Chair of Computer Science 6 (Machine Learning and Reasoning)

RWTH AACHEN UNIVERSITY

The present work was submitted to the Chair of Machine Learning and Reasoning.
Diese Arbeit wurde vorgelegt am Lehrstuhl für Maschinelles Lernen und Inferenz.

Learning Robot Motion Policies for Pushing Objects that Generalize

Master Thesis
Masterarbeit

Presented by / Vorgelegt von

Ümit Diker
407523

Supervised by / Betreut von Daniel Swoboda, M.Sc.

1st Examiner / 1. Prüfer Prof. Hector Geffner, Ph.D.

2nd Examiner / 2. Prüfer Prof. Dr. Christopher Morris

Aachen, September 7, 2025

Abstract

Generalization remains a key challenge in robotic manipulation, especially for non-prehensile tasks such as pushing. This thesis addresses the problem by reformulating pushing as a keypoint prediction task, where a single 3-D contact point defines an effective pushing strategy. Building on the CORN framework, the proposed architecture combines point cloud embeddings with cross-attention and additional observations to predict contact keypoints. Experiments in simulation with the Franka Emika Panda robot demonstrate that this abstraction improves generalization compared to direct Behavior Cloning. The model achieved robust performance under size variations of training objects and showed high success rates on novel objects. While our approach significantly improves performance over a baseline, excessive data generation pipelines and the reliance on additional object properties limit the policy's deployment to simulation.

Acknowledgments

I want to thank my supervisor, Daniel Swoboda, for his support, valuable feedback, and guidance throughout this thesis. I am also grateful to Prof. Hector Geffner and Prof. Christopher Morris for giving me the opportunity to pursue this work. I would also like to thank my colleagues and friends for the many discussions, advice, and encouragement along the way.

I am deeply grateful to my family, my mother Züleyha and my father Naçi, whose constant support and belief in me have enabled me to pursue my studies and reach this stage of my life. I also want to thank my sisters, Arzu and Cansu. Finally, I am thankful to Juana for her motivation, support, and encouragement at every stage of my studies.

Contents

1 Introduction

Daily object manipulation by humans relies on intuitive action rather than explicit planning. For robots, however, such tasks are far from trivial. In this work, we focus on non-prehensile pushing tasks. Pushing is a fundamental yet challenging manipulation skill for robots. Unlike grasping, which requires a precise grip, pushing only requires contact with an object's surface, which, however, still needs to be precise. It enables a robot to handle items that are too large, heavy, or irregularly shaped to be grasped easily. This flexibility makes pushing valuable in industrial applications, such as warehouse automation (e.g., sliding boxes into place), and household scenarios (e.g., rearranging objects on a countertop).

Real world environments are unpredictable, and objects vary in shape, size, or mass. A robot must be able to generalize across these variations. Classical motion planning approaches rely on accurate object geometry and physical parameters, which are often unavailable in practice [1, 2]. Alternative approaches of learning motion policies are Behavior Cloning (BC) with large, diverse datasets [3, 4] and Reinforcement Learning (RL) over millions of iterations [5]. However, BC based methods often fail on out-of-distribution inputs (e.g., objects of novel shape or unexpectedly high weight) [6], while RL can be computationally expensive and unstable.

In this thesis, we propose a method that reformulates pushing as a keypoint prediction problem. Instead of directly predicting robot actions, our model predicts a single 3-D keypoint that defines a suitable pushing contact. This compact representation abstracts away irrelevant details and emphasizes task-relevant geometry, thereby supporting transfer to new and unseen objects. Our approach builds upon CORN (Contact-based Object Representation for Non-prehensile Manipulation) [7], which we adapt to predict keypoints rather than actions. We evaluate the method against a BC baseline and analyze its ability to generalize across different conditions.

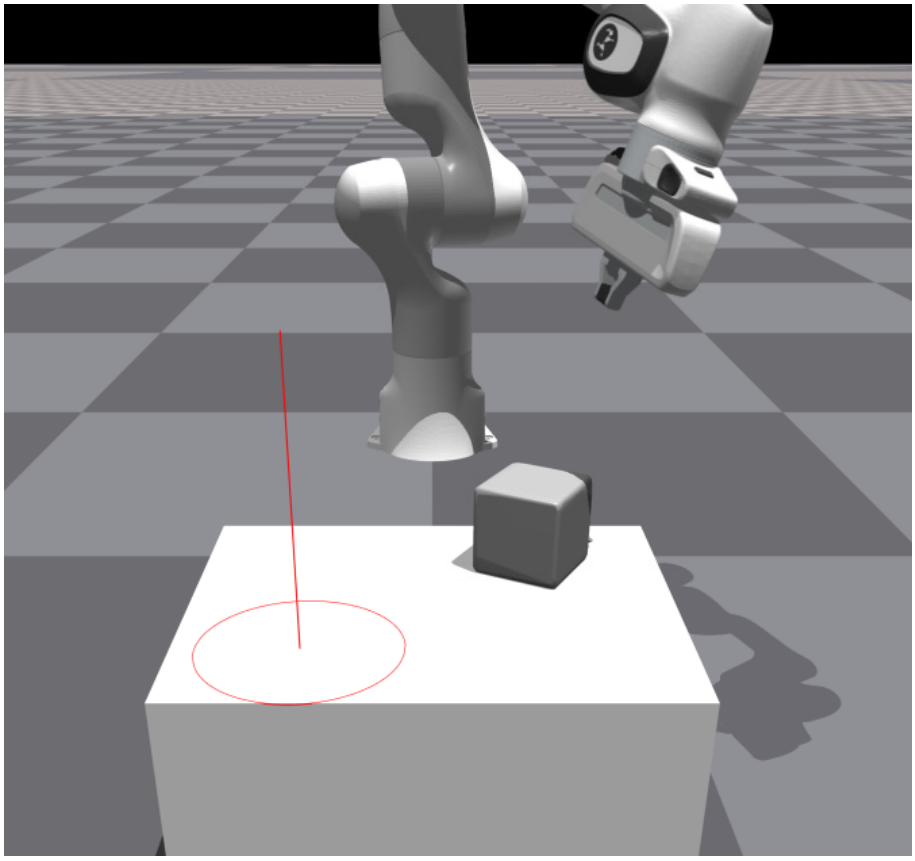
The remainder of this thesis is structured as follows: Section 1.1 provides motivating examples. Section 2 introduces the concept of generalization in robot learning and gives an overview of the theoretical background and techniques we use in the work. Section 3 reviews related work in Imitation and Reinforcement learning for robotic manipulation, planning approaches, as well as learning approaches to extract features from point clouds. Section 4 describes our method and architecture in detail, while Section 5 presents the experimental evaluation. In the experiments we will train and test on a variety of objects. Section 6 concludes with a discussion of the findings and future directions.

1.1 Motivation

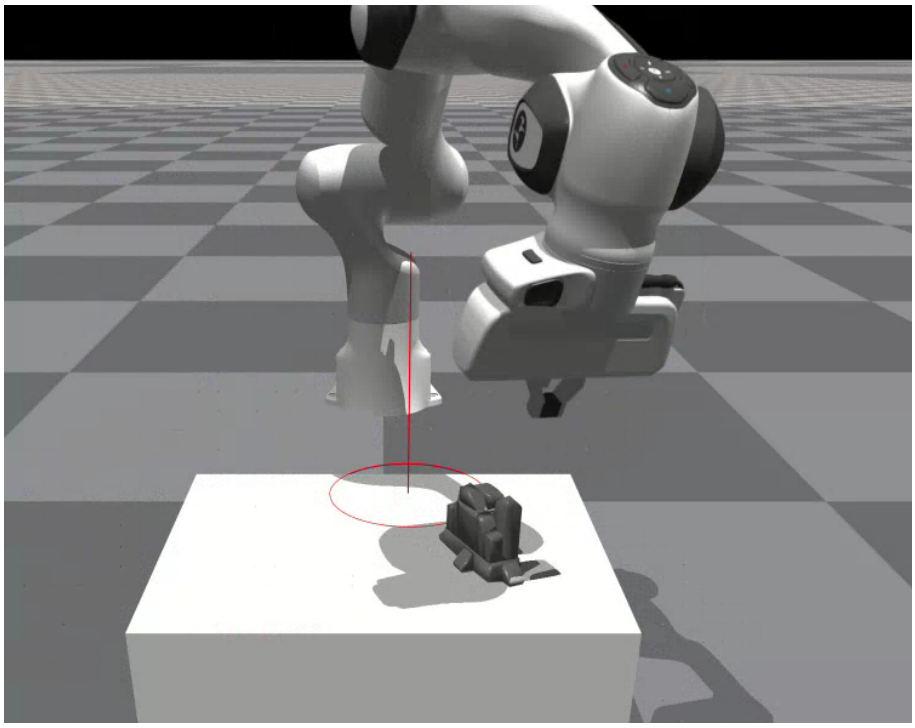
Consider a scenario where the robot is trained on a simple object such as a block but later has to manipulate an object with a very different geometry, like a toy covered with bumps, as shown in Figure 1.1. A direct imitation policy may fail in such cases, as it has only experienced a smooth and regular shape during training. This could possibly lead to movements, which cause the toy to flip, since the bumps give an uncertainty.

