliding window estimator, in particular for learning a mapping from image data to a task success reward. This approach becomes directly applicable when simplifying bin picking to *planar* grasping. In general, planar manipulation limits the end-effector orientation to be orthogonal to the image plane. This way, the grasp point is described by four DoFs: the translation x and y, the rotation α within the image plane, and the height z (Figure 3.3). Then, the grasp orientation equals the camera orientation, and the gripper fingers are parallel to the viewing direction.

Usually, planar grasping is combined with top-down images of the bin. Then, the grasp orientation is limited to be orthogonal to the world plane ($\beta = \gamma = 0$).

For bin picking, planar grasping is a surprisingly small restriction. For a bin with a large number of randomly filled objects, a good planar grasp is very probable in practice. The bin usually prohibits planar grasps near its edges or corners; planar grasps might not be able to grasp objects here. This becomes noticeable if the bin only contains a few objects at these positions, so that the robot might not be able to empty the bin completely. Our robotic setup uses long fingers, e.g. longer than the bin height, to prevent collisions between the gripper base and the bin and to mitigate these issues.

5.1.1 Primitives

To apply the concept of primitive-based manipulation (subsection 3.3.3) to grasping, we define a general grasp primitive as the following sequence:

1. The robot moves the gripper to the *approach* pose, and its gripper to the defined pre-shaped gripper width d_m by the primitive m. This trajectory is planned by a collision-aware motion planner.

2. The robot approaches the given grasp pose parallel to its gripper finger (along the local z-axis). If a collision is detected by a force sensor, the robot stops and retracts a mm (so that it avoids being in contact).

3. The robot then closes the gripper and detects a possible object using the force-feedback and width sensor within the gripper. Then, let the current pose be the *final pose* with the *final gripper stroke*.

4. If an object is detected, the robot lifts the object and moves above the (possibly different) filing bin. Then, the final *grasp success* is measured again.

This way, an entire grasping action is parametrized by the pose (x, y, z, α) and the pre-shaped gripper width d_m. To further simplify planar grasping, we restrict possible gripper widths d_m to a small set of commonly four discrete values. The widths should span the gripper's width and be uniformly distributed. We interpret these as four different primitives $\mathcal{M} = \{d_1, d_2, d_3, d_4\}$ to keep them only dependent on the pose.

Furthermore, we calculate the height z directly from the depth image using classical computer vision. Briefly, the height at the grasp point (x, y) is read and transformed into a global frame with a fixed offset for the gripper. The idea is that a pre-programmed method for calculating the height z provides successful values for (at least) a wide range of scenarios, and the policy then learns the reward estimation and all other parameters with this in mind. The reward of a grasping primitive is given by

$$r = \begin{cases} 1 & \text{if} \quad \textit{grasp until filing successful,} \\ 0 & \text{else} \end{cases} \tag{5.1}$$

Although the given reward r is binary, predicting the grasping probability is not a classification but a regression task: The grasping success is intrinsically probabilistic, mostly because the MDP is only partially observable and the state s based on the visual observations does not have full knowledge about the scene. Eventually, the grasping policy needs to learn four parameters (x, y, α, m) from the visual input.

5.1.2 Model Architecture

Let s be a top-down image of the bin, and $\hat{s}$ be a cropped window at a known pose (x, y, α). To implement a sliding-window approach, we define the action-value-function $Q_m(\hat{s}, a)$ per primitive m as mapping the image window $\hat{s}$ to the binary grasp reward r as defined by Equation 5.1. In particular, all relevant information about the pose is contained in the cropped image window $\hat{s}$. The action-value function is approximated by an FCNN called ψ.

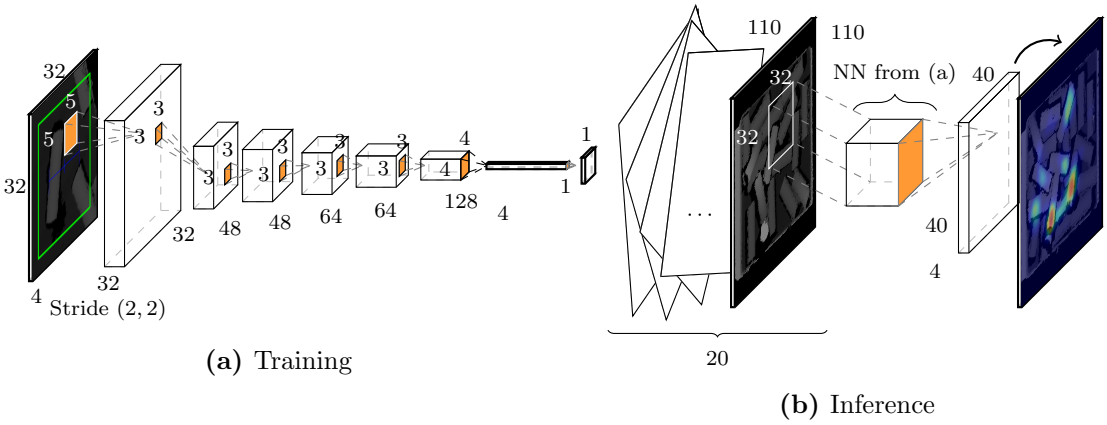


Figure 5.1: (a) Our neural network (NN) architecture. The training output has a size of $(1 \times 1 \times 4)$, corresponding to three gripper distances d. (b) During inference, the NN calculates from the overview image an output of size $(40 \times 40 \times 3)$, corresponding to the (x, y, d) parameters. For the rotation a, the input image is pre-transformed and the NN is recalculated for 20 angles. A single output can be interpreted as a grasp probability heatmap.

Training Figure 5.1 shows the architecture of the NN. During training, the NN takes a cropped window of size $32 \times 32 \times 4$ pixels as input. The last dimension corresponds to the four RGBD channels of the image. For depth or RGB images, a smaller number of input channels can be used without loss of generality. The NN outputs four scalar predictions $(\psi_1, \psi_2, \psi_3, \psi_4)$ for each manipulation primitive and corresponding gripper width. The output is of size $1 \times 1 \times 4$, as the NN should predict reward predictions for all manipulation primitives at *every* pose. The number of manipulation primitives is hence implemented within the *depth* channel of the final layer. While the exact NN implementation might differ within the experiments, Figure 5.1 shows details such as the number of kernels for a

typical architecture. In general, we use a stride of $(2, 2)$ in the first layer to half the feature size. Then, we continue to shrink the feature size with kernel sizes of $(3, 3)$, reducing the feature size by $(2, 2)$ per layer. Simultaneously, we increase the number of kernels over the layer depth from 32 up to 128. We use a leaky ReLU for the activation function and batch normalization in every hidden layer. We apply standard dropout with a rate of 0.3 and a l_2 regularization with a low 0.01 weight. A final sigmoid function maps the NN output to binary reward predictions $\in [0, 1]$. A typical NN has around 200 000 learnable parameters. Although used commonly in image classification, we do not make use of pooling layers. Manipulation requires exact spatial information which would get lost during pooling. The NN is trained to minimize the loss

$$\mathcal{L} = -\frac{1}{|\mathcal{M}|} \sum_{m=1}^{|\mathcal{M}|} \delta_{mn} \left(r_n \log \psi_m + (1 - r_n) \log(1 - \psi_m) \right), \tag{5.2}$$

the binary cross entropy between the predicted reward and the measured value for a specified manipulation primitive m (gripper width).

Inference During inference, the *same* NN takes an image of the complete scene as input. The output size X' scales with the input size X according to

$$X' = (X - 32)/2 + 1 \tag{5.3}$$

due to the stride of 2 in the first layer. A typical input size of $(110 \times 110 \times 4)$ is then mapped to $(40 \times 40 \times 4)$ reward predictions. These correspond to predictions at a *grid* of poses over (x, y). In particular, the index position (i, j) within this grid is sufficient to calculate the position (x, y) of the reward prediction, and the calibrated orthographic image allows to compute the real-world position. The angle α is incorporated by rotating the input image first, usually with 20 uniformly distributed orientations. This leads to a number of 32 000 different grasp poses with four different primitives each, resulting in a total action space size of $|\mathcal{A}| = 128\,000$ actions. The final output tensor has a size $(20, 40, 40, 4)$

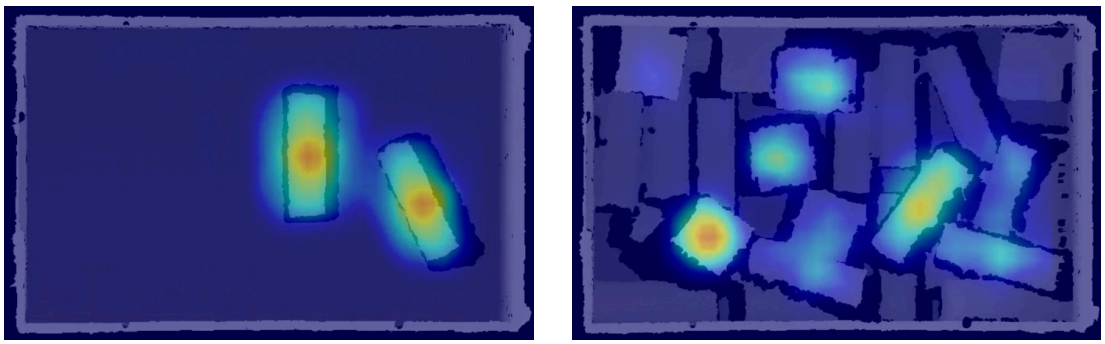


Figure 5.2: Example heatmaps for planar grasping. The estimated grasp reward, averaged over all rotations α and gripper widths d, is shown ranging from 0 (blue) to 1 (red).

reward predictions corresponding to (α, x, y, m). This can be interpreted as a heatmap over the spatial dimensions (x, y). Figure 5.2 shows exemplary heatmaps that are averaged over rotations α and primitives m. Typical resolutions are approximately 4 mm in the x-y-plane, 15 mm for the gripper distance d_m, and less than 9 ° for the rotation α. For binary rewards such as r, the estimated grasping action-value function $Q(s, a)$ is interpreted as a grasp success probability ψ.

5.1.3 Training Process

Chapter 4 motivated and derived techniques for self-supervised learning. We apply them for scaling the real-world training process to a large and diverse dataset. In particular, *two* bins are used for continuous and diverse training: The robot learns to grasp objects out of one bin and drops them into the second bin. This way, the fill level of the bin varies and the objects do not repeat that closely over time. If a bin is empty or a number of grasp attempts fail, the robot switches the bins and starts grasping from the then-filled bin. If not otherwise stated, we use a threshold of 12 consecutive failed grasp attempts. Usually, a human changes the objects or the bin once every 10 h. We apply a simple curriculum learning strategy, starting with a few objects and increasing the number and complexity of object types over time.

Random grasps result in a success rate of only 2 % to 3 %. However, successful grasp actions contain more valuable information, as (1) they should be inferred during the application phase, and (2) are harder to collect. This motivates a systematic approach to exploration to use already learned knowledge to improve the learning itself. This exploration is implemented by the selection function σ that chooses the final action from the tensor of all grasp reward predictions. The general idea of the exploration is to prefer grasps that have a higher chance of success, all while keeping a given (over time decreasing) level of randomness. Besides uniform sampling for random grasps, we primarily make use of the *Prob(1)* and *Prob(2)* methods (section 3.7). We avoid a purely greedy strategy during training to avoid repeating the same, potentially failing grasping action. Furthermore, switching is implemented *softly* by a probability threshold of choosing a first strategy over a second one. During the application phase, we use the greedy policy $\sigma = \arg\max$ named *Top(1)*. If a grasp fails, we switch to the *Top(10)* selection.

In parallel to the real-world data generation, our system of two GPUs is able to train the NN from the most current dataset. The NN training takes in the order of 10 min. This guarantees that grasp reward predictions are always up-to-date, which improves the data efficiency of the training procedure. Two further effects need to be considered: Firstly, an average reward of around 0.5 for the overall dataset is preferred for a normalized reward output, otherwise, the NN overfits to the specific, skewed distribution. Secondly, both grasping error types are asymmetric. A false negative error is a missed chance for grasping, while there are typically a multitude of positive grasping poses. However, false positive errors correspond to real grasping errors, which should be minimized with higher priority.

(a) Simple Objects **(b)** Unknown Objects

Figure 5.3: Object sets for evaluating planar grasping.

Both effects are implemented as two successive, reward-depending weights in the calculation of the cross entropy loss.

To increase the quality of grasps, the clamping force f of the gripper is reduced during training in comparison to application time. This minimizes grasping torques during training and draws grasps towards the object's center of mass. For improved evaluation and easier debugging, we use a deterministic split into training and test sets. As new measurements are added, a hash function maps the grasp timestamp to a number $\in [0, 1]$. If this value is less than 0.2, the grasp is assigned to the test set. The exact set sizes are therefore not predetermined. The NN is trained by the Adam optimizer with a learning rate of 10^{-4}.

5.1.4 Experimental Results

We use the general experimental setup specified in section 4.3 with the Realsense D435 camera only. Moreover, the experiments make use of three object sets: (1) simple wooden primitives for training, (2) M10-80 screws for challenging reflective surfaces, and (3) a variety of more than 50 *unknown* objects for testing only (Figure 5.3).

Dataset

We train two separate datasets furthermore: (1) A main set for training on simple objects as well as the evaluation of unknown objects, and (2) a dataset for screws specifically. The main dataset includes 25 000 grasp attempts, recorded in 100 h in total. The 20 000 attempts of the screw dataset were taken in 80 h. A grasp attempt takes 15 s on average. Figure 5.4 gives examples of overview RGBD images with corresponding grasp actions for both sets. It also highlights the noisy depth image due to the reflective screws in clutter, causing the major challenge of grasping reflective or transparent objects. Figure 5.6 shows cropped image windows $\hat{s}$ that are used as input for training the NN.

Complete random grasps were conducted for the first 2000 grasp attempts. Then we switched to the *Prob(n=1)* selection method for exploration and increased n to 4 till 15 000 grasp attempts. We then used a ε-greedy exploration until the loss saturated at 25 000

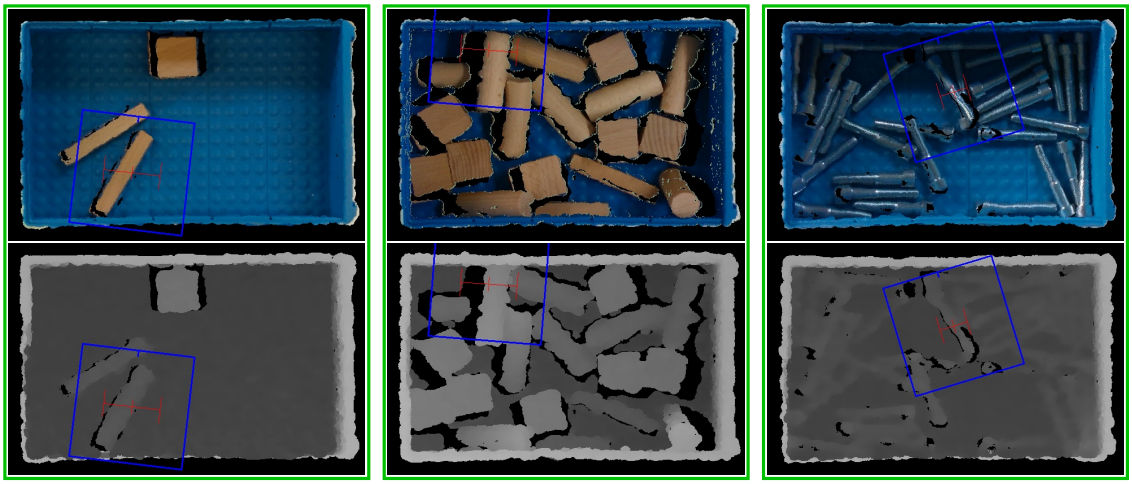


Figure 5.4: Example images of the bin with its border (white) and a successful grasp (red) and the window used for its calculation (blue).

or 20 000 grasp attempts respectively. Figure 5.5 shows this convergence of the binary crossentropy (BCE) loss and the accuracy of the NN depending on the dataset size for training, validated on the evaluation set of the final dataset. For the ε-greedy exploration,

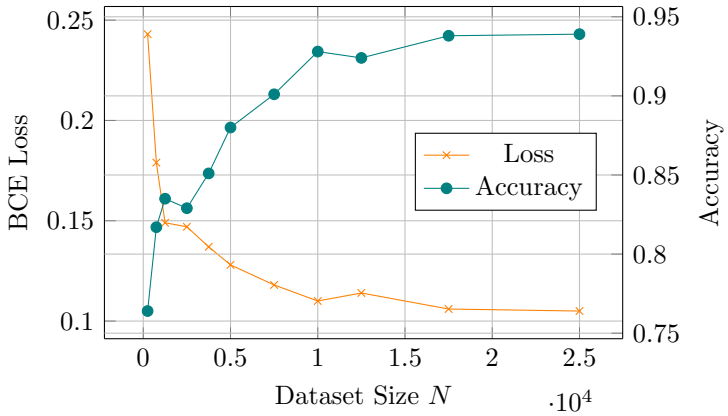


Figure 5.5: The BCE loss and the accuracy of the NN depending on the training progress, corresponding to the training set size N.

a *Prob*(n=2) selection was chosen with probability ε, otherwise, a greedy *Top(25)* action was executed. ε was reduced linearly from 1 to 0.2 after 15 000 grasp attempts until the end of the training. We retrain the NN on the most current dataset repeatedly, on average every 200 grasp attempts depending on the duration of the NN training.

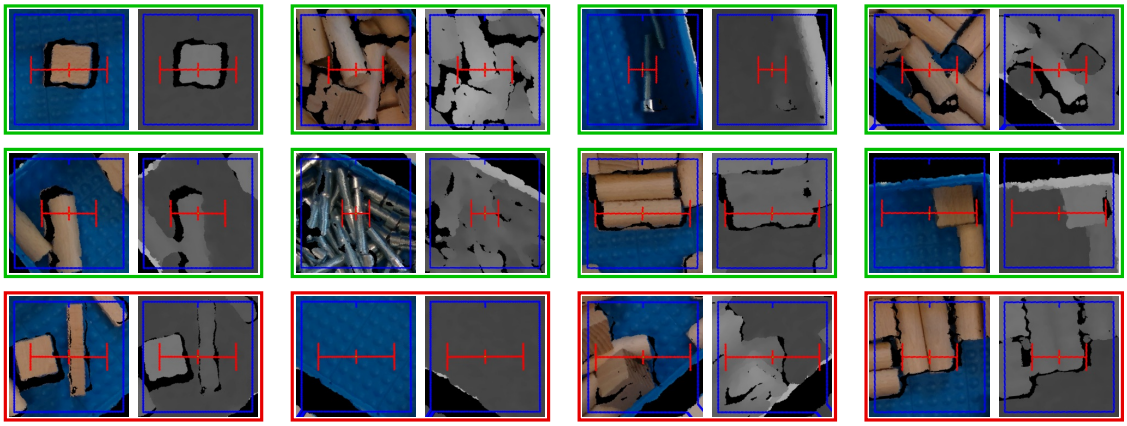


Figure 5.6: Examples of cropped image windows $\hat{s}$ around the tool center point (TCP) (red). The border indicates a grasp success (green) or failure (red).

Grasp Rate

The grasp rate is measured by counting the attempts of grasping n objects without replacement out of a bin filled with a total of m objects. Table 5.1 lists grasp rates for different bin picking scenarios. We differ between the used object set as well as the number of ob-

Table 5.1: Grasp rate for planar grasping in different bin picking scenarios.

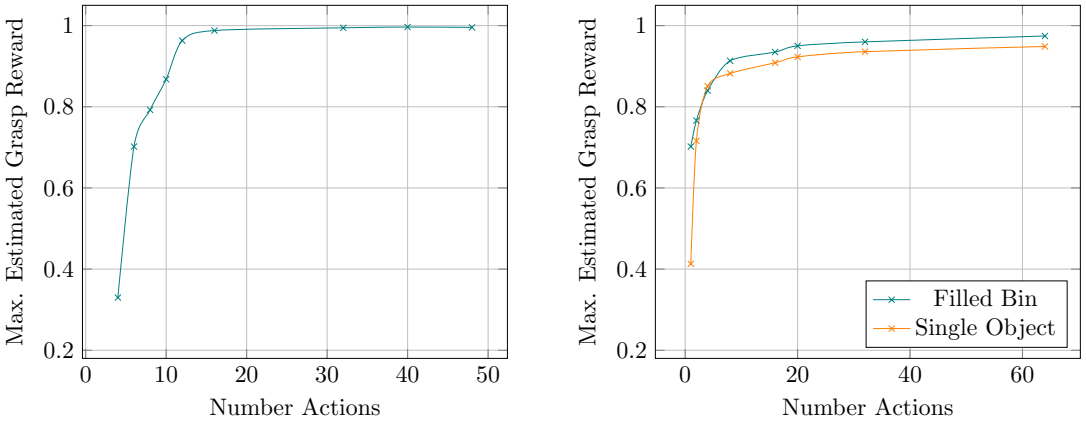
Objects	n out of m	Grasp Attempts	Grasp Rate %
Simple	1 out of 1	213	99.5 ± 0.5
	1 out of 5	205	99.0 ± 0.6
	5 out of 5	183	98.4 ± 0.9
	5 out of 20	219	95.9 ± 1.2
	10 out of 20	240	93.8 ± 1.6
	15 out of 20	192	93.8 ± 2.4
	25 out of 30	216	92.6 ± 1.4
Screws	1 out of 1	114	97.4 ± 1.5
	5 out of 20	135	96.3 ± 1.5
	10 out of 40	156	89.7 ± 2.2
Unknown (simple)	1 out of 1	101	92.1 ± 2.5
	5 out of 10	114	92.1 ± 3.2
Unknown (complex)	1 out of 1	154	68.8 ± 3.8
	5 out of 10	114	61.4 ± 4.8

jects n and m. The same training and screw object set were used during both training and evaluation. However, the *unknown* object set was trained only with the *simple* object set. For typical bin picking scenarios of grasping 5 out of 20 objects, a grasp rate of (95.9 ± 1.2) % for the wooden objects was measured. As expected, the grasp rate improves

with decreasing clutter and increasing choice. For example, clutter might lead to stacked and partially hidden objects, gaps in the depth information, and interlocked objects. The grasp rate decreases for n near m and large m. Still, the system achieves a grasp rate of $(92.6 \pm 1.4)\,\%$ even for $m = 30$ objects inside the bin. For highly reflective screws in dense clutter ($m = 40$), the grasp rate decreases to $(89.7 \pm 2.2)\,\%$.

Despite being trained only on simple wooden primitives, the robot is able to generalize to novel object types never seen during training (Figure 5.3b). We differ between simple and complex unknown objects depending on the similarity of the shape and material to the trained objects. For simple objects, the robot can keep a grasp rate of over 92 % even for clutter. For more difficult objects, however, the grasp rate drops to 68.8 % even for isolated objects. Problems arise with unknown objects larger than the trained objects. As larger objects may look like multiple smaller objects, the system attempts a grasp that would separate possible objects. Hence, the grasping success depends strongly on the similarity with the trained object type.

Another typical grasp failure case is due to missing depth data. In orthographic images, an elevated object creates a "shadow" of missing depth data around it. Additionally, geometric occlusions, reflective objects, or camera noise might increase these regions of missing depth data significantly. Then, unexpected objects or obstacles might be within these regions of missing data that result in a collision and failed grasp. Section 5.3.3 discusses this and other failure cases in more detail.



(a) Translational Resolution **(b)** Rotational Resolution

Figure 5.7: The maximum estimated grasp reward over the translational and rotational resolution.

The output of the FCNN includes 128 000 estimated grasp rewards in a grid-like structure over 20 rotations and 40 translations for x and y each. We further investigate the dependency of the action space resolution on the maximum grasp reward. A low resolution might skip good grasp candidates, resulting in a low maximum grasp reward. A high reso-

lution decreases the computing performance, so that a trade-off for finding *successful* grasps needs to be considered. Figure 5.7 shows the maximum grasp reward for the total number of translations and rotations. We find that the grasp reward saturates at a surprisingly low number of around 15 translations and 20 rotations. This corresponds to a grid resolution of 11 mm and 9 °.

Calculation Duration

We measure an average calculation duration of (23.2 ± 3.1) ms on our control hardware. This includes (1) 7.2 ms for pre-processing, e.g. for rotating and scaling the images, (2) 14.2 ms for the NN inference, and (3) 1.8 ms for selecting and calculating the final grasp action. Figure 5.8 shows the calculation duration over the side length of a (quadratic) bin. We generate (hypothetical) images by scaling real images accordingly. As expected, we find

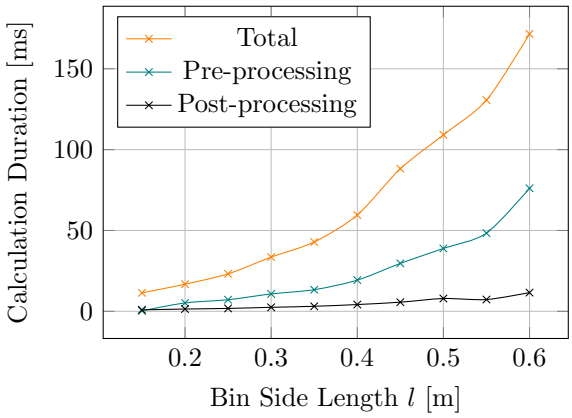


Figure 5.8: The calculation duration depending on the size of the bin with constant resolution.

that the total calculation duration as well as the NN inference scales quadratically with the bin size. For a side length of 0.6 m, the policy estimates the grasp rewards for 1 230 080 actions. Still, the calculation duration stays below 200 ms.

Transfer Learning

Besides generalization to unknown objects, we investigate the *transfer* learning between object types. In particular, the robot trains on a dataset of wooden cylinders only. We then evaluate it on sharp wooden cubes as the novel object type. Figure 5.9 shows the robot's capability of transfer learning based on different training datasets. With either a large dataset of cylinders or a small dataset of cubes, the grasp success rate stays below 72 %. In the first case, it fails to generalize to out-of-distribution objects, in the latter case it overfits on too few data samples. However, combining both datasets results in a grasp rate comparable to prior trainings as in Table 5.1.

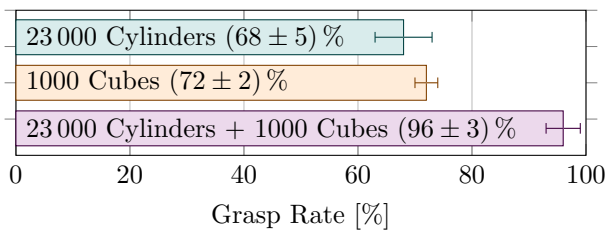


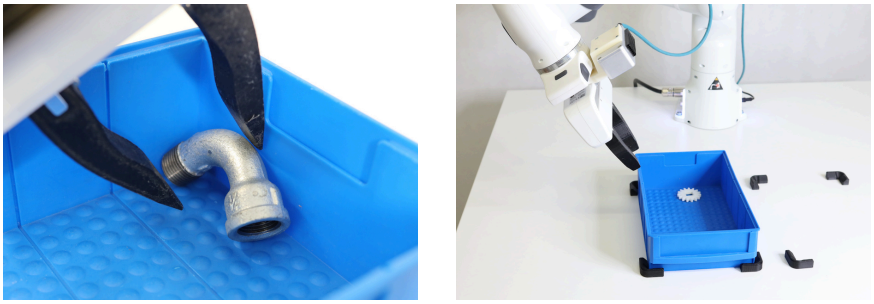
Figure 5.9: The grasp rate of cubes (10 out of 15 objects), depending on the trained object type (label) and their corresponding dataset sizes N.

5.2 Model-based Adaptive Primitives for 6 DoF Grasping

So far, we focused on planar grasping as a simplification towards the general 6 DoF grasping problem. In the following two sections, we want to derive two approaches for 6 DoF grasping that both gently extend the methods introduced in chapter 3. The key idea of the first approach is to use *adaptive primitives*. Here, the manipulation primitives are, besides the planar grasp pose, parametrizable by a small number of additional variables. By using an analytical model for these parameters next to the learned grasp reward estimation, a *hybrid* approach is derived for 6 DoF grasping. This section is based on [10].

5.2.1 Advantages and Challenges of 6 DoF Grasping

6 DoF grasping allows for two primary advantages over planar grasping: First, the additional DoFs of the orientation might allow new grasps that would otherwise be in collision and therefore improve the general grasp quality. When calculating grasps based on top-down images of the bin, this appears mostly for collision avoidance between the gripper and the bin. With 6 DoF grasps, the robot is able to approach a grasp point laterally to e.g. grasp objects near the edges of the bin (Figure 5.10a). Second, 6 DoF grasping allows for more flexible viewpoints. Assuming a flat object on a table, the robot needs to grasp the object in a specific orientation (Figure 5.10b). 6 DoF grasping relaxes the requirement of a top-down image to more practical camera positions, e.g. found in service or humanoid robotics with a camera mounted on a mobile platform.



(a) Collision avoidance **(b)** Lateral viewpoint

Figure 5.10: Two advantages of 6 DoF over planar grasping: First, it allows more flexible grasps and improved collision avoidance (left). Second, it allows lateral points of view instead of top-down images (right), even for flat objects on a table.

However, these advantages are bought dearly by challenges that are avoided intentionally in planar grasping. First and foremost, the additional DoFs reinforce the *curse of dimensionality* that results in an exploding action space. For example, section 5.1 discretized the action space to 128 000 grasp poses. Extending the action space by two more DoFs increases its size exponentially, e.g. by a factor of 400× to 51 200 000 actions when keeping the same resolution for all three rotations. First, this would slow down the infer-

ence calculation to around 16 s. This would not only be impractical for any applications, but would also more than halve the real-world training throughput. Second, the task of 6 DoF grasping yields significantly more diverse input data, and therefore requires more training data for similar performance metrics. Moreover, the extended action space makes exploration much more challenging in the real world. At the beginning of the training, a number of successful grasps need to be observed so that the robot can start learning to classify between grasp successes and failures. Using more complex selection strategies relies on a working yet possible erroneous reward estimation that was kickstarted by random sampling. As grasp failures are very easy to collect, we argue that the initial training time scales with the random grasp success rate. 6 DoF grasping reduces the random grasp rate to around 0.25 % from 2.5 % for planar grasping. To collect 50 grasp successes, random planar grasping requires around 2000 grasp attempts. For 6 DoF grasping, this increases by an order of magnitude to around 20 000 attempts.

Planar grasp primitives were introduced in subsection 5.1.1. For 6 DoF grasping, we allow β and γ to be non-zero. In particular, the grasp pose is defined by a complete 6 DoF frame. A grasping action is parametrized by the pose $(x, y, z, \alpha, \beta, \gamma)$ and the pre-shaped gripper width d_m. We keep four distinct pre-shaped gripper widths which are implemented as separate manipulation primitives m.

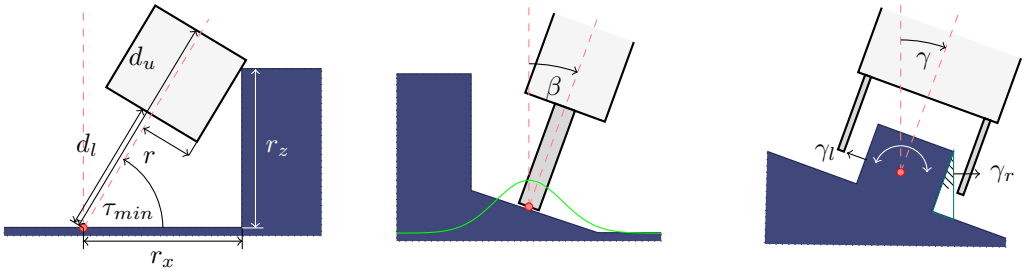
5.2.2 Model-based Orientation

To mitigate the challenges associated with the introduction of the β and γ DoFs, we extend planar grasping by calculating these by a manually defined, model-based controller. This way, we keep learning the planar DoFs only, and therefore avoid an exponentially increased action space for the learned policy. In the following, we derive a controller $C_{\beta\gamma}$

$$C_{\beta\gamma} : \hat{\mathcal{S}} \times \mathcal{M} \mapsto [-\pi/2, \pi/2]^2 \tag{5.4}$$

mapping each image window $\hat{s}$ and primitive m to the relevant half-space of the angles β and γ. Similar to the height z, the NN needs to implicitly estimate the controller $C_{\beta\gamma}$ to predict the final grasp reward. Therefore, we constrain the mapping to be simple in the sense of *deterministic* and *continuous* with a finite and low Lipschitz-constant. As the grasp reward is generally a multi-modal distribution, a simple controller can only contribute as a first approximation. However, it suffices that the controller benefits the grasp on average, as negative (false positive) cases should be learned and eventually avoided by the NN. For an efficient implementation, we restrict our algorithm input to depth images. In summary, the robot should utilize the additional two DoFs to:

- Avoid **collisions** between the gripper and a possible bin or other objects.
- Maximize the **grasp quality**. We briefly define the quality using the antipodal score. In general, we minimize the mean scalar product of the normal vectors between both gripper jaws and the object surface.



(a) Collision Avoidance **(b)** Angle β **(c)** Angle γ

Figure 5.11: Sectional drawings of depth profiles $s(x, y)$ (blue) to illustrate the controller derivation for a given grasp point (red). In (c), the hatched area marks an undercut from the camera view from above.

- Minimize **uncertainty** as grasping in 6 DoF naturally has to deal with back sides, undercuts, and viewing shadows. In this regard, the uncertainty increases with increasing grasp angles β and γ.
- Maximize the **grasp surface** between the gripper jaws and the object.

In a simplified manner, the grasp quality of a planar two-finger gripper is only changed by γ. Furthermore, we separate the calculation for each DoF.

Angle β The primary task of this DoF is to avoid collisions (Figure 5.11b). Let $s(x, y)$ be the height profile given by a depth image of size $x \in [-d_x, d_x]$ and $y \in [-d_y, d_y]$. We calculate the gradient in regard to the rotational axis x and average over all values in y

$$\tan\beta = -\frac{1}{4d_x d_y}\int_{-d_y}^{d_y}\int_{-d_x}^{d_x} w(x)\frac{\partial s}{\partial x}(x,y)\,\mathrm{d}x\,\mathrm{d}y$$

The resulting angle β approximates an orthogonal vector at the grasp point, averaged over the depth profile of the gripper's environment. A centered Gaussian $w(x) = \mathcal{N}(0, 1.5 \cdot$ finger width) is used as a weighting function.