Instead of reasoning about every detail, a keypoint-based approach can identify local contact regions that are stable and predictable. By focusing on such keypoints, the robot can push effectively even when the object is unfamiliar. This abstraction enables the policy to generalize by consistently detecting and exploiting reliable contact zones, thereby minimizing slippage or tipping.

This example motivates our central research question: can keypoint prediction provide a robust representation that improves generalization in robotic pushing tasks and how can we learn such representations?



(a) Training on a block object.



(b) Testing on a toy with multiple bumps.

Figure 1.1: Examples with the Franka Emika Panda robot [8]. The policy is trained on a block object and tested on a novel toy object with bumps.

2 Background

The purpose of this chapter is to introduce the fundamental concepts and terminology that form the basis of this thesis. In this work the focus lies on non-prehensile pushing tasks, where the robot moves an object from a start to a target position through a pushing motion.

2.1 Learning for Manipulation

We now introduce the basic concepts that will be used in the following sections, along with possible learning approaches.

Non-Prehensile Manipulation Unlike prehensile manipulation, where objects are mostly grasped, non-prehensile manipulation refers to interactions without stable grasp, such as pushing, sliding, or throwing. In this work, we focus on pushing, where a robot moves an object across a surface toward a goal. Although simple to describe, pushing is challenging due to uncertainties in frictional contact and object dynamics, and has therefore become a standard benchmark for studying generalization in manipulation learning [9].

Multilayer Perceptrons (MLPs) A common building block in learning-based approaches is the multilayer perceptron (MLP) [10]. An MLP maps an input vector $\mathbf{x} \in \mathbb{R}^d$ through a sequence of affine transformations and nonlinear activations,

$$\mathbf{h}^{(l+1)} = \sigma\big(W^{(l)}\mathbf{h}^{(l)} + b^{(l)}\big),$$

where $W^{(l)}$ and $b^{(l)}$ are the weights and biases of layer l, and σ is a non-linear activation function (e.g., ReLU). During training, these parameters are adjusted to minimize a task-specific loss function, using gradient-based optimization with backpropagation to compute the weight updates. MLPs are widely used to approximate policies, as they can learn non-linear mappings from states to actions.

Point cloud A point cloud is a collection of points in 3-D space representing the shape and surface geometry of an object or scene,

$$\mathcal{P} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\},$$

where $\mathbf{x}_i \in \mathbb{R}^3$. Point clouds can vary in size and density depending on the sensor or sampling method, and they often contain noise or outliers [11].

Embeddings An embedding refers to a compact latent representation of high-dimensional input data [10]. In the context of robotic manipulation, embeddings are typically learned from observations such as images or point clouds. The goal is to transform raw sensory inputs into a structured feature space that captures the most task-relevant properties while discarding noise or redundant information. Embeddings reduce the dimensionality of the input, making subsequent policy learning more efficient. They also provide invariances, for instance, embeddings can be structured so that small perturbations in sensor noise do not alter the feature vector significantly. Finally, embeddings can capture semantic or geometric similarities in objects or object regions that afford similar interactions, which may be mapped to nearby points in the latent space. In practice, such embeddings are often obtained through an encoder network.

Encoder–Decoder Architectures Another widely applied concept in manipulation learning is the encoder–decoder architecture [12]. The encoder compresses high-dimensional observations such as point clouds or images into a latent representation, while the decoder transforms this representation into task-specific outputs (e.g., contact points, action distributions). This separation allows the network to first extract generalizable features and then map them flexibly to different prediction tasks.

Transformer Architecture Transformer [13] is widely used in natural-language processing. It consists of an encoder and a decoder. For our task we only need the encoder.

Figure 2.1 shows the encoder of a standard transformer. The input tokens are embedded and augmented with a positional encoding that restores order information. In the multi-head attention block the embeddings are linearly projected to queries Q, keys K and values V. Attention weights are obtained from the scaled dot product $QK^\top/\sqrt{d}$, where d is the dimensionality of the query and key vectors (i. e. the model dimension per head). Several heads run in parallel, allowing the model to focus on different relational patterns. Each attention block is followed by a residual connection and layer normalization. A position-wise MLP adds non-linearity, again wrapped by residual and normalization. Stacking L such layers yields contextualized token features that aggregate information from the entire sequence.

Cross Attention A key component in transformers in the decoder part, is the cross attention, which we will use in our approach. Cross attention links two sequences: a query set $X \in \mathbb{R}^{N_q \times d_{\text{in}}}$ and a context set $Y \in \mathbb{R}^{N_k \times d_{\text{in}}}$. It produces a context aware representation for each query.

First are the input vectors mapped into a common feature space:

$$Q = XW^Q, \quad K = YW^K, \quad V = YW^V, \tag{2.1}$$

with $W^Q, W^K, W^V \in \mathbb{R}^{d_{\text{in}} \times d}$ to be learned.

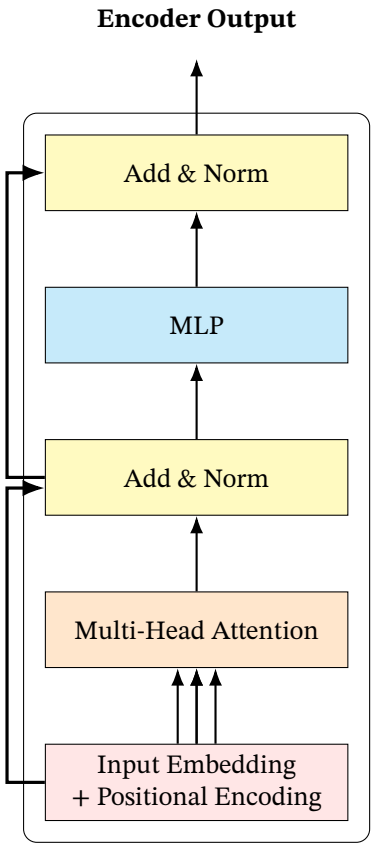


Figure 2.1: Transformer encoder block [13]. The input sequence is embedded and enriched with positional encoding, followed by multi-head attention, MLP, and each sub-layer is wrapped with a residual connection followed by layer normalization, denoted as Add & Norm. The output represents each token as a feature vector enriched by information from the surrounding context.

Afterwards, attention scores are computed as

$$A(Q,K,V) = \text{softmax}(\frac{QK^\top}{\sqrt{d}})V. \tag{2.2}$$

$A \in \mathbb{R}^{N_q \times d}$ is the attended representation.

The result is then transformed by $W^O \in \mathbb{R}^{d \times d_{\text{out}}}$:

$$Z = AW^O. \tag{2.3}$$

All matrices W^Q, W^K, W^V, W^O are parameters learned via backpropagation. In cross attention mostly the multi head variant is used to stabilize training and parallel computation. Here we split d into h heads ($d_h = d/h$), apply Eq. (2.2) per head, concatenate, then apply W^O (2.3).

Cross attention provides global access. Each query can attend to all key locations, enabling long range dependencies. It is input flexible and works with text, images, or scene embeddings. Attention weights are interpretable and allow insight into model behavior.

Learning based approaches There are several types of approaches that have been applied to robotic manipulation, such as Imitation Learning, Reinforcement Learning or Inverse Reinforcement Learning (IRL). In this work, we will focus on IL, especially BC. However having our own approach from an RL approach.

IL techniques employ expert demonstrations to learn tasks. In BC, these demonstrations are used as training data in a supervised learning framework to learn a policy. However, many BC-trained policies have a poor generalization performance, because they struggle when encountering unseen states or variations in the environment [6]. Moreover, performance in pure BC is limited by the quality of the demonstrator's behavior.

In RL however, a model learns by interacting with the environment, taking actions and receiving feedback in the form of rewards. The goal is to learn a policy that maximizes the expected reward.

In IRL, as described by Ng and Russell [14], we learn the reward function by observing an existing policy. The policy can be an expert or a working policy (e.g. a policy that can perform the task to a sufficient degree but is not necessarily optimal). The intuition of learning the reward function is that the reward function is the most succinct, robust, and transferable definition of the task. Once the reward function is learned, RL can be used to optimize a policy under that reward. IRL additionally solves the problem of reward engineering in RL, especially in complex environments, which can aid improving generalization.

In contrast to purely learning-based approaches, classical motion planning algorithms compute collision-free trajectory for the robot from a start to a goal state [15]. Such methods typically require an accurate model of the robot and its environment.

2.2 Generalization

Generalization for robotic manipulation involves the robot's ability to apply learned skills to scenarios that differ from those seen during training. Generalization can generally be distinguished based on two categories, In-distribution (ID) generalization and Out-of-distribution (OOD) generalization [16]:

- **ID generalization:** Variations remain within the same overall distribution as the training data (e.g., slight changes in object shape that still resemble the training examples).
- **OOD generalization:** The task conditions deviate significantly from anything the robot has seen before (e.g., objects with completely new shapes).

Mathematically, we can formalize it as follows. Let $\mathcal{D}$ be the overall distribution of the state representations and Π^* be the set of all possible expert policies $\pi^* \in \Pi^*$. Then, the generalization error of a policy π can be defined as:

$$\text{Error}(\pi, \mathcal{D}) = \arg\min_{\pi^*} \mathbb{E}_{s\sim\mathcal{D}}\Big[(\pi(s) - \pi^*(s))^2\Big]$$

Here $\mathbb{E}$ refers to the expectation and the goal is to find a policy π which makes this error small. ID generalization would only consider the distribution of the training set, which can differ from the overall distribution. OOD generalization would consider a distribution which differs from the training distribution. In practice, calculating this error is not possible, due to the lack of knowledge of the state distribution, as well as the set of expert policies. Instead, we use the empirical error. This allows us to apply an IL approach, or a RL approach [17] if no expert knowledge is available. More details are provided in Section 2.1, Section 2.3 and Section 2.4.

We aim to generalize to new objects with varying sizes and shapes. Achieving robust OOD generalization is a challenging task. In this work we will focus on both, ID and OOD generalization.

We now describe general challenges that can lead to poor generalization performance.

Distribution shifts When learning policies for robotic manipulation, one of the main challenges arises from distribution shifts, as described in [18]. This concept is related to the problem of generalization described earlier. A distribution shift occurs whenever the distribution of the input data at test time differs from the distribution observed during training. As a result, the policy may perform well on the training distribution but degrade in accuracy or robustness once evaluated under new conditions. In the context of robotics, such shifts are almost unavoidable, since the environment is dynamic and contains uncertainties that can not be fully captured in the training data.

There are several types of distribution shifts that we aim to address in this work [18], including:

- Covariate shift,
- Selection bias,
- Domain shift,
- Imbalanced data,
- Prior Probability Shift.

Covariate shift This refers to a mismatch between the distribution of states encountered during training and those encountered during execution. In IL, covariate shift often occurs because the training data is collected from expert demonstrations, while the learned policy may visit states that the expert never encountered. We will investigate examples of covariate shift in

this thesis. More randomization of the training set is one solution to reduce the potential for covariate shifts.

Selection bias Another related phenomenon is selection bias. Selection bias occurs when the training data does not adequately represent the true underlying distribution of situations the robot will face. For example, if demonstrations or simulated trajectories are biased toward certain object types, poses, or environmental conditions, the resulting policy may fail when encountering less represented or unseen conditions. This is particularly problematic in robotic manipulation, where variability in object geometry is very high. Addressing selection bias requires careful data collection and often strategies such as domain randomization to expose the policy to a broader range of conditions. We will test our approach on this type of bias in the evaluation.

Domain shift A further important case is domain shift, which occurs when the entire domain of training differs from the deployment domain. The most common example in robotics is the sim-to-real gap. Policies trained purely in simulation may fail on real robots because of mismatches in dynamics, sensor noise, or visual appearance. Even small differences in lighting, textures, or friction parameters can lead to significant performance drops. This type of shift will not play a major role in this thesis, since the variations lie in the objects while the environmental conditions remain fixed in their distribution.

Imbalanced data Imbalanced data refers to situations where some classes or outcomes are heavily underrepresented compared to others. For example, in binary classification, one class may account for 95% of the training data while the other makes up only 5%. This imbalanced data can often lead to poor performance on the minority class, which may be the more important one. For example, consider an autonomous driving task where the dataset contains many instances of ”no parking” signs but only a few of traffic lights. In this case, the model may become good at recognizing ”no parking” signs but unreliable at detecting traffic lights, even though traffic lights are far more critical for safe driving.

There are many approaches to deal with imbalanced data, as described by Provost [19]. A common strategy is to rebalance the dataset, for example by oversampling the minority class or undersampling the majority class.

In this work, we ensure that each object in the dataset, is represented by the same number of episodes. Although the number of state–action pairs per episode varies, potential imbalance effects are expected to be small. More details about the objects and training are provided in Chapter 5.

Prior probability shift Another type of distribution shift is prior probability shift, also referred to as label shift. It occurs when the marginal distribution of the output labels $p(y)$

changes between training and test time, while the conditional distribution $p(x|y)$ remains fixed. For example, in classification tasks the frequency of certain classes in the test set may differ from the training set.

In the context of our work, prior probability shift would correspond to changes in the frequency of object categories or task outcomes between training and deployment. Since we balance the number of training episodes across objects, we explicitly minimize such label imbalance effects in the dataset. However, in evaluation the effective priors may still differ, e.g., if some objects are more difficult to manipulate and therefore more likely to fail.

Overall, distribution shifts are a key aspect for reliable generalization in robotic manipulation.

2.3 Imitation Learning

In Imitation Learning (IL) the agent learns the behaviour of an expert. We assume access to a dataset $\mathcal{D} = \{(s_i, a_i)\}_{i=1}^{N}$ of state–action pairs recorded from an expert policy π_{E}. The task is to learn a policy $\pi_\theta : \mathcal{S} \to \mathcal{A}$ that reproduces the expert's decisions as closely as possible. In practice, demonstrations can be obtained in different ways, for example through human teleoperation, or kinesthetic teaching where the robot arm is physically guided by a human [20].

Behavior Cloning (BC) is a commonly used supervised learning method. The states play the role of inputs and the expert actions serve as fixed targets.

In BC we minimize the empirical risk [21]

$$\mathcal{L}(\theta) \;=\; \frac{1}{N}\sum_{i=1}^{N} \ell(\pi_\theta(s_i),\, a_i),$$

where ℓ is a loss function that measures the error between the prediction of the policies and the action of the experts. Depending on the action space, different loss functions are appropriate for IL. For discrete action spaces (e.g., $\mathcal{A} = \{\text{left}, \text{right}\}$), the cross-entropy loss

$$\ell_{\mathrm{CE}}(\pi_\theta(s), a) = -\log \pi_\theta(a \mid s)$$

measures how well the predicted probability distribution over actions matches the expert's chosen action. Minimizing it increases the probability assigned to the correct expert action.

In continuous action spaces (e.g., torque vectors for a robot arm), the mean-squared-error loss

$$\ell_{\mathrm{MSE}}(\pi_\theta(s), a) = \|\pi_\theta(s) - a\|_2^2$$

penalizes the squared difference between the predicted and target action[22].