Angle γ The primary task of this DoF is to maximize the grasp quality score. We define the angles γ_l and γ_r (Figure 5.11c) as

$$\tan\left(\gamma_l + \frac{\pi}{2}\right) = -\max_y \frac{1}{2d_x}\int_{-d_x}^{d_x}\frac{\partial s}{\partial y}(x,y)\,\mathrm{d}x$$

$$\tan\left(\gamma_r - \frac{\pi}{2}\right) = -\min_y \frac{1}{2d_x}\int_{-d_x}^{d_x}\frac{\partial s}{\partial y}(x,y)\,\mathrm{d}x$$

As the gradients can be quite prone to depth noise, we calculate the mean gradient

$$\tan\gamma_m = -\frac{1}{4d_x d_y}\int_{-d_y}^{d_y}\int_{-d_x}^{d_x}\frac{\partial s}{\partial y}(x,y)\,\mathrm{d}x\,\mathrm{d}y$$

averaging the depth profile without weights. Let $\rho \geq 0$ be the so-called *certainty* parameter. We combine these three angles via

$$\gamma = \begin{cases} \frac{1}{2}(\gamma_l + \gamma_r) + \gamma_m & \text{if } \gamma_l > -\frac{\pi}{2} \text{ and } \gamma_r < \frac{\pi}{2} \\ \frac{1}{1+\rho}(\rho\gamma_l + \gamma_m) & \text{if } \gamma_r \geq \frac{\pi}{2} \\ \frac{1}{1+\rho}(\rho\gamma_r + \gamma_m) & \text{if } \gamma_l \leq -\frac{\pi}{2} \\ \gamma_m & \text{else} \end{cases}$$

with the following intuition: If an undercut exists ($|\beta|$ or $|\gamma| \geq \frac{\pi}{2}$), the system will try a grasp orthogonal to the surface γ_m (for $\rho = 0$) or antiparallel to the visible surface normal of the object γ_l or γ_r (for $\rho \gg 0$). Furthermore, we use $\rho = 0.5$.

5.2.3 Collision Avoidance

Both angles should be guaranteed to be without collision (Figure 5.11a). For the following derivation, we introduce the angle τ substitutional for both angles β and γ. We derive the interval $[\tau_{min}, \tau_{max}]$ around planar manipulation $\tau = \pi/2$ without collision. For each angle, we use the maximum profile $s_b = \max_x s(x, y))$ or $s_c = \max_y s(x, y))$ regarding the rotation axis. Then, we calculate the maximum height h_i at a discrete set of distances r_{xi} from the grasp pose. We assume a simple rectangular collision model (with side lengths $d_u - d_l$ and $2r$) of the gripper. Simple geometric considerations lead to

$$\tau_{min} = \begin{cases} \tan^{-1}\frac{r}{d_l} & \text{if } d_u < r_x, \\ \tan^{-1}\frac{r}{d_u} + \tan^{-1}\frac{\sqrt{d_u^2+r^2-r_x^2}}{r_x} & \text{if } \frac{d_u^2+r^2}{r_x^2+r_z^2} < 1, \\ \tan^{-1}\frac{r_x}{r_z} + \tan^{-1}\frac{\sqrt{r_x^2+r_z^2-r^2}}{r} + \pi & \text{if } \frac{d_l^2+r^2}{r_x^2+r_z^2} < 1, \\ \tan^{-1}\frac{r_z}{r_x} + \tan^{-1}\frac{r}{d_l} & \text{else} \end{cases}$$

depending on the finger length d_l and the considered gripper height d_u. Similar results hold for the other side to calculate τ_{max}. Finally, the angles β and γ are clipped to be within their respective intervals.

Given a planar pose (x, y, α) with a corresponding cropped window $\hat{s}$, the controller calculates the additional orientations (β, γ) in a deterministic manner. Additionally, we keep the (trivial) calculation of the height z by reading the value at the center of the depth images plus a fixed gripper-dependent offset. Similar to subsection 5.1.2, a NN $\psi(\hat{s})$ maps the image state $\hat{s}$ to the single-step grasp reward r (given by Equation 5.1).

Again, we apply a fully convolutional neural network (FCNN) to efficiently implement a sliding-window approach for the *planar* DoFs. A key contribution is the integration of the model-based controller within the FCNN (Figure 5.12): The FCNN predicts the grasp reward at a grid of planar poses. At the selected planar pose (x, y, α) (e.g. the maximum

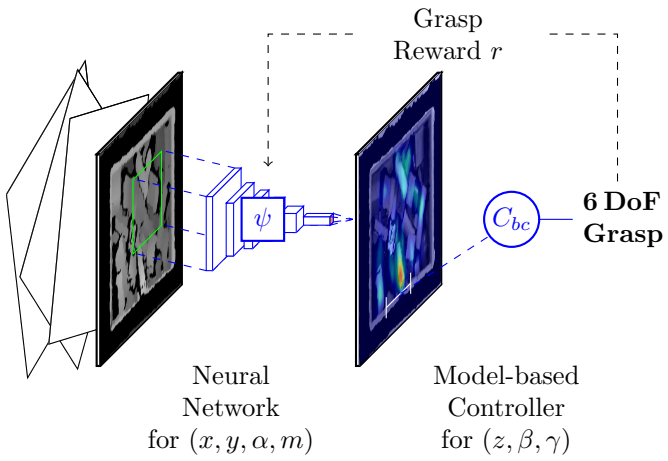


Figure 5.12: During inference, a fully convolutional neural network (FCNN) estimates the reward for a grid of grasp poses (heatmap). A model-based controller calculates the lateral DoF (z, β, γ) based on the selected planar grasp (x, y, α, m). During training, the reward of the *adapted* grasp is fed into the NN.

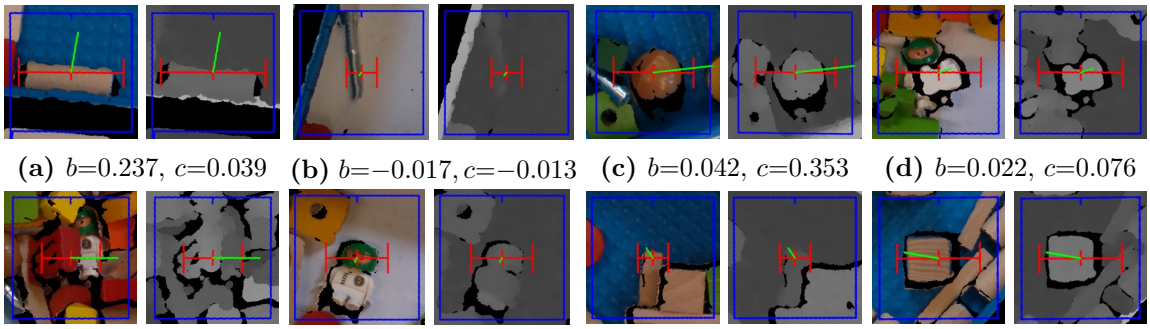
predicted grasp reward), the controller takes the cropped image window as input and calculates the remaining DoFs (z, β, γ). The grasp reward r is measured for the complete 6 DoF grasp. While the NN does not predict and only knows about the exact orientation indirectly, it learns to predict the grasp reward for a full 6 DoF grasp. In fact, the pipeline first estimates the grasp reward before computing the explicit values for β and γ. The planar DoFs are learned and predicted by the FCNN, while the remaining DoFs are calculated by the analytic model introduced above.

5.2.4 Experimental Results

Again, we use the experimental setup introduced in section 4.3 with the Realsense D435 camera only. We evaluate the approach on (1) simple wooden primitives for training and (2) more than 50 *unknown* objects for testing (Figure 5.3).

Training Process

The NN was trained with a dataset of 27 000 grasp attempts in an overall training time of 120 h. For exploration, random grasps were conducted for the first 4000 grasp attempts. Then, the *Prob(n=1)* selection strategy was used with linearly increasing n until 18 000 attempts. Then the ε-greedy *Top(25)* strategy was applied, falling back to *Prob(n=1)* with a probability of $\varepsilon = 0.1$. We retrain the NN on the most current dataset every 200 grasp attempts. In the end, the BCE of the validation dataset starts to saturate in our experimental setup.



(a) b=0.237, c=0.039 **(b)** b=−0.017, c=−0.013 **(c)** b=0.042, c=0.353 **(d)** b=0.022, c=0.076

(e) b=−0.002, c=0.281 **(f)** b=−0.028, c=−0.013 **(g)** b=0.057, c=−0.038 **(h)** b=0.035, c=−0.202

Figure 5.13: Examples of successful grasps, showing the RGBD image window $\hat{s}$. The projected approach vector (green) represents the angles b and c, the result of the model-based adaption. Furthermore, the grasp and its stroke are shown (red).

Learned Model-based Orientation

The adaption of the grasping primitive for the lateral angles β and γ are shown for several examples in Figure 5.13. We find angle β to be robust and working well for general collision avoidance (e.g. Figure 5.13a, e). By adapting γ, the robot avoids other objects (e.g. Figure 5.13c, i, j) or increases the grasp quality (Figure 5.13f). In general, the system often uses near-planar grasps (e.g. Figure 5.13b, g) to minimize uncertainty.

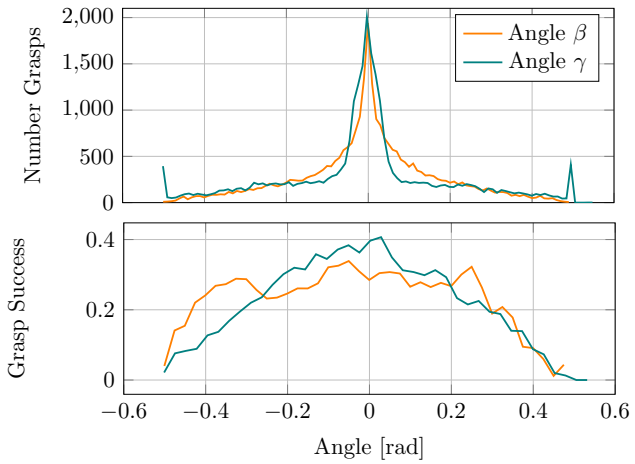


Figure 5.14: Histogram of the lateral angles β and γ (top) and their corresponding mean grasp success (bottom) during training.

Figure 5.14 shows a histogram of the angles during the training phase. As expected, we find that our model is very much limited to angles of around 0.5 rad (30 °). However, due to the integral limits depending on d_m as well as taking max and min values, γ is more sensitive to camera noise. We therefore crop the lateral angles, leading to small peaks of γ

at the border. As expected, we find that the grasp success drops off for non-planar grasps.

Grasp Rate

As before, we define the *grasp rate* as the ratio of successful grasps ($r = 1$) over the total number of grasp attempts without object replacement. In Table 5.2, we report grasp rates for several scenarios: For trained objects, we measure grasp rates near 100 % for isolated objects. In bin picking scenarios (e.g. 20 out of 30 objects), the grasp rate decreases to 92 %. In comparison to planar grasps with similar training, 6 DoF grasps are around 3 % less robust. If we apply our lateral adaption to a NN trained for planar grasps, the grasp rate decreases significantly. This emphasizes the ability of the NN to learn grasping *in conjunction* with the model-based adaption. For isolated unknown objects (Figure 5.3), the robot achieves grasp rates of 85 %.

Table 5.2: Grasp rate for different bin picking scenarios: (a) 6 DoF or planar grasping, with (b) normal or short grippers, and (c) either trained or unknown (novel) objects. The robot needs to grasp n objects out of a bin filled with m objects.

		Object Count	Grasp Rate	Number
6 DoF		1 out of 1	100 %	100
		1 out of 5	(99.0 ± 1.0) %	103
		5 out of 20	(96.3 ± 1.6) %	109
		20 out of 30	(92.1 ± 2.2) %	152
Planar		20 out of 30	(95.2 ± 2.0) %	126
Planar + Model		20 out of 30	(73.5 ± 2.0) %	136
Short	6 DoF	5 out of 20	(90.9 ± 2.5) %	110
	Planar	5 out of 20	(37.0 ± 6.9) %	108
Unknown	6 DoF	1 out of 1	(85.1 ± 3.3) %	215
		5 out of 20	(81.7 ± 3.4) %	104

To emphasize a real-world benefit of 6 DoF bin picking, we evaluate the grasp rate for *short* gripper fingers smaller than the height of the bin. While the planar grasp rates drop naturally to below 40 % due to collisions, our approach keeps the grasp rate above 90 %. Here, the collision avoidance enables grasps that would not be possible with planar grasps.

For bin picking applications, the inference time directly reduces the PPH. During real-world experiments, we achieve up to 296 ± 3 PPH. Table 5.3 lists processing times between camera input and grasp pose output for 6 DoF approaches. On average, our proposed algorithm requires less than 50 ms and scales only with the observed bin area. The model-based adaption takes a constant overhead of 6 ms. In comparison to related work, the presented approach requires significantly less processing in front of the NN inference.

Table 5.3: Processing and NN inference time of different grasping approaches. Comparisons to related work by [96].

	Processing [ms]	Inference [ms]	Overall [ms]
6 DoF (Ours)	15.3 ± 4.3	33.0 ± 0.4	48.3 ± 4.6
Planar (Ours)	9.8 ± 0.7	32.8 ± 0.5	42.6 ± 1.1
GPD [118]	24 106	1.5	24 108
Qin et al. [96]	5804	12.6	5817

5.3 Fully Convolutional Actor-Critic for 6 DoF Grasping

In the prior section, we investigated how analytical models can supplement a learned approach to keep the training complexity and inference calculation manageable. In this section, we propose a *fully learned* approach for 6 DoF grasping. We keep our general approach from chapter 3 by using an FCNN for reward prediction, but extend it incrementally to a fully convolutional *actor-critic* architecture. The actor calculates pose parameters for the non-planar DoFs of the system, in particular the height z and the angles β and γ, in a way that maximizes the estimated grasp reward. This way, the proposed architecture might also be used for other tasks as a general 6 DoF pose calculator.

5.3.1 Model Architecture

Let us briefly motivate the general proposed model architecture. Commonly, the task of grasping is split into grasp *generation* and grasp *ranking*. At first, a number of grasp hypotheses are generated, and subsequently evaluated by a scalar grasp score. Then, the highest-ranked grasp is chosen for execution. A key idea of this section is to approach this distinction between generation and ranking for each DoF separately. In particular, both generating and ranking grasps can be parallelized efficiently for *some* DoFs. For these DoFs, a large number of grasp hypotheses can be calculated quickly and in a trivial way. For the remaining DoFs, we generate only a small number of however high-quality pose hypotheses, in particular as many as the compute budget allows for. Finally, all hypotheses are ranked to find the best possible grasp. The planar DoFs (x, y, α) are computed in a sliding-window approach as before, while the remaining non-planar DoFs (z, β, γ) are generated continuously.

We use a first deep neural network (NN) called *actor* to generate grasps, and a second *critic* NN to rank them. This corresponds to the common *actor-critic* approach in RL (as introduced in chapter 3). Figure 5.15 illustrates the NN architecture during training (a) and inference (b). At first, the critic is trained to map the image window $\hat{s}$ as well as the non-planar grasp pose (z, β, γ) to the reward r. Afterwards, the actor is trained to maximize the critic's reward by computing corresponding grasp actions. In particular, the training propagates the gradients of the critic to the actor's output, enabling an efficient optimization. The actor and critic share their base convolutional layers and differ only in the final layers called actor or critic *head*. The critic head takes the action (z, β, γ) as input and concatenates these values with the first convolutional layer of input size $1 \times 1 \times N$. In practice, the computationally expensive base layers are needed to be computed only once for both action (z, β, γ) and reward r prediction. Moreover, the actor generates multiple (in the order of three) action hypotheses and optimizes for their joint *maximum* reward. The actor might then better approximate a multi-modal problem such as grasping. Intuitively, the actor might propose different grasps that are far away from each other, resulting in improved coverage of the grasp space before the critic's ranking.

During training, the complete bin image s, rotated over N_r discrete orientations, is

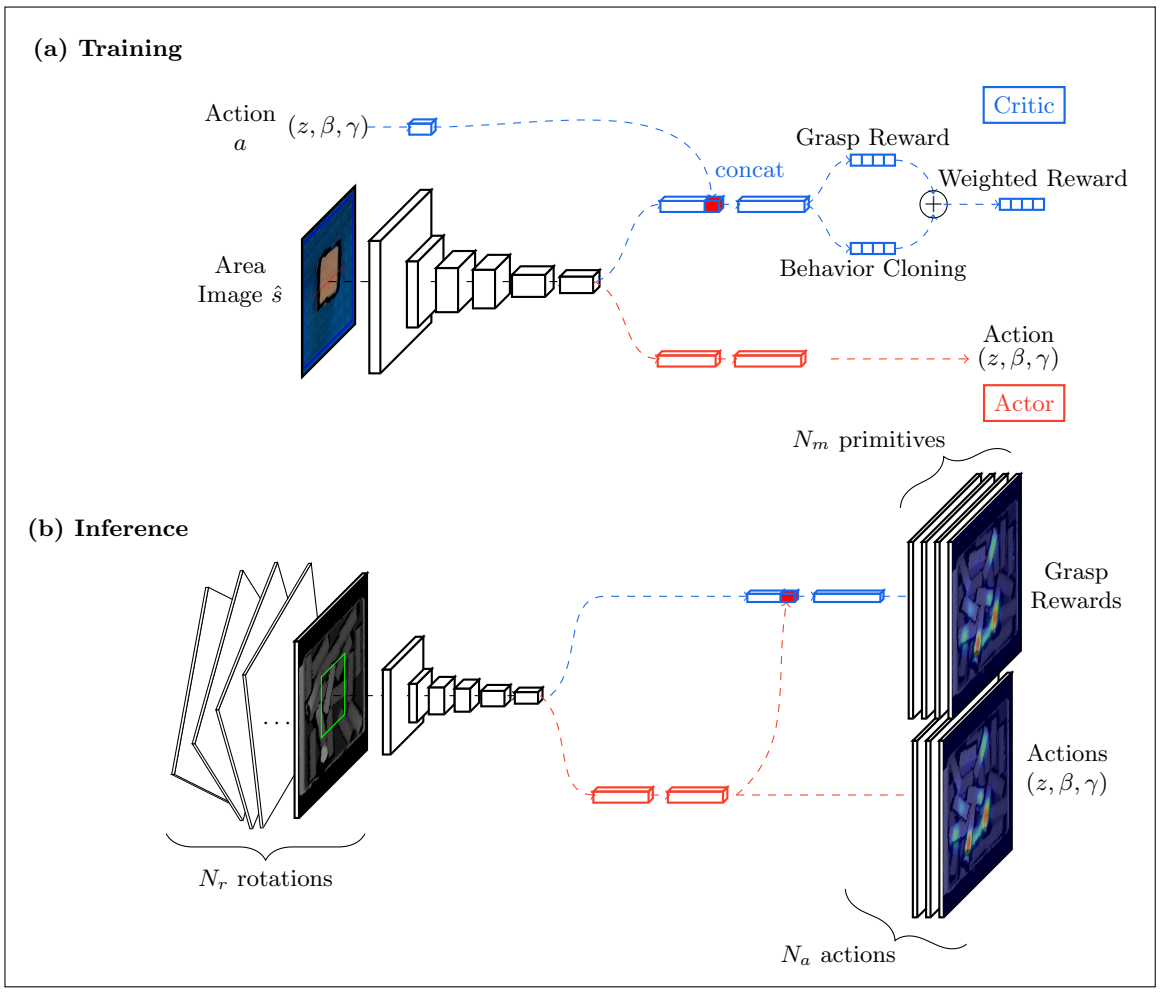
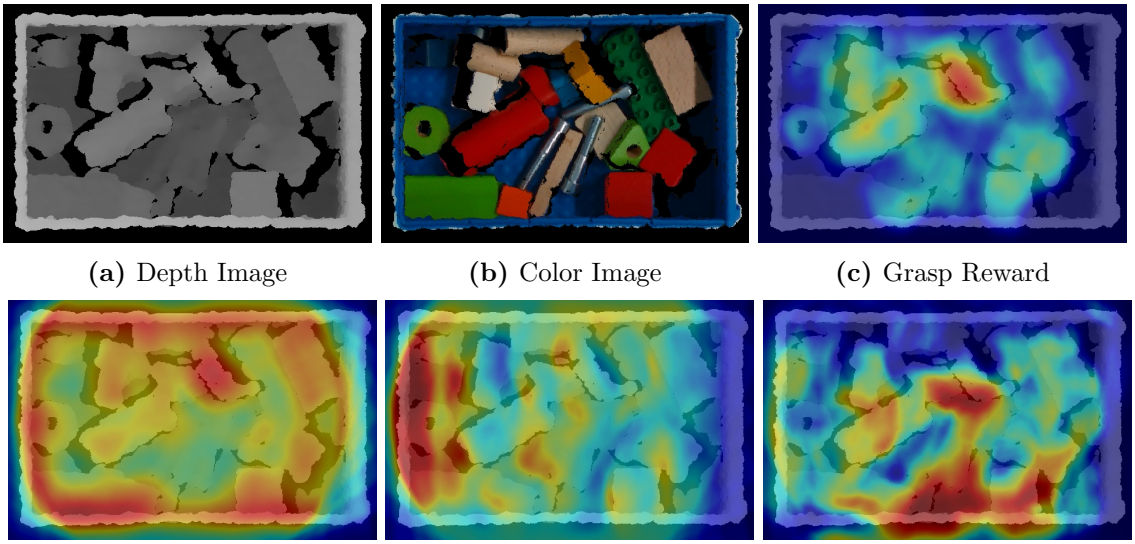


Figure 5.15: The neural network (NN) architecture during training (a) and inference (b). The overall system is divided into the base (black), the critic (blue), and the actor NN. During training, the critic maps an area around the gripper and the action of an either demonstrated or self-supervised grasp to the corresponding reward. The actor learns to maximize the weighted reward by predicting the action. All NN are fully convolutional. During inference, the same NN is applied to a complete image of the scene, resulting in a heatmap of actions and rewards.

taken as input for the shared base layers. First, the actor calculates N_a predictions for (z, β, γ) at each planar pose (x, y, α). This results in $N_r \times N_x \times N_y \times N_a$ generated grasp poses with a full 6 DoF pose that are subsequently evaluated by the critic head. Finally, the selection policy selects the best possible grasp for training or inference. The proposed architecture generates non-planar action hypotheses for all manipulation primitives m at once, and only the critic then maps them to a grasp reward for each individual primitive. Figure 5.16 shows that both the resulting grasp reward as well as action proposals can be interpreted as a heatmap.



(a) Depth Image **(b)** Color Image **(c)** Grasp Reward

(d) Action z **(e)** Action b **(f)** Action c

Figure 5.16: Example images with corresponding heatmaps for the fully convolutional actor-critic NN architecture. Given an RGBD input (a) and (b), the NN predicts the grasp reward at a grid of planar poses (c), here averaged over all rotations. Additionally, the actor generates an action height z (d), rotation b (e), and c (f) that maximizes the grasp reward at this specific pose. The reward heatmap is given from low (blue) to high (red), while the actions are normalized from their minimum value (blue) to their maximum (red). Exemplarily, (e) shows that the NN predicts a maximum rotation b near the left border of the bin that rotates the gripper away from the border.

5.3.2 Combined Self-supervised and Imitation Learning

Self-supervised Learning

As introduced in chapter 4, the training needs to be self-supervised in order to scale the real-world data generation. The **grasp reward** is defined as

$$r_g(s, a) = \begin{cases} 1 & \text{if grasp and filing successful,} \\ 0 & \text{else.} \end{cases} \tag{5.5}$$

To allow for a continuous and variable training, there are two bins (with changing grasp and filing) with a variety of objects. During training, we apply a simple curriculum learning strategy by introducing more complex objects, different boxes and deliberately changing the lighting condition over time.

Imitation Learning

Subsection 5.2.1 lists challenges of 6 DoF grasping, in particular related to the slow exploration and the more complex action space. We pre-estimate that the low purely random grasp success rate of around 0.25 % will make hundreds of thousands of grasp attempts necessary, unfortunately overstraining our robotic resources. To reduce the required exploration, we want to learn from human grasp demonstrations instead. By collecting a dataset up front, an initial policy is trained for the subsequent self-supervised training.

A grasp demonstration is given as a pose $\vec{p}_h$ with a continuous gripper stroke d_h, corresponding to a successful grasp (as supposed by the human annotator). We map the continuous gripper stroke d_h to a grasp primitive m by choosing the primitive with the next larger gripper width. In addition to the grasp reward, we train the robot to imitate the human grasp demonstrations. The reward for **behavior cloning** is defined as a contrastive loss

$$r_b(s,a) = \begin{cases} 1 & \text{if grasp was demonstrated,} \\ 0 & \text{else from noise distribution.} \end{cases} \tag{5.6}$$

using a pre-defined negative noise distribution. We overlay two distributions: First, a uniform random distribution over the complete action space, constrained by the size of the bin. Second, given a grasp demonstration $\vec{p}_h$ at the planar position (x, y, α), we uniform randomly set (z, β, γ) only. This way, the noise distribution focuses on the output of the actor. In other words, the critic learns to differentiate between *human* and *random* grasps, and the actor then tries to fool the critic.

Combining Imitation and Self-supervised Learning

We collect successful human grasp demonstrations first, and subsequently continue with the self-supervised exploration. Let the total reward r be the weighted sum of the behavior cloning and grasp reward

$$r = w_b\, r_b + (1 - w_b)\, r_g \tag{5.7}$$

with the behavior cloning weight w_b. For the critic, both rewards r_b and r_g are estimated at inference run-time. For training the actor, however, this weight needs to be fixed. Then, we start the self-supervised training only based on behavior cloning (with $w_b = 1$). In general, we reduce w_b linearly to zero over time. At the end of the training, the robot should only predict grasps based on its *own* experience.

This way, the purely random exploration is replaced by behavior cloning. For the self-supervised exploration, however, we introduce two stochastic elements to the training:

1. We apply the *Prob(n)* selection method from section 3.7 with $n \in \{1, 2, 4, 6\}$. While the *Top(n)* method has little randomness, we combine it in a ε-greedy strategy with *Prob(n)* as the fallback.

2. We apply a Gaussian noise with zero mean on the action (z, β, γ) via

$$(z, \beta, \gamma)' = (z, \beta, \gamma) + \varepsilon_a \mathcal{N}\left[0, (0.01\,\text{cm}, 0.1\,\text{rad}, 0.1\,\text{rad})\right]. \tag{5.8}$$

   The standard deviations are scaled with the ε_a parameter.

In particular, we avoid purely (uniform) random exploration *completely* because of the high-dimensional action space and the hence very low random grasp success rate. Instead, behavior cloning is used to find an initial policy that enables downstream self-supervised training. However, the reduced exploration increases the chances of converging to a (worse) local minima instead of the optimal global minima. The data collection for imitation learning is crucial for drawing the local near the global minima.

Neural Network Training

In parallel to the self-supervised data collection, the NN is continuously trained on the most recent dataset. Both actor and critic NNs are split into three parts (Figure 5.15): The critic head, the actor head, and a shared base of convolutional layers. The critic and actor NNs are trained consecutively, while the base weights are only updated during critic training. Here, we use a common binary crossentropy (BCE) loss regarding the reward r. Afterwards, the base is kept fixed for the actor training. The actor generates N_a action hypotheses. Then, the loss function of the actor is given by

$$\mathcal{L} = \sum_{i \in N_b} \max_{j \in N_a} |1 - \text{critic}(\hat{s}_i, \text{actor}_j(\hat{s}_i))| \tag{5.9}$$

with the batch size N_b. In addition to the data augmentation techniques presented in section 3.6.5, we make use of the height correspondence between z and the depth image. A random offset Δz is applied to the depth image by changing its brightness by a constant factor; the orthographic color image is invariant.

5.3.3 Experimental Results

Again, we use the experimental setup introduced in section 4.3 with the Realsense D435 camera only.

Recording Grasp Demonstrations

Successful grasp demonstrations are recorded as a pair of grasp pose p and continuous gripper width d. Furthermore, we evaluate and compare two approaches for human grasp demonstrations (Table 5.4).

Table 5.4: Methods for Recording Grasp Demonstrations.

	Human Tracking	Kinesthetic Teaching
Calibration (per session)	15 min	-
Grasp Recording	5 s	16 s
Post-processing	6 s	-
Total (per grasp)	15 s	16 s
Mean Deviation	8 mm	< 1 mm

First, **human tracking** allows to estimate (1) the hand position using an *ARTrack* infrared marker system and (2) the relative position of the fingertips using a *CyberGlove* data glove. We measure the position of the thumb and index fingertip, and derive the grasp pose as the center and the gripper width as the distance between the two fingertips. Instead of relying on a human hand model for calculating the finger position based on angles, we calibrate the CyberGlove by learning directly from the raw sensor values. Typical errors are in the order of a few mm for the marker-based system and around a cm of accuracy for the relative fingertip positions. This way, post-processing of the grasp poses is required. Moreover, we found that the CyberGlove needs to be re-calibrated quite regularly after a session of around 250 grasps due to sensor shift.

Second, we move the collaborative Franka Emika robot to successful grasp poses by hand, referred to as **kinesthetic teaching**. While this approach achieves optimal accuracy (at least the same absolute position accuracy that applies to the grasping process itself), teaching is slower *per grasp* than by human tracking. However, no calibration or post-processing is required. In total, we recorded around 1000 grasp demonstrations in 5 h for each approach.

Training

After training the NN with 2000 demonstrated grasps for the first time, the robot collected over 38 000 grasp attempts in a self-supervised manner. For each grasp attempt, one top-down and two additional RGBD images from lateral viewpoints were recorded per grasp, resulting in a dataset of over 100 000 images. Typical durations for a grasp attempt and three images are between 15 s and 20 s, resulting in 150 h of self-supervised training on top of 10 h of human grasp demonstrations. Figure 5.19 shows the accuracy as the absolute difference between the estimated and measured grasp reward over the training progress.

(a) Simple Objects **(b)** Unknown Objects

Figure 5.17: Object sets for evaluating 6 DoF grasping with an actor-critic architecture

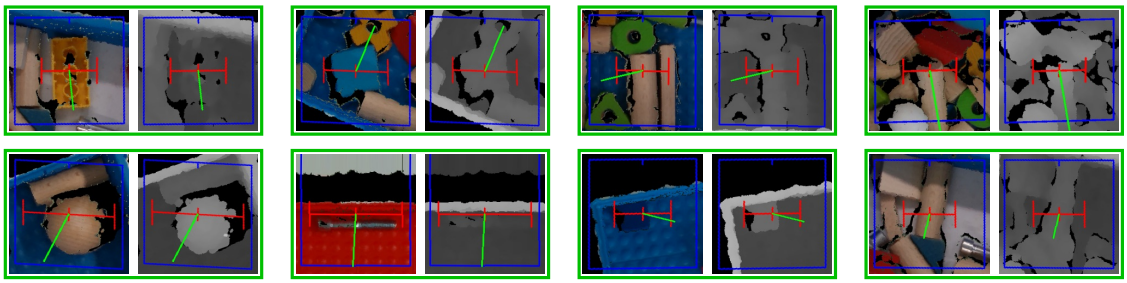


Figure 5.18: Example windows $\hat{s}$ of (successful) human grasp demonstrations

In general, a variety of simple objects, mostly wooden geometric primitives, were used throughout training (Figure 5.17a). We applied a simple curriculum learning strategy: First, we introduced more complex objects (e.g. screws) over time. Second, we increased the number and diversity of objects within the bin. In the end, the dataset included around 40 000 grasp attempts with a mean success rate of 31 %. Then, the NN training required around 2 h, corresponding to 450 newly collected grasp attempts.

Grasp Rate

As before, the grasp rate is defined as the ratio of successful grasps ($r = 1$) over the total number of grasp attempts without object replacement. In particular, the grasp rate is measured by grasping n objects out of a bin filled with m objects. Table 5.5 shows grasp rates for a variety of bin picking scenarios. Apart from the training objects, *unknown* (novel) objects were used for evaluation. They were never seen before, and include household as well as office objects, toys, tools, and industrial parts (Figure 5.17b).

In typical bin picking scenarios, our approach can grasp trained objects with success rates of over 95 %. However, this rate decreases with an increasing number of objects in the bin (due to clutter) as well as an increasing number of objects to be grasped out (due

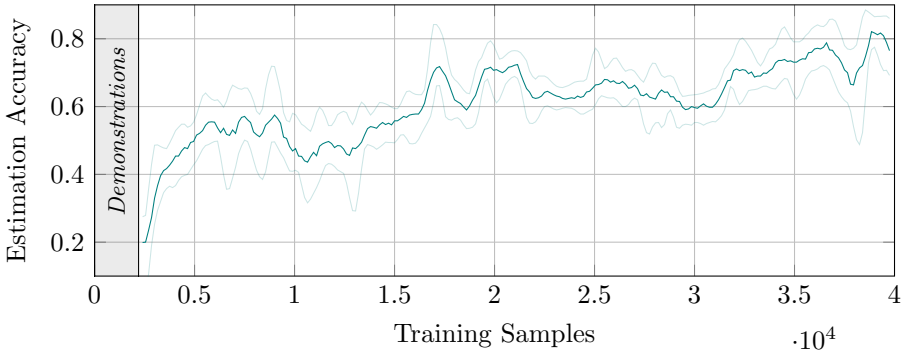


Figure 5.19: Estimation accuracy (absolute difference between estimated and measured reward) over the training progress. The first 2500 training samples were recorded as human grasp demonstrations, following a self-supervised real-world training.

to using the best grasps first). For unknown objects, we measured grasp rates of 92 % in isolated settings and 84 % in clutter. Furthermore, we compare to a planar grasping baseline and three alternative approaches.

In general, 6 DoF grasping has two major advantages over planar grasping in the context of bin picking: First, the robot is better able to avoid collisions, in particular between gripper and bin. Accordingly, our approach is able to grasp objects out of the corner of the bin. We test objects from both the trained and unknown sets that are not possible to grasp in a planar way (Figure 5.10a). In comparison, we achieve a grasp rate of 87 % in this setting. Second, 6 DoF grasping allows for a more flexible viewpoint of the camera. For example, a robot is then able to grasp flat objects from lateral points of view (Figure 5.16f).

Inference Performance

Table 5.5 shows the grasp calculation duration for all evaluated approaches. On our hardware, our approach takes on average 67 ms for inferring the next best grasp. Regarding related work, this is two orders of magnitude faster than PointNetGPD calculating 6 DoF grasps with specialized NN architecture point clouds. Table 5.6 shows detailed timings about the calculation performance. Interestingly, inferring the NN takes less than half of the total time, as rendering orthographic images from point clouds takes a fixed 24 ms. In comparison to planar grasps, 6 DoF grasping shows an overhead of around 20 ms for the larger actor-critic NN. To highlight the efficient implementation of the FCNN as a sliding window approach, we further compare this to a non-fully convolutional NN that crops each input image individually. In comparison, the FCNN accelerates the inference over two orders of magnitude.