We use gradient-based optimization [23], such as Adam (Adaptive Moment Estimation), of BC updates θ so that the probability of the expert's action increases.

Unlike RL [17], IL does not rely on an explicitly defined reward function but instead directly leverages expert demonstrations. This makes it attractive in domains where designing reward signals is difficult or unsafe.

One limitation of BC is its high data demand. Since BC treats imitation as a supervised learning problem, it requires a large number of diverse demonstrations to cover the relevant state space. If demonstrations are too few or lack variability, the policy overfits to the training distribution and fails to generalize to unseen states. Specifically, because BC ignores the long-term consequences of actions, it is vulnerable to covariate shifts [24]. Small prediction errors can move the agent into states that lie outside the training distribution, where errors compound. In practice, mitigating techniques such as data augmentation, noise injection or interactive approaches like Dataset Aggregation (DAgger) are often employed [24].

Furthermore, training examples obtained from demonstrations are naturally limited by the capabilities of the teacher. In some cases, human teachers may be slower or less precise than the robot itself. Thus, the policy may be constrained by the teacher's abilities, such that

$$\mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)] \;\leq\; \mathbb{E}_{\tau\sim\pi_E}[R(\tau)],$$

where $R(\tau)$ denotes the return of a trajectory τ (e.g. success rate).

Beyond BC, more advanced approaches exist, such as Inverse Reinforcement Learning (IRL) [14] or Generative Adversarial Imitation Learning (GAIL) [25], which aim to overcome the limitations of supervised imitation by either inferring underlying reward functions or training in an adversarial framework.

2.4 Reinforcement Learning

Reinforcement Learning (RL) [17] is a framework in which an agent learns to make decisions by interacting with an environment. Unlike supervised learning, where learning signals are explicit target labels, RL agents receive only scalar feedback in the form of rewards. Through trial and error, the agent adapts its policy, in order to maximise long-term cumulative reward.

Most RL problems are modelled as a Markov Decision Process (MDP) $\langle \mathcal{S}, \mathcal{A}, P, r, \gamma \rangle$. $\mathcal{S}$ represents the state space, $\mathcal{A}$ the action space, $\mathcal{P}$ the transition probability function, r the reward function and γ the discount factor. As in Figure 2.2, at discrete time step t the agent is in state $s_t \in \mathcal{S}$, selects an action $a_t \in \mathcal{A}$ according to its current policy $\pi(\cdot \mid s_t)$, and receives a reward $r_{t+1} = r(s_t, a_t, s_{t+1})$. The environment then evolves to a new state s_{t+1} according to the

transition probability function $P(s_{t+1} \mid s_t, s_t)$. The Markov property asserts that the distribution of the next state depends on the current state–action pair, not on the entire past.

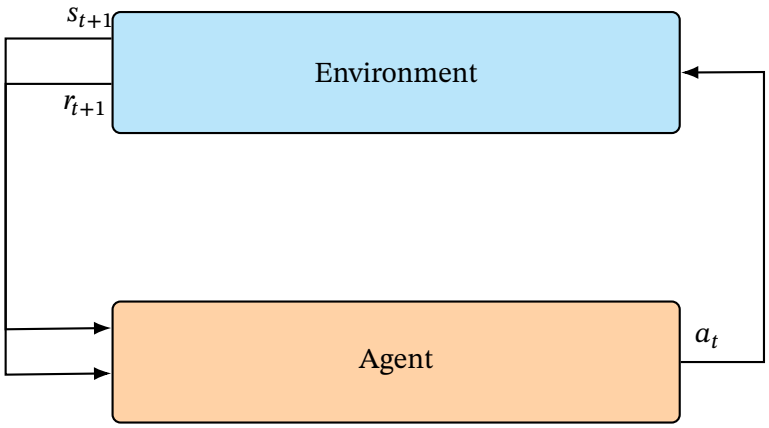


Figure 2.2: Interaction between agent and environment: the agent observes s_t, takes action a_t, and receives reward r_{t+1} and next state s_{t+1}.

The goal in RL is to learn a policy that maximizes the expected reward,

$$J(\pi) = \mathbb{E}_{\tau\sim\pi}[G_0].$$

The cumulative reward, collected along a trajectory $\tau = (s_0, a_0, s_1, \dots)$ is

$$G_t = \sum_{k=0}^{\infty} \gamma^k \, r_{t+k+1},$$

where the discount factor $\gamma \in [0,1]$ controls the importance of near term versus distant rewards.

Two auxiliary quantities underpin many RL algorithms, the value function and state value function:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\big[G_t \mid S_t = s\big],$$
$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\big[G_t \mid S_t = s, A_t = a\big].$$

The Bellman expectation equations express the value of a state (or state–action pair) under a policy in terms of the immediate reward and the value of the successor state:

$$V^{\pi}(s) = \mathbb{E}_{\pi}[R_{t+1} + \gamma \, V^{\pi}(S_{t+1}) \mid S_t = s],$$
$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[R_{t+1} + \gamma \, Q^{\pi}(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a].$$

These recursive relations allow dynamic programming and bootstrapped temporal-difference updates.

The optimal value function

$$V^*(s) = \max_{\pi} \mathbb{E}_{\pi}\big[G_t \mid S_t = s\big]$$

serves as a benchmark, indicating the best achievable long-term reward from any state.

Because the transition dynamics and reward structure are unknown, the agent must balance

- *Exploitation*: choosing actions that yield high rewards under current knowledge.
- *Exploration*: selecting uncertain or novel actions to gather information.

A simple strategy is ε-greedy sampling, which initially explores widely ($\varepsilon \approx 1$) and gradually shifts towards greedy exploitation as learning progresses.

In many real-world settings, the agent perceives only partial information, yielding a Partially Observable MDP (POMDP). Limited observability complicates reward attribution and often motivates the use of memory. When the agent has full information about the environment, the observation o_t equals the true state s_t (a fully observed MDP). The actionspace can be, like in BC, discrete or continuous.

Classical RL employed tabular value estimates or function approximators, however Deep RL replaces these with deep neural networks. This enables generalization in high-dimensional input spaces (images, point clouds).

We can distinguish between policy-based and value-based methods.

- *Policy-based* approaches learn $\pi_\theta(a \mid s)$ directly, often via gradient ascent on $J(\pi_\theta)$. Policies may be deterministic, $a = \mu_\theta(s)$, or stochastic, sampling from $\pi_\theta(\cdot \mid s)$, encouraging exploration.
- *Value-based* approaches estimate $Q^\pi(s, a)$ or $V^\pi(s)$ and derive a greedy or ε-greedy policy from these estimates.
- *Actor–critic* hybrids maintain both: the actor proposes actions while the critic supplies low-variance value targets.

On-policy algorithms (e.g. SARSA [26], PPO [27]) update the policy using data generated by that same policy, ensuring a matched state–action distribution. Off-policy algorithms (e.g. Q-learning, Deep Deterministic Policy Gradient [28]) reuse experience gathered under different behaviour policies, improving sample efficiency through replay buffers or importance weighting.

2.5 Proximal Policy Optimization

Proximal Policy Optimization (PPO) [27] is a RL algorithm that operates in an on-policy manner. This helps avoid the distribution shift issues that commonly plague off-policy methods. PPO is also a first order, gradient-based technique that pairs implementation simplicity with reliably stable updates. Here, first order refers to optimization methods that use only the gradient (first derivative) of the objective function with respect to the model parameters, as opposed to second-order methods that also use curvature information (e.g., the Hessian).

Advantage function In PPO the advantage function is used. The advantage function $A^\pi(s, a)$ measures how much better (or worse) taking action a in state s is compared to the average action under policy π. It is defined as the difference between the action-value function and the value function:

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s).$$

A positive advantage means that action a is better than average at state s, while a negative value suggests the opposite.

Clipped surrogate objective Rather than solving a constrained optimization problem exactly (as Trust-Region Policy Optimisation (TRPO)) [27], PPO maximizes a clipped surrogate objective (or, in an alternative variant, a Kullback-Leibler (KL) penalized objective) over several epochs of stochastic gradient ascent. The clipping or KL penalty keeps each policy step from staying too far from the previous one, delivering TRPO like robustness without the computational overhead of second order line searches. The procedure is as follows:

For each sampled state–action pair (s_t, a_t), form the probability ratio

$$r_t(\theta) \;=\; \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}$$

and multiply it by the estimated advantage $\hat{A}_t$. To keep updates from drifting too far, clip the ratio to $[\,1-\beta,\ 1+\beta\,]$ $(\beta \in (0,1))$:

$$J^{\text{clip}}(\theta) \;=\; \mathbb{E}\big[\min\!\big(r_t(\theta)\hat{A}_t,\ \text{clip}\big(r_t(\theta), 1-\beta, 1+\beta\big)\hat{A}_t\big)\big].$$

Maximizing this objective with a standard optimizer (e.g. Adam) yields the update rule

$$\theta_{k+1} \;=\; \theta_k \;+\; \alpha\, \nabla_\theta J^{\text{clip}}(\theta_k),$$

where α is the learning rate.

KL-penalty variant An alternative version, replaces the hard clip with a soft KL-divergence penalty:

$$J^{\text{KL-pen}}(\theta) \;=\; \mathbb{E}\big[r_t(\theta)\hat{A}_t - \beta\, D_{\text{KL}}(\pi_\theta \parallel \pi_{\theta_{\text{old}}})\big].$$

After each update, the coefficient β is adjusted to keep the mean KL close to a target ε (e.g. 0.1):

- If $d < \frac{\varepsilon}{1.5}$, set $\beta \leftarrow \frac{\beta}{2}$;
- If $d \geq 1.5\,\varepsilon$, set $\beta \leftarrow 2\beta$,

where $d = \mathbb{E}\big[\, D_{\text{KL}}\big(\pi_\theta \parallel \pi_{\theta_{\text{old}}}\big)\big]$.

2.6 Patch Transformer

To handle point clouds, we must account for their unordered nature. The points can be presented in any permutation, yet the underlying geometry they represent is unchanged. Standard convolutional neural networks rely on a fixed pixel grid, which point clouds lack. Because the points can appear in any order, a suitable model must be permutation invariant.

Patch Transformers [7] address this challenge by grouping nearby points into patches, tokenizing them, and applying a transformer architecture to these patches for prediction.

Patch Tokenization The first step of a Patch Transformer is to identify and tokenize patches. To determine patches in a point cloud, we first select N representative points using farthest point sampling (FPS). FPS is an algorithm that selects N points which are maximally spread out across the point set. The procedure is as follows:

1. **Initialization:** Select an initial point $\mathbf{s}_1$ randomly from the point set (e.g., point cloud) $\mathcal{P} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ and set $\mathcal{S} = \{\mathbf{s}_1\}$.
2. **Iterative selection:** For each $i = 2, \dots, N$, select the point

   $$\mathbf{s}_i = \arg\max_{\mathbf{x}\in\mathcal{P}} \min_{\mathbf{s}\in\mathcal{S}} \|\mathbf{x} - \mathbf{s}\|_2$$

   and update the sampled set: $\mathcal{S} \leftarrow \mathcal{S} \cup \{\mathbf{s}_i\}$. That is, select the point with the maximum distance to its closest point in the current set.
3. **Repeat** until $|\mathcal{S}| = N$ points have been selected.

After applying FPS to obtain N representative points, we use k-nearest neighbors (kNN). It is a method widely applied in both classification and regression tasks [29]. Specifically, each point in the point cloud is assigned to its closest representative among the N selected points, thereby forming the patches.

To embed the local shape of the patches, each patch is normalized by subtracting its center from its representative points. Afterwards, the patches are tokenized using a multilayer perceptron (MLP). These tokens are then processed by a transformer.

2.7 CORN (Contact-based Object Representation for Non-prehensile Manipulation)

Cho *et al.* [7] introduced CORN, a RL framework that learns where and how a robot hand interacts with a target object in order to perform actions such as pushing, rolling, or toppling. The core idea is to focus on those regions of the object that can actually come into contact with the robot's gripper rather than modeling the entire object geometry.

CORN consists of two parts, the contact network and the "teacher" policy. In this thesis, we work with the contact network and modify the "teacher" policy. Figure 2.3 shows the CORN architecture.

Contact Network In the first step, the contact network is trained. The contact network in CORN consists of a point cloud encoder that produces patch-wise embeddings and a contact-prediction decoder that classifies whether each patch intersects with the gripper. In the encoder (red), the point cloud is divided into local patches and processed by a patch-based transformer to embed them into a set of vectors, as described in Section 2.6. The decoder is trained to predict whether a patch is colliding with the robot's hand. Afterwards, the decoder is discarded and the weights of the encoder is frozen. To train the contact network, gripper and object poses are randomly sampled in SE(3) space, and the gripper is moved toward the object's surface with added Gaussian noise to create near-collision or collision configurations. Points sampled from the object's surface are then labeled based on whether they intersect with the gripper, producing data for training the contact-prediction network.

Teacher Policy Afterwards, the frozen encoder outputs patch embeddings to a RL module (the "teacher" policy). These patch embeddings serve as the keys and values in the cross-attention mechanism, representing local object regions. In parallel, robot state information (hand state, joint states, previous action, and relative goal) is processed by a Tokenize-Query MLP to produce the queries. The cross-attention learns key object patches that the policy deems relevant for impending or ongoing contact, allowing CORN to focus on precisely those regions needed for each manipulation task. The final policy module consists of an actor and a critic, both implemented as MLPs. They receive the cross-attended embeddings and additional robot state information. The policy is trained using PPO as described in section 2.5.

The reward function is defined as a weighted sum of task success, goal reaching, and contact inducing terms, minus an energy penalty,

$$r = r_{\text{suc}} + r_{\text{reach}} + r_{\text{contact}} - c_{\text{energy}},$$

where r_{suc} indicates whether the object is within a tolerance of the goal, r_{reach} and r_{contact} are shaped rewards based on distance metrics, and c_{energy} penalizes high joint torques.

Student Policy Finally, a "student" policy is distilled from the teacher but only sees partial real-world observations. The distillation process works by having the student policy imitate the teacher policy's action distribution. At each step, the teacher generates actions for the current full observation s, and the student generates actions for the current partial observation o. The student policy is trained to match these actions using a loss function such as the KL-divergence

$$\mathcal{L}_{\text{KL}} = D_{\text{KL}}\Big(\pi_\theta^{\text{teacher}}(\cdot \mid s) \parallel \pi_\phi^{\text{student}}(\cdot \mid o)\Big),$$

which measures how well the student's action distribution π_ϕ matches that of the teacher π_θ given state s and o. This encourages the student to reproduce the teacher's decisions while operating on its own partial observations.

CORN not only scales well to high-dimensional point clouds in massively parallel simulation, but also efficiently identifies the critical contact locations needed to manipulate unseen real-world objects.

Limitations While CORN demonstrates strong performance on non prehensile manipulation, it also comes with several limitations. The approach requires extensive pretraining with a large and diverse set of objects in simulation. This makes the method computationally expensive, both in terms of the number of training iterations and the amount of simulated interaction data needed to achieve stable performance. Furthemore, the approach requires a lot of objects and iterations for the RL training. Also, while the contact based representation is effective, it still requires high quality point cloud observations. Noisy or incomplete sensory data as commonly encountered in the real world can worse the policy.