Table 5.5: Grasp Rate in Different Bin Picking Scenarios.

	Calculation [ms]	Objects	n out of m	Grasp Attempts	Grasp Rate %
6 DoF (ours)	66.7 ± 1.3	Simple	1 out of 1	152	100
			1 out of 5	122	99.2 ± 0.8
			5 out of 5	122	98.4 ± 1.1
			5 out of 20	124	96.8 ± 1.5
			10 out of 20	220	95.5 ± 1.2
			15 out of 20	193	93.3 ± 1.3
		Unknown	1 out of 1	116	92.2 ± 2.5
			5 out of 10	203	83.7 ± 2.2
Planar (ours)	47.3 ± 2.3	Simple	1 out of 1	163	98.8 ± 0.9
			10 out of 20	195	92.3 ± 1.7
		Unknown	1 out of 1	120	73.3 ± 3.6
			5 out of 10	117	51.3 ± 4.3
Model-based Convolution	36.1 ± 1.8	Simple	1 out of 1	108	82.4 ± 4.2
			10 out of 20	116	69.0 ± 2.7
		Unknown	1 out of 1	120	73.3 ± 3.6
			5 out of 10	117	51.3 ± 4.3
Dex-Net [82]	650 ± 39	Simple	1 out of 1	108	85.8 ± 3.1
			10 out of 20	114	78.9 ± 2.5
		Unknown	1 out of 1	112	79.5 ± 3.3
			5 out of 10	107	74.8 ± 2.2
PointNetGPD [75]	9650 ± 2060	Unknown	1 out of 1	106	68.9 ± 3.3
			5 out of 10	115	56.5 ± 1.9

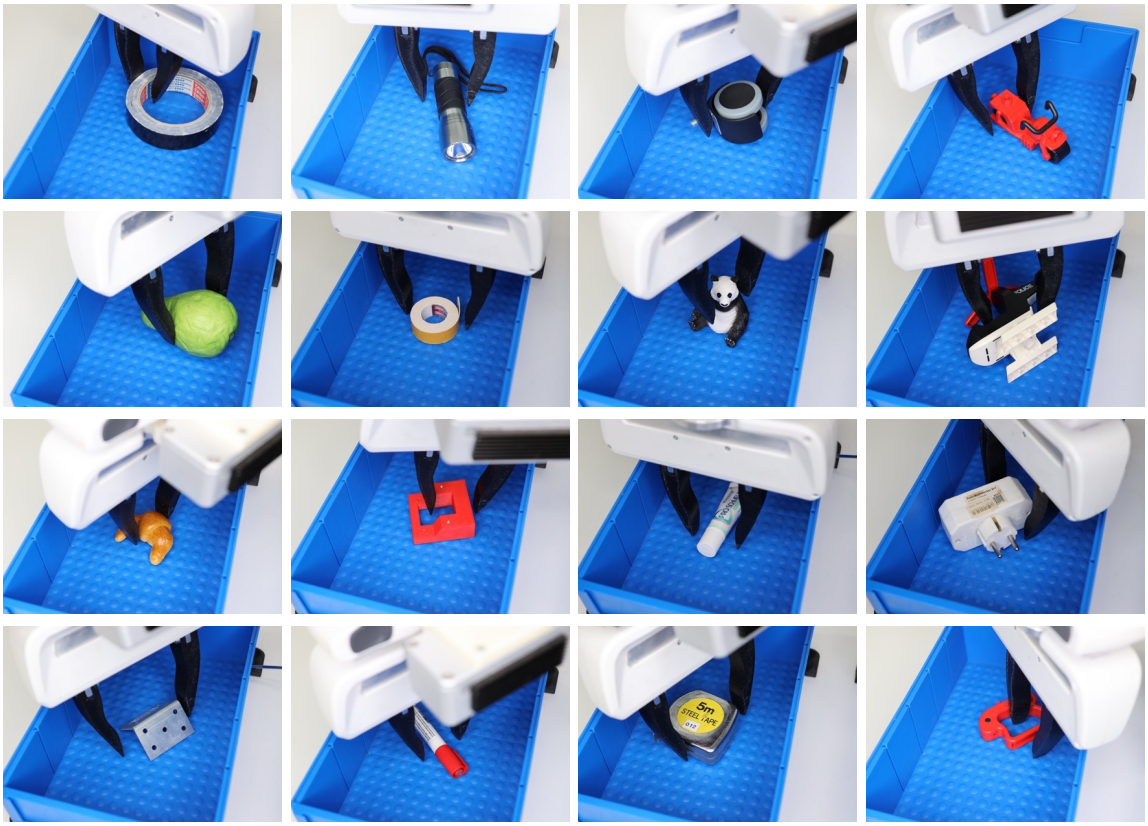


Figure 5.20: Example grasps with 6 DoF on unknown objects, computed by a fully convolutional actor-critic NN architecture trained on human demonstrated and self-supervised data collection.

Table 5.6: Details about the Calculation Duration [ms].

	Non-FC 6 DoF	Planar	6 DoF
Rendering	23.9 ± 0.4		
Preprocessing	4919 ± 45	7.6 ± 1.0	7.6 ± 0.9
NN Inference	3980 ± 8	13.5 ± 0.9	32.5 ± 0.3
Selection	191.0 ± 0.4	2.3 ± 0.1	2.7 ± 0.2
Total	9114 ± 38	47.3 ± 2.3	66.7 ± 1.3

Collision Rate

We define a collision when the external force exceeds 7 N during the grasp approach motion. In most cases, a collision should be avoided due to sensitive objects and the robot's wear and tear. In some cases, however, a collision can enable a grasp that wouldn't be possible otherwise, e.g. when a finger needs to move into a small gap between objects. Our approach allows three modes to handle collisions: First, the default setting (called collisions as failures) explicitly learns to avoid collisions. During the training, both the target pose and the pose just above the collision are saved. Then, the collision pose is used as the successful grasp, while the original target pose is learned as a grasp failure. Second, a model-based check finds collisions as intersections between a bounding box of the gripper and the point cloud. In case of a predicted collision, the next best possible grasp is selected in an iterative manner. Third, collisions can be ignored. For these modes, Table 5.7 compares collision and grasp rates. Despite treating collisions as failures, we measure a collision rate of 19 %,

Table 5.7: Collision Rate (10 out of 20).

Mode	Collision Rate [%]	Grasp Rate [%]
Ignore Collisions	34 ± 3	94.2 ± 1.6
Collisions as Failures	19 ± 2	95.5 ± 1.2
Model-based Check	3 ± 1	94.6 ± 1.1

improving from a baseline 34 % without collision handling. With the model-based check, we reduce the collision rate to 3 %.

Ablative Study

Furthermore, we investigate the system in an ablative study. For some parameters, we measure the influence on the *validation metrics* of the corresponding NN. While there is some correlation to the actual grasp rate, this is highly nonlinear. In particular, the true grasp error rate is a rate of false positive predictions.

Fully Convolutional Constraints While a fully convolutional architecture allows for an efficient implementation of a sliding-window approach, it limits possible designs of the NNs.

Primarily, the concatenation of the action onto the critic is required to be a vector without spatial dilation. In contrast, an unrestricted NN might tile the action and therefore merge the action within lower layers. Table 5.8 compares a fully convolutional and an unrestricted NN with fixed training otherwise. Note that the fully convolutional NN improves the performance by two orders of magnitude compared to cropping and evaluating each window individually.

Table 5.8: Accuracy of a fully convolutional vs. unrestricted NN architecture.

Architecture	BCE	Accuracy	Duration [ms]
Fully Convolutional	0.295	85.9 %	66.7 ± 1.3
Unrestricted	0.276	86.7 %	9114 ± 38

Locality The side length l of the cropped image area $\hat{s}$ is a parameter for the locality of the grasp prediction. We set $l = 10\,\text{cm}$ as a default, slightly exceeding the size of the maximum gripper width of 8.6 cm, and corresponding roughly to the maximum object size in the training set. Figure 5.21 shows the dependency of the critic's accuracy on the locality. We kept the NN architecture fixed while retraining on scaled input images. For purely *local* actions, we expect a maximum at the maximum gripper length. However, maximum accuracy is reached at side lengths $l \approx 0.12\,\text{cm}$. This is due to non-local effects, e.g. collisions between the gripper and the bin. For lateral orientations, the projected approach vector might even be outside the image area. As expected, the accuracy drops off for increased side length e..g due to the decreased resolution.

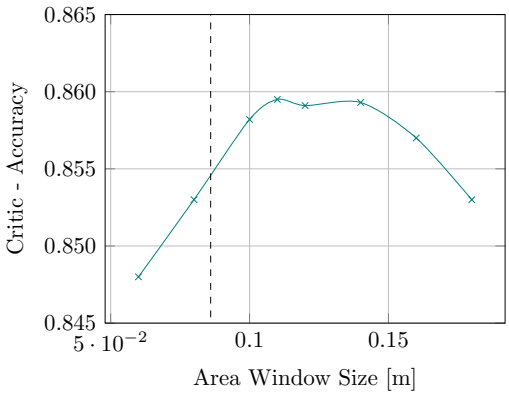


Figure 5.21: Accuracy of the critic NN depending on the window size of the input image.

Number of Action Hypotheses The actor predicts multiple action hypotheses to be evaluated by the critic. While this comes with the cost of a lower inference performance, an increased number of actions improves the mean absolute error metric of the actor (Figure 5.22). In our experiments, we set the number of action hypotheses to $N_a = 3$.

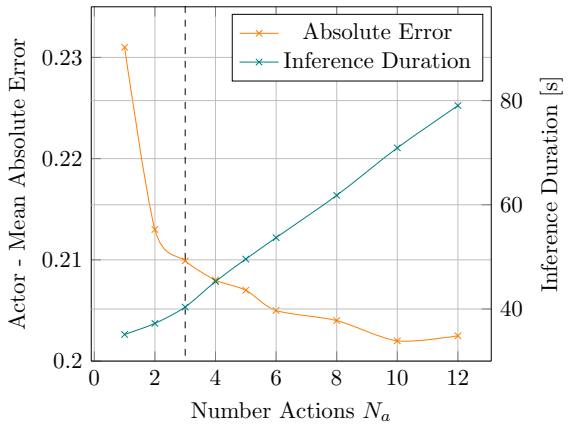


Figure 5.22: The actor's loss and inference duration depending on the number of non-planar action hypotheses.

Color Information Depending on the camera, the image provides color (RGB), depth (D), or both (RGBD) information as input channels to the NNs. Incorporating color information improves the accuracy of the critic from 85.5 % (D) to 86.0 % (RGBD). In comparison, the critic achieves an accuracy of 82.9 % only using color information (RGB). Note however that depth information is required for the orthographic projection. As expected, we find color input in particular helpful when depth information is missing or inaccurate.

Viewpoint Encoding So far, the NNs are trained assuming the manipulation to be viewpoint invariant under translations and *all* rotations. However, as described in section 3.4, gravity influences the physics of manipulation. The three DoFs (α, β, γ) change the local gravity vector. As these are given in the camera frame, the camera orientation becomes essential. Therefore, we input information about the current viewpoint orientation into the system. Figure 5.23 compares the NN training result with and without the viewpoint encoding. We find that the actor model is able to achieve a higher grasp rate with a viewpoint encoding, in particular for orientations that are far from top-down ($\beta = 0$) grasps.

Behavior Cloning Weight Figure 5.24 shows the grasp rate depending on the behavior cloning weight w_b, influencing the final reward according to Equation 5.7. The grasp rate decreases monotonously with increasing focus on behavior cloning. When using only the 2000 human grasp demonstrations and ignoring the self-supervised training ($w_b = 1$), the system achieves a grasp rate of $(73 \pm 4)\,\%$. As the grasp rate reaches its maximum at $w_b = 0$, the human demonstrations are only required by the training, but not the final grasp policy. Trivially, we expect a benefit from the larger combined dataset size, however, the shortcoming might be caused by a mismatch between human demonstrations and the subsequent transfer to the robot's experience.

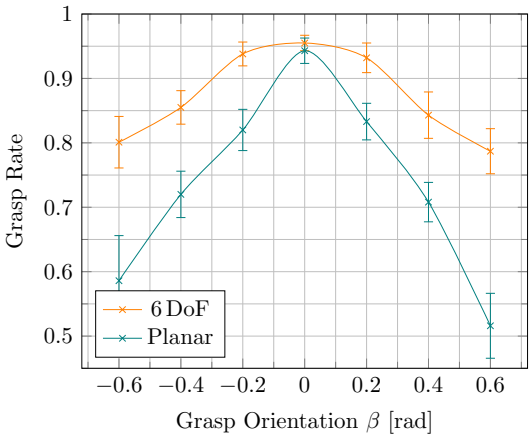


Figure 5.23: Grasp success rate depending on the viewpoint orientation β (the grasp orientation given in the camera frame).

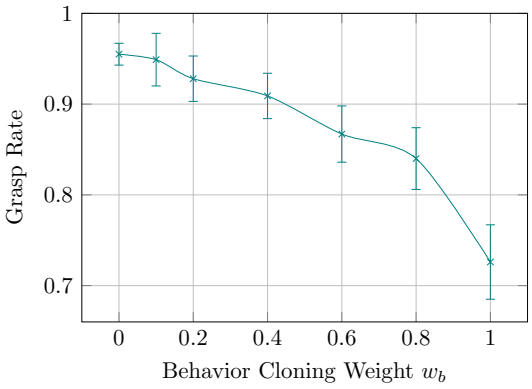
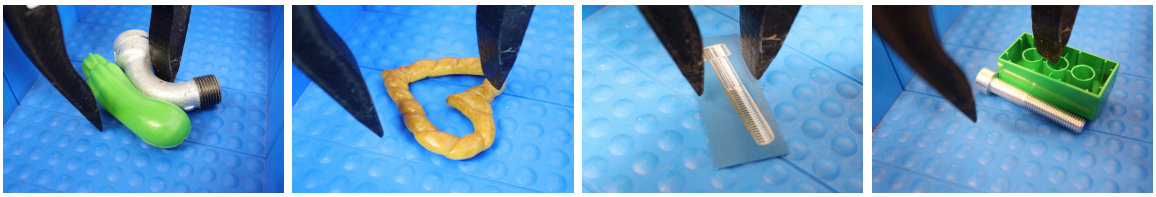


Figure 5.24: Grasp success rate depending on the weight w_b of the behavior cloning (based on human grasp demonstrations). The grasp rate is measured by grasping 10 out of 20 simple objects out of a bin.

Failure Cases

Similar to other data-driven approaches, grasp decisions and in particular its failures are not necessarily *explainable*. Nonetheless, we find several primary failure cases (Figure 5.25) showing the limitation of the current system.



(a) Multiple Objects **(b)** Adversarial Object **(c)** Confusing Texture **(d)** Missing Depth Data

Figure 5.25: Common failure cases of our 6 DoF grasping pipeline.

Multiple Objects As the robot does not know about the concept of objects, it may try to grasp multiple objects at once. This often leads to unstable configurations due to objects moving relative to each other. In general, the robot fails to predict whether two object *parts* are fixed.

Adversarial Objects Despite training on a variety of objects, we are still able to discover objects that result in repeated grasp failures. This is mostly due to the geometric shape being not similar enough to the trained objects, corresponding to an out-of-distribution case for our data-driven approach.

Confusing Texture Furthermore, we find that texture can confuse the robot, despite the seemingly small improvement of 0.5 % in the critic's accuracy when using RGBD input. As an example, the robot estimates the grasp reward of a printed image of a screw to (80 ± 4) %, corresponding to an obvious false positive classification.

Missing Depth Data In orthographic depth images, every object has, depending on its height and distance to the camera, a "shadow" of missing depth data around it. However, camera failures, unfavorable occlusions, or reflective objects might increase these regions of missing depth data significantly. Currently, the NN takes these regions with a constant zero as input, and therefore dismisses the explicit uncertainty that the system knows about. Incorporating uncertainty into the grasp reward prediction, e.g. by mapping the uncertainty of the input data to the reward, might help to make the robot more robust to unexpected objects or obstacles in the missing depth data. While the NN implicitly learns about this uncertainty, it still sometimes tries to insert the robot's fingers into missing depth data. This might also be caused by the depth representation mapping zero values to far distances. Currently, around a third of grasp failures are within regions missing sufficient depth information. As discussed in section 5.6, some of the related work fill missing depth data with

neighboring values. However, we find that this creates a range of even greater challenges and assumptions, dismisses uncertainty completely, and does not improve the grasping rate in practice.

Both failure cases of *multiple objects* and *missing depth data* are regularly seen during training and in the dataset. However, there are also positive examples of both failure cases. For example, the gripper finger might reach into the gap of the Duplo block. We find that grasp failures are often due to this *stochastic* behavior of the system. Incorporating uncertainty might help finding a risk-averse policy in the future.

5.4 Pre-grasping Manipulation

In general, grasping is more complex than a single clamping action. For example, surrounding obstacles might prevent all possible grasps of a specific object. In this case, it needs to be moved first so that it can be grasped afterwards. While pre-grasping manipulations are trivial for humans, they require complex interactions like sliding, pushing, or rolling objects. In the context of bin picking, pre-grasping is essential to empty a bin completely, since the bin itself might block grasps at its edges or corners. Additionally, when items are stored as space-efficiently as possible, objects often prevent each other from being grasped in densely filled bins. We focus on planar grasping again to further motivate the necessity for pre-grasping manipulation. This section is based on [8].

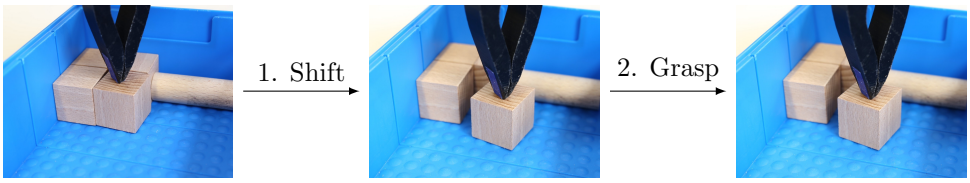


Figure 5.26: A shift primitive for pre-grasping manipulation allows the robot to grasp objects that are blocked due to collisions. It learns to apply a shift action to explicitly increase the predicted grasp success.

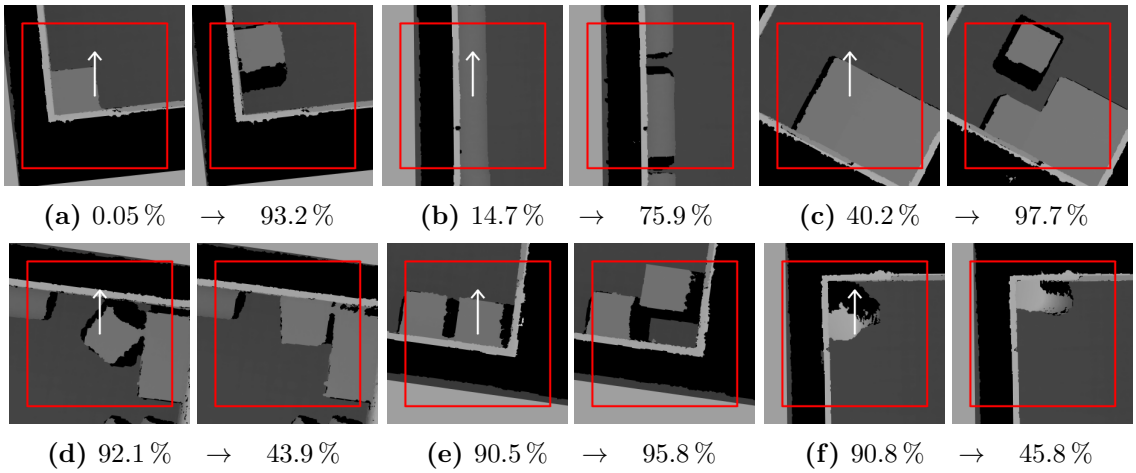
For this, we add two motion primitives for *shifting* as a pre-grasping manipulation (Figure 5.26). In both cases, the gripper closes completely and approaches the given shift pose parallel to its gripper jaws. If a collision occurs, the approach motion stops. Then, the robot moves either 30 mm in positive x-direction or positive y-direction of the gripper frame. Note that the gripper is not rotational invariant, so these primitives are distinct regarding the gripper's collision shape.

In a trivial manner, the MDP could be prepended by a number of pre-grasping actions before the final grasp, and the robot could then learn to maximize the final grasp reward (similar to Equation 5.1). However, this introduces a time dependency into the system, as the grasp is now dependent on all previous pre-grasping actions. The complexity of the learning problem would scale exponentially with the number of pre-grasping actions, as the robot would need to learn the effect of each action on the grasp in a data-driven manner. Moreover, there is only a single reward at the end of each episode, leading directly to the problem of *sparse rewards*. The robot would need to learn the outcome of multiple actions based on a single scalar, and therefore would need to try out an exponential number of combinations to find the pre-grasping actions causing the improving grasp success. In the following, we will present an approach to avoid this challenging complexity.

5.4.1 Learning for Shifting

A key contribution of this section is making one primitive (shifting) explicitly dependent on the other (grasping). This way, we bypass the problem of sparse rewards in multi-step MDPs. Besides faster and more data-efficient training, this also allows to learn the skills successively. Therefore, we can reuse prior, successful approaches of learning for grasping (e.g. from section 5.1) and focus solely on pre-grasping manipulation.

For grasping, we again define the set of motion primitives $m \in \mathcal{M}$ as gripper clamping actions starting from three different pre-shaped gripper widths d_m. Subsection 5.1.1 describes the primitives and their trajectories in more detail. The grasp reward is given by the binary success (Equation 5.1). Based on section 5.1, we make use of a NN mapping a cropped image window $\hat{s}$ to an estimated grasp reward ψ and use it for: (1) Estimating the grasp probability at a given position (x, y, α), (2) calculating the best grasp (x, y, α, d), (3) calculating the maximum grasp probability in the entire bin $\lceil\psi\rceil$, and (4) calculating the maximum grasp probability $\lceil\hat{\psi}(s, x, y, \alpha)\rceil$ in a cropped window with a given side length l centered around a given pose (x, y, α). Figure 5.27 shows exemplary image windows before and after a shifting primitive and their respective maximum grasp probability $\lceil\hat{\psi}\rceil$ around the gripper's pose.



(a) 0.05 % → 93.2 % **(b)** 14.7 % → 75.9 % **(c)** 40.2 % → 97.7 %

(d) 92.1 % → 43.9 % **(e)** 90.5 % → 95.8 % **(f)** 90.8 % → 45.8 %

Figure 5.27: Examples of depth image windows $\hat{s}$ before (left) and after (right) an applied motion primitive. The maximum grasp probability within the red window $\lceil\hat{\psi}\rceil$ before and $\lceil\hat{\psi}\rceil'$ after are given below; their difference is then estimated by a fully convolutional neural network (FCNN).

We integrate prior knowledge about the relationship between shifting and grasping by making the reward function for shifting *explicitly* dependent on the grasping probability. More precisely, the system predicts the influence of a motion primitive on the maximum grasp probability $\lceil\psi\rceil$. We train a second NN using the reward function

$$r_s(s) = \frac{1}{2}\left(\lceil\hat{\psi}\rceil(s', x, y, \alpha) - \lceil\hat{\psi}\rceil(s, x, y, \alpha) + 1\right) \tag{5.10}$$

mapping the image s to the difference of the maximum grasping probability in a window $\lceil\hat{\psi}\rceil(s, x, y, \alpha)$ before s and after s' the manipulation primitive. Therefore, depth images before and after the shifting attempt are recorded and applied to the grasp probability NN. Additionally, the reward is re-normalized to $r_s \in [0, 1]$. The window w is approximately 50 % larger than for the grasping action, the latter corresponding roughly to the maximum gripper width. In comparison to the grasping reward function $r_g \in \{0, 1\}$, estimating shift rewards is a pure regression task. We further denote the estimated reward for shifting $Q_s(s, a)$ as ρ. The NN is trained to optimize the mean squared loss between the predicted and actual reward ρ and r_s. Finally, all predictions for the five grasping and shifting primitives are combined, and computed by a single NN at 20 rotations, 40 x and y translations each, resulting in 160 000 evaluated manipulation actions. Figure 5.28 shows two exemplary heatmaps of a shifting primitive averaged over all rotations.

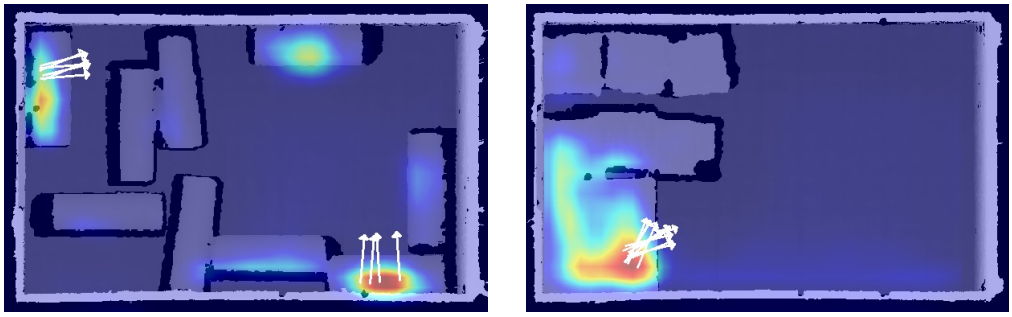


Figure 5.28: Examples of heatmaps for the shifting primitive. The NN predicts the maximum positive reward ρ at a given pose (x, y) for all rotations a and motion primitives d. The rewards are shown from low (blue, $\rho = 0.5$) to high (red, $\rho = 1$), the direction is shown (white) for the ten highest rewards.

Since data generation is a limiting factor in robot learning, it is important that the training is self-supervised and requires as little human intervention as possible (chapter 4). To guarantee continuous training, the system first estimates the overall maximum grasping probability $\lceil\psi\rceil$. If $\lceil\psi\rceil < 0.2$, the system tries to increase the grasping probability until $\lceil\psi\rceil \geq 0.8$ is reached. Then, the system tries to decrease the maximum grasping probability until $\lceil\psi\rceil < 0.2$ again. This is done by exploring the negative action-value-function $-Q_s(s, a)$ while keeping the selection function σ constant. Intuitively, the robot *both* learns to shift objects to improve and worsen the grasp probability, so that it can efficiently explore randomized scenes in a self-supervised manner. The training starts with a single object in the bin, further ones were added manually over time for curriculum learning.

5.4.2 Combined Learning and Inference

For the task of bin picking, grasping and shifting needs to be combined into a single controller. Besides inference itself, combined learning also enables matching data distributions

for training and application. First, let ψ_g be a threshold probability deciding between a grasping and a shifting attempt. Second, let ρ_s denote a threshold probability between a shift attempt and the assumption of an empty bin. As shown in Figure 5.29, the system first infers the maximum grasping probability $\lceil\psi\rceil$. If $\lceil\psi\rceil$ is higher than ψ_g, the robot grasps, else it evaluates the shifting NN and estimates the maximum shifting reward $\lceil\rho\rceil$. If $\lceil\rho\rceil$ is larger than ρ_s, the robot shifts and restarts the grasp attempt, else it assumes the bin to be empty. ψ_g can be interpreted as a high-level parameter corresponding to the

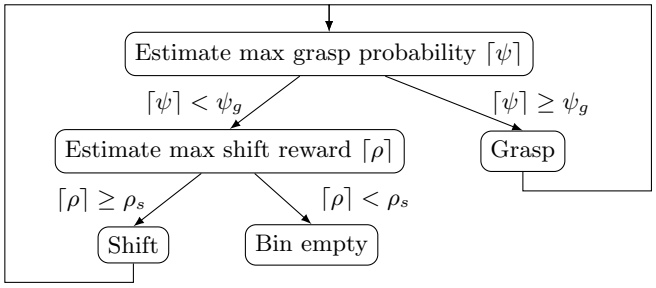


Figure 5.29: State diagram of the combined grasping and shifting controller; common threshold parameters are $\psi_g \approx 0.75$ and $\rho_s \approx 0.6$.

system's cautiousness for grasp attempts as well as its readiness to rather increase the grasp probability by shifting over trying a grasp attempt.

5.4.3 Experimental Results

Experiments were performed on the system introduced in section 4.3. Inferring the NN takes around 30 ms on a single GPU, and calculating the next action including image capturing takes less than 100 ms. The source code, trained models and a supplementary video showing our experimental results are open-source[1].

Data Recording

In bin picking, shifting objects becomes either necessary because of static obstacles like the bin or dynamic obstacles like other objects. This first case is enforced by using a small bin of size 18 cm × 28 cm. The latter case is emphasized by using cubes as the most suitable shape for blocking grasps among themselves. Additionally, we train with wooden cylinders as a second object primitive (Figure 5.30). For our final data set, we trained 25 000 grasps in around 100 h. For learning to shift, we recorded 2500 attempts in around 9 h. We find that grasping and shifting need different amounts of training time; separate training allows for easy stopping at an appropriate success measure and an overall data-efficient recording. Furthermore, we generated data using the combined training approach for the last 10 % of

[1] Available at `https://github.com/pantor/learning-shifting-for-grasping` (accessed on December 9th, 2022)

the training time. For robust and low-noise calculations of ρ, the grasping probability ψ needs to be trained reliably for at least 15 000 grasp attempts.

(a) Training Objects **(b)** Unknown Objects

Figure 5.30: The object sets for training and testing the system's ability to generalize to unknown objects. All objects can be placed within a bin so that they can not be grasped directly.

Shifting Objects

The recorded shifting data set has a mean reward of 0.521 ± 0.112. The trained NN achieves a cross-validated mean squared loss of 0.053, corresponding to a mean error of the grasp probability difference $|\lceil\hat{\psi}\rceil' - \lceil\hat{\psi}\rceil|$ of around 14 %. Setting $\rho_s = 0.6$ is a robust threshold for perceiving empty bins. The output of the NN can be interpreted as a heatmap over the original input image. Figure 5.28 shows qualitatively that the system learned to shift objects either apart or away from the bin's edge for improved grasping. We denote non-graspable positions as object poses, where at least one shift is required before grasping. As given in Table 5.9, on average 2 % of shifts are not sufficient for grasping single objects from those positions. Moreover, shifting objects enabled the robot to empty the bin completely (grasping m out of m objects) throughout the evaluation (Table 5.9).

Table 5.9: Picks per hour (PPH), grasp rate, and shifts per grasp in different bin picking scenarios. In the experiment, the robot grasped n objects out of a bin with m objects without replacement. The random grasp rate was approximately 3 %.

n out of m	PPH	Grasp Rate	Shifts per Grasp	Grasp Attempts
1 out of 1	323 ± 3	100 %	$0\,02 \pm 0\,01$	100
1 out of 1 *(non-graspable)*	170 ± 3	(98.2 ± 1.8) %	1.02 ± 0.02	56
10 out of 10	272 ± 6	(98.4 ± 1.1) %	0.10 ± 0.02	122
10 out of 20	299.6 ± 1.4	100 %	0	120
20 out of 20	274 ± 3	(98.4 ± 1.0) %	0.07 ± 0.01	122

Picks Per Hour

For realistic scenarios like grasping 20 objects out of a bin with 20 objects without replacement, our robotic setup achieved 274 ± 3 PPH. The fewer shifts per grasp are necessary for the bin picking scenario, the higher the PPH. Using the grasp threshold parameter ψ_g, we can adapt the grasp strategy to be more cautious. Let the grasp rate be the number of grasp successes over the total number of grasp attempts. As expected, Figure 5.31 shows

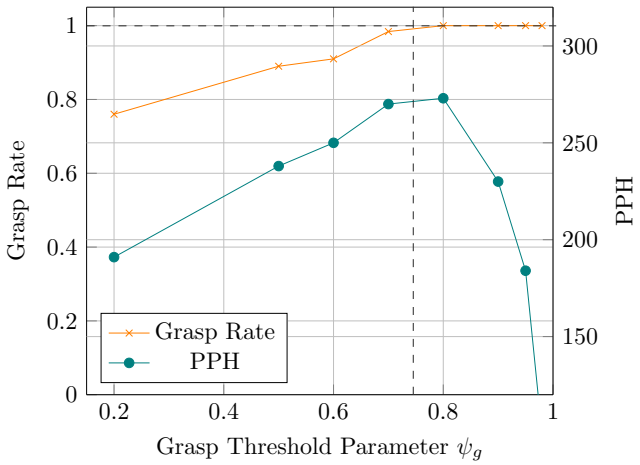


Figure 5.31: Grasp rate and picks per hour (PPH) depending on the minimum grasp probability ψ_g deciding between a grasp or shift. While the robot grasped 10 objects out of a bin with 10 objects without replacement, an optimal threshold $\psi_g \approx 0.75$ regarding PPH was measured.

that a higher ψ_g results in an improved grasp rate. In particular, the system is able to achieve a 100 % grasp rate already for $\psi_g > 0.8$. For this reason, the grasp rate loses its significance as the major evaluation metric. Instead, we can directly optimize the industrially important metric of picks per hour (PPH). At low ψ_g, frequent failed grasps take a lot of time. For high ψ_g, the improved grasp rate comes with the cost of more shifts. Since some shifts might be superfluous and worsen the PPH, an optimal threshold of $\psi_g \approx 0.75$ exists. Interestingly, the corresponding grasp rate is less than 1 in that case.

Unknown Objects

The ability to generalize to novel (unknown) objects is important for extending the range of applications. While training only with cylinders and cubes, we further evaluated the system on the object test set shown in Figure 5.30b. On average, the robot was able to grasp novel objects from non-graspable positions after 1.2 ± 0.3 shifts (Table 5.10). Then, the robot achieved an average grasp rate of (92.1 ± 7.8) %. As expected, we find a qualitative relation between the similarity to the training objects and the success rate of each test object. However, the most common cause of failure is missing depth information from the stereo camera and the resulting confusion by large black regions (e.g. shown in Figure 5.27).

Table 5.10: Grasp rate and shifts per grasp for novel (unknown) objects from non-graspable positions.

Object	Grasp Rate	Shifts	Grasp Attempts
Brush	77 %	1.3	10
Cardboard box	94 %	1	15
Duplo bricks	91 %	1.3	10
Folding rule	83 %	1.1	10
Marker	100 %	1.9	10
Pliers	83 %	1.2	10
Screw driver	83 %	1.2	10
Shape primitives	98 %	1	25
Table tennis balls	100 %	1.1	10
Tape	92 %	1.5	10
Tissue wrap	100 %	1	10
Toothpaste	100 %	1.1	10
	(92 ± 7) %	1.2 ± 0.3	140

Furthermore, the shifting motion primitives are not equally suitable for each object. For example, the marker was usually shifted twice, as it did not roll far enough else. The brush did sometimes rebound back to its original position, which could be prevented by a larger shifting distance.