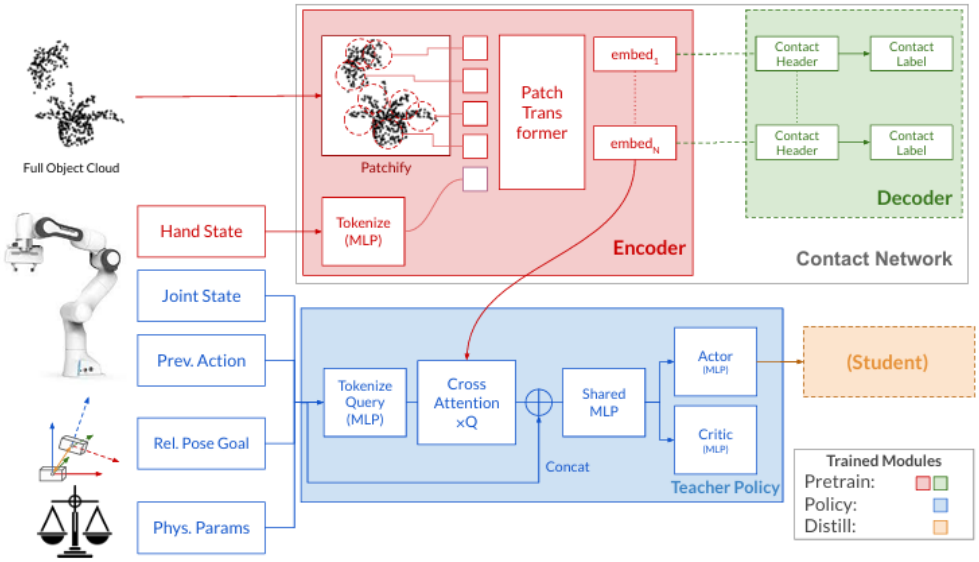


Figure 2.3: Model Architecture of CORN [7]: The contact network features a point cloud encoder (red) and a contact-prediction decoder (green), which transmit point cloud embeddings to the teacher policy module (blue). The teacher policy uses cross-attention to combine embeddings with robot state information, and outputs actions through an actor MLP and state-value estimates through a critic MLP.

3 Related Work

In the following, we will describe work related to our focus on generalization in robotic manipulation. We begin with an overview of Imitation Learning (IL) approaches, showing how they have been applied to manipulation tasks and in particular how they address generalization. Afterwards, we turn to Reinforcement Learning (RL) approaches and discuss their strategies and limitations with respect to generalization. After RL, we review several Inverse Reinforcement Learning (IRL) methods that have been investigated in robotics. Most of these approaches focus on grasping, as comparatively little research exists on pushing. Afterwards, we will also discuss planning approaches. Finally, we outline methods for processing and extracting point clouds, which are the central observation source for our method.

As described in Chapter 2, we distinguish in-distribution (ID) generalization , which means robustness to moderate variations within the training distribution, from out-of-distribution (OOD) generalization, with larger variations.

3.1 Imitation Learning for Manipulation Tasks

Generalization of motion policies using IL is often achieved on large data sets with a variety of different objects and tasks. Recent research on data scaling laws in robotic manipulation [3, 4] shows how system performance can improve as the size and diversity of training data increases, thus guiding strategies for collecting large datasets from different environments. However, such large scale data collection in robotics is expensive and time consuming.

Nevertheless, there are other approaches than using large datasets, such as learning affordance regions [30, 31], invariance regions [32], predicting keypoints [33] of the objects, using stability loss functions [34] or constraints [35].

Rana *et al.* [30] propose an affordance-centric policy-learning method for a pick-and-place task using BC. Their approach aligns and positions a task frame based on key affordance regions (most important region of an object for performing a specific task). The observation space includes the complete pose of the tool, the orientation of the affordance frame, and the state of the gripper. This method enables ID generalization, allowing a policy trained on a single instance of an object category to successfully transfer across intra-category variations in shape, size, and color. Another approach regarding affordance learning is being used by Borja-Diaz *et al.* [31], in which key affordances from human-teleoperated play data is being extracted and model-based planning with model-free RL is integrated.

Furthermore Zhang and Boularias [32] propose IMOP (Invariance Matching One-shot Policy Learning) a one-shot IL algorithm for grasping tasks. IMOP identifies invariant regions within the state space for a specific task. Subsequently, it determines the pose of the end effector by matching these invariant regions between demonstrations and test scenarios. The policy uses point clouds, invariant regions, and the correspondence matrix (which maps matching points between the demonstration and the new scene) as input. This approach enables ID generalization, which also generalizes to large shape variations.

Manuelli *et al.* [33] utilizes semantic 3D keypoints as object representations to define and execute manipulation tasks, allowing flexible specification of manipulation targets through geometric costs and constraints on keypoints. This method enables the robot to handle shape variations and perform tasks such as perceiving, grasping, and placing objects.

Adapting the loss function can lead to better ID generalization. Mehta *et al.* [34] applies BC with a stability loss to mitigate covariate shifts by ensuring robot policies remain close to demonstrated behaviors, enhancing reliability in dynamic environments. Wang *et al.* [35] uses BC with RL as a reward constraint for stability.

More vision-centric approaches, such as VIEW (Visual Imitation learning with Waypoints) [36] or GROOT (Generalizable Robot Manipulation Policies for Visuomotor Control) [37] have been developed. VIEW is an approach to use video demonstrations from humans for learning to grasp objects. The robot processes a video of a human picking up a small box. From this video, it extracts waypoints that capture critical steps of the task, such points of the approach path, and the moment of grasping. These waypoints then act as a prior for the robot's own trajectory, guiding it to approach, grasp, and lift the box in a manner similar to the demonstrated actions. Through iterative exploration around these waypoints and repeated interaction with the environment, the robot refines its trajectory to accurately imitate the demonstrated behavior. GROOT utilizes point clouds and a transformer-based policy to generalize setups with different conditions, namely different visual distractions, changed camera angles, and new objects (color and size, ID generalization).

These methods go beyond the scope of only using pure BC. One major problem is the ability to adapt its behavior from executing wrong predictions and change its behavior to help for generalization. Table 3.1 gives the summary of the main approaches. OOD generalization is one of the major problems the approaches does not account.

Year	Task	Method	Observation	Score	Gen.
2024 [30]	Grasping & Placing	BC	Pose, Affordance, Gripper State	82 %	ID
2019 [33]	Grasping & Placing	Optimization	RGB-D	88-97%	ID
2024 [32]	RLBench Task	One Shot IL	Point clouds, Invariant regions	70%	ID

Table 3.1: IL Methods Summary

3.2 Reinforcement Learning for Manipulation Tasks

CORN [7] demonstrates state-of-the-art performance in non-prehensile manipulation, showing generalization both ID and OOD. Other RL approaches are applied in robotic manipulation tasks to achieve generalization, such as memory based approaches [38, 39] or Q-learning approaches [40–43]. One major problem in using RL is the need for a large amount of data. A summary of the approaches can be seen in Table 3.2.

Cong *et al.* [38] propose a vision-proprioception RL model for planar object pushing, which uses latent representations to encode the characteristics of an object. Building on this approach, Bergmann *et al.* [39] introduce the current state-of-the-art memory-based vision-proprioception RL for non-prehensible manipulation. The main contributions are an improved sampling of the sliding friction force during training and using of a GRU-layer to provide the agent with a memory. The RL agent performs fewer corrective movements around an object. Yet, in the experiments, they used only two simple object shapes, varying only slightly in length, radius, and weight.

Kalashnikov *et al.* [40] introduce a scalable self-supervised vision-based RL framework QT-OPT for real world grasping tasks. It learns from a large dataset of more than 580k attempts and 28k on-policy iterations. This approach achieves ID generalization, demonstrating a 96% success rate when grasping unseen objects. One disadvantage is that a lot of data is required.

Al-Shanoon *et al.* [41] propose a Deep Reinforcement Grasp Policy (DRGP), another real world grasping policy. The researchers simultaneously trained a deep neural network consisting of a visual network to extract features from visual observations and a policy network to map the features into the action space. It performs with a success rate of 70% on six unknown objects in a crowded situation, and 93% in single arrangement of unknown objects. Compared to QT-OPT, the objects are more different in test scenario, leading to worse performance.

Lang and Al-Shanoon [42] introduce Shift-to-grasp (StG), a self-learning strategy for grasping novel objects, where adjacent objects are cluttered together. In StG, the robot initially learns to shift objects to make room for grasping using Q-learning, before mastering the grasp itself. It is trained in a simulation environment and then tested on real-world objects.

Zeng *et al.* [43] introduce a pushing and grasping synergy model for clutter clean up, using Q-learning. It has grasping success rate of 68% to objects that fall within a similar shape distribution as that of the training objects. The approach struggles with completely new shapes.