5.5 Semantic Grasping

Semantic grasping refers to the task of grasping a specific object class out of a scene. Action-centric approaches allow for flexible grasping regarding object types, and even allow for variation within a single scene. Then, targeted semantic grasping might become desirable.

Furthermore, we want to explicitly keep the flexibility of grasping unknown objects in the sense of specifying *unknown object classes* as well. Here, one-shot learning is essential to define the target object class during run-time based on a single label only. In the following, the system takes a first image of the target object class and a second image of the grasping scene. It is then able to output not only the general grasp probability, but the probability of grasping a *specific* target object class. This way, flexible manipulation tasks could be designed on top of a semantic grasping skill.

5.5.1 Object Correspondence Reward

We presume a system and a learned policy for general planar grasping. We extend this system by the concept of *object correspondence* whether the grasped object was an element of the target object class. In this regard, we keep the binary grasp reward r_g (Equation 5.1) to maximize the grasp success. We define the *object correspondence reward* r_o to be

$$r_o = \begin{cases} 1 \text{ if grasped object is in the target class} \\ 0 \text{ otherwise.} \end{cases}$$

The final semantic grasping reward r is the weighted geometric mean of both rewards

$$r = r_g^{\omega}\, r_o^{1-\omega}, \quad \omega \in [0, 1] \tag{5.11}$$

with the hyperparameter weight ω. The overall objective of the robot is to maximize the final reward r, as this corresponds to maximizing both the grasp success *and* the object correspondence. For this, we extend the output of the NN architecture to include both rewards r_g and r_o. The separation allows to learn both rewards independently, and in particular to *reuse* data learned from grasping alone.

We extend the prior approach of planar grasping (section 5.1) using a fully convolutional NN and the orthographic image of the scene s. Again, let $\hat{s}$ be the cropped image window around the gripper pose. We denote the additional image of the target object class (or an instance thereof to be precise) as the object image s_o. In comparison to the bin image, the object image does not necessarily need to be in an orthographic projection. In particular, it is possible to use simple RGB images (such as in Figure 5.33). Given these two images $\hat{s}$ and s_o, the NN architecture should predict the object correspondence reward r_o. The key idea is to simplify the correspondence to a similarity metric based on the temporal causality of the images. During the training process, the robot can grasp objects out of a bin and

take images s_o of the grasped, isolated object. In this case, the images $\hat{s}$ and s_o were from the same grasp measurement and their objects (obviously) correspond. Moreover, we can generate a distribution of false temporal causality by mixing $\hat{s}$ and s_o images from different, randomly sampled measurements. Then, the objects do not correspond to each other and the NN is able to learn that grasp actions at $\hat{s}$ do not retrieve objects pictured in s_o. This way, we reformulate the object correspondence as a contrastive reward r'_o with

$$r'_o = \begin{cases} 1 \text{ if } \hat{s} \text{ and } s_o \text{ were from the same grasp measurement} \\ 0 \text{ otherwise.} \end{cases} \tag{5.12}$$

Both rewards r'_o and r_o are equal if all objects are from different classes. However, there might be multiple measurements of the same object class within the dataset, leading to both false positives and false negatives in the contrastive reward. To mitigate this issue, we apply data augmentation to the object class image. In particular, we translate, rotate, and mirror the images to learn a robust similarity metric between the input and object class image. In short, our approach solves semantic grasping by actually learning to grasp inside the bin so that the robot retrieves an object most similar to the one in the object class image.

5.5.2 Embedding Learning

Information of the scene image s and the object image s_o need to be merged within the NN architecture (Figure 5.32). In particular, the main NN should, as before, be fully convolutional to output the rewards r_g and r_o at a large number of grid-like poses. In contrast, the object class image should be used globally for evaluation, and information thereof should be entirely available at each reward prediction. This constraints the NN architecture to use an intermediate *embedding* of the object image.

To calculate this embedding, the so-called object NN takes the object image as input. As this is a regular RGB image, we apply the popular MobileNetV2 architecture [104] including its pre-trained weights. To allow for faster manipulation training, we incorporate a pre-training strategy to predict useful embeddings without any real-world manipulation data. Here, we apply the contrastive loss as a self-supervised (in the machine learning definition) learning task. Ideally, the embeddings should be similar (or close) for the same object class and diverse (or distant) embeddings for different classes. This way, object classes can be easily compared by downstream manipulation tasks. Let x_1 and x_2 be two embedding vectors. The contrastive loss is then defined by

$$L(x_1, x_2, y) = \frac{y}{2}\|x_1 - x_2\|^2 + \frac{(1-y)}{2}\max\left(0, m - \|x_1 - x_2\|\right)^2 \tag{5.13}$$

with the equality label y, a pre-defined distance m, and the L2-norm $\|\cdot\|$. Minimizing this loss pushes two embeddings of the same class together and else apart. Given a set of various object class images, we randomly select two distinct object classes to create an inequality sample $y = 0$). However, an equality sample $y = 1$ is generated by applying

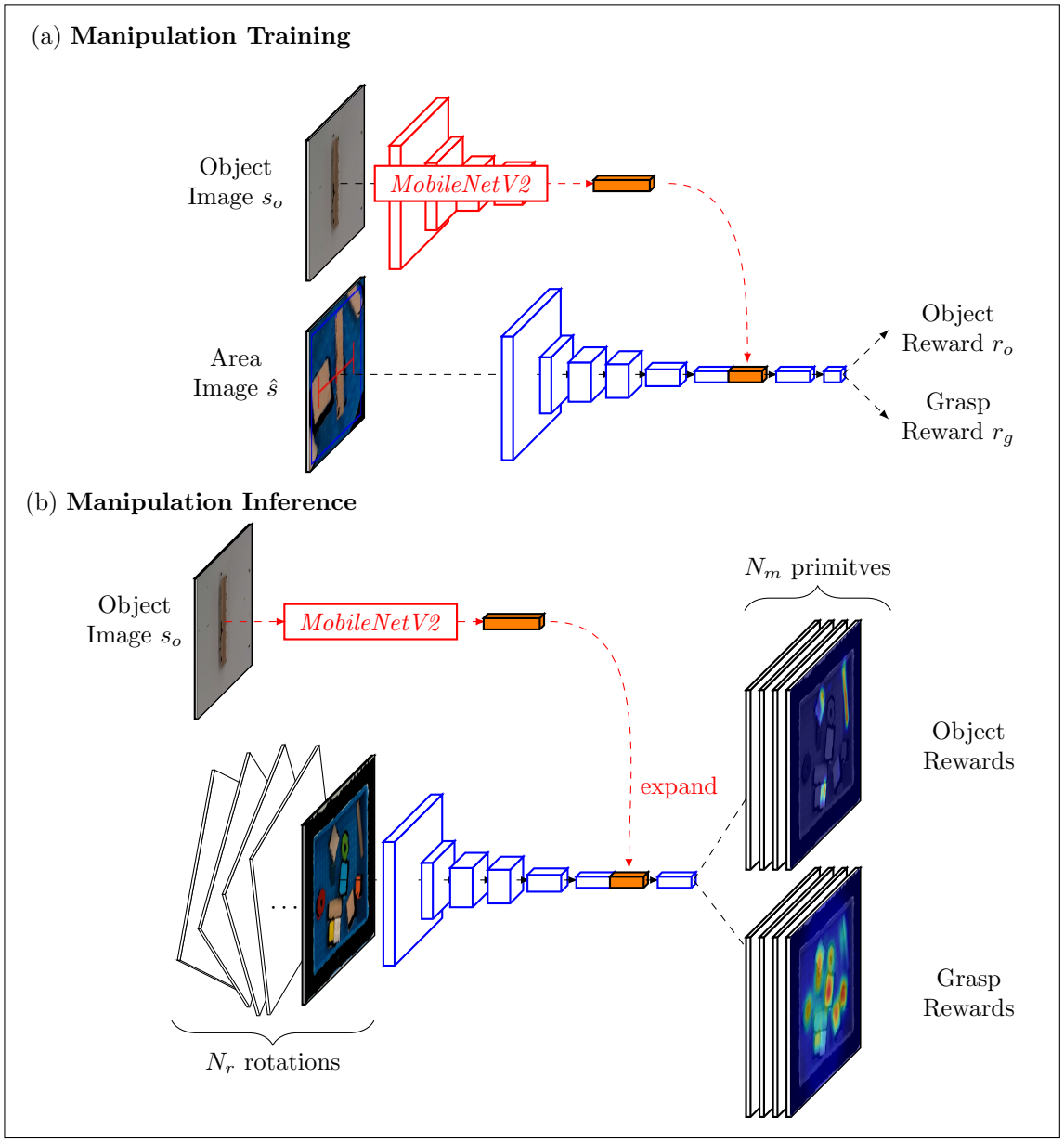
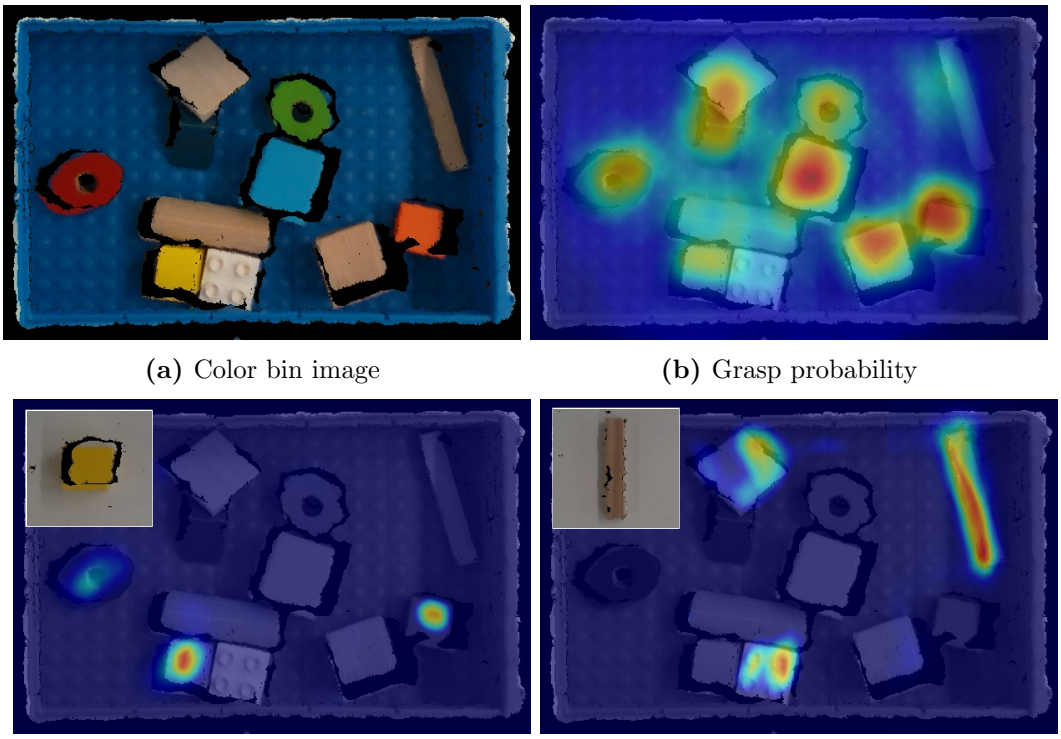


Figure 5.32: The neural network (NN) architecture for learning semantic grasping during manipulation training (a) and inference (b). At first, the object NN (red) learns to predict embeddings from object images for usage in the downstream manipulation task. During manipulation training (a), the manipulation NN (blue) learns to predict the object correspondence reward (whether the object from s_o matches the grasped object from $\hat{s}$) and the grasp success reward. The object NN is kept fixed during this training step, and the embedding is concatenated with an intermediate layer of the manipulation NN. During inference (b), the same NNs are used to predict reward heatmaps for a large number of poses. While the manipulation NN is fully convolutional, the object NN is not, and therefore the embedding must be expanded to match the heatmap size.

distinct data augmentation to the same image, using the same augmentation pipeline as in subsection 5.5.1. We then fine-tune the last two layers of the object NN only to this object pre-training task.

The predicted embedding is then combined with the FCNN, the so-called manipulation NN. The embedding vector is concatenated to the first layer that has a width and height of 1 during training, and the same size as the final reward output during inference. In the final architecture, this corresponds to a concatenation in the third last layer, with two following convolutional layers that have a kernel size of (1×1) respectively (Figure 5.32). The manipulation NN is trained to minimize the loss of the combined reward r similar to Equation 5.2. The object NN is kept fixed. To allow reusing data collected without any semantic labels, we weigh the object reward in the loss function and omit any gradient update during training in case no object image was given. During inference, the final output can be interpreted as two heatmaps, one as a spatial map of general grasp rewards ψ_g and one of the object rewards ψ_o conditioned on the object image s_o (Figure 5.33).



(a) Color bin image **(b)** Grasp probability

(c) Object probability **(d)** Object probability

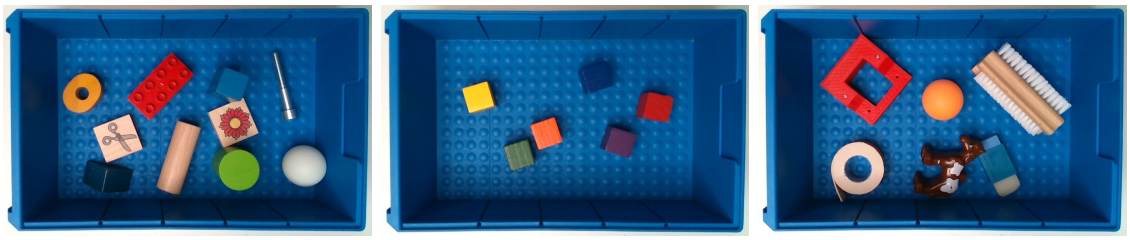
Figure 5.33: Example heatmaps of semantic grasping. Based on the bin image (a), the neural network (NN) calculates the estimated grasp reward (b) and a probability that the shown object will be grasped. Two examples of the yellow cube (c) and the peg (d) are given.

5.5.3 Experimental Results

We evaluate the proposed approach for semantic grasping as a proof of concept on our common hardware setup (section 4.3). The experiments focus on success rates of different selection tasks. Besides the overall grasp success rate, the rate of grasping the *correct* object class is of primary interest.

Training

Training of semantic grasping builds upon a planar grasping policy trained according to section 5.1. Similarly, the robot tries to grasp objects out of one bin and places them randomly in the second bin. This enables a consecutive training, as the robot switches the bins in case no more objects can be grasped. For recording the object image of semantic grasping, a third scene is defined next to the bins on the planar table. After an object is successfully grasped, the robot places the object at this pre-defined pose, by moving towards the table until a force is detected. Then, the robot stops and opens the gripper. The object image is taken from a pre-defined pose. Afterwards, the robot again grasps the object with the previously learned policy and drops it randomly into the filing bin. In general, this procedure allows for a self-supervised training without human intervention.



(a) Trained Objects **(b)** Colored Cubes **(c)** Unknown Objects

Figure 5.34: The object sets for training and evaluating semantic grasping.

A dataset of 400 successful grasps was recorded using the objects shown in Figure 5.34a. The training took around 3 h. The *Top(5)* selection method was applied during training to match the grasp distribution seen in real-world evaluation. Grasp failures were discarded and not included in the dataset. The dataset was recorded without intermediate training of the NN, so no closed-loop feedback typical for RL was available during training.

Selection Rate

We define the *selection attempt rate* as the number of correctly grasped target objects over the number of grasp attempts. In comparison, the *selection grasp rate* is the number of correctly grasped target objects over the number of successful grasps. Table 5.11 shows selection rates for different settings. Similar to grasping alone, the robot needs to grasp n

out of m objects of a given target class out of the bin. If $n > 1$, objects are grasped without replacement, resulting in a higher success rate for random grasps. For example, random grasps for 1 out of 15 objects lead to $1/15 \approx 6.7\,\%$, while 15 out of 15 objects result in a random average selection rate of 22.1 %.

Table 5.11: Success rates of selecting the correct target object in different scenarios of semantic grasping. The robot should grasp n out of m objects without replacement.

Objects	n out of m	Selection Attempt Rate	Selection Grasp Rate	Grasp Rate	Attempts
Colored Cubes	1 out of 6	38 %	40 %	97 %	52
Trained	1 out of 6	88 %	90 %	98 %	52
Trained	1 out of 15	75 %	76 %	97 %	52
Trained	15 out of 15	76 %	83 %	91 %	66
Unknown	1 out of 6	49 %	63 %	77 %	53

For trained objects, a selection attempt rate of up to 88 % in the 1 out of 6 objects configuration was achieved. This rate decreases to 75 % for $m = 15$ objects within the bin, as both the general grasp rate decreases due to clutter, as well as the selection grasp rate decreases due to a larger choice of objects. For trained objects, a selection grasp rate of around 80 % was measured. This rate is halved when grasping only colored cubes (Figure 5.34b), as only color (and no depth) information can be used for distinguishing objects. For unknown objects (Figure 5.34c), we measured a selection grasp rate of 63 %.

Grasp Rate

In general, the grasp success rate decreases in comparison to planar grasping alone, as the object correspondence reward influences the final grasp independently of the estimated grasp success. The parameter ω_o weights the object reward within the final reward for action selection. For $\omega_o = 0$, semantic grasping equals planar grasping. For $\omega_o = 1$, the system outputs a grasp that corresponds only to the estimated object without considering its success. Figure 5.35 shows the grasp success rate, the selection attempt rate, and the selection grasp rate depending on the object reward weight ω_o. As expected, the grasp rate corresponds to the case of planar grasping when $\omega_o = 0$. It decreases with increasing weight, however remains over the random grasp rate for $\omega_o = 1$ by an order of magnitude. At the cost of this decreasing grasp rate, the selection attempt rate increases at a higher object reward weight. As the selection grasp rate is the product of the grasp rate and selection grasp rate, the selection grasp rate achieves a global maximum for $\omega_o \in (0, 1)$. Our experiments show a maximum selection attempt for $\omega_o = 0.5$.

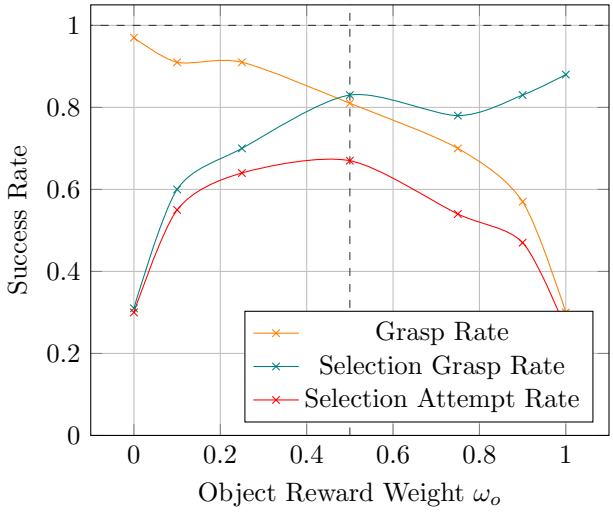


Figure 5.35: The grasp success rate, selection rate over all attempts, and the selection rate over all grasps depending on the object reward weight ω_o. Measurements were taken in the 1 out of 3 setting for both trained and unknown objects.

Inference Performance

Calculating a semantic grasp takes around 200 ms on our hardware, around four times as long as for planar grasping alone. Table 5.12 shows detailed information about the inference performance. The slower calculation for semantic grasping is caused by: (1) rendering and pre-processing of the additional object image, (2) inferring the larger NN, in particular for the object embedding, and (3) taking the square root and product of both rewards. Most of these additional calculations only need to be repeated when a new target object image is used. Otherwise, the object image rendering and pre-processing as well as embedding can be calculated once. Then, the embedding is saved and reused directly in the manipulation NN, increasing the inference performance to around 80 ms.

Table 5.12: Calculation Duration of Semantic Grasping [ms].

	Planar Grasping	Semantic Grasping
Rendering	23.9 ± 0.4	43.5 ± 0.6
Pre-processing	7.6 ± 1.0	9.7 ± 1.3
NN Inference	13.5 ± 0.9	133 ± 8
Selection	2.3 ± 0.1	14.2 ± 1.8
Total	47.3 ± 2.3	200 ± 11

5.6 Discussion

In this chapter, we introduced a general approach for learning several manipulation tasks related to *grasping*. Foremost, we compare the experimental results of the entirely learned 6 DoF grasping based on the fully convolutional actor-critic architecture with the following baselines and related work.

Model-derived Convolution As a simple baseline, we apply a model-derived filter representing the geometric profile of the gripper to calculate planar grasps. The kernel is then convoluted over the depth image of the bin. As the filter rewards nearer depth values between the gripper and farther values around and at the gripper, this approach represents a primitive collision avoidance that favors high objects. In comparison, our method improves the grasp rates of this simple baseline by 23 % on average.

Dex-Net 2.0 Mahler et al. [82] calculate planar grasps targeted for bin picking applications. For our experimental setup, we adapted the rendering of the depth images to the specifications of the pre-trained model. In general, we find the default *CEM* policy to work more robustly than its fully convolutional counterpart. Then, Dex-Net achieves a grasp rate of 82 % for the simple and 73 % for the unknown object set. Note however that the simple object set is also *unknown* to the pre-trained model. Despite setting a corresponding *segmentation mask*, grasp failures are commonly caused by collisions between the gripper and bin. Accordingly, we find that Dex-Net focuses strongly on the local contact between gripper fingers and objects. This is expected, as the NN is learned on analytical data focusing on the grasp points alone. For example, this results in grasps of jammed objects. For unknown objects without (nearby) bin, [82] reported a grasp rate of over 94 %. As commonly found in related work, Dex-Net calculates grasps with a fixed pre-shaped gripper width, which we find to miss many possible grasps in *dense* clutter.

PointNetGPD Liang et al. [75] calculate 6 DoF grasps directly from point clouds. We found two drawbacks regarding our experimental setup: First, PointNetGPD is not applicable to bin picking, but only to a heap of objects on a table surface. Second, it is focused on physically larger objects, so that we were only able to evaluate a selected unknown object set. Moreover, the table needs to be cropped from the point cloud manually. On our setup, PointNetGPD achieves a grasp rate of 69 % for unknown objects. In comparison, the experiments by [75] measured a success rate of over 66 %, significantly less than our proposed method for 6 DoF grasping. Moreover, our approach reduces the grasp calculation by over 100× due to the efficiency of the fully convolutional NN.

Real-world Training Our implementation works on a simple experimental setup and improves time and data efficiency regarding other works trained with real-world experience, e.g. by Pinto and Gupta [94] or Levine et al. [73]. We use a depth camera in contrast to [94, 73, 57] that includes relevant geometric information about the scene and allows

for an orthographic projection. An average grasp attempt lasts less than 15 s, leading to a fourfold reduction compared to [94] with a similar setting. Our algorithm for planar grasps runs at 50 frames per second and outperforms [99], despite their focus on real-time capability. Although our random grasp rate of around 3 % is an order of magnitude lower, the final dataset is only required to be half the size of [94].

Pre-grasping

We presented a real-world solution for self-supervised learning of pre-grasping manipulation to improve the expected grasp success. We find two implications of our work particularly interesting: First, emptying a bin completely is important for industrial applications. Second, we integrated prior knowledge into the learning algorithm by making the reward of one skill (shifting) dependent on the other (grasping). This way, we were able to bypass sparse rewards for data-efficient learning. In contrast, both Kalashnikov et al. [61] and Zeng et al. [128] rewarded grasp success in a time-dependent manner. Additionally, we focused on bin picking scenarios of densely filled, industrial storage bins. For this task, we find multiple gripper openings, neglected by both [61, 128], inevitable. While our approach is more similar to [128], we highlight four concrete improvements: First, we integrated multiple motion primitives into a single NN. Second, our controller can change its readiness to assume risk. This way, our robot achieves arbitrary grasp rates, so that we can directly optimize in relation to picks per hour (PPH). Third, while we find their contribution of training grasping and pushing in synergy necessary, it is not ideal on its own. By splitting both the rewards and training procedures for grasping and shifting, we incorporate different training complexities for different skills. Fourth, our algorithm is around an order of magnitude faster, resulting in increased PPH. Regarding [61], the PPH are fundamentally restricted by their repeating inference and motion steps.

Semantic grasping

The presented approach for semantic grasping works flexibly based on a target image (similar to Jang et al. [58]), and not on pre-defined classes or other priors as in [26, 57, 55]. In this regard, [58] learns a representation based on object persistence, and uses this embedding to identify object instances within the scene. In their prior work [57], they introduced a separation into a general and a target object specific grasp reward, the so-called *dorsal* and *ventral* stream. Moreover, their policy was also trained on real-world experience. In contrast, they control the robot with a planar Cartesian velocity-like control (building upon the work of Levine et al. [73]), and therefore require multiple time steps and large amounts of training data for the grasping strategy. They also reuse data from prior experiences for learning to grasp alone, and can reduce the required data for the *semantic* learning to hundreds of labels per class. Our action space of single-step primitives is closer related to Iqbal et al. [55]. Here, a pipeline for semantic segmentation analyzes the scene and selects a corresponding grasp. Our approach is however learned in an end-to-end manner, making it more robust in dense clutter where segmentation might become challenging.

Chapter 6

Pick-and-place

A common task in robotics is not only the *grasping*, but also the subsequent targeted *placing* of objects. Machine feeding, blister packaging, logistics, or general assembly require the object to be in a specific pose relative to a reference frame. First, the approach should work for unknown objects without a model for great flexibility. Second, it needs to ensure high reliability for picking objects out of dense clutter or an obstacle-rich environment like a randomly filled bin. Third and important for real-world applications, the computation needs to be as fast as possible, all while keeping the desired placing precision.

To approach this problem in the context of primitive-based manipulation, we introduce a learned *place primitive*. In this chapter, we explain the challenges of pick-and-place over grasping (section 6.1). We derive an algorithm for learning planar pick-and-place based on a visual reference that extends our prior approaches for planar grasping. Furthermore, we will specify the action space (section 6.2), the state space observations (section 6.3), and the learned reward function (section 6.4) in detail. The system is evaluated in real-world experiments based on the object placement error and task success rates (section 6.7). This chapter is based on [9].

6.1 Challenges

In comparison to grasping alone, pick-and-place introduces two major challenges: First, the action space of a combined pick and place action becomes exponentially more complex. Second, the reward should minimize the distance to a target pose. This pose needs to be defined even for unknown objects.

Action Space We define a pick-and-place action as a temporal sequence of a grasp and a place (Figure 6.1). Then, the place action is conditioned on the grasp: The place pose after releasing a grasped object depends on the pose of the object within the closed gripper, which in return depends on the grasp pose itself. Moreover, both actions might constrain each other: A target pose might be infeasible to achieve with a given grasp, e.g. due to

collisions with the environment. This makes a *combined* approach necessary. Here, both the pick and place action are calculated simultaneously.

Figure 6.1: A pick-and-place action with a grasp (left) and the subsequent place (right.

Our approaches so far have used a sliding window (e.g. implemented by an FCNN) to evaluate a reward prediction at a discrete grid of actions. The action space for a grasp action commonly includes 10^5 actions. Given the same number for a place action, the combined action space increases to 10^{10} pick-and-place actions. The curse of dimensionality prevents the *naive* approach of evaluating every possible pick-and-place action.

Reward definition The pose of an unknown object without a model is ill-defined, since a reference point (origin) would need to be known. To still allow for flexible applications with unknown objects, we allow a single demonstration to be used as a goal state. This results in an approach in the field of *one-shot imitation learning*. Ideally, the reward should resemble an inverse distance metric (e.g. between an estimated predicted place pose and the target pose). However, calculating a distance based on visual sensor data is challenging - and would again require a rigid object model that might not be available. We achieve a desired action-centric approach by *learning* a similarity metric between an expected and the target state directly within the visual space. In comparison to grasping, the reward becomes more complex than measuring a binary success using the gripper's sensors.

6.2 Pick-and-place Primitives

A pick-and-place action $a \in \mathcal{A}_g \times \mathcal{A}_p$ is a *grasp* $a_g \in \mathcal{A}_g$ following a *place* action $a_p \in \mathcal{A}_p$. We limit both action types to a set of manipulation primitives, given by predefined motions at a specified pose. As planar primitives, each action space $\mathcal{A}_g$ and $\mathcal{A}_p$ is given by $\mathrm{SE}(2) \times \mathbb{R} \times \mathbb{N}$ with actions parametrized by a tuple (x, y, α, z, m) with the planar translation (x, y) parallel to the table surface, the 2D rotation α around the z-axis, and the index of the manipulation primitive m. The height z is analytically calculated from the depth image, similar to our approach for planar grasping in section 5.1. In general, the controller needs to decide where to apply which manipulation primitive m.

We define four *grasp* primitives, which differ in the gripper opening as a pre-shaped gripper width, similar to subsection 5.1.1. We also assume the grasp success definition in-

cluding the measurement just before the place primitive. Here, a single *place* action opens the gripper at a given pose using the same approach trajectory as grasping. In particular, the robot retracts a mm if its force sensors detect a collision during the approach.

While both learning to grasp from a dense reward and finding missing objects in the goal state is comparably easy, the challenge of precise pick-and-place is to find corresponding grasp and place actions. In this regard, we propose an approach to efficiently handle the Cartesian product $\mathcal{A}_g \times \mathcal{A}_p$ of both action spaces.

6.3 State Space

Given the state space $\mathcal{S}$, let s be a set of images in orthographic projection. Each image is either a depth or an RGBD image, depending on the chosen camera configuration. Due to the orthographic projection, each translation or rotation in the image space corresponds to a planar transformation (x, y, α) of the robot in the task space. To observe the entire bin without occlusion, we use top-down views of the scene.

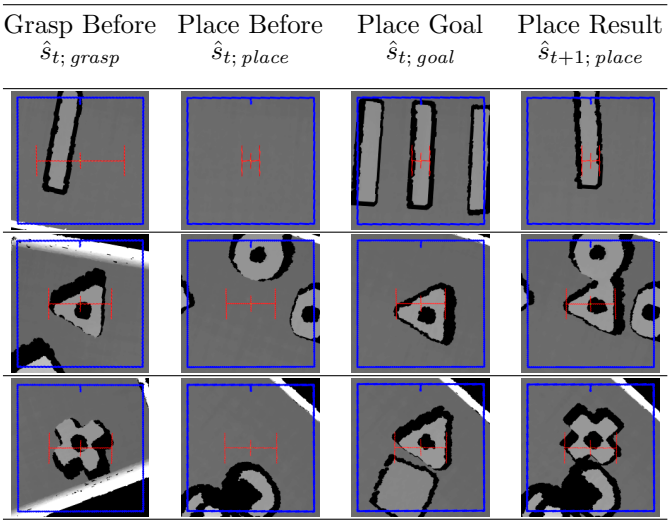


Figure 6.2: Dataset samples from successful pick-and-place actions. Our approach uses four relevant observations, in particular the shown image windows $\hat{[}s]$ of the scene around the robot's tool center point (TCP). The robot grasps an object out of the *grasp before* image, and places it into the *place before* image with a given *place goal* in mind. The subsequent *place result* is only required for training. In a simplified manner, the robot should choose pick-and-place actions so that it cannot differentiate between the *place goal* and the *place result* image.

Let $\hat{s} \subset s$ be an image window around the TCP of the robot, which is defined by the tip of the closed gripper. As common in robot learning for manipulation, we train an FCNN to estimate the action-value function Q_a of an action given the corresponding image window $\hat{s}$. During inference, we use the same FCNN to estimate Q_a at a grid of poses efficiently.

This corresponds to a pixel-wise sliding-window approach for (x, y), while α is calculated by pre-rotating the image inputs. The window's side length l remains roughly equal to the maximal object size. In terms of RL, the policy $\pi(s) = \sigma \circ Q_a(s, a)$ can be defined by a so-called selection function σ composed with the action-value function Q_a. We make use of selection strategies introduced in section 3.7.

We consider the grasp and place actions to be in different scenes. Importantly, all images of the same scene are taken using the same top-down camera pose. Then, we define the state space $\mathcal{S}$ by the set s of following three observations:

1. $s_{\texttt{grasp}}$, state of the grasp scene before the grasp a_g,
2. $s_{\texttt{place}}$, state of the place scene before the place a_p, and
3. $s_{\texttt{goal}}$, goal state of the place scene. Usually, this is an image of the scene configured by a human demonstrator.

Furthermore, the state of the place scene after the place a_p is a fourth important observation. It is denoted as $s_{t+1;\texttt{place}}$ and can be understood as the *result* of a pick-and-place action. It is only available and required for training. Examples of the four relevant observations are shown in Figure 6.2.

6.4 One-shot Imitation Learning

Since the robot should imitate the given goal state, the reward r needs to capture the similarity between $s_{\texttt{goal}}$ and the expected $s_{t+1;\texttt{place}}$, conditioned on both before states $s_{\texttt{grasp}}$ and $s_{\texttt{place}}$. In general, the robot should choose those actions that most probably result in the *place goal* image as the real *place result* image. We interpret this as a task of *density estimation* and apply noise contrastive estimation (NCE) [48] to this unsupervised learning problem. Given a dataset D of pick-and-place actions a, a model is trained to discriminate between the real data D (with label $C = 1$) and a noise distribution Q (with $C = 0$). Then, a classifier is able to measure the probability r_p of an image $\hat{s}$ being a realistic outcome of a pick-and-place action:

$$r_p(\hat{s}) := p(C = 1 | \hat{s}, \hat{s}_{\texttt{grasp}}, \hat{s}_{\texttt{place}})$$

We refer to this probability as the *place reward* $r_p \in [0, 1]$. Besides, the estimated *grasp reward* $r_g \in [0, 1]$ of the binary grasp success can be interpreted as a grasp probability. For NCE, the design of the noise distribution Q is crucial for estimating the density of the data D. For each sample in D with a given grasping image $\hat{s}_{\texttt{grasp}}$, we augment the remaining pair of place images. For the data distribution ($C = 1$) in a pick-and-place task, we generate two positive samples:

Hindsight Foremost, the real measured pair of $\left(\hat{s}_{\texttt{place}}, \hat{s}_{t+1;\texttt{place}}\right)$ is the basic positive sample in the data distribution.

Further Hindsight If further objects are placed into the same bin for $t_{\texttt{bin}}$ steps after the current pick-and-place action, the images $\left(\hat{s}_{\texttt{place}}, \hat{s}_{t+n;\texttt{place}}\right)$ for $1 \leq n \leq t_{\texttt{bin}}$ are used as positive samples as well. This sample enables the robot to consider future actions when multiple pick-and-place actions are needed to achieve a goal state, e.g. for box stacking.

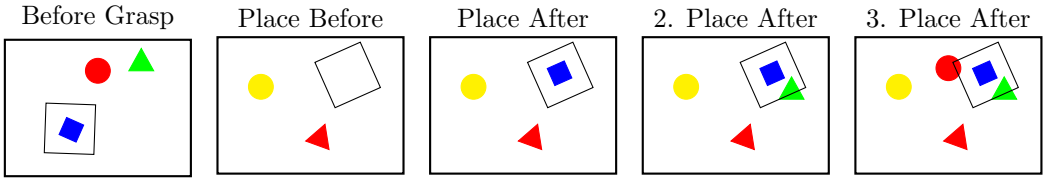


Figure 6.3: Further hindsight: Subsequent episodes are also used as positive examples in the contrastive loss. Then, other objects might be added to the reference window.

For the noise distribution ($C = 0$), we generate a range of negative samples as follows:

Negative Foresight We create negative pairs using either identical images (for both $\hat{s}_{\texttt{place}}$ and $\hat{s}_{t+1;\texttt{place}}$) or positive pairs in the wrong temporal order $\left(\hat{s}_{t+1;\texttt{place}}, \hat{s}_{\texttt{place}}\right)$.

Augmented Hindsight Moreover, we generate additional negative samples by jiggling the pose of the place images $\hat{s}$. In particular, we jiggle real hindsight images randomly with a minimum displacement of 1 mm or 3 °.

Goal The goal image pair $\left(\hat{s}_{\texttt{place}}, \hat{s}_{\texttt{goal}}\right)$ is used as a negative sample. As the training progresses and accuracy improves, both images should converge and lead to a median contrastive loss. To circumvent this effect, we apply methods of *confident learning*: If a trained classifier predicts a goal image sample to be from the real data distribution with given certainty, we remove the goal image from the noise distribution furthermore.

Other Hindsight Finally, place images of independent actions a_g and a_p are used. In particular, this results in mismatching object types as negative samples.

Using NCE, the imitation learning problem simplifies to ordinary supervised learning. The contrastive loss is defined using the BCE

$$\begin{aligned} y_i &= z_i - \log p(i) \\ \text{BCE} &= \sum_i^N C_i \log \sigma(y_i) - (1 - C_i) \log(1 - \sigma(y_i)) \end{aligned} \tag{6.1}$$

with the logits z_i adapted by the probability p of the sample i, and the sigmoid function σ. We train a system of three NNs by minimizing the BCE of the contrastive loss (Equation 6.1) as well as the binary grasp reward.

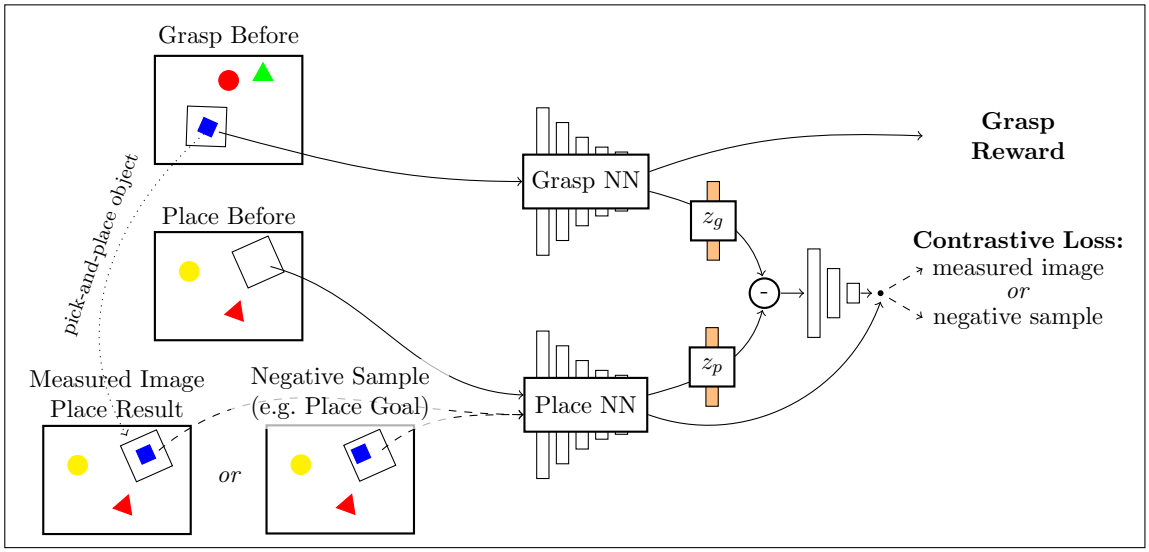


Figure 6.4: Our neural network (NN) architecture during training: Given the images of both the grasping and placing scene, a system of three NNs predicts whether a given third image is a real measured image based on an executed action or a negative sample such as the goal image. While both the *Grasp NN* and the *Place NN* are limited to their corresponding scene, a third NN predicts a combined contrastive loss via the difference between the z_g and z_p embeddings. We interpret this contrastive loss as a *place reward*. Additionally, the fully convolutional Grasp and Place NN predict the grasp success and the place reward as an initial estimation, respectively.

6.5 Model Architecture

Three constraints are applied to our NN architecture (Figure 6.4): First, we define two separate *grasp* and *place* NNs as FCNNs and limit their input to their corresponding scene. Second, the grasp NN predicts the grasp reward r_g as an additional output. Similarly, the place NN pre-estimates the place reward, however without information about the grasp action. We denote this prediction by the place NN as r_{p1}. Third, both NNs calculate action embeddings z_g or z_p respectively. They are combined in the (non-convolutional) *merge* NN using their element-wise difference. Since information from both the grasp and the place scene are joined here, the place reward r_p can then be predicted with significantly improved accuracy.

During the inference phase, we first rotate the images s of the grasp and place scene given a discrete set of rotations (Figure 6.7). The image batches are fed into their corresponding FCNN, resulting in action rewards r_g and r_{p1} with their corresponding embeddings z_g and z_p for a discrete set of action poses. The final action space $\mathcal{A}$ for pick-and-place scales by the number of combinations $\mathcal{A}_g \times \mathcal{A}_p$. For performance reasons, not all possible combinations can be evaluated in the merge NN. Therefore, a small fraction of grasp and place actions

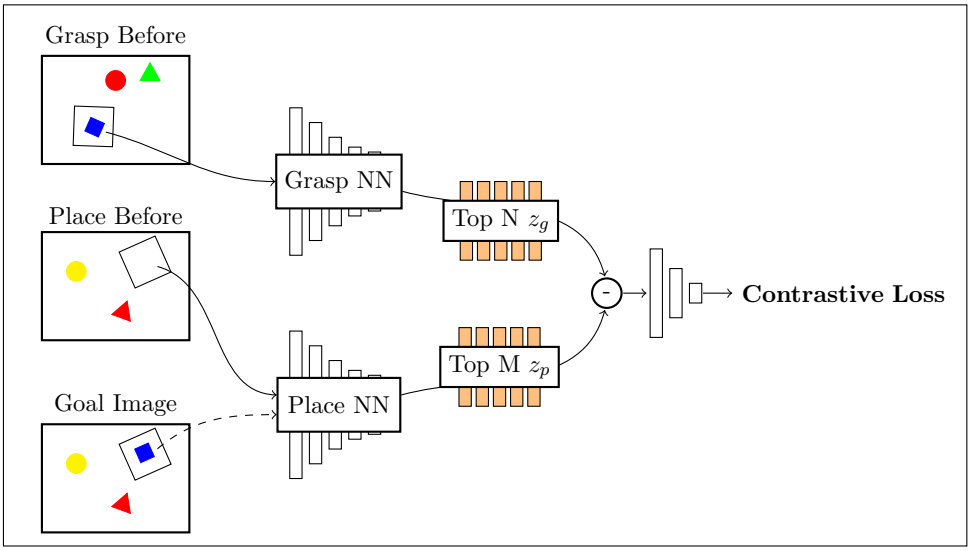


Figure 6.5: Our neural network (NN) architecture during inference: Given the images of both the grasping and placing scene, a system of three NNs predicts whether a given third image is a real measured image based on an executed action or a negative sample such as the goal image. While both the *Grasp NN* and the *Place NN* are limited to their corresponding scene, a third NN predicts a combined contrastive loss via the difference between the z_g and z_p embeddings. We interpret this contrastive loss as a *place reward*. Additionally, the fully convolutional Grasp and Place NN predict the grasp success and the place reward as an initial estimation, respectively.

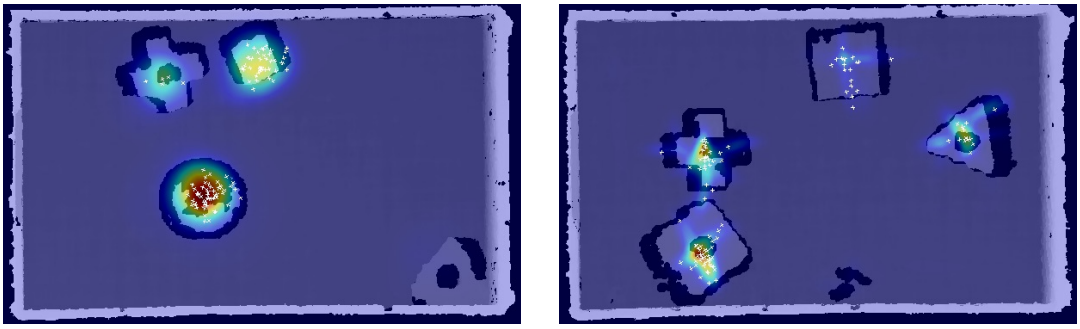
are pre-selected using the probability distribution

$$p(a_g|\psi_g) = \frac{\psi_g^\alpha}{\sum_{\mathcal{A}_g} \psi_g^\alpha}, \quad p(a_p|\psi_{p1}) = \frac{\psi_{p1}^\alpha}{\sum_{\mathcal{A}_p} \psi_{p1}^\alpha}, \quad \alpha > 1.$$

based on the estimated grasp reward ψ_g and the pre-estimated place reward ψ_{p1}. Furthermore, we set $\alpha = 6$ and sample $N := N_g = N_p = 200$ action proposals without replacement, respectively. Figure 6.6 illustrates the grasp and place rewards r_g and r_{p1} as heatmaps, moreover indicating exemplarily positions of sampled action proposals.

For each combination of a proposed grasp a_g and place a_p, the merge NN predicts the refined place reward r_p from the embedding difference $z_g - z_p$. Since we only train the place and merge NN on pick-and-place actions with successful grasps, the place reward r_p is conditioned on $r_g = 1$. This way, we extend common approaches for learning to grasp incrementally, and are able to learn the grasp confidence with additional or prior grasp data *independently*. Finally, the pick-and-place reward $r = r_g \cdot r_p$ is defined by the product of grasp and place reward.

One of three selection functions σ is applied to the set of N^2 proposed pick-and-place actions: First, a uniform random selection for initial exploration. Second, a sampling-based approach based on *Prob(N)* using $a_g, a_p \sim \psi_g \cdot \psi_p$. Third and most important, an ε-greedy



(a) Grasp reward ψ_g **(b)** Place reward ψ_{p1}

Figure 6.6: Example heatmaps of the estimated grasp and place reward, ranging from low (blue) to high (red). The estimations of each FCNN are averaged over rotations. For visualization, we reduce the number of sampled action proposals to $N = 80$ (white dots).

selection method *Top(N)* $a_g, a_p = \arg\max \psi_g \cdot \psi_p$ is used for application. Then, given the scene images, the robot chooses the grasp and place combination that makes the scene after the executed action look *most* like the goal image.

6.6 Self-supervised Learning

In order to scale the real-world training, the learning process needs to work with minimal human interaction (chapter 4). Therefore, two bins are used for continuous and diverse training. We apply a simple curriculum learning strategy, starting with single objects and increasing the number and complexity of object types over time. In the beginning, we sample grasps from a given grasping policy and place objects in the second bin randomly. Later on, we sample goal states from a range of previously seen before states, denoted as the goal database. As our approach is off-policy, we train the NN using the most current dataset in parallel to the real-world data collection. Then, we sample the pick-and-place action with the *Prob(N)* strategy, increasing N over time. At the end of the training, we switch to the *Top(N)* selection strategy.

Optionally, the training of the grasp reward can be improved using single grasps without place action. As pick-and-place is an extension to the planar grasping task (section 5.1), the training can be bootstrapped by reusing prior data.

6.7 Experimental Results

We use the hardware setup introduced in section 4.3 for the following experiments.

Data: Images $s_{\texttt{grasp}}$, $s_{\texttt{place}}$, $s_{\texttt{goal}}$
Result: Grasp a_g, Place a_p

1: Create set of rotated grasp images S_g
2: Create set of rotated place and goal images S_p
3: Calculate grasp rewards ψ_g and embeddings z_g for each pose in S_g using grasp FCNN
4: Calculate place rewards ψ_{p1} and embeddings z_p for each pose in S_p using place FCNN
5: Sample N_g grasps z_{gi} with probability $(\psi_{g,i})^\alpha$
6: Sample N_p places z_{pj} with probability $(\psi_{p1;j})^\alpha$
7: **for** each combination (z_{gi}, z_{pj}) **do**
8: Subtract z_{gi} and z_{pj} element-wise
9: Calculate final place reward ψ_p using merge NN
10: **end for**
11: Select actions greedily via $a_g, a_p = \arg\max_{ij} \psi_g \cdot \psi_p$

Figure 6.7: Algorithm of inferring a pick-and-place action using FCNNs and reward pre-estimation.

6.7.1 Data Collection and Training

We evaluate our approach with two distinct models: First, a *specialized* model for pick-and-placing screws (M10×60) using RGBD images. This model was trained on around 3500 pick-and-place actions. We reuse 12 000 sole grasps from prior experiments for improving the grasp reward estimation of the screw model. Second, a *general* model was trained for all remaining object types and experiments. Due to reliability issues of the RealSense camera, this model uses only depth images of the Ensenso N10. It was trained on wooden primitive shapes with side lengths of ≈ 4 cm for around 25 000 pick-and-place actions, corresponding to around 120 h.

Both grasp and place NNs are fully convolutional and share the first few layers between the reward and embedding outputs, respectively. The merge NN is a three-layer dense neural network. We double the loss weight of the final place reward and optimize the NNs using Adam [63] with a learning rate of 2×10^{-4}. After 100 epochs, we remove goal images with a predicted contrastive loss of above 0.7 from the training set. For further details, we refer to the open-source implementation[1].

6.7.2 Object Placement Error

The precision of the pick-and-place task is evaluated using the placement error of a single object. We define this error as the distance between the object's pose within the goal state and the pose after the executed pick-and-place action. Given the robot's high repetition accuracy and a well-calibrated depth-camera, the goal and result images are taken from the

[1] Available at `https://pantor.github.io/learning-pick-and-place/` (accessed on December 9th, 2022)

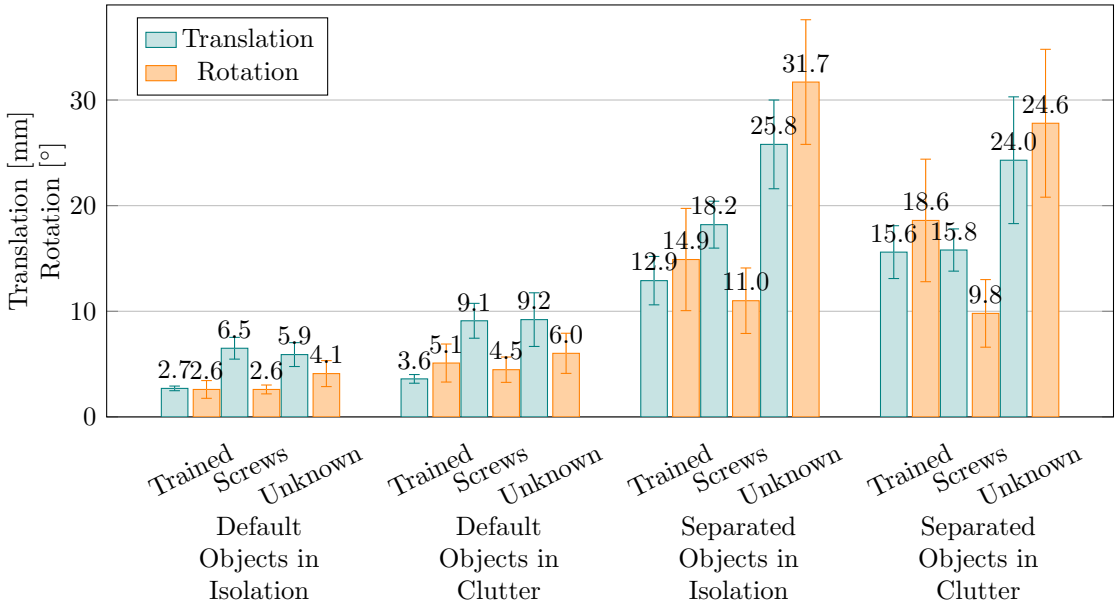


Figure 6.8: The translational and rotational mean placement errors in different settings. We compare the *default* case (N=200) with the *separated* case where the best grasp and best place actions are chosen independently (N=1). Here, N is the number of proposed action embeddings for further combination. Moreover, we differentiate between experimental results for grasping isolated objects and out of clutter.

same camera pose. Then, we measure the placement error by determining the 2D transformation bringing $\hat{s}_{\texttt{goal}}$ and $\hat{s}_{t+1;\texttt{place}}$ into alignment.

Figure 6.8 shows the translational and rotational placement error in various settings. The placement error is investigated for isolated objects as well as objects in clutter. For the latter, we fill the grasping bin with 25 trained objects, 80 screws, or 10 unknown objects respectively and measure only places of the correct object type. Additionally, we compare the results of our proposed default system with a separated approach: Then, we set $N = 1$, leading to a system that chooses the best grasp action and the best place action independently of each other, and does not make use of the merge NN. For trained objects, our robot achieves an average placement error below of 3 mm and 3 °. Interestingly, the translational error falls just below the placing resolution. We assume the worse results for screws to be caused by less training data and a more complex, e.g. reflective visual appearance. For unknown objects in our test set (Figure 6.9), the robot is still able to achieve an average precision of around 6 mm and 4 °. In clutter, the precision decreases in particular for screws and unknown objects. Here, typical grasp rates lie around 95 % for trained objects, and around 85 % for unknown objects. Over all experiments, we find an average error of 1.6 mm in the direction of the gripper jaws constraining the object, and 3.9 mm orthogonal thereto. These findings suggest that the initial object displacement

Figure 6.9: The evaluation set of 50 unknown objects.

caused by the closing gripper might influence the placing precision significantly.

6.7.3 Insertion Task

Although the robot did not learn to insert objects directly, we investigate this capability despite tolerances of around 1 mm using the peg game (Figure 6.1), pushing the robot to its pick-and-place precision limits. On average, the robot achieves a success rate of 72 % for the insertion of isolated objects, however depending heavily on the object type (Table 6.1). For example, we observed that predicting the displacement of the cross shape

Table 6.1: Success rates of inserting a given object onto a peg

	Default		Separated	
Object	Isolation	Clutter	Isolation	Clutter
Circle	9 / 10	7 / 10	5 / 10	1 / 10
Triangle	8 / 10	4 / 10	0 / 10	0 / 10
Square	9 / 10	7 / 10	1 / 10	0 / 10
Oval	6 / 10	6 / 10	3 / 10	0 / 10
Cross	4 / 10	2 / 10	1 / 10	0 / 10
	72 %	52 %	20 %	2 %

during clamping is difficult to do - again suggesting that the grasp displacement is a primary source of imprecision. The system can increase success rates by around 50 % in comparison to a separated approach. Moreover, the peg game also represents a perturbation of the environment. Success rates of up to 90 % suggest that the system can generalize to unknown environments. We classify a wrong object type while grasping in clutter as a failure.

6.7.4 Selection Error

Given multiple object types in the grasping scene, the task of selecting the correct object to place arises naturally. We evaluate this task by choosing five objects randomly from the

set of either all known or unknown (Figure 6.9) objects. Then, the robot needs to select the solely shown object from the goal state. For this 1-out-of-5 selection task, the robot achieves a success rate of $(86 \pm 5)\,\%$ for trained, and $(60 \pm 7)\,\%$ for unknown objects, in comparison to a random success rate of $20\,\%$ (Figure 6.10).

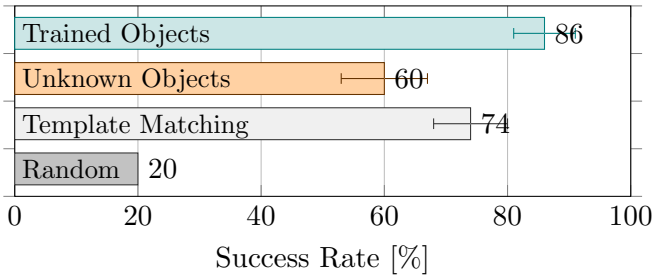


Figure 6.10: Success rates of grasping the demonstrated objects out of a set of five distinct objects (1-out-of-5 selection task), independent of the final place precision. This comparison uses depth images only.

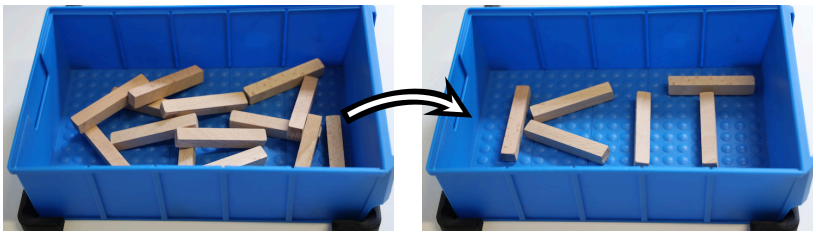
However, the evaluated model does not make use of color images, which we assume to be helpful for this task. We additionally compare our model against a simple *template-matching* baseline. First, the target object is detected using the difference between the goal and place image. Second, we apply this template to match a corresponding object within the grasp image. Although we find that this baseline is sensitive to depth shadows, it still outperforms our approach for unknown objects.

6.7.5 Multiple-step Tasks

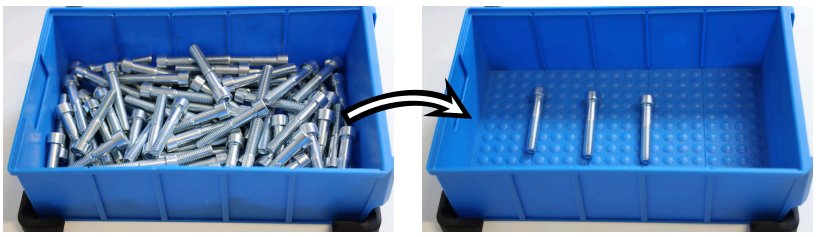
Our approach allows to infer multiple actions from a single goal image when updating the grasp and place images after each action. Given a single demonstration, the robot can place multiple objects out of clutter, with examples shown in (Figure 6.11a and 6.11b). Moreover, we can take a sequence of goal images as an instruction list. This allows a wide range of easy-programmed pick-and-place tasks (Figure 6.11c). Videos of all three examples are included in the supplementary material linked above.

6.8 Discussion

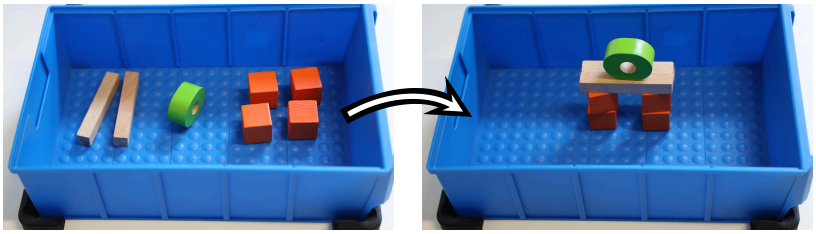
We presented an approach for learning pick-and-place tasks in a self-supervised manner. Since our approach does not depend on an object model and instead takes a demonstrated goal state as an input, it allows for the flexible yet precise placing of unknown objects. The evaluated system was trained in the real world with up to 25 000 pick-and-place actions, resulting in average placement errors of $(2.7 \pm 0.2)\,\mathrm{mm}$ and $(2.6 \pm 0.8)\,^\circ$ for trained objects. A separated approach, where grasp and place actions are calculated independently, results in four to five times lower precision. For unknown objects, the translational placement error



(a) Placing the logo of our Alma Mater (KIT) (6 actions, 1 goal).



(b) Placing screws in an industrial scenario (3 actions, 1 goal).



(c) Building a house with wooden blocks (6 actions, 4 goals).

Figure 6.11: Given a single goal state, our robot can infer *multiple* pick-and-place actions from the (cluttered) grasp scene (left) to the place scene (right). Using an instruction list of multiple goal states, the robot can reproduce more complex examples (c).

increases to around 6 mm and 9 mm for grasping in clutter. The robot learns to select the demonstrated objects out of five alternatives with up to 86 % accuracy. A second model was trained specially for screws in an industrial use case. Moreover, we demonstrated the robot's capability to pick and place multiple objects from a single goal state.

We see our system as a combination of two common approaches in robot learning: First, we build upon learning of manipulation primitives, estimating their reward for planar poses using FCNNs. So far, we found that this was mostly done for grasping or pre-grasping [82, 130, 129], as the reward can here be defined and measured more easily. Second, we integrate methods of one-shot imitation learning. In particular, the work of Singh et al. [110] used a contrastive loss approach to classify goal states from policy-generated, self-executed states. Similarly, we use a demonstrated goal state to define a precise object pose for placing.

Regarding other approaches to pick-and-place, we see both advantages as well as shortcomings. In comparison to Finn et al. [39], we limit our policy to a single time step and a discrete action space. While this diminishes the generality and ignores the trajectory in between, our approach increases the final object precision significantly. Moreover, we extend a common, state-of-the-art approach of learning for grasping, i.a. resulting in the capability of pick-and-place out of clutter. Due to the use of manipulation primitives, our trained models depend only on the gripper and generalize in principle to other robot arms.

Gualtieri et al. [47] learn a policy for pick-and-place in full 6 DoF. Despite their advantages over planar manipulation, we see shortcomings in the restricted generalization capability as well as a reward-based placing objective. In contrast, our approach allows for wider generalization, easy training of additional object types, and flexible place poses. Additionally, our robot is limited by its data consumption of real-world training in comparison to learning in simulation.

As a part of a pick-and-place pipeline, Zhao et al. [132] predicted the object displacement during the gripper's closing action. We found that this displacement is a major source of imprecision in our experiments. Still, our overall pick-and-place precision is similar to their displacement prediction error. In comparison, our contributions allow to learn the entire pick-and-place pipeline at once. Their approach would still require a pose estimation of the object model and a grasp point detection for targeted placing. Furthermore, our approach was also evaluated in clutter.

From a more general point of view, we proposed a method for learning two interdependent actions in an open-loop manner. We find that the presented ideas could be transferred easily to other applications. In fact, chapter 8 uses bimanual manipulation and is based on the idea of pre-selection and combining corresponding embeddings. Here, the two actions are executed simultaneously instead of successively.

Chapter 7

Transition Model for Composed Primitives

So far, the approach of manipulation primitives has reduced the controller to a single time step. However, the discarded time dependency is often useful for practical applications. Amongst others, it allows to plan longer manipulation sequences or second, optimize for time-dependent multiple-step criteria. In this chapter, we propose to keep learning the manipulation primitives in a single-step and data-efficient manner, while introducing a transition model on top. This visual transition model is learned from pairs of images *before* and *after* an executed manipulation action, and is then able to *predict* the resulting state of an action. As a transition model adds errors into the system, we further predict the uncertainty of the transition model. By estimating the final uncertainty of the manipulation action, the robot can manipulate in a *risk-aware* manner. As before, we address this to the robotic task of *bin picking*. It highlights several challenges for transition models, e.g. unknown and partially hidden objects, as well as an obstacle-rich environment. We *compose* manipulation primitives from planar grasping primitives (section 5.1) and pre-grasping (section 5.4) to complex primitive sequences. This chapter is based on [11].

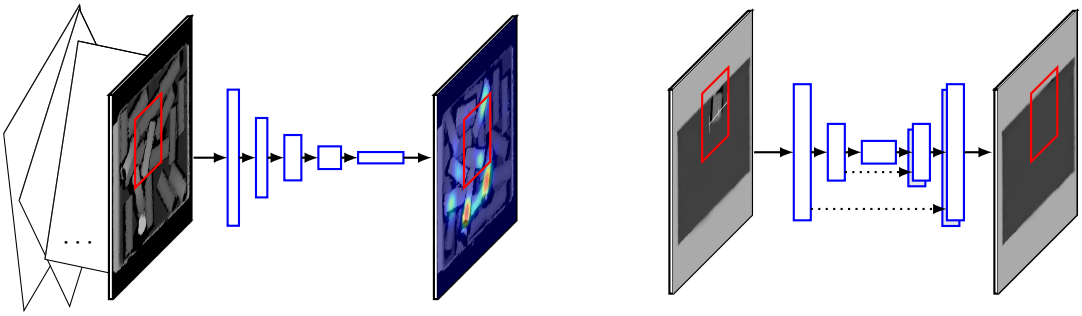
7.1 Model Architecture

In RL, a transition model $\mathcal{T}$ predicts the distribution over states s_{t+1} at the next timestep

$$p(s_{t+1}) = \mathcal{T}(s_t, a_t) \tag{7.1}$$

based on the current state s_t and the chosen action a_t. A transition is generally non-deterministic to reflect real-world stochasticity and incomplete knowledge that might not be available to the agent. As the states s_t and s_{t+1} are in the visual domain in this work, our transition model maps one image to another image.

Section 3.5 analyzed the locality of manipulation actions. In this regard, all prior approaches have simplified manipulation planning to take only an area around the action's



(a) Manipulation Model M: The fully convolutional manipulation NN is trained on an image window only, but is able to predict a reward heatmap averaged over rotations and primitive types for a complete image of the bin.

(b) Transition Model T: The *U-Net* architecture predicts the next image state s_{t+1} depending on the prior state s_t, the action a_t, and its reward r_t. The predicted window around the action (red) is then patched into the original image.

Figure 7.1: We make use of two NNs, one for the *manipulation* policy (a) and one for the *transition* model (b). The locality of manipulation was so far used to restrict the manipulation model to a window around the TCP of the robot (red). Likewise, the same arguments allow us to simplify our *local* transition model.

pose into account. For example, a NN evaluating a primitive's reward takes a cropped image window $\hat{s}$ and not the complete image s of the bin as input. The same arguments hold true for the transition model $\mathcal{T}$. In this regard, we learn to predict only a window $\hat{s}_{t+1}$ around the gripper's pose for the next step. Likewise, a *local* transition model has two major advantages: First, while the window size could in principle match to the overall image s, we limit the window size to twice the maximal object size. This reduces the size of the NN or the resolution of the transition model. Second, we center the robot's TCP within the image windows. This way, all spatial dependencies (x, y, α) can be removed from the transition model. To estimate the distribution over s_{t+1}, the state image s_t is patched with $\hat{s}_{t+1}$ at the affine transformation (x, y, α)

$$p(s_{t+1}) = \mathcal{T}(s_t, a_t) \approx \mathcal{T}(\hat{s}_t, m_t, r_t) \tag{7.2}$$

with the manipulation primitive type m_t. Additionally, we condition the transition model on the reward r_t of the action. This is motivated by the following arguments:

- By using the estimated reward ψ_t instead, the calculations of this difficult and high-level feature can be outsourced from the transition model to the already learned manipulation NN. This is particularly important for grasp actions: Here, the next states depend strongly on the binary grasp success. Therefore, we train the manipulation model to learn the grasp success explicitly (as it is measurable) and reuse this information in the transition model.
- Real reward measurements r_t can be used in the transition model. For example, the grasp reward can be quickly measured by the robot's gripper.

- Note that there is no loss of generality by conditioning the transition model on the reward, as we can substitute the manipulation model as a reward predictor.

If not stated otherwise, we simplify the transition model to

$$p(s_{t+1}) \approx \mathcal{T}\left(\hat{s}_t, m_t, M(\hat{s}_t)\right). \tag{7.3}$$

Let $\hat{s}'_{t+1}$ denote the image prediction with the highest probability.

We employ the BicycleGAN architecture by Zhu et al. [133] for our transition model. In a nutshell, the architecture is based on a generator, discriminator, and an encoder NN. The generator tries to mimic images from the underlying distribution, taking a noise vector z from a prior latent distribution as input. While playing a min-max-game, the discriminator aims to distinguish the generated data from the real data. The BicycleGAN introduces an encoder to calculate a representation from an image within the latent space z. By constraining the latent distribution between the real and generated images, as well as a similarity to a Gaussian, the architecture is able to output a multi-modal distribution. This way, the BicycleGAN architecture is in principle able to capture the stochastic physics as a transition distribution.

We use the generator NN as the transition model $\mathcal{T}$. It uses a *U-Net* architecture with (64×64) pixel input and output size (Figure 7.1b). We extend the BicycleGAN architecture by additionally condition the generator on the manipulation primitive type m_t (as a one-hot encoded vector) and the reward r_t.