Year	Task	Method	Observation	Reward	Score	Gen.
2024 [7]	Non-prehensile	PPO	Embedding, Gripper State	Dense	71%	ID, OOD
2024 [39]	Pushing	SAC + GRU	Image	Binary	90%	ID
2018 [40]	Grasping	Q-Learning	Image	Binary	96%	ID
2021 [41]	Grasping	Q-Learning	Heightmap	Binary	93%	ID
2022 [42]	Grasping	Q-Learning	Heightmap	Binary	80%	ID
2018 [43]	Pushing & Grasping	Q-Learning	Heightmap	Binary	68%	ID

Table 3.2: RL Methods Summary

3.3 Inverse Reinforcement Learning Approaches

IRL is applied in various robotic tasks, such as grasping [44, 45], or where it can be difficult to find a reward function [46], such as autonomous driving [47]. IRL has the capability to outperform the expert demonstrations in comparison to BC. Brown *et al.* [48] enables a reinforcement learning agent to exceed the performance of the suboptimal demonstrator by learning to optimize this extrapolated reward function.

IRL approaches include [49–52]. Ziebart et al. introduced the Maximum Entropy (MaxEnt) Framework [49], by learning a reward and a policy that is as uncertain as possible, while still explaining the demonstrations. Fu *et al.* [50] introduces Adversial IRL, which uses a learned discriminator to distinguish between expert and agent states. Ni *et al.* [51] introduces f-IRL another MaxEnt IRL based method. Szot *et al.* [52] presents the Behavior Cloning IRL (BC-IRL) framework and demonstrated that BC-IRL learns more generalizable rewards that provide meaningful rewards beyond the expert demonstrations. BC-IRL shows in the fetch reach task, success rates (number of completion of the task) twice as high as compared to other IRL approaches in challenging generalization settings. It also generalizes better than those trained with pure BC.

A problem that occurs with IRL is that it requires large amounts of iterations for training the RL agent. Also, in contrast to standard RL, IRL must first learn the reward from demonstrations before training a policy, requiring more iterations. However, IRL can learn more generalizable policies than RL, due to a better reward function that more accurately defines the task [52]. One thing to consider is that noisy demonstration can produce inaccurate rewards and thus suboptimal policies. Compared to IL, in IRL the agent learns to perform corrective movements.

3.4 Planning-based Approaches for Manipulation Tasks

Since we focus on pushing tasks, we compute the push analytically [53]. More details are given in Section 4.3. However, non-learning approaches for non-prehensile manipulation also include trajectory optimization and planning .

Optimization-based formulations typically model the task as a trajectory subject to discontinuous contact dynamics [1, 2]. The main challenge is that such dynamics make the optimization problems hard to solve and often yield motions that are not physically realistic.

Kaelbling and Lozano-Pérez [54] introduce an integrated task-and-motion planning framework, accounting for uncertainty in perception and execution. Their approach combines symbolic reasoning with geometric motion planning, enabling conditional plans that include both manipulation actions and information gathering steps. In general, such integrated planning is computationally expensive.

Another line of work casts manipulation as a discrete search problem, where robot and object states, as well as contact modes, are represented as nodes in a graph and feasible transitions as edges [55]. Such methods must reason over large hybrid state–action spaces, which makes real-time deployment difficult [56].

A main advantage of optimization and planning-based approaches is that they provide physically interpretable solutions and allow explicit reasoning about contacts. This can be beneficial for generating precise motions in structured settings or for tasks where accurate models are available. In contrast, these approaches also require exact knowledge of object and environment properties, such as mass and friction, which are not directly available in the real world. These limitations motivate a shift toward data-driven alternatives, where policies are learned from interaction data and can generalize to novel objects without requiring precise physical models.

3.5 Learning Representations from Point Clouds

Recent advances in self-supervised learning have led to powerful methods for extracting representations from 3-D point clouds. Several approaches illustrate these developments.

Earlier foundational work such as PointNet [57] introduced an architecture for directly processing raw point sets by applying shared multilayer perceptrons (MLPs) to each point and aggregating features through a symmetric function, ensuring permutation invariance. However, PointNet has limited ability to capture local geometric structures, since it mainly relies on global pooling across all points. Its extension PointNet++ [58] addressed this issue by recursively applying PointNet to local neighborhoods. By this it is able to capture both local and global context.

Other approaches such as PointContrast [59] and OcCo [60] leverage contrastive objectives or partial point reconstruction to learn features useful for downstream tasks. More recently, masked point modeling methods like Point-BERT [61] and PointMAE [62] have demonstrated state-of-the-art performance on shape classification and segmentation benchmarks.

In robotics, these representations have been adapted to encode object geometry and pose for control policies. The CORN framework [7] represents the current state-of-the-art. It uses a point cloud–based contact network to predict where and how a robot gripper can interact with an object, enabling robust manipulation across various tasks.

4 Approach

This thesis tackles the problem of generalization for robot pushing skills to improve handling of unseen objects, shapes, and ”weird” objects. Here, by ”weird” objects we mean objects, where the center of mass is changed in an unexpected way. Instead of directly predicting actions, we formulate the task as a keypoint prediction problem. Given a raw point-cloud observation of the object, the robot hand state, the center of mass of the object, and a desired goal position, our network predicts a single 3-D keypoint that constitutes an optimal pushing contact. We compress high dimensional input into a compact representation that simplifies the problem and hopefully enables generalization across different objects.

We give an overview of the individual components in the remainder of this chapter:

- *Problem:* Section 4.1 formalizes the problem we are dealing with in this work.
- *Observation and Action Space:* Section 4.2 formalizes the corresponding observation and action space that we deal with in our scenario.
- *Data Generation:* In Section 4.3 we describe the simulation pipeline that collects expert pushing trajectories, including the reset heuristic that mitigates OOD errors during pushes.
- *Network Architecture:* Section 4.4 entails details how we use and adapt CORN for our approach and train the network with an ℓ_2 loss on keypoints.
- *Limitations:* In Section 4.5, we provide information about the limitations of our approach.
- *Summary:* In Section 4.6 we provide an overall summary of our approach.

4.1 Problem Formulation

We distinguish between two types of problems. First, we define the pushing task itself. Second, we formulate the problem of generalization, which addresses how well a learned policy can transfer to new situations.

Pushing Task In the first step, we consider a pushing task. A specific object is placed on the table along with a defined goal position. The objective of the robot is to push the object to this target without letting it fall. We use the Franka Emika Panda robot, a torque-controlled lightweight manipulator introduced in 2017, designed for advanced human-robot interaction [8]. As illustrated in Figure 4.1, the setup includes the object (here a bottle), the Franka Emika Panda robot, and the goal position (marked by the red line inside the circle).

A push is considered successful if two conditions are fulfilled:

1. The object must reach the goal within a tolerance.
2. The object must remain stable on the table.

More precisely, at the end of an episode, the object's origin lies within 5 cm of the goal position and the object remains on the table without tipping over. Formally, let $o_T \in \mathbb{R}^2$ denote the final position of the object's origin in the tabletop plane and $g \in \mathbb{R}^2$ the goal position. Success is defined as

$$\|o_T - g\|_2 \ < \ \epsilon, \qquad \epsilon = 5\,\text{cm},$$

under the constraint that the object remains stable on the table.

For stability, we can describe it as following. Let z^0_{COM} and z^T_{COM} denote the initial and final heights of the object's center of mass, and z^0_i, z^T_i the heights of additional reference points i on the object surface. Stability requires

$$|z^T_{\text{COM}} - z^0_{\text{COM}}| < \delta,$$
$$|z^T_i - z^0_i| < \delta \quad \forall i,$$

with a small tolerance δ. If this condition is violated, the object is considered unstable (e.g., tilted or toppled).

Generalization Despite successful training, learned policies may fail to generalize. More details about generalization is described in Section 2.2. Typically, two types of failure can occur: either the policy underfits the training data and does not even solve the original task well, or it overfits i.e., it solves the training tasks but performs poorly on new or unseen situations [63].