7.2 Uncertainty Prediction

Given the image distribution $p(s_{t+1})$, we are able to estimate a pixel-wise uncertainty of the prediction by sampling from its latent space z. Let the uncertainty $\hat{\sigma}_s^2$ be the variance of the prediction $\hat{s}'_{t+1}$

$$\hat{\sigma}_s^2 = \mathbb{E}\left[\left(\hat{s} - \hat{s}'_{t+1}\right)^2\right] = \sum_i^N p(\hat{s}_i)\left(\hat{s}_{t+1,i} - \hat{s}'_{t+1}\right)^2 \tag{7.4}$$

given the number of samples N. The state uncertainty $\sigma_{s,t+1}^2$ of the overall image is estimated by zero-padding the uncertainty window $\hat{\sigma}_s^2$ at (x, y, α). The state uncertainty σ_s^2 can be propagated towards the estimated reward ψ_t of the manipulation action. A first-order Taylor series of the manipulation NN M yields

$$\sigma_a^2 \approx \sum_{i,j} \left|\frac{\partial M(\hat{s}'_{t+1})}{\partial s_{i,j}}\right|^2 \sigma_{s;i,j}^2 \tag{7.5}$$

summing over all pixels (i, j). The approximation presumes a zero covariance between individual pixels and only small uncertainties σ_s^2. A key contribution of our work is to

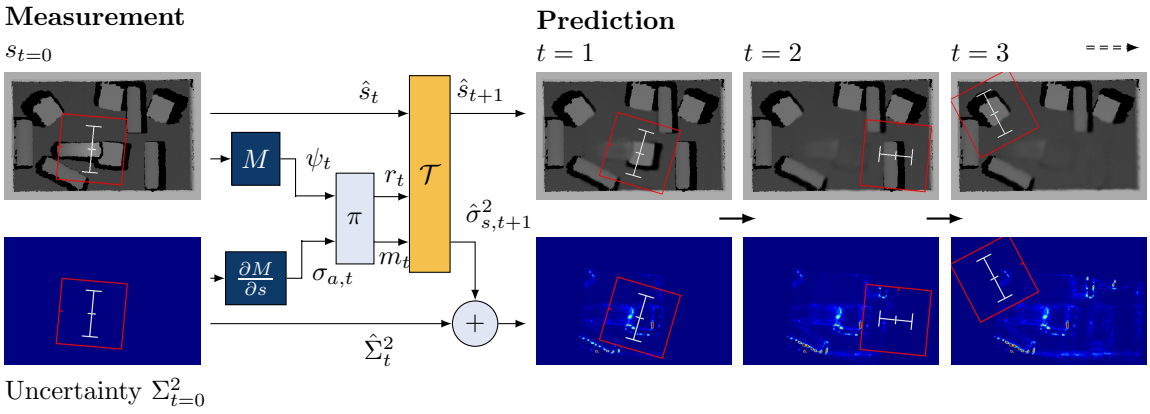


Figure 7.2: Iterative multiple-step predictions of the state s_t with corresponding uncertainty Σ^2_t given an initial measurement $s_{t=0}$. The computational flow during a prediction step is as follows: The manipulation neural network (NN) M estimates rewards ψ_t for a discrete set of action primitives m_t, its gradients are used to propagate the cumulative input uncertainty Σ^2_t towards the estimated reward $\sigma_{a,t}$. The policy π chooses the final action a with corresponding reward r using the lower bound of the estimated reward. Then, the transition model $\mathcal{T}$ predicts the new state $\hat{s}_{t+1}$ around the action a with pixel-wise uncertainties $\hat{\sigma}^2_{s,t+1}$. The uncertainties are summed up ranging from low (blue) to high (red).

implement Equation 7.5 in a fully convolutional way: The gradient of the fully convolutional manipulation NN regarding the pixel-wise image input is fully convolutional itself. Then, the gradient is convoluted over the mean prediction and multiplied with the pixel-wise uncertainty. This results in an efficient computation of both the estimated rewards and its uncertainties $\psi_t \pm \sigma_a$. Note that we only consider the input uncertainty; other uncertainties, in particular from M itself, are neglected.

7.3 From Predicting to Planning

Following, we assume the independence of each state's uncertainties $\hat{\sigma}_{s,t}$ over time t. Due to the linearity of the variance, we define the cumulative uncertainty

$$\Sigma^2_t = \sum_{i=0}^{t} \sigma^2_{s;i} \tag{7.6}$$

as the accumulated uncertainty of the state. Furthermore, we denote t as the number of action steps after an image measurement at $t = 0$. At $t = 0$, we consider the sensor to be perfect and reset the cumulative uncertainty to zero. Moreover, we adapt the policy π to select the action a_t maximizing the lower confidence bound of the estimated reward ψ_t

given by

$$a_t^* = \arg\max_a \left(M(s_t) - w_c \left| \frac{\partial M(s_t)}{\partial s} \right| \Sigma_t \right) \tag{7.7}$$

using the cumulative uncertainty Σ_t in contrast to Equation 7.5. Let $w_c > 0$ denote a parameter for weighing the estimated reward of actions against their confidence. If the uncertainty rises above a threshold σ^2_{image} a new image is taken. If the lower bound of the estimated reward falls below a threshold, the bin is assumed to be empty. Furthermore, given both a policy π and a transition model $\mathcal{T}$, states and actions can be predicted multiple steps ahead by iteratively applying π and $\mathcal{T}$ alternately. Figure 7.2 shows the computational flow of a single prediction step, allowing to plan ahead and optimize for multiple-step criteria. As we don't focus on planning algorithms itself, we implemented a simple breadth-first tree search. To allow diverse tree paths, we adapt Equation 7.7 to sample from the $N \approx 5$ highest actions uniformly randomly. This way, we can predict the overall tree ahead of time and return the action path maximizing the criteria during inference.

7.4 Experimental Results

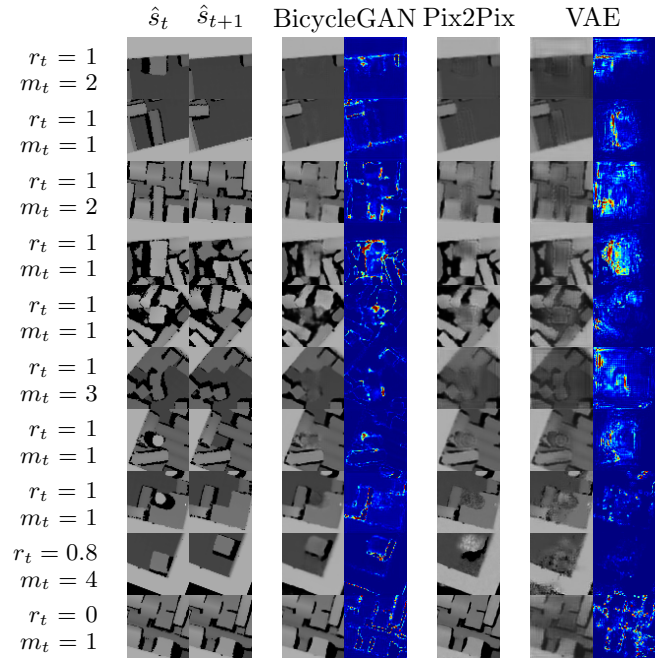
We use the same experimental setup introduced in section 4.3 with both the Ensenso N10 and RealSense D435 camera. For further details on the implementation, additional videos, experimental results, and datasets, we refer to the project website[1].

7.4.1 Image Prediction

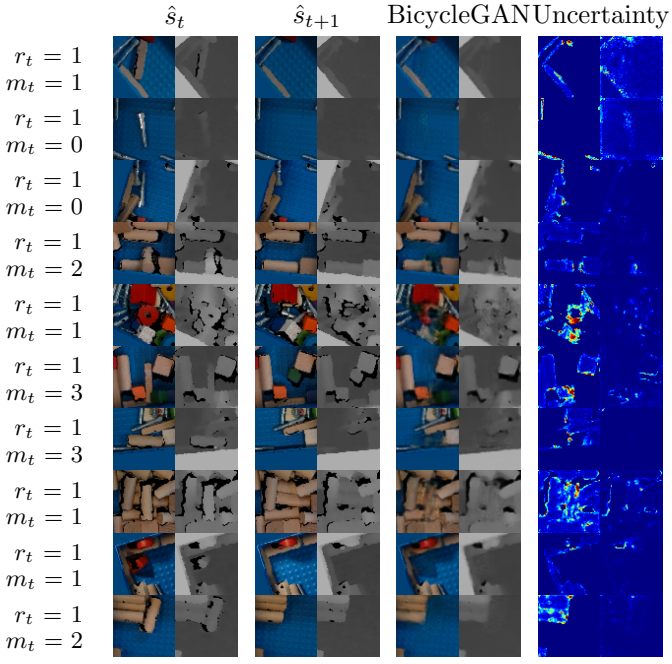
During around 180 h of real-world data collection, the robot attempted 40 000 grasps and 2400 shifts. The robot used two bins with a variety of objects for training. After a successful grasp, the robot moves the objects to the other bin, however returns after filing to take an image *after* the manipulation. After 4000 random actions, the robot used the *Prob(N)* selection strategy for learning manipulation. N was increased continuously from 1 to 6 and transitioned into the *Top(N)* selection at 30 000 actions, leading to an average reward of $\overline{r} = 0.55$. The manipulation NN was trained around every 500 actions. In the end, we train the BicycleGAN architecture on the complete dataset of state-action-state-tuples $(s_{t=0}, p_{t=0}, r_{t=0}, s_{t=1})$.

To stabilize the training of the generative adversarial networks (GANs), we smooth the labels and add random noise to the discriminator input. In comparison to the original implementation, we use a training batch size of 32. Otherwise, we keep the training process of the BicycleGAN architecture. Figure 7.3 shows example predictions given the manipulation primitive type m_t and the reward r_t. We include the ground truth as well as results of the Pix2Pix architecture (without uncertainty estimation) and a variational

[1] Available at `https://pantor.github.io/learning-transition-for-manipulation/` (accessed on December 9th, 2022)



(a) Comparison of predictions by the BicycleGAN, Pix2Pix (without uncertainty) and VAE model.



(b) Predictions by the BicycleGAN, for both RGB (not used for downstream manipulation) and depth images.

Figure 7.3: Example predictions given an image window $\hat{s}_t$ before a manipulation action of type m_t and reward r_t. $\hat{s}_{t+1}$ corresponds to the ground truth. The normalized pixel-wise uncertainties are shown from low (blue) to high (red). Action types $m_t \in \{0, 1, 2, 3\}$ are grasps of different pre-shaped gripper width; $m_t \in \{4, 5\}$ are shifting motions.

autoencoder (VAE) for visual comparison. Predicting both a state of 64×64 pixels with corresponding uncertainties using $N = 20$ samples needs around 60 ms.

While our transition model can generate realistic images, we find two common failure modes: First, low-level failures like large-scale deviations in the background or blurred edges. Second, wrong high-level features like cropped, blurred, or visual fused objects. The latter case occurs in particular for either shifting actions or occluded layers beneath the grasped object. Regions of high predicted uncertainty can be found primarily near edges, in particular of regions without depth information (black) and within the manipulated objects themselves. Examples of iterative predictions are shown in Figure 7.2.

7.4.2 Grasp Rate

We define the grasp rate as the percentage of successful grasps for grasping 15 objects out of a bin with 25 objects without replacement. For a bin picking scenario with multiple object types, Figure 7.4 shows the dependency between the grasp rate and the prediction step t, equivalent to the number of actions since the last image measurement.

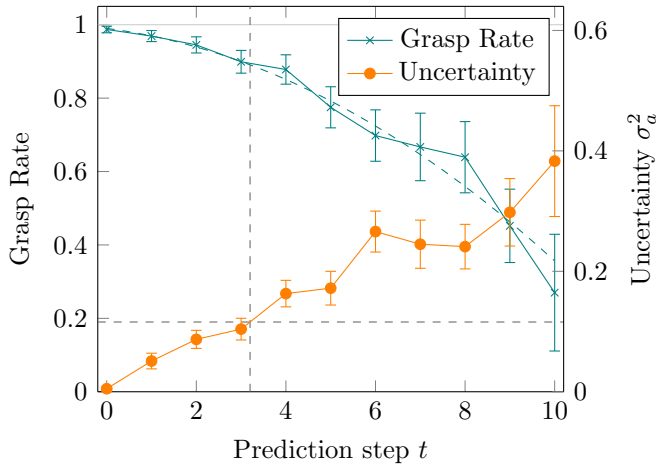


Figure 7.4: Grasp rate and uncertainty of the estimated reward σ_a^2 depending on the number of prediction steps after an initial image measurement at $t = 0$. Based on 1080 grasp attempts in the defined bin picking scenario.

The grasp rate decreases from 98.7 % at $t = 0$ to around 90 % for $t = 3$. Then, the average uncertainty of the estimated reward σ_a^2 increases to around 0.1, resulting in a significant term in Equation 7.7. In order to minimize the sum of lost time due to both grasping mistakes and image acquisitions, we estimate an optimal uncertainty threshold σ^2_{image} given a rough estimate of the execution times of the robot. We identified the durations $t_G = 6.5\,\text{s}$ of the grasp and $t_I = 2.5\,\text{s}$ of the image measurement and related motions. The picks per

hour (PPH) are then given by

$$\text{PPH} = \frac{\text{Grasp Rate}}{t_G + \frac{1}{t} t_I} \approx \frac{\text{-}0.00354\, t^2 - 0.0228\, t + 0.987}{6.5 + 2.5\, t^{-1}} \tag{7.8}$$

with the optimal prediction step $t^* \approx 3.2$. The quadratic regression is plotted in Figure 7.4. To optimize the PPH in further experiments, we set the threshold σ^2_{image} for taking a new image to the corresponding uncertainty $\sigma_a^2 \approx 0.12$.

7.4.3 Benchmarking Picks Per Hour

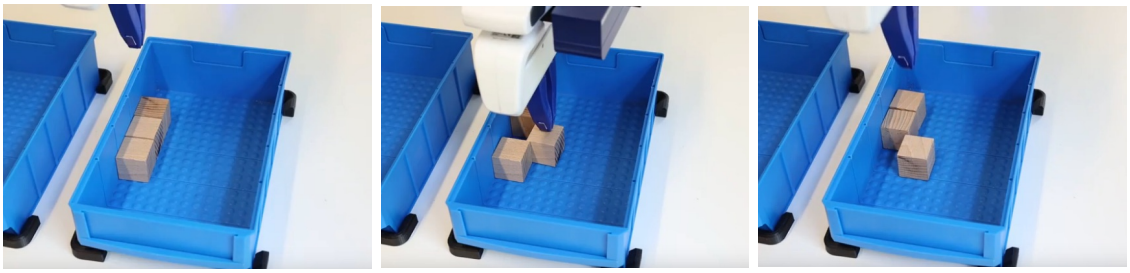
We evaluate our robotic system against the Box and Blocks Test from the YCB benchmark suite [22]. Within 2 min, the robot should grasp and place as many cubes (2.5 cm side length) as possible out of one bin into another. Additionally, we introduce a setting where the robot is allowed to grasp multiple objects at once. Based on the work of semantic grasping [57], we extend the manipulation NN to predict whether the closed gripper distance will be larger than 4 cm. This corresponds to grasping at least 2 objects; the policy π then maximizes the lower bound of the expected number of grasped objects. Table 7.1 shows the grasp rate, object count, and final PPH for each combination of grasps with/without prediction and of single/multiple objects, and in comparison to the heuristic of Calli et al. [22]. Random grasps are evaluated by sampling randomly from the entire action space. We repeated every experiment 5 times. Our robot achieves a peak performance of (23.5 ± 0.5) grasps with (3.3 ± 0.2) images, corresponding to (702 ± 14) PPH. On average, using an approach with prediction improves the PPH by 15 %.

Table 7.1: Results for the Box and Blocks Test, including the grasp success rate, the object count after 2 min, and the corresponding picks per hour (PPH).

	Grasp Rate	Count	PPH
Heuristic [22]	80 %	10.8	324
Random	(13 ± 4) %	2.2 ± 0.7	66 ± 20
Single	(97 ± 2) %	12.8 ± 0.3	384 ± 10
Single, Prediction	(94 ± 2) %	16.4 ± 0.5	492 ± 14
Multiple	(96 ± 2) %	20.4 ± 1.0	612 ± 29
Multiple, Prediction	(94 ± 2) %	23.4 ± 0.5	702 ± 14

7.4.4 Planning

We demonstrate the ability to plan ahead for the task of emptying a bin with as few actions as possible. Given an object configuration of three cubes in a row at the side of the bin, the robot can grasp all objects in 4 steps by shifting the middle cube first. If an outer cube is shifted otherwise, the robot needs at least 5 steps. In this configuration, planning can reduce the average number of action steps from 4.77 ± 0.20 to 4.14 ± 0.13, measured



(a) Start **(b)** 4 Actions **(c)** 5 Actions

Figure 7.5: Example scene for evaluating the planning capabilities of the transition model. This way, the robot can infer and minimize the total number of actions needed to empty the bin depending on the initial action.

by 25 successful episodes with a success rate of 78 %. Further illustrative material can be found on the project website[2]. Calculating the planning tree with 3 leaves and a depth of 5 results in a computation time of around 10 s.

7.4.5 Generalization

We qualitatively evaluate the system for objects that were never seen during training (Figure 7.6). We find that our system is able to generalize well to compact objects, naturally

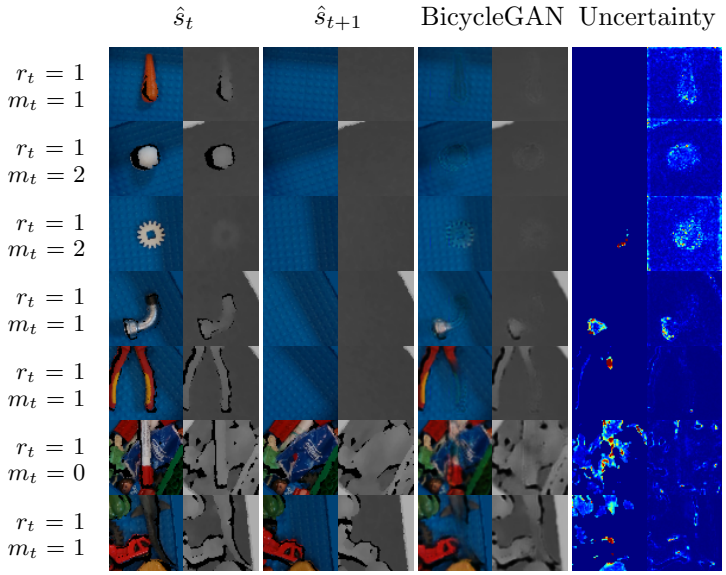


Figure 7.6: Example predictions of the visual state $\hat{s}_{t+1}$ after an executed manipulation primitive m_t for novel (unknown) objects. The uncertainties are shown from low (blue) to high (red).

[2]https://pantor.github.io/learning-transition-for-manipulation/ (accessed on December 9th, 2022)

depending on the similarity to trained objects. From the perspective of instance segmentation, this problem is challenging for unknown objects, as the system needs to estimate the connection of objects. For example, the pincers (5th image) are only connected outside out of the image. For long and irregular objects, the model often splits single objects into multiple parts. However, it often captures the boundaries for other objects well, e.g. like the red object in the bottom row.

7.5 Discussion

In this chapter, we presented an approach for learning a generative transition model between the visual states and their pixel-wise uncertainty of a manipulation action. This way, the robot is able to plan sequences of single-step manipulation primitives, which were learned data-efficiently without sparse rewards. Furthermore, we demonstrated two more novel skills of our robot: By skipping image measurements, it was able to increase the PPH by around 15 % in the Box and Blocks Test [22]. Second, the robot was able to optimize regarding multiple-step criteria in a flexible way, e.g. to minimize the number of actions for emptying a bin. This becomes possible through three major contributions: First, we simplify the transition model by conditioning it on the already learned reward estimation as well as using the spatial invariance of the state space. Second, we propose a model architecture generating samples from a multi-modal distribution, allowing to estimate a prediction uncertainty. Third, the latter was propagated efficiently towards the estimated reward.

Our transition model works in a direct image-to-image setting similar to [17, 37, 32]. In comparison, our model additionally estimates a pixel-wise uncertainty measure and is applied to more difficult, obstacle-rich environments. From a broader perspective, we see several advantages in our approach: In comparison to model-based RL, our transition model does not interact with the single-step policy during training and extends the robotic system *optionally*. This allows to condition the transition model on the estimated reward and reuse the manipulation model learned with model-free RL. Oftentimes, a transition model is used for model-based policy training to reduce sample complexity [50]. However, we find that this is not useful for grasping: The following states depend strongly on the grasp success, which can be measured easily by the gripper and learned explicitly by a manipulation model.

Still, our approach allows for flexible optimization criteria and - even more important for real-world applications - off-policy RL of time-dependent manipulation sequences. In this regard, we learned simple manipulation primitives similar to [128], but were able to combine multiple primitives to more complex action sequences. Although model errors are inevitable for robotic manipulation [37, 32], our uncertainty-aware policy does *actively* avoid these model limitations. However, while our transition model can generate realistic images, we find that it fails to capture high-level features of the stochastic physics. Instead, it usually learns the simplest solution and provides low-level pixel uncertainties.

Chapter 8

Garment Folding

Garment handling such as folding and packing are common tasks in textile manufacturing and logistics, industrial and household laundry, healthcare, and hospitality. These tasks are largely done manually due to the complex configuration space as well as the highly non-linear dynamics of deformable objects. In the following, we introduce an end-to-end system for fast and efficient garment folding. In this regard, we see this chapter as the extension of our general approach for learning primitives to (1) *dual-arm* manipulation and (2) dynamic and more complex applications. This chapter is based on [4].

8.1 Problem Statement

Given a visual observation $o_t \in \mathbb{R}^{W \times H \times C}$ of the garment's configuration s_t at time t, the objective is to compute and execute an action a_t to transfer the garment from an arbitrary configuration to a desired user-defined s^* goal configuration. In particular, s^* is invariant under the garment's position and orientation in the workspace. Moreover, we ignore which side of the garment (e.g. the front or back of a t-shirt) points upwards in a smoothed configuration. We assume an overhead observation with a calibrated pixel-to-world transformation, as well as a garment that is easily distinguishable from the workspace.

We consider a dual-arm robot with parallel-jaw grippers executing actions of type $m \in \mathcal{M}$ from a discrete set of pre-defined action primitives. In particular, we parameterize each primitive by two planar gripper poses

$$a_t = \langle m, (x_1, y_1, \theta_1), (x_2, y_2, \theta_2) \rangle$$

for each arm respectively, in which (x_i, y_i) are coordinates in pixel space, and θ_i is the end-effector rotation about the z axis. We further assume a flat obstacle-free workspace and a motion planner that computes collision-free trajectories for a dual-arm robot. However, collision avoidance and limited reachability are common challenges in bimanual manipulation and especially deformable manipulation that need to be addressed fundamentally.

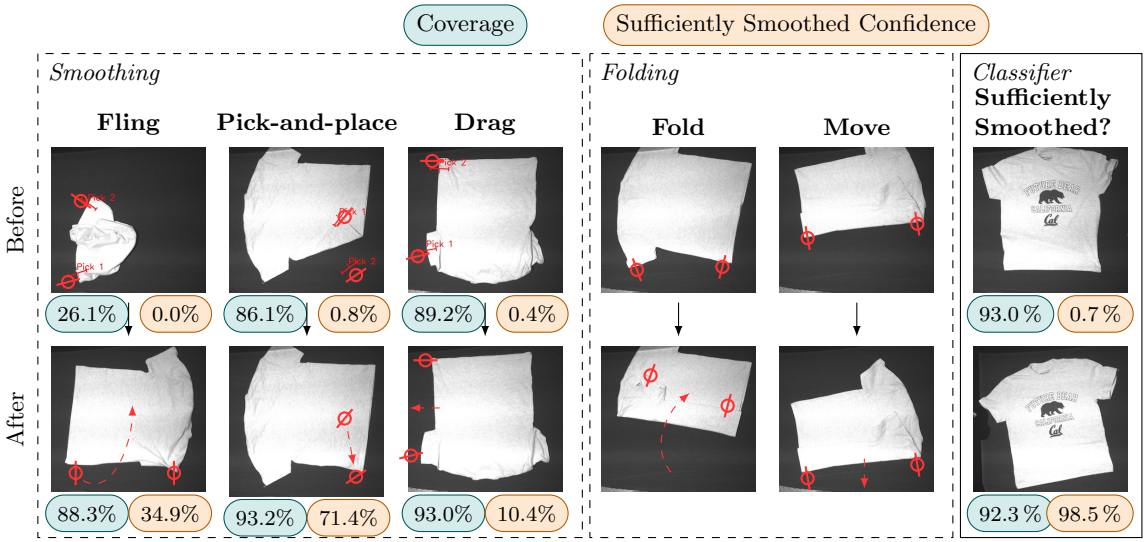


Figure 8.1: Action primitives for garment folding: Given an overhead RGBD image, we select a smoothing action from a discrete set of primitives (**left** box), and compute a pair of end-effector poses. Coverage calculation (in blue) is insensitive to wrinkles in the fabric (**right** box, **top**) and therefore it learns to classify configurations as Sufficiently Smoothed (**right** box, **bottom**) (in yellow). Folding a t-shirt from a smooth configuration is done through a sequence of folding primitives (**center** box).

8.2 Primitives

We are interested in the set of quasi-static and dynamic action primitives that enable the robot to (1) transfer an arbitrary garment configuration s_t to a folded goal configuration s^* (completeness), (2) reduce the number of action steps (efficiency), and (3) with a reduced number of primitives (minimality). Each action primitive is defined through a pair of poses as well as a motion trajectory. All primitives share a common procedure to reliably grasp the garment with parallel jaw grippers: Each gripper moves 4 cm above the grasp pose a_t, rotates 8° so that one fingertip is below the other, and moves 1 cm towards the direction of the higher fingertip. This motion improves the success of grasping in particular at the edge of the garment. We define the following learned primitives (Figure 8.1 left box):

Fling: Given two pick poses, the arms first lift the garment above the workspace and stretch it until a force threshold is reached, measured using the arms' internal force sensors. Next, the arms apply a dynamic motion, flinging the garment forward and then backward while gradually reducing the height toward the workspace. Similar to [49], we find the fling motion to be robust under changes of velocity and trajectory parameters, and therefore we keep these parameters fixed. The fling primitive allows to significantly increase the garment's coverage in only a few steps, but oftentimes does not yield a smooth configuration.

Pick-and-place: Given a pick and a corresponding place pose, a single arm executes this quasi-static action, while the second arm presses down the garment at a point on a line extending the pick from the place pose. Pick-and-place enables the robot to fix local faults such as corners or sleeves folded on top of the garment.

Drag: Given two pick points, the robot drags the garment for a fixed distance away from the garment's center of mask. Dragging leverages the friction with the workspace to smooth wrinkles or corners such as sleeves folded below the garment.

We define the following heuristic-based primitives (Figure 8.1 center box):

Fold: Both arms execute a pick-and-place action simultaneously to fold the garment. The heuristic for calculating the pick and place poses is explained in section 8.6.

Move: While similar to drag, this primitive's pick poses and its drag distance are calculated by a heuristic so that the garment's center of mask is moved to a goal target point. Usually, the robot drags the garment towards itself to mitigate reachability issues in subsequent actions. section 8.6 provides details about the pose calculation.

We define an additional learned primitive to switch from garment-smoothing to folding (Figure 8.1 right box):

Sufficiently Smoothed: We find that deciding whether a garment is ready to be folded purely from coverage computation, as done in prior works [53, 108, 49], is not sufficient. In particular, even a high coverage is prone to wrinkles or faults that might reduce the subsequent fold quality significantly (as described in Figure 8.1). Instead of relying on the coverage, the system returns a smoothness value given an overhead image. While this primitive is not used to change the configuration of the garment, it is used to switch from garment-smoothing to folding.

8.3 Model Architecture

Predicting a single pose from an overhead image is commonly done by first estimating a pixel-wise value map per gripper z-axis rotation θ, in which each pixel value represents a future expected reward (e.g., grasp success, increase in garment coverage, etc.), and then selecting the maximum greedily [128, 8, 44, 49]. Extending this approach to two *corresponding* planar poses $(x, y, \theta)_{1,2}$ conditioned on each other is however challenging primarily due to the exponential scaling of possible end-effector poses with the number of dimensions. In particular, this is a multi-modal problem, and the predicted unconditioned value maps $Q_{unc}(x, y, \theta)$ have multiple peaks (as in Figure 8.3). While unconditioned value maps may provide information relevant to downstream bimanual tasks, such as the grasp success, they provide no information regarding their correspondences. To address this we define *correspondence descriptors*

$$\vec{d} = (Q_{unc}, x, y, \sin\theta, \cos\theta, m, \vec{e})$$

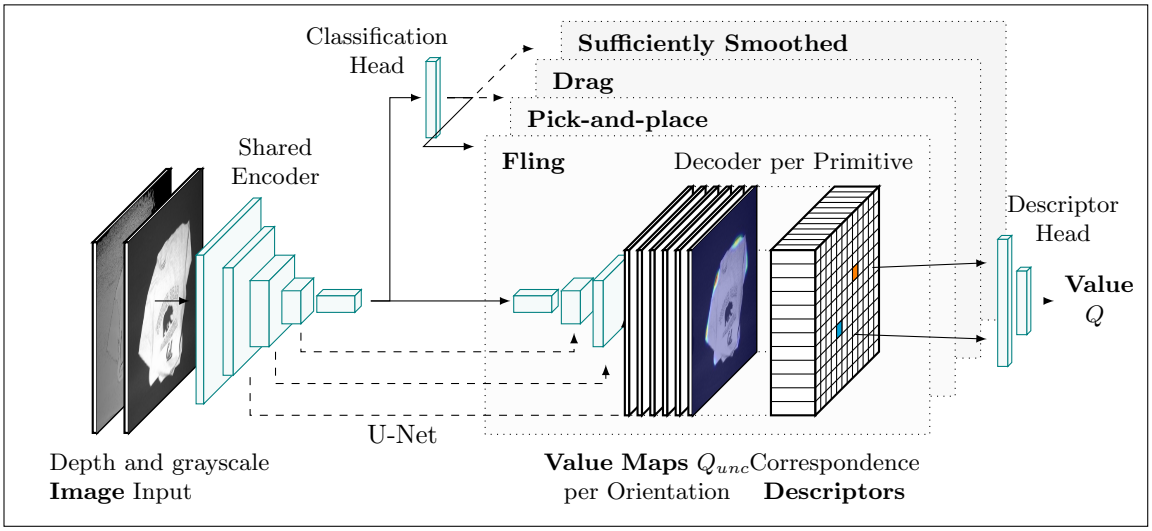
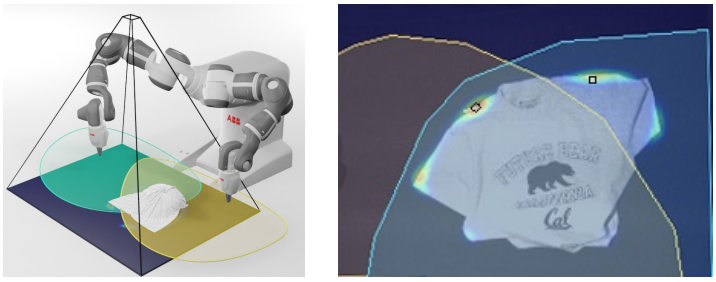


Figure 8.2: Our BiMaMa-Net architecture first maps an image to a manipulation primitive type via its shared encoder and classification head. Given a primitive, it predicts dense unconditioned value maps for a discrete set of gripper orientations. It then calculates pixel-wise correspondence descriptors. A descriptor pair, representing a bimanual action, is combined in the descriptor head to predict its joint value.

where $\vec{e} \in \mathbb{R}^M$ is a predicted embedding for each pixel (disregarding orientations θ) concatenated with the unconditioned value Q_{unc}, positional encodings, and the action primitive type m. Then, the final conditioned value $Q(\vec{d}_1, \vec{d}_2)$ depends on a descriptor pair.

Figure 8.2 shows the complete information flow of BiMaMa-Net: A shared encoder using a ResNext-50 [127] backbone maps an input image (e.g. depth and grayscale) to high-level features. First, a classification head predicts the manipulation primitive m. For a *Sufficiently Smoothed* primitive, no further action is required. For all other learned primitives, a U-Net [100] decoder predicts value maps for a discrete number N of end-effector orientations θ. We choose a U-Net architecture over fully convolutional NN used in prior robotics manipulation works [9, 131, 49, 8] as U-Nets are better suited for high-resolution inputs that we find necessary for detecting edges and wrinkles for garment smoothing.

Then, *BiMaMa-Net* samples a set of poses from the value map, where pixels with higher values are more likely to get sampled. During training, BiMaMa-Net samples from $p(a|s) \sim \sqrt{Q_{unc}(a,s)}$ to allow for sampling also negative examples to better estimate the underlying distribution of action values. We empirically found that this strategy is a good compromise between too simple (e.g. uniform randomly) and too hard (e.g. by heatmap Q_{unc}) counterexamples. For inference, BiMaMa-Net samples from $p(a|s) \sim Q_{unc}(a,s)^2$ which emphasize action poses with high values. It then calculates the correspondence descriptors for each pose, and a final NN head combines all descriptor pairs $\vec{d}_1$ and $\vec{d}_2$ to output the final conditioned action value Q.



(a) Workspace **(b)** Reachability mask

Figure 8.3: Reachability for bimanual manipulation. (a) We perform a boundary search to compute separate reachability masks for the left (yellow) and right (blue) robot arms. (b) BiMaMa-Net guarantees at least one pick pose (black) from the value map within each mask.

If two poses are interchangeable (e.g., during a fling or a drag action), a single decoder predicts the value maps per θ. However, if a certain relation between the poses must be maintained (e.g., the conceptual difference between the pick and the place poses in a pick-and-place action) then separate decoders compute two value maps Q^1_{unc} and Q^2_{unc}. Due to this simplification, the robot is not able to consider whether the front or back of the fabric points upwards after a fling.

At run-time, the NN samples $N_1 = N_2 = 200$ descriptors and combines them into a list of $N_1 \times N_2 = 40\,000$ correspondence descriptor pairs. The descriptor head is a comparably small three-layer dense NN to avoid becoming the bottleneck due to the high number of descriptor combinations. This pre-selection is similar to the approach in chapter 6, and is also useful for a range of tasks related to garment folding: For example, the robot needs to *grasp* first to allow for downstream tasks such as flinging. This requirement reduces the number of possible poses significantly even without considering two conditioned poses. While two conditioned action poses arise naturally in bimanual manipulation, we also apply it to subsequent poses in the *pick-and-place* primitive. As the NN predicts an unconditioned Q_{unc} value without knowing about the second pose, the estimated value is only a first approximation.

8.4 Reachability Calibration

As shown in Figure 8.3, to ensure reliable garment smoothing and folding, the robot should compute the actions that maximize the expected reward within the reachable space. To find the robot's reachable space, we perform a one-time boundary search along a discretized grid in the action space (x, y, θ) for each gripper, assuming a constant height z above the table. The search, done separately for each θ, starts with a fixed lower value of y and increases x until the inverse kinematics fails to find a solution. Afterwards, it repeatedly increases y or decreases x so that the search is confined to the continuous boundary at which reachability

fails. As a result, we get separate masks M_l and M_r for the left and right arms

$$M(x, y, \theta) \rightarrow \{0, 1\}$$

that can be incorporated into the algorithm as spatial binary constraints by restricting the action sampling to the masks. To ensure that each reachability mask contains at least one pose, we create up to four masked value maps from Q^1_{unc} (or Q^2_{unc}) by multiplying them with M_l or M_r: $\{Q^{1l}_{unc}, Q^{1r}_{unc}, Q^{2l}_{unc}, Q^{2r}_{unc}\}$. An action value $Q_{unc} = 0$ is ignored in the sampling process. We then sample and combine the correspondence descriptors from Q^{1l}_{unc} and Q^{2r}_{unc}, and vice versa for Q^{1r}_{unc} and Q^{2l}_{unc}.

We find that using a calibrated reachability mask for each end-effector orientation θ separately significantly reduces the number of false negatives that arise when using approximations, such as a circular mask. After selecting the final, reachable poses $(x, y, \theta)_1$ and $(x, y, \theta)_2$ during run-time, we check for possible collisions due to inter-arm interaction. If a potential collision is detected, the next best action is selected until reachable and collision-free poses are found.

8.5 Training for Smoothing

We train BiMaMa-Net via self-supervised real-world learning to predict the manipulation primitive type m and the corresponding action poses $(x, y, \theta)_1$ and $(x, y, \theta)_2$ given an overhead image of a garment.

In order to scale real-world interaction, the learning process is designed for minimal human intervention. First, we collect examples of smooth configurations to train a classifier outputting the confidence $p(\textit{Sufficiently Smoothed}|s)$. Additionally, let $cov(s)$ be the coverage of the garment at configuration s observed from an overhead perspective, calculated by background subtraction and color filtering. We define the reward r

$$r_t = \max\left(\tanh\left[\alpha\ \left(cov(s_{t+1}) - cov(s_t)\right) + \beta\ \left(p(\textit{smoothed}|s_{t+1}) - p(\textit{smoothed}|s_t)\right)\right], 0\right)$$

as the sum of the *change* of coverage and *Sufficiently Smoothed* confidence with tuned weights α and β respectively. It is scaled to $r \in [-1, 1]$ first and then clipped to a non-negative value, so that no change equals zero reward. To ensure continuous training, the robot resets the garment configuration by grasping it at a random position on its segmented mask and dropping it from a fixed height. We train the NN using the complete, most recent dataset in parallel to the real-world, *off-policy* data collection. The robot explores different actions by randomly choosing from the set of N_s best possible actions uniformly sampled.

To avoid a purely random and sample-inefficient initial exploration, we kickstart the training with human annotations. We differentiate between self-supervised and human-annotated data within the training process in three ways: (1) We set the reward of human-annotated data to a fixed r_h. (2) Besides training the value map at the specific annotated pixel position and orientation, we follow [44] and introduce a Gaussian decay centered around each pose as a global target value instead. (3) The classification head is trained only with data that has a reward higher than a tuned threshold $r \geq r_c$.

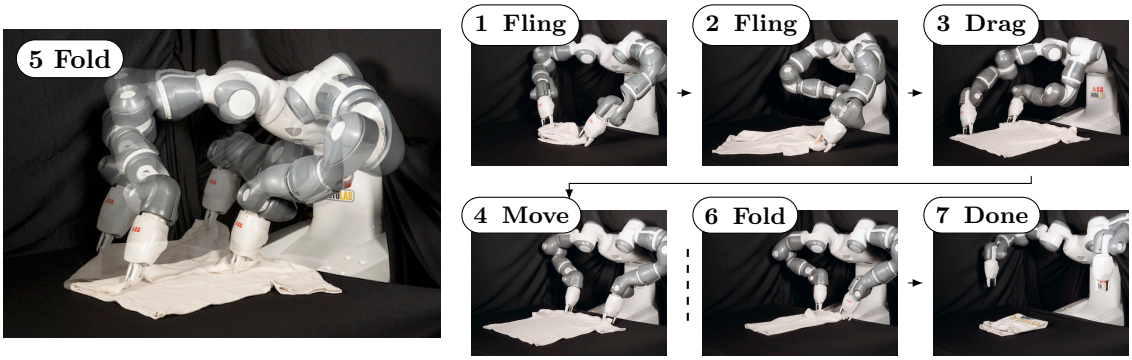


Figure 8.4: We learn to fold garments from arbitrary configurations: Given a crumpled t-shirt, the robot unfolds it by applying two fling actions (1, 2), smooths the t-shirt with a drag action (3) until it is *sufficiently smoothed*. It then moves the t-shirt for better reachability (4), and applies three folds (5-6) to achieve the user-defined goal configuration (7).

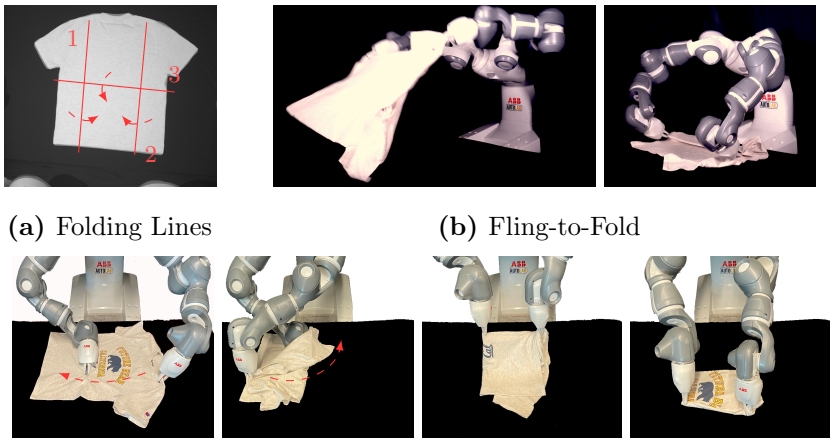
8.6 Folding Pipeline

We compare three approaches for folding: *instruction-based folding*, which can be adapted to different garments and different folding techniques, *"2 seconds" fold*, a known heuristic for surprisingly fast t-shirt folding, and *fling-to-fold* (F2F), a novel technique that can increase the number of folds-per-hour (FPH) by leveraging prior knowledge about the t-shirt's dimensions.

Instruction-based Folding: As shown in Figure 8.5 (left), given a mask of a smoothed garment, the robot iteratively folds the garment along user-specified *folding lines*. These allow to define the goal configuration of a smooth garment precisely without using high-dimensional visual goal representations [124, 53]. A complete user instruction includes: (1) A binary mask called template and (2) a list of folding lines relative to the template (Figure 8.5 left). The folding direction is defined with respect to the line according to the right-hand rule.

To apply the folding lines to the garment, a particle-swarm optimizer computes an affine transformation by registering the template with the current image. Afterwards, the algorithm calculates corresponding poses for a bimanual fold action: Let $\vec{p}_{ent}$ be the first and $\vec{p}_{exit}$ be the second intersection point of the line and the mask, where the line enters and exits the mask respectively. This splits the mask into a *base* and a *fold-on-top* part. On the contour of the latter, the algorithm finds two pick points $\vec{p}_1$ and $\vec{p}_2$ so that the area of the four-sided polygon $(\vec{p}_{ent}, \vec{p}_1, \vec{p}_2, \vec{p}_{exit})$ is maximized. We use the normal at the pick point for the gripper orientation θ. The place poses are calculated by mirroring the pick poses at the folding line.

"2 Seconds" Fold: For specific garments such as t-shirts, there exist heuristics for efficient folding. Given a smooth configuration, the so-called "2 seconds" fold follows a



(a) Folding Lines **(b)** Fling-to-Fold

(c) "2 Seconds" Fold

Figure 8.5: We compare three folding approaches. Left: A template mask with a sequence of folding lines that is compiled into a number of bimanual pick-and-place actions. Center: A so-called "2 seconds" folding heuristic that applies only very few steps however is for t-shirts only [1]. Right: A *fling-to-fold* primitive that combines a fling with an immediate folding action. Here, the garment does not need to be fully smoothed, however, prior knowledge is required.

specific set of steps that requires using two arms simultaneously (Figure 8.5), and is therefore well suited for a bimanual robot [1].

Fling-to-fold (F2F): We observe that (1) a fling action while grasping a sleeve and the non-diagonal bottom corner is especially effective and (2) the first fold action grasps the same points. We conclude that these two steps can be merged to reduce imaging and motion time. We implement F2F by adding a learned primitive to BiMaMa-Net that computes these pick points if visible. The primitive's motion is implemented by combining a fling with a consecutive fold action (Figure 8.5). To ensure that the t-shirt is folded correctly, prior knowledge about the t-shirt's dimension is required to adapt the height of each arm prior to the fold.

8.7 Experimental Results

We experimentally evaluate the garment smoothing and folding performance of the full system on a known t-shirt, as well as on two garments unseen during training. For further details on the implementation, the source code, additional videos, and datasets, we refer to the project website[1]

[1] Available at `https://pantor.github.io/speedfolding/` (accessed on December 9th, 2022)

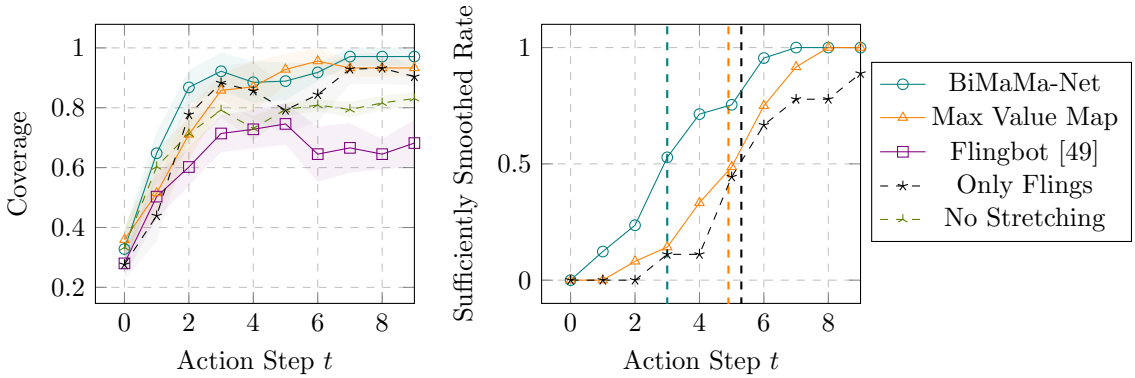


Figure 8.6: Garment smoothing until it is ready to fold. We compare the normalized coverage (left) and prediction of the learned *Sufficiently Smoothed* classifier (right) over the number of action steps with different baseline methods.

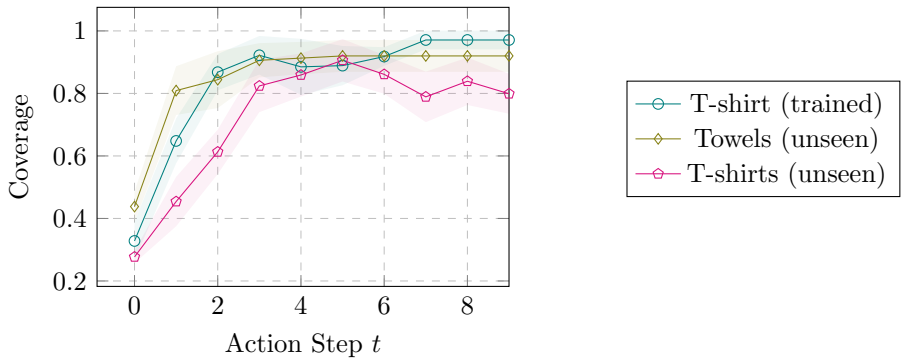


Figure 8.7: The system can generalize to unseen garments of different color, pattern, and material.

8.7.1 Experimental Setup

We perform experiments on a physical ABB YuMi robot with parallel-jaw grippers. The gripper's fingertips are extended by small 3D-printed teeth to improve grasping. A thin sponge mattress is placed on the workspace to allow the grippers to reach below the garment without colliding. A Photoneo PhoXi captures overhead grayscale and depth images of the workspace, generating observations $o_t \in \mathbb{R}^{256\times192\times2}$. As the garment is frequently outside the camera's field of view, a 1080P GESMATEK RGB webcam is mounted above the workspace and used for coverage calculation. Computing is done on a system using an Intel i7-6850K CPU, 32GB RAM, and an NVIDIA GeForce RTX 2080 Ti.

We first perform data collection, and train (section 8.5) on a single t-shirt. Initially, 600 scenes of random garment configuration were recorded and manually annotated. After training a first NN, the robot collected 2200 self-supervised actions in 16 h. To include data of less frequently observed actions, we copied and re-annotated 1500 actions, resulting in a dataset of 4300 actions in total. We used a single t-shirt shown in Figure 8.4 throughout the training. For training the NN, we make excessive use of data augmentation to offset the limited data collection due to real-world interaction. This includes random translations, rotations, flips, resizes, brightness, and contrast changes. We use $N = 20$ gripper orientations equally distributed over $[0, 2\pi)$ to implement the decoder as described in section 8.3. For training, we set $N_s = 50$, $r_h = 0.8$ and $r_c = 0.3$.

We designed a set of garment smoothing and folding experiments to evaluate the system. Initial garment configurations are generated by environment resets as described in section 8.5. Each experiment is averaged over 15 trials. We ignore experiments that fail due to a motion planning error, as this is not the focus of this paper. A trial is considered unsuccessful if either the garment was not successfully folded according to the majority vote of three reviewers or the number of actions exceeded a maximal horizon of $H = 10$. We define a grasp success if the gripper holds the garment *after* an executed action. For known garments, BiMaMa-Net achieves an average grasp success rate of over 96 %.

8.7.2 Sufficiently Smoothed

We evaluate garment smoothing using two metrics: The garment coverage, computed from an overhead image, and a binary Sufficiently Smoothed value, predicted using the Sufficiently Smoothed classifier. We compare BiMaMa-Net to two baselines (1) *Max Value Map*, a variant that computes the pick points directly from the value maps Q_{unc} by computing the maximum over the map to find two pick points without using correspondence descriptors, (2) *Only Flings*, a variant restricted to fling actions only, and (3) *Flingbot*, the pre-trained method from [49] (Figure 8.6). Results suggest that our system can smooth a known t-shirt to a Sufficiently Smoothed configuration in ~ 3 fewer steps compared to the baselines requiring ~ 5. Although the increase of coverage is similar to Only Flings, the latter reaches a Sufficiently Smoothed configuration later or even fails to do so, confirming the need for additional action primitives to fully smooth a garment.

We note that the FlingBot baseline fails to reach an 80 % coverage as reported in [49],

Table 8.1: End-to-end folding for different NN architectures, folding approaches, and garments. 15 trials were averaged for each experiment. The duration of an end-to-end fold is averaged over successful folds, while the cycle time and folds per hour are averaged over both successful and unsuccessful folds.

Method	**Folding Approach**	**Garment**	**Smoothing Actions**	**Duration** [s]	**Fold Success**	**Cycle Time** [s]	**Folds Per Hour (FPH)**
Max Value Map	Instruction	T-shirt	5.1 ± 0.5	133.9 ± 7.4	80 %	167.4 ± 9.2	21.5 ± 1.2
BiMaMa-Net	Instruction “2 Seconds” Fold	T-shirt	3.0 ± 0.4	108.7 ± 7.3	**93 %**	116.9 ± 7.9	30.8 ± 2.1
			3.0 ± 0.4	97.3 ± 4.8	53 %	182.4 ± 5.4	19.7 ± 0.6
	Fling-to-fold		$\mathbf{1.8 \pm 0.2}$	$\mathbf{81.7 \pm 4.3}$	**93 %**	$\mathbf{87.9 \pm 4.7}$	$\mathbf{40.9 \pm 2.2}$
Garments Unseen During Training							
BiMaMa-Net	Instruction	Towel	1.7 ± 0.2	59.2 ± 3.8	87 %	68.1 ± 4.4	52.9 ± 3.4
		T-shirt	4.8 ± 0.4	141.1 ± 8.7	80 %	176.3 ± 10.9	20.4 ± 1.3

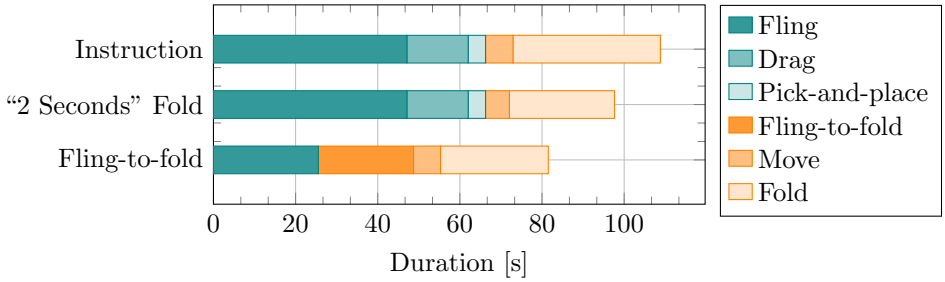


Figure 8.8: Timings for calculating and executing the action primitive types depending on the folding approach. Instruction-based and "2 seconds" fold share the same smoothing actions (blue), but differ in folding (orange). By introducing a combined Fling-to-fold primitive, a smoothed state is not required before folding. However, the "2 seconds" fold is suited for manipulating a t-shirt only, and Fling-to-fold assumes prior knowledge of the garment.

as we observe frequent grasp failures presumably due to differences in the physical setting. We ablate the stretching motion before a fling and observe that stretching leads to higher coverage.

8.7.3 Folds per Hour

Table 8.1 shows results of end-to-end garments folding experiments. BiMaMa-Net manages to (1) successfully fold garments in over 90 % of the trials on known garments and (2) 30 % faster than the Max Value Map baseline using the Instruction-based folding approach. The "2 seconds" fold achieves an additional speedup of 10.4 % when executed successfully, however, we find that it is sensitive to the t-shirt's orientation in a Sufficiently Smoothed configuration and suffers from a low fold success rate. With prior information on the t-shirt's dimensions, F2F uses 40 % less smoothing actions and imaging time. As a result, it achieves a speedup of over 25 % compared to the instruction-based approach, leading to 40.9 folds per hour on average. Calculating an action using the BiMaMa-Net architecture takes (126.0 ± 0.9) ms on our hardware.

8.7.4 Generalization to Unseen Garments

We explore how the system, trained on a single t-shirt, can generalize to garments unseen during training. In these experiments, we use (1) a t-shirt with a different color and stiffness and (2) a rectangular towel with a different color compared to the original t-shirt. We evaluate on unseen garments using instruction-based folding, as this is the only approach that easily adapts to general garments. We run the same experiments on the unseen t-shirt with no changes to the BiMaMa-Net model or the folding template. In contrast, when we run the experiments with the towel we observe that the system fails to classify a Sufficiently Smoothed configuration, as the object's shape is different from that BiMaMa-Net was trained on. To address this, we add 20 examples of Sufficiently Smoothed towel

images to the dataset and re-train BiMaMa-Net. Table 8.1 suggests that the system can generalize to garments with different color, stiffness, and shape.

8.7.5 System Limitations

The YuMi's reach of 56 cm is the primary limiting factor of our system. Despite using the presented reachability calibration algorithm, and the use of fling and move primitives addressing the reachability, the robot may encounter situations in which the desired pick poses are out of reach. This may happen, for example, while folding a large towel.

Grasp failures, especially during a fling motion, can decrease the garment's coverage dramatically. We find that most grasp failures happen due to losing the grip during the stretching motion prior to a fling action. This limitation can be mitigated using improved force feedback or by adding visual feedback. We observe a frequent failure case during top-down grasps while executing the first step of the "2 seconds" fold. These grasps may require different gripper jaws that are better suited for top-down grasps. If the grasp fails at an outermost position during stretching, the garment is then not reachable by both arms after a fling. A single-arm grasp for resetting could be integrated as an additional primitive.

As common with data-driven methods, our system can generalize to *similar* unseen garments. For example, textile patterns may be more challenging to detect and classify correctly. This limitation can be addressed through additional data augmentation. Generalization to different garment shapes may also be limited, and can be addressed by adding examples of Sufficiently Smoothed configurations to the dataset, as described in subsection 8.7.4 for the towel example.

8.8 Discussion

While prior works e.g. by Maitin-Shepard et al. [83] or Doumanoglou et al. [30] achieved a high success rate for end-to-end cloth folding, cycle times for a single fold were on the order of 3 to 6 folds per hour (FPH). In comparison, our proposed system achieves 30 FPH to 40 FPH. Inspired by Ha and Song [49], this cycle time reduction is enabled by a dynamic fling primitive that is able to unfold the garment in a few number of actions. In contrast, however, we introduce additional action primitives that enable the robot to reach a configuration that is smooth and ready to fold. Ha and Song learns in simulation with the prevalent sim-to-real gap for deformable objects. By using the techniques introduced in chapter 4 and 5.3, our system can generate real-world data in a self-supervised manner. Moreover, Ha and Song use a 4 DoF parameterization that introduces several constraints for while being tailored to flings. We assume that these tight restrictions lead to their grasp rate of below 75 % and suboptimal fling actions topping out at 80 % coverage. In comparison, our method uses all 6 DoF to parameterize two planar poses for general bimanual actions.

As introduced in chapter 6, pick-and-place is a common robotic task requiring two

conditioned poses. Prior work has developed NN architectures to predict actions [9, 131]. BiMaMa-Net extends the idea of combining embedding vectors to (1) multiple manipulation primitives and (2) correspondence descriptors by encoding additional information. In contrast to [131, 49] or prior approaches in this work, however, BiMaMa-Net is not fully convolutional. As FCNNs disallows the usage of pooling layers, they hinder the usage of high-resolution input that we found necessary for clothes smoothing (in particular detecting edges and wrinkles of garments). Moreover, garment folding is not a *local* problem, as the garment size can commonly be half the size of the image input. In this regard, section 3.4 and 3.5 introduced FCNNs to utilize the *locality* and *viewpoint invariance* of manipulation actions. As our approach to garment folding could only make limited use of these simplifications, the focus shifted to human annotations to minimize the general amount of required data. Similar to 6 DoF grasping in section 5.3, we apply imitation learning to compensate for a large action space and its resulting data-inefficient exploration.

Chapter 9

Conclusion

This work presented an action-centric approach to learning *robotic manipulation* based on real-world interaction. We implemented, adapted, and evaluated our approach on various manipulation tasks like grasping for bin picking, pick-and-place, or garment folding. The first key idea is to simplify the robot's actions to manipulation *primitives*. Then, the controller reduces to the decision *where* to apply *which* pre-defined primitive. Motivated by the viewpoint invariance and locality of manipulation, we derived a sliding window approach that predicts the reward of primitives at a grid of poses. We implemented this *efficiently* as a fully convolutional neural network (FCNN) for planar DoFs.

Second, we focused on learning from real-world data. The key to scaling real-world data generation lies in an autonomous and *self-supervised* implementation. In particular, a safe and continuous training process for contact-rich manipulation requires online trajectory generation (OTG). We presented *Ruckig*, the first algorithm calculating time-optimal trajectories for arbitrary state-to-state motions while considering third-order constraints (for velocity, acceleration, and jerk). The proposed algorithm is real-time capable for control cycles as low as 1 ms. This enables practical collision handling as well as faster motions. We apply both key approaches of manipulation primitives and self-supervised learning to the following manipulation tasks (Figure 9.1):

Grasping We presented an approach to planar (top-down) grasping for bin picking. Here, an FCNN learns to estimate a grasp's success as a binary reward. We use multiple pre-shaped gripper widths as different primitives, leading to over 100 000 reward predictions that can be calculated in under 40 ms. Real-world training collected around 25 000 grasp attempts in 100 h on a single robot. This way, grasp rates of over 90 % for trained objects could be achieved. As common throughout this work, the data-driven and action-centric approach allows to generalize to unknown objects and environments.

We proposed two approaches for 6 DoF grasping: By making the manipulation primitives parametrizable for lateral orientations, we are able to combine a model-based and data-driven controller into a hybrid approach. While the planar DoFs and the gripper width were learned in a data-driven manner as before, the lateral DoFs were optimized

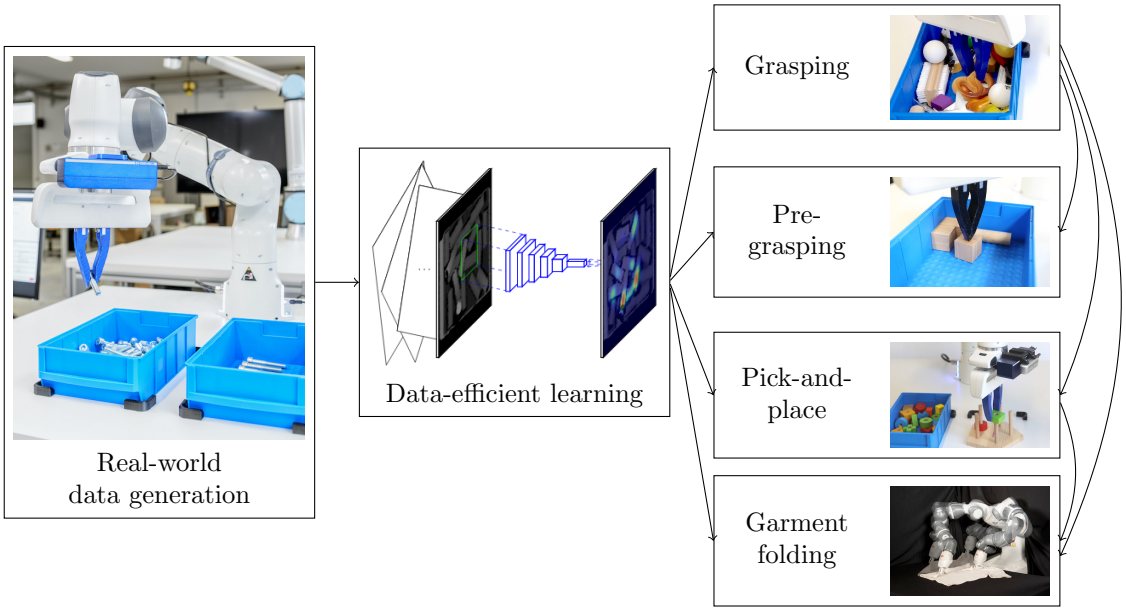


Figure 9.1: The contributions of this work build upon each other to solve common tasks for robotic manipulation. The images visualize: (left) The overall robot system including an eye-in-hand camera and the objects to manipulation; (mid) a fully convolutional neural network (FCNN) for the efficient prediction of an action's success for a large number of possible positions, orientations, and manipulation primitives; (right) execution of different tasks.

analytically. This way, we were able to keep an efficient FCNN for the grasp reward estimation. Our approach achieved grasp rates of up to 92 % in dense clutter, despite processing times of less than 50 ms.

Second, we extended grasping to a fully convolutional actor-critic architecture. The key idea is to rank a large number of grasps along the planar DoFs that can be implemented efficiently, and generate only a few grasp hypotheses for the non-planar DoFs. To kickstart the real-world training and reduce the required exploration, we recorded human grasp demonstrations. Afterwards, the robot started a self-supervised training phase for transferring and adapting the human policy. After 40 000 grasp attempts, the robot was able to grasp objects with a success rate of over 95 % in typical bin picking scenarios. For unknown objects, it can achieve a grasp rate of over 92 % in isolated settings.

Pre-grasping manipulation We proposed an approach to self-supervised learning for pre-grasping manipulation to improve the expected grasp success afterwards. First, this allowed to empty a bin completely even with planar grasping. Second, we integrated prior knowledge into the learning algorithm by making the reward of one skill (shifting) dependent on the other (grasping). This way, we were able to bypass sparse rewards for data-efficient learning. By choosing between a grasp with an estimated probability and a pre-grasping action for improving said probability, the system can optimize the grasping speed to 274 ± 3

PPH. Again, the robot is able to generalize to unknown objects.

Semantic grasping We extended planar grasping by choosing a specific goal object type at run-time. By using an image of the goal object as input, the object type doesn't need to be known at training time. Data is generated in a self-supervised manner by grasping objects out of clutter and taking an isolated image afterwards. The system makes use of contrastive learning to choose an object based on similarity: Here, an embedding is used to combine semantic grasping with an already learned grasp reward estimator. For a diverse set of trained objects, selection rates of 90 % were achieved using a dataset of 400 semantic grasps only.

Pick-and-place We presented an approach for learning pick-and-place tasks in a self-supervised manner. Since our approach does not depend on an object model but instead takes a demonstrated goal image as input, it allows for the flexible yet precise placing of unknown objects. Similar to semantic grasping, it predicts an embedding to combine a grasping primitive with a corresponding placing primitive. The evaluated system was trained in the real world with up to 25 000 pick-and-place actions, resulting in average placement errors of (2.7 ± 0.2) mm and $(2.6 \pm 0.8)^\circ$ for trained objects. A separated approach, where grasp and place actions are calculated independently, resulted in over four times lower precision. For unknown objects, the translational placement error increased to around 6 mm and 9 mm for grasping in clutter. The robot learns to select the demonstrated objects out of five alternatives with up to 86 % accuracy. Moreover, we demonstrated the robot's capability to pick-and-place multiple objects from a single goal state.

Transition model A transition model for robotic manipulation, for both the next visual state and its pixel-wise uncertainty, allowed the robot to plan sequences of single-step manipulation primitives. This way, they could be reused and learned data-efficiently without sparse rewards. By skipping image measurements, the robot was able to increase the PPH by around 15 %. Furthermore, the robot was able to optimize regarding multiple-step criteria in a flexible way, e.g. to minimize the number of actions for emptying a bin. This becomes possible through three major contributions: First, we simplify the transition model by conditioning it on the already learned reward estimation as well as using the locality of the state space. Second, we propose a model architecture generating samples from a multi-modal distribution, allowing to estimate a prediction uncertainty. Third, the latter was propagated efficiently towards the estimated action's success reward.

Garment folding Garment folding showed that manipulation primitives can scale to more complex real-world tasks. We presented a bimanual robotic system for the efficient folding of garments from arbitrary initial and crumpled configurations. At its core, a novel bimanual NN architecture predicts two poses to parameterize a rich set of manipulation primitives. A correspondence descriptor, extending the embedding introduced for pick-and-place, was proposed to efficiently calculate conditioned bimanual poses. After learning from

4300 human-annotated or self-supervised actions, the robot can fold garments in under 120 s on average with a success rate of 93 %. Again, the presented approach allowed the robot to generalize to unknown garments.

In the bigger picture, these various manipulation tasks offered some common insight. Throughout, a NN predicted the reward of manipulation primitives at a large number of possible poses. We find that ranking actions (by a critic) is easier than generating them (by an actor), which might be caused by the inherent multi-modality of manipulation. This affects both the data generation, as there is no "best grasp" that could be used for training, as well as the NN itself due to the high non-linearity of manipulation. While ranking is, as implemented in this work, limited to a discrete action space, we find that this has negligible effects on the evaluated manipulation tasks.

Many tasks like semantic grasping, pick-and-place, or bimanual garment folding require multiple actions (or inputs) to be calculated jointly. Then, embeddings can be used to condition actions on each other. Pre-selecting actions based on a tentative metric allows to keep the number of combinations small and mitigates an exploding action space for fast calculation.

Semantic grasping or pick-and-place require human input during run-time. Contrastive learning allows to compare multiple images in a one-shot manner. Given a current image of the scene, the approach learns the probability that a human-given image might be the outcome of a specific action, in contrast to a task-specific and pre-defined noise distribution. This way, we allow flexible task executions based on a single human-demonstrated goal state.

Most importantly, we confirm that "it is often much better to gather more data than to improve the learning algorithm" (Goodfellow et al. [43, p. 414]). Of course, different approaches regarding action spaces, control frequency, or machine learning algorithms might have a large impact on the task performance. Still, the task performance scales with the amount, quality, and variety of the data. We argue that data generation is, and should be seen, as a primarily *scientific* problem. Setting up training pipelines remains cumbersome, mostly due to robotic issues that pop up in unpredictable contact-rich manipulation. If a task, e.g. garment folding, is not solvable by purely self-supervised learning in practical time, we added an initial solution either by human imitation or annotation. This way, the self-supervised learning was kickstarted to transfer the policy to the robot. Throughout this work, we restricted ourselves to methods that allow generalization to new objects or environments. Given that premise, we found generalization to be mostly a matter of training data, again.

Chapter 10

Outlook

Flexible robotic manipulation remains an essential and challenging problem and will continue to do so in the foreseeable future. Robot learning promises to deal with the great complexity of manipulation, in particular regarding the required flexibility and possible deviations of the *exact* task. Usually, this refers to manipulating unknown objects in changing states or environments. Learned approaches are, in principle, able to generalize to these situations when trained on a sufficiently large dataset. Amongst others, this work has shown trade-offs between the robustness, the flexibility, and the training effort for a manipulation task. Future work should further focus on improving one or multiple criteria for various manipulation tasks.

Improving the manipulation robustness, for example for unknown objects, remains first and foremost a matter of training data. Reducing the required training data for learning tasks is an active field of research, both in the machine learning as well as the more applied robotics community. However, various tasks in machine learning, such as natural language synthesis [21] or image generation [98], show that the task performance scales with ever-growing datasets and NN capacity. We expect this effect to transfer to manipulation tasks.

We are therefore interested in increasing the general dataset size by order of magnitudes. Unfortunately, real-world data generation is comparably difficult and expensive in robotics. Moreover, datasets are often specific to a robotics setup or environment, and are therefore not easily shareable. An increased focus on techniques and tools for fast and large-scale data generation might be helpful instead. Still, very large datasets might not be achievable to gather in academic research. We argue that, for this reason, future work should see learning for manipulation even more in an industrial context. While industrial tasks are constrained by their relevance for applications, they might scale larger according to their business model. In addition, they can be defined and controlled precisely, allowing for a good learning environment as well as various tasks of incremental difficulty. In the end, large-scale data generation might not be a research problem but a question of a valid business model.

Novel tasks could be investigated for primitive-based manipulation, in particular by using new and more complex primitives. For example, robotic assembly consists of longer sequences of actions to solve multiple sub-tasks sequentially. Common subtasks include peg-in-hole or fitting that might be solved by extending adaptive primitives from section 5.2. A peg-in-hole primitive might include force feedback and be parametrized by the robot's stiffness. More complex primitives like a (classical) spiral search strategy or even a sub-policy learned by RL could be incorporated.

Mechanical search is the task of searching objects for semantic grasping. Then, pre-grasping primitives similar to section 5.4 for excavating objects become necessary. They could be extended by merging into a grasp-push primitive.

Soft object or fabric manipulation offers various interesting tasks beyond garment folding. Industrial use cases need methods that can learn to manipulate a garment with a novel shape given a few demonstrations. Combining garment folding with an upstream bin picking task would allow the isolation of various garment types.

For many complex tasks, the agent needs to take information about the goal into account. In contrast, naive RL specifies the goal entirely in the reward definition. In chapter 6, we proposed to use a goal image for the pick-and-place task. However, this might not be easily transferable to novel tasks. Instead, more abstract instructions, logical sequences, or even language input might be a better solution. We think that the *goal description* of a task so that it can be useful for a wide range of applications is a crucial field within robot learning.

6 DoF pick-and-place could be a straightforward extension of this work. Combining the approach of pick-and-place (chapter 6) based on a single goal image and a fully convolutional actor-critic architecture (section 5.3) could enable flexible and precise pick-and-place tasks with 6 DoF. In particular, placing unknown objects like shown in a single human demonstration could be applied for machine tending, simple assembly, or many service robotics tasks.

Bimanual grasping could allow the robots to grasp objects larger than the maximum gripper width by using two grippers to hold the object in between. This way, grasping could become even more flexible for a wider range of object sizes. We think that general bimanual grasping could be solved similarly to the primitives for garment folding (chapter 8). Bimanual grasping might become more important in the future if costs for robots and grippers decrease.

Closed-loop primitives could bridge the gap to more flexible *end-to-end* learning. The manipulation primitives introduced in chapter 3 are open-loop and therefore are required to have a *predictable* outcome. This might not hold in practice, either due to incomplete information or underlying process stochasticity. Then, intermediate sensor feedback might help to adapt the current primitive and improve its task performance. For example in pick-and-place (chapter 6), sensor information about the pose of the object within the

gripper is essential for a precise place pose. Similarly, sensor information about the pose of a garment within a closed gripper might be useful for high-quality garment folding. An in-hand camera or tactile sensors within the fingers might be beneficial, but require expensive open-loop training with long time horizons and sparse rewards. Keeping the number of sensor feedback events small allows for a comparably small action space. From a more general perspective, closed-loop primitives can be interpreted as a discrete simplification of continuous end-to-end learning.

Representation learning might find an intermediate representation between lower-level perception and higher-level manipulation features. So far, the reward of each manipulation primitive is estimated by taking low-level sensor data as input. An intermediate representation would allow learning a separate perception and a manipulation NN, and therefore sharing and reusing the perception NN for different manipulation tasks. The challenge is finding a task objective for the intermediate representation that is universal and generalizable. For example, one could estimate the mass density or material properties of objects, as they are relevant information for all downstream manipulation. Pick-and-place benefits from correspondence information between objects in different images. A spatial perception model of the environment might combine different cameras and a memory buffer for a refined scene understanding. However, splitting the NN into a *perception* and *manipulation* module might come with downsides: In general, manipulation of unknown objects is not really *abstractable*, as arbitrary small and low-level geometric information might influence an action's outcome.

In the long term, general artificial intelligence might lead to human-like performance and robustness even for arbitrarily complex manipulation tasks. Even these high expectations might be insufficient, as infants need *years* to learn robust grasping or manipulation. In the meantime, it remains the puzzle of the robotics community to find, design, and solve useful tasks with restricted domain knowledge.

List of Figures

2.1 Object- or action-centric manipulation with model- or learning-based approaches. . . . 5
2.2 Examples of classical approaches of grasp generation. . . . 7
2.3 Learning enables skipping an intermediate object representation prevalent in prior manipulation approaches, opening the door for generalization to unknown objects. . . . 9
2.4 Examples of robot manipulation with different extent of learning. . . . 10

3.1 A summary of reinforcement learning (RL). . . . 22
3.2 Example of perspective and orthographic image projections. . . . 25
3.3 Pose definition in the scene frame. . . . 27
3.4 Viewpoint invariance of a manipulation model. . . . 29
3.5 Localities around the robot's gripper. . . . 31
3.6 Sliding-window approach for translational invariance and bounded locality. 31
3.7 Multiplication step of a convolution operation. . . . 37

4.1 Speeding-up real-world robot learning. . . . 42
4.2 A time-optimal seven-step profile for trajectory generation. . . . 47
4.3 Decision tree for the optional brake pre-trajectory. . . . 48
4.4 A single blocked interval in state-to-state trajectory generation. . . . 52
4.5 Example of finding the minimum non-blocked duration of multiple DoFs. . 53
4.6 Calculation performance depending on the number of degrees of freedom (DoFs). . . . 56
4.7 Example application for OTG in the field of HRC. . . . 57
4.8 Example of offline planning with Ruckig. . . . 58
4.9 General hardware setup of the Franka Emika Panda robot arm. . . . 60
4.10 Example images of the bin taken by the respective cameras. . . . 62
4.11 Screenshot of the database viewer. . . . 63

5.1 Neural network architecture for planar grasping. . . . 67

5.2 Example heatmaps for planar grasping. The estimated grasp reward, averaged over all rotations α and gripper widths d, is shown ranging from 0 (blue) to 1 (red). 68
5.3 Object sets for evaluating planar grasping. 70
5.4 Example images of the bin with its border (white) and a successful grasp (red) and the window used for its calculation (blue). 71
5.5 The BCE loss and the accuracy of the NN depending on the training progress, corresponding to the training set size N. 71
5.6 Examples of cropped image windows $\hat{s}$ around the tool center point (TCP) (red). The border indicates a grasp success (green) or failure (red). 72
5.7 The maximum estimated grasp reward over the translational and rotational resolution. 73
5.8 The calculation duration depending on the size of the bin with constant resolution. 74
5.9 The grasp rate of cubes (10 out of 15 objects), depending on the trained object type (label) and their corresponding dataset sizes N. 75
5.10 Advantages of 6 DoF over planar grasping 76
5.11 Sectional drawings of the depth profiles to illustrate the controller's model-based adaptions. 78
5.12 During inference, a fully convolutional neural network (FCNN) estimates the reward for a grid of grasp poses (heatmap). A model-based controller calculates the lateral DoF (z, β, γ) based on the selected planar grasp (x, y, α, m). During training, the reward of the *adapted* grasp is fed into the NN. 80
5.13 Example of successful 6 DoF grasps with model-based adaption. 81
5.14 Histogram of the lateral angles β and γ (top) and their corresponding mean grasp success (bottom) during training. 81
5.15 Neural network architecture for 6 DoF grasping. 85
5.16 Example images with corresponding heatmaps for the fully convolutional actor-critic architecture 86
5.17 Object sets for evaluating 6 DoF grasping with an actor-critic architecture . 90
5.18 Example windows $\hat{s}$ of (successful) human grasp demonstrations 90
5.19 Estimation accuracy over the training progress. 91
5.20 Example 6 DoF grasps on unknown objects 93
5.21 Accuracy of the critic NN depending on the window size of the input image. 95
5.22 The actor's loss and inference duration depending on the number of non-planar action hypotheses. 96
5.23 Grasp success rate depending on the viewpoint orientation β (the grasp orientation given in the camera frame). 97
5.24 Grasp success rate depending on the weight w_b of the behavior cloning (based on human grasp demonstrations). The grasp rate is measured by grasping 10 out of 20 simple objects out of a bin. 97
5.25 Common failure cases of our 6 DoF grasping pipeline. 98

5.26 A shift primitive for pre-grasping manipulation allows the robot to grasp objects that are blocked due to collisions. 100
5.27 Examples of depth images $\hat{s}$ before and after a shifting primitive. 101
5.28 Example heatmaps of the shifting primitive for pre-grasping manipulation. . 102
5.29 State diagram of the combined grasping and shifting controller; common threshold parameters are $\psi_g \approx 0.75$ and $\rho_s \approx 0.6$. 103
5.30 The object sets for learning pre-grasping manipulation. 104
5.31 Grasp rate and picks per hour (PPH) depending on the threshold probability between shifting and grasping. 105
5.32 Neural network architecture for semantic grasping. 109
5.33 Example heatmaps of semantic grasping. 110
5.34 The object sets for training and evaluating semantic grasping. 111
5.35 The selection attempt rate depending on the object reward weight ω_o. . . . 113

6.1 A pick-and-place action with a grasp (left) and the subsequent place (right. 118
6.2 Dataset samples from successful pick-and-place actions 119
6.3 Further hindsight for subsequent pick-and-place episodes 121
6.4 Neural network architecture for pick-and-place during training 122
6.5 Neural network architecture for pick-and-place during inference 123
6.6 Example heatmaps of the estimated grasp and place reward. 124
6.7 Algorithm of inferring a pick-and-place action using FCNNs and reward pre-estimation. 125
6.8 Translational and rotational mean placement error 126
6.9 The evaluation set of 50 unknown objects. 127
6.10 Success rates of pick-and-placing the demonstrated objects for the 1-out-of-5 selection task. 128
6.11 Given a single goal state, our robot can infer multiple pick-and-place actions from the grasp scene to the place scene. 129

7.1 NN architecture with separated manipulation and transition models 132
7.2 Iterative multiple-step predictions of the state s_t with corresponding uncertainty Σ_t^2. 134
7.3 Example predictions of an image window $\hat{s}_{t+1}$ after an executed manipulation primitive. 136
7.4 Grasp rate and uncertainty of the estimated reward σ_a^2 depending on the number of prediction steps t. 137
7.5 Example scene for evaluating planning capabilities of the learned transition model. 139
7.6 Example predictions of the visual state $\hat{s}_{t+1}$ after an executed manipulation primitive m_t for novel (unknown) objects. The uncertainties are shown from low (blue) to high (red). 139

8.1 Manipulation primitives for garment folding 142

8.2 Neural network architecture for garment folding 144
8.3 Reachability of the robot 145
8.4 Folding sequence of the robot 147
8.5 Folding approaches 148
8.6 Coverage and sufficiently smoothed prediction over the number of steps . . 149
8.7 The system can generalize to unseen garments of different color, pattern, and material. 149
8.8 Timings for garment folding 152

9.1 Summary of relevant contributions and investigated tasks. 156

List of Tables

4.1 Steps of Constant Jerk of an Extremal Profile. 49
4.2 Time-Optimal Profile Types for the UP Direction. 50
4.3 Calculation Performance for 7 DoFs. 56
4.4 Example waypoints for offline trajectory planning. 58

5.1 Grasp rate for planar grasping in different bin picking scenarios. 72
5.2 Grasp rate of 6 DoF or planar grasping for different bin picking scenarios. . 82
5.3 Processing and NN inference time of different grasping approaches. Comparisons to related work by [96]. 83
5.4 Methods for Recording Grasp Demonstrations. 89
5.5 Grasp Rate in Different Bin Picking Scenarios. 92
5.6 Details about the Calculation Duration [ms]. 94
5.7 Collision Rate (10 out of 20). 94
5.8 Accuracy of a fully convolutional vs. unrestricted NN architecture. 95
5.9 Picks per hour (PPH), grasp rate, and shifts per grasp in different bin picking scenarios. 104
5.10 Grasp rate and shifts per grasp for novel (unknown) objects from non-graspable positions. 106
5.11 Success rates of selecting the correct target object in different scenarios of semantic grasping. 112
5.12 Calculation Duration of Semantic Grasping [ms]. 113

6.1 Success rates of inserting a given object onto a peg 127

7.1 Picks per hour in the Box and Blocks Test. 138

8.1 Experimental results for end-to-end folding for different NN architectures, folding approaches, and garments 151

Bibliography

[1] How to fold a t-shirt in two seconds. `https://www.wikihow.com/Fold-a-T-Shirt-in-Two-Seconds`. Accessed: 2022-03-01.

[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. {TensorFlow}: a system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016.

[3] Kitak Ahn, Wan Kyun Chung, and Youngil Yourn. Arbitrary states polynomial-like trajectory (aspot) generation. In *30th Annual Conference of IEEE Industrial Electronics Society, IECON*, volume 1, pages 123–128. IEEE, 2004.

[4] Yahav Avigal, Lars Berscheid, Tamim Asfour, Torsten Kröger, and Ken Goldberg. Speedfolding: Learning efficient bimanual folding of garments. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2022.

[5] Devin J Balkcom and Matthew T Mason. Robotic origami folding. *The International Journal of Robotics Research*, 27(5):613–627, 2008.

[6] Christian Bersch, Benjamin Pitzer, and Sören Kammel. Bimanual robotic cloth manipulation for laundry folding. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1413–1419. IEEE, 2011.

[7] Lars Berscheid and Torsten Kröger. Jerk-limited real-time trajectory generation with arbitrary target states. *Robotics: Science and Systems XVII*, 2021.

[8] Lars Berscheid, Pascal Meißner, and Torsten Kröger. Robot learning of shifting objects for grasping in cluttered environments. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 612–618. IEEE, 2019.

[9] Lars Berscheid, Pascal Meißner, and Torsten Kröger. Self-supervised learning for precise pick-and-place without object model. *IEEE Robotics and Automation Letters*, 5(3):4828–4835, 2020.

[10] Lars Berscheid, Christian Friedrich, and Torsten Kröger. Robot learning of 6 dof grasping using model-based adaptive primitives. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4474–4480. IEEE, 2021.

[11] Lars Berscheid, Pascal Meißner, and Torsten Kröger. Learning a generative transition model for uncertainty-aware robotic manipulation. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4670–4677. IEEE, 2021.

[12] Marius Beul and Sven Behnke. Analytical time-optimal trajectory generation and control for multirotors. In *International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 87–96. IEEE, 2016.

[13] Antonio Bicchi and Vijay Kumar. Robotic grasping and contact: A review. In *Proceedings 2000 ICRA. Millennium conference. IEEE international conference on robotics and automation. Symposia proceedings (Cat. No. 00CH37065)*, volume 1, pages 348–353. IEEE, 2000.

[14] Alexander Bierbaum, Matthias Rambow, Tamim Asfour, and Rüdiger Dillmann. Grasp affordances from multi-fingered tactile exploration using dynamic potential fields. In *2009 9th IEEE-RAS International Conference on Humanoid Robots*, pages 168–174. IEEE, 2009.

[15] Jeannette Bohg, Kai Welke, Beatriz León, Martin Do, Dan Song, Walter Wohlkinger, Marianna Madry, Aitor Aldóma, Markus Przybylski, Tamim Asfour, et al. Task-based grasp adaptation on a humanoid robot. *IFAC Proceedings Volumes*, 45(22):779–786, 2012.

[16] Jeannette Bohg, Antonio Morales, Tamim Asfour, and Danica Kragic. Data-Driven Grasp Synthesis - A Survey. *IEEE Transactions on Robotics*, 30(2):289–309, April 2014.

[17] Byron Boots, Arunkumar Byravan, and Dieter Fox. Learning predictive models of a depth camera & manipulator from raw execution traces. In *2014 IEEE International Conference on Robotics and Automation*, pages 4021–4028, 2014.

[18] Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt Konolige, et al. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4243–4250. IEEE, 2018.

[19] Gary Bradski. The opencv library. *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, 25(11):120–123, 2000.

[20] Xavier Broquere, Daniel Sidobre, and Ignacio Herrera-Aguilar. Soft motion trajectory planner for service manipulator robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2808–2813. IEEE, 2008.

[21] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[22] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar. Benchmarking in manipulation research: Using the yale-cmu-berkeley object and model set. *IEEE Robotics Automation Magazine*, 22(3):36–52, Sep. 2015. ISSN 1070-9932. doi: 10.1109/MRA.2015.2448951.

[23] Lillian Y Chang, Siddhartha S Srinivasa, and Nancy S Pollard. Planning pre-grasp manipulation for transport tasks. In *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2697–2704. IEEE, 2010.

[24] Dmitry Chetverikov, Dmitry Svirko, Dmitry Stepanov, and Pavel Krsek. The trimmed iterative closest point algorithm. In *2002 International Conference on Pattern Recognition*, volume 3, pages 545–548. IEEE, 2002.

[25] Erwin Coumans. Bullet physics simulation. In *ACM SIGGRAPH 2015 Courses*, page 1. 2015.

[26] Hao Dang and Peter K Allen. Semantic grasping: Planning robotic grasps functionally suitable for an object manipulation task. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1311–1317. IEEE, 2012.

[27] Coline Devin, Pieter Abbeel, Trevor Darrell, and Sergey Levine. Deep object-centric representations for generalizable robot learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7111–7118. IEEE, 2018.

[28] Rosen Diankov and James Kuffner. Openrave: A planning architecture for autonomous robotics. *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34*, 79, 2008.

[29] Mehmet R Dogar and Siddhartha S Srinivasa. Push-grasping with dexterous hands: Mechanics and a method. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2123–2130. IEEE, 2010.

[30] Andreas Doumanoglou, Jan Stria, Georgia Peleka, Ioannis Mariolis, Vladimir Petrik, Andreas Kargakos, Libor Wagner, Václav Hlaváč, Tae-Kyun Kim, and Sotiris Malassiotis. Folding clothes autonomously: A complete pipeline. *IEEE Transactions on Robotics*, 32(6):1461–1478, 2016.

[31] Yan Duan, Marcin Andrychowicz, Bradly Stadie, OpenAI Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. In *Advances in neural information processing systems*, pages 1087–1098, 2017.

[32] Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018.

[33] Aaron Edsinger and Charles C Kemp. Two arms are better than one: A behavior based control system for assistive bimanual manipulation. In *Recent progress in robotics: Viable robotic service to human*, pages 345–355. Springer, 2007.

[34] Javier Felip, Janne Laaksonen, Antonio Morales, and Ville Kyrki. Manipulation primitives: A paradigm for abstraction and execution of grasping and manipulation tasks. *Robotics and Autonomous Systems*, 61(3):283–296, 2013.

[35] Carlo Ferrari and John Canny. Planning optimal grasps. In *Proceedings 1992 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2290–2295. IEEE, 1992.

[36] Bernd Finkemeyer, Torsten Kröger, and Friedrich M Wahl. Executing assembly tasks specified by manipulation primitive nets. *Advanced Robotics*, 19(5):591–611, 2005.

[37] C. Finn and S. Levine. Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2786–2793, May 2017.

[38] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep Spatial Autoencoders for Visuomotor Learning. In *International Conference on Robotics and Automation (ICRA)*, pages 512–519. IEEE, 2016.

[39] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot visual imitation learning via meta-learning. In *Conference on Robot Learning*, pages 357–368, 2017.

[40] Peter R Florence, Lucas Manuelli, and Russ Tedrake. Dense object nets: Learning dense visual object descriptors by and for robotic manipulation. In *Conference on Robot Learning*, pages 373–385. PMLR, 2018.

[41] Aditya Ganapathi, Priya Sundaresan, Brijen Thananjeyan, Ashwin Balakrishna, Daniel Seita, Jennifer Grannen, Minho Hwang, Ryan Hoque, Joseph E Gonzalez, Nawid Jamali, et al. Learning dense visual correspondences in simulation to smooth and fold real fabrics. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11515–11522. IEEE, 2021.

[42] Irene Garcia-Camacho, Martina Lippi, Michael C Welle, Hang Yin, Rika Antonova, Anastasiia Varava, Julia Borras, Carme Torras, Alessandro Marino, Guillem Alenya, et al. Benchmarking bimanual cloth manipulation. *IEEE Robotics and Automation Letters*, 5(2):1111–1118, 2020.

[43] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[44] Jennifer Grannen, Priya Sundaresan, Brijen Thananjeyan, Jeffrey Ichnowski, Ashwin Balakrishna, Vainavi Viswanath, Michael Laskey, Joseph Gonzalez, and Ken Goldberg. Untangling dense knots by learning task-relevant keypoints. In *Conference on Robot Learning*, pages 782–800. PMLR, 2021.

[45] Marcus Gualtieri and Robert Platt. Learning 6-dof grasping and pick-place using attention focus. In *Proceedings of 2nd Conference on Robot Learning (CoRL 2018)*, 2018.

[46] Marcus Gualtieri, Andreas Ten Pas, Kate Saenko, and Robert Platt. High precision grasp pose detection in dense clutter. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 598–605. IEEE, 2016.

[47] Marcus Gualtieri, Andreas ten Pas, and Robert Platt. Pick and place without geometric object models. In *Proceedings of 2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[48] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 297–304, 2010.

[49] Huy Ha and Shuran Song. Flingbot: The unreasonable effectiveness of dynamic manipulation for cloth unfolding. In *Conference on Robot Learning*, pages 24–33. PMLR, 2022.

[50] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pages 2555–2565. PMLR, 2019.

[51] Robert Haschke, Erik Weitnauer, and Helge Ritter. On-line planning of time-optimal, jerk-limited trajectories. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3248–3253. IEEE, 2008.

[52] Shogo Hayakawa, Weiwei Wan, Keisuke Koyama, and Kensuke Harada. A dual-arm robot that autonomously lifts up and tumbles heavy plates using crane pulley blocks. *IEEE Transactions on Automation Science and Engineering*, 2021.

[53] Ryan Hoque, Daniel Seita, Ashwin Balakrishna, Aditya Ganapathi, Ajay Kumar Tanwani, Nawid Jamali, Katsu Yamane, Soshi Iba, and Ken Goldberg. Visuospatial foresight for multi-step, multi-task fabric manipulation. *Robotics: Science and Systems (RSS)*, 2020.

[54] Yang Hu, Lin Zhang, Wei Li, and Guang-Zhong Yang. Robotic sewing and knot tying for personalized stent graft manufacturing. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 754–760. IEEE, 2018.

[55] Shariq Iqbal, Jonathan Tremblay, Andy Campbell, Kirby Leung, Thang To, Jia Cheng, Erik Leitch, Duncan McKay, and Stan Birchfield. Toward sim-to-real directional semantic grasping. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7247–7253. IEEE, 2020.

[56] Stephen James, Andrew J Davison, and Edward Johns. Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task. In *Conference on Robot Learning*, pages 334–343. PMLR, 2017.

[57] Eric Jang, Sudheendra Vijayanarasimhan, Peter Pastor, Julian Ibarz, and Sergey Levine. End-to-end learning of semantic grasping. In *Conference on Robot Learning*, pages 119–132, 2017.

[58] Eric Jang, Coline Devin, Vincent Vanhoucke, and Sergey Levine. Grasp2vec: Learning object representations from self-supervised grasping. In *Conference on Robot Learning*, pages 99–112. PMLR, 2018.

[59] Michael Janner, Sergey Levine, William T Freeman, Joshua B Tenenbaum, Chelsea Finn, and Jiajun Wu. Reasoning about physical interactions with object-oriented prediction and planning. In *International Conference on Learning Representations*, 2018.

[60] Yun Jiang, Marcus Lim, Changxi Zheng, and Ashutosh Saxena. Learning to place new objects in a scene. *The International Journal of Robotics Research*, 31(9):1021–1043, 2012.

[61] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning*, 2018.

[62] Jonas C Kiemel and Torsten Kröger. Learning robot trajectories subject to kinematic joint constraints. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4799–4805. IEEE, 2021.

[63] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[64] Kilian Kleeberger and Marco F Huber. Single shot 6d object pose estimation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6239–6245. IEEE, 2020.

[65] Kilian Kleeberger, Richard Bormann, Werner Kraus, and Marco F Huber. A survey on learning-based robotic grasping. *Current Robotics Reports*, 1(4):239–249, 2020.

[66] Oliver Kroemer, Scott Niekum, and George Konidaris. A review of robot learning for manipulation: Challenges, representations, and algorithms. *arXiv:1907.03146*, 2019.

[67] Torsten Kröger. Opening the door to new sensor-based robot applications - the reflexxes motion libraries. In *IEEE International Conference on Robotics and Automation*, pages 1–4. IEEE, 2011.

[68] Torsten Kröger and Friedrich M Wahl. Online trajectory generation: Basic concepts for instantaneous reactions to unforeseen events. *IEEE Transactions on Robotics*, 26 (1):94–111, 2009.

[69] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[70] Robert Lee, Daniel Ward, Vibhavari Dasagi, Akansel Cosgun, Juxi Leitner, and Peter Corke. Learning arbitrary-goal fabric folding with one hour of real robot experience. In *Conference on Robot Learning*, pages 2317–2327. PMLR, 2021.

[71] Puttichai Lertkultanon and Quang-Cuong Pham. A certified-complete bimanual manipulation planner. *IEEE Transactions on Automation Science and Engineering*, 15 (3):1355–1368, 2018.

[72] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-End Training of Deep Visuomotor Policies. *Journal of Machine Learning Research*, page 40, 2016.

[73] Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. Learning Hand-Eye Coordination for Robotic Grasping with Large-Scale Data Collection. In *International Symposium on Experimental Robotics*, pages 173–184. Springer, 2016.

[74] Ying Li, Jiaxin L Fu, and Nancy S Pollard. Data-driven grasp synthesis using shape matching and task-based pruning. *IEEE Transactions on visualization and computer graphics*, 13(4):732–747, 2007.

[75] Hongzhuo Liang, Xiaojian Ma, Shuang Li, Michael Görner, Song Tang, Bin Fang, Fuchun Sun, and Jianwei Zhang. Pointnetgpd: Detecting grasp configurations from point sets. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3629–3635. IEEE, 2019.

[76] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous Control with Deep Reinforcement Learning. *arXiv:1509.02971 [cs, stat]*, September 2015. URL `http://arxiv.org/abs/1509.02971`.

[77] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[78] Jens Lundell, Francesco Verdoja, and Ville Kyrki. Beyond top-grasps through scene completion. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 545–551. IEEE, 2020.

[79] Sonja Macfarlane and Elizabeth A Croft. Jerk-bounded manipulator trajectory planning: design for real-time applications. *IEEE Transactions on robotics and automation*, 19(1):42–52, 2003.

[80] Jeffrey Mahler and Ken Goldberg. Learning Deep Policies for Robot Bin Picking by Simulating Robust Grasping Sequences. In *Conference on Robot Learning*, pages 515–524, 2017.

[81] Jeffrey Mahler, Florian T. Pokorny, Brian Hou, Melrose Roderick, Michael Laskey, Mathieu Aubry, Kai Kohlhoff, Torsten Kroger, James Kuffner, and Ken Goldberg. Dex-Net 1.0: A cloud-based network of 3d objects for robust grasp planning using a Multi-Armed Bandit model with correlated rewards. pages 1957–1964. IEEE, May 2016. ISBN 978-1-4673-8026-3. doi: 10.1109/ICRA.2016.7487342.

[82] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. In *Robotics: Science and Systems (RSS)*, 2017.

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

[84] Christian Mandery, Ömer Terlemez, Martin Do, Nikolaus Vahrenkamp, and Tamim Asfour. The kit whole-body human motion database. In *2015 International Conference on Advanced Robotics (ICAR)*, pages 329–336. IEEE, 2015.

[85] Matthew T Mason. Progress in nonprehensile manipulation. *The International Journal of Robotics Research*, 18(11):1129–1141, 1999.

[86] Andrew T Miller and Peter K Allen. Graspit! A Versatile Simulator for Robotic Grasping. *IEEE Robotics & Automation Magazine*, 11(4):110–122, 2004.

[87] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. *arXiv:1312.5602 [cs]*, December 2013. URL `http://arxiv.org/abs/1312.5602`.

[88] Arsalan Mousavian, Clemens Eppner, and Dieter Fox. 6-dof graspnet: Variational grasp generation for object manipulation. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2901–2910. IEEE, 2019.

[89] Adithyavairavan Murali, Arsalan Mousavian, Clemens Eppner, Chris Paxton, and Dieter Fox. 6-dof grasping for target-driven object manipulation in clutter. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6232–6238. IEEE, 2020.

[90] Robert Paolini, Alberto Rodriguez, Siddhartha S Srinivasa, and Matthew T Mason. A data-driven statistical framework for post-grasp manipulation. *The International Journal of Robotics Research*, 33(4):600–615, 2014.

[91] T. Pardi, R. Stolkin, and A. M. Ghalamzan E. Choosing grasps to enable collision-free post-grasp manipulations. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pages 299–305, Nov 2018. doi: 10.1109/HUMANOIDS.2018.8625027.

[92] Peter Pastor, Ludovic Righetti, Mrinal Kalakrishnan, and Stefan Schaal. Online movement adaptation based on previous sensor experiences. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 365–371. IEEE, 2011.

[93] Jan Peters and Stefan Schaal. Reinforcement Learning of Motor Skills with Policy Gradients. *Neural Networks*, 21(4):682–697, May 2008. ISSN 08936080. doi: 10.1016/j.neunet.2008.02.003.

[94] Lerrel Pinto and Abhinav Gupta. Supersizing Self-supervision: Learning to Grasp from 50k Tries and 700 Robot Hours. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 3406–3413. IEEE, 2016.

[95] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[96] Yuzhe Qin, Rui Chen, Hao Zhu, Meng Song, Jing Xu, and Hao Su. S4g: Amodal single-view single-shot se (3) grasp detection in cluttered scenes. In *Conference on Robot Learning*, pages 53–65, 2020.

[97] Deirdre Quillen, Eric Jang, Ofir Nachum, Chelsea Finn, Julian Ibarz, and Sergey Levine. Deep reinforcement learning for vision-based robotic grasping: A simulated comparative evaluation of off-policy methods. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6284–6291. IEEE, 2018.

[98] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pages 8821–8831. PMLR, 2021.

[99] Joseph Redmon and Anelia Angelova. Real-time Grasp Detection using Convolutional Neural Networks. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 1316–1322. IEEE, 2015.

[100] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[101] Andrei A Rusu, Matej Večerík, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets. In *Conference on robot learning*, pages 262–270. PMLR, 2017.

[102] A. Sahbani, S. El-Khoury, and P. Bidaud. An Overview of 3d Object Grasp Synthesis Algorithms. *Robotics and Autonomous Systems*, 60(3):326–336, March 2012.

[103] JK Salisbury. Kinematics and force analysis of articulated hands, phd thesis, 1982.

[104] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[105] Ashutosh Saxena, Justin Driemeyer, and Andrew Y Ng. Robotic Grasping of Novel Objects Using Vision. *The International Journal of Robotics Research*, 27(2):157–173, 2008.

[106] Connor Schenck and Dieter Fox. Perceiving and reasoning about liquids using fully convolutional networks. *The International Journal of Robotics Research*, 37(4-5): 452–471, 2018.

[107] Philipp Schmidt, Nikolaus Vahrenkamp, Mirko Wächter, and Tamim Asfour. Grasping of unknown objects using deep convolutional neural networks based on depth images. In *2018 IEEE International Conference on Robotics and Automation*, pages 6831–6838. IEEE, 2018.

[108] Daniel Seita, Aditya Ganapathi, Ryan Hoque, Minho Hwang, Edward Cen, Ajay Kumar Tanwani, Ashwin Balakrishna, Brijen Thananjeyan, Jeffrey Ichnowski, Nawid Jamali, et al. Deep imitation learning of sequential fabric smoothing from an algorithmic supervisor. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9651–9658. IEEE, 2020.

[109] Daniel Seita, Pete Florence, Jonathan Tompson, Erwin Coumans, Vikas Sindhwani, Ken Goldberg, and Andy Zeng. Learning to rearrange deformable cables, fabrics, and bags with goal-conditioned transporter networks. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4568–4575. IEEE, 2021.

[110] Avi Singh, Larry Yang, Kristian Hartikainen, Chelsea Finn, and Sergey Levine. End-to-end robotic reinforcement learning without reward engineering. In *Robotics: Science and Systems (RSS)*, 2019.

[111] Christian Smith, Yiannis Karayiannidis, Lazaros Nalpantidis, Xavi Gratal, Peng Qi, Dimos V Dimarogonas, and Danica Kragic. Dual arm manipulation - a survey. *Robotics and Autonomous systems*, 60(10):1340–1353, 2012.

[112] Shuran Song, Andy Zeng, Johnny Lee, and Thomas Funkhouser. Grasping in the wild: Learning 6dof closed-loop grasping from low-cost demonstrations. *IEEE Robotics and Automation Letters*, 5(3):4978–4985, 2020.

[113] Alexander Sorokin, Dmitry Berenson, Siddhartha S Srinivasa, and Martial Hebert. People helping robots helping people: Crowdsourcing for grasping novel objects. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2117–2122. IEEE, 2010.

[114] Li Sun, Gerarado Aragon-Camarasa, Paul Cockshott, Simon Rogers, and J Paul Siebert. A heuristic-based approach for flattening wrinkled clothes. In *Conference Towards Autonomous Robotic Systems*, pages 148–160. Springer, 2013.

[115] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, Mass., nachdr. edition, 1998. ISBN 978-0-262-19398-6.

[116] Daisuke Tanaka, Solvi Arnold, and Kimitoshi Yamazaki. Emd net: An encode–manipulate–decode network for cloth manipulation. *IEEE Robotics and Automation Letters*, 3(3):1771–1778, 2018.

[117] Kenta Tanaka, Yusuke Kamotani, and Yasuyoshi Yokokohji. Origami folding by a robotic hand. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2540–2547. IEEE, 2007.

[118] Andreas ten Pas, Marcus Gualtieri, Kate Saenko, and Robert Platt. Grasp pose detection in point clouds. *The International Journal of Robotics Research*, 36(13-14): 1455–1473, 2017.

[119] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

[120] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[121] Jacob Varley, Chad DeChant, Adam Richardson, Joaquín Ruales, and Peter Allen. Shape completion enabled robotic grasping. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 2442–2447. IEEE, 2017.

[122] Mingli Wang, Juliang Xiao, Fan Zeng, and Guodong Wang. Research on optimized time-synchronous online trajectory generation method for a robot arm. *Robotics and Autonomous Systems*, 126:103453, 2020.

[123] Jonathan Weisz and Peter K Allen. Pose error robust grasping from contact wrench space metrics. In *2012 IEEE international conference on robotics and automation*, pages 557–562. IEEE, 2012.

[124] Thomas Weng, Sujay Man Bajracharya, Yufei Wang, Khush Agrawal, and David Held. Fabricflownet: Bimanual cloth manipulation with a flow-based policy. In *Conference on Robot Learning*, pages 192–202. PMLR, 2022.

[125] Bryan Willimon, Stan Birchfield, and Ian Walker. Model for unfolding laundry using interactive perception. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4871–4876. IEEE, 2011.

[126] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. *Robotics: Science and Systems (RSS)*, 2018.

[127] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[128] Andy Zeng, Shuran Song, Stefan Welker, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4238–4245. IEEE, 2018.

[129] Andy Zeng, Shuran Song, Kuan-Ting Yu, Elliott Donlon, Francois R Hogan, Maria Bauza, Daolin Ma, Orion Taylor, Melody Liu, Eudald Romo, et al. Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[130] Andy Zeng, Shuran Song, Johnny Lee, Alberto Rodriguez, and Thomas Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics. *IEEE Transactions on Robotics*, 36(4):1307–1319, 2020.

[131] Andy Zeng, Pete Florence, Jonathan Tompson, Stefan Welker, Jonathan Chien, Maria Attarian, Travis Armstrong, Ivan Krasin, Dan Duong, Vikas Sindhwani, et al. Transporter networks: Rearranging the visual world for robotic manipulation. In *Conference on Robot Learning*, pages 726–747. PMLR, 2021.

[132] Jialiang Zhao, Jacky Liang, and Oliver Kroemer. Towards precise robotic grasping by probabilistic post-grasp displacement estimation. In *Proceedings of Field and Service Robotics (FSR) 2019*, 2019.

[133] Jun-Yan Zhu, Richard Zhang, Deepak Pathak, Trevor Darrell, Alexei A Efros, Oliver Wang, and Eli Shechtman. Toward multimodal image-to-image translation. In *Advances in neural information processing systems (NIPS)*, pages 465–476, 2017.