To formalize this, let $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$ denote the training and test distributions over object geometries, $\mathcal{L}$ a loss function, τ the expert trajectory. We aim to learn a policy π_θ that minimizes the expected loss on $\mathcal{D}_{\text{train}}$ and generalizes to $\mathcal{D}_{\text{test}}$:

$$\theta^* = \arg\min_\theta \ \mathbb{E}_{\tau\sim\mathcal{D}_{\text{train}}}[\mathcal{L}(\pi_\theta, \tau)] \quad \text{such that} \quad \mathbb{E}_{\tau\sim\mathcal{D}_{\text{test}}}[\mathcal{L}(\pi_\theta, \tau)] \text{ remains low.}$$

Overfitting becomes apparent when the policy performs well on training tasks but poorly on test distributions:

$$\mathbb{E}_{\tau\sim\mathcal{D}_{\text{train}}}[\mathcal{L}(\pi_\theta, \tau)] \ll \mathbb{E}_{\tau\sim\mathcal{D}_{\text{test}}}[\mathcal{L}(\pi_\theta, \tau)].$$

The test distribution may match the training distribution, as is the case in in-distribution (ID) generalization. However, our goal is to achieve generalization on different test distributions. For example, by using entirely new objects or by shifting the object's center of mass. The center of mass is a critical factor in determining how to push optimally.

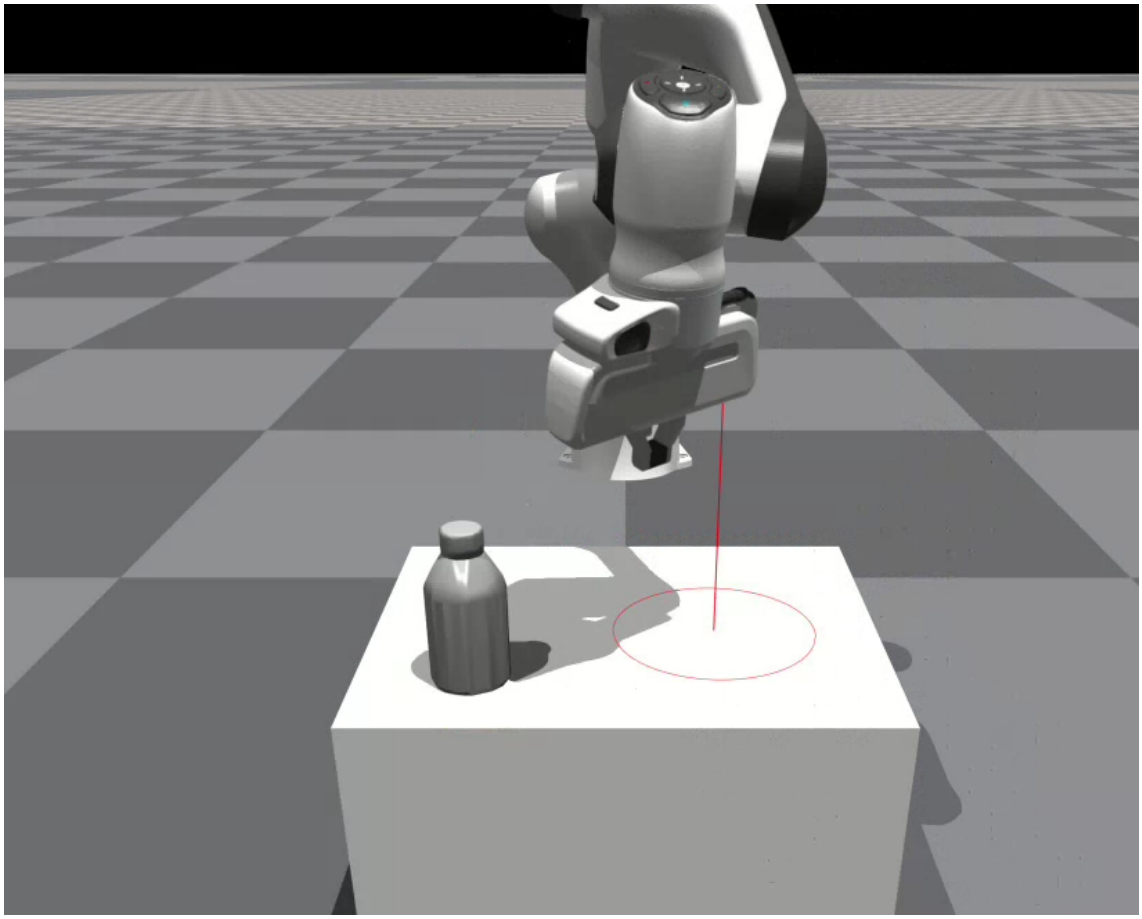


Figure 4.1: Experimental setup with the Franka Emika Panda robot (7-DoF) [8] and a bottle as an object. The red circle denotes the goal region (radius 5 cm), and the red vertical line indicates the target position. The goal is to push the object inside the circle. The object must remain upright (no tipping) for a successful push.

Keypoints We simplify the problem by learning keypoints. In our formulation, a keypoint $\mathbf{k} \in \mathbb{R}^3$ denotes the contact location on the object surface where the robot end effector establishes interaction. During training, we define the ground truth keypoint as the actual contact point between the robot and the object, i.e.

$$\mathbf{k}_t = \mathbf{p}_{\text{contact}}^{(t)}, \quad t = 1, \dots, N,$$

where $\mathbf{p}_{\text{contact}}^{(t)}$ is the 3-D contact position in episode t. This keypoint serves as the target output for the prediction network, as described in Section 4.4.

The pushing direction can be derived by combining $\mathbf{k}$ with the goal position $\mathbf{g}$, resulting in a push vector

$$\mathbf{u} = \frac{\mathbf{g} - \mathbf{k}}{\|\mathbf{g} - \mathbf{k}\|}.$$

Thus, the learning problem reduces to predicting a stable contact location rather than directly regressing the full action. Keypoints abstract away irrelevant variations in the input and highlight geometric features that are stable across varying distributions. This abstraction allows the policy to generalize more effectively from training to test settings.

Therefore, to enable generalization, we learn a keypoint function $k_\phi : \mathcal{O} \to \mathbb{R}^3$ that maps observations $\mathbf{o}_t$ to a task-relevant representation $\mathbf{k}_t = k_\phi(\mathbf{o}_t)$. The control policy then operates on this low-dimensional representation:

$$\pi_\theta(\mathbf{o}_t) := \pi_\theta\big(k_\phi(\mathbf{o}_t)\big).$$

However, the final policy is not learned directly. More details of the architecture is provided in Section 4.4.

Research question In this thesis, we investigate whether keypoints offer a representation to support generalization. This leads us to the following guiding research questions:

- Does 3-D keypoint prediction improve the generalization of pushing policies to unseen objects?
- How can we learn meaningful keypoints to support generalizable manipulation behavior?

In the following we will describe the observation and action space, we are dealing with in our problem.

4.2 Observation and Action Space

We use the observation, or a subset of it, as input to the keypoint predictor network. The action space, in contrast, is defined by the motion policy. The actual prediction of the network

corresponds to a 3-D keypoint, which is later mapped to an action within the policy framework. In this way, the observations provide the necessary state information, while the predicted keypoints serve as an intermediate representation that guides the robot's actions.

Observation Space We define the observation $\mathbf{o}_t$ at time step t as the combination of the object's point cloud, the 3-D center of mass $\mathbf{c}_m$, the 2-D goal position $\mathbf{g}$, and the 9-D hand state $\mathbf{e}_t$:

$$\mathbf{o}_t = (\mathbf{p}_t, \mathbf{c}_m, \mathbf{g}, \mathbf{e}_t) \in \mathcal{O} \subset \mathbb{R}^{512\times3} \times \mathbb{R}^3 \times \mathbb{R}^2 \times \mathbb{R}^9,$$

where $\mathbf{p}_t \in \mathbb{R}^{512\times3}$ is the object point cloud, $\mathbf{c}_m \in \mathbb{R}^3$ is the center of mass, $\mathbf{g} \in \mathbb{R}^2$ is the horizontal goal position, and $\mathbf{e}_t \in \mathbb{R}^9$ represents the hand state, consisting of 3-D position and 6-D orientation.

We chose the observation modalities based on their relevance for the pushing task. Point clouds provide a rich 3-D description of the object's geometry, including local surface details, location and overall shape. Compared to image-based observations, point clouds are viewpoint-invariant and avoid problems caused by lighting, texture, or background.

The center of mass is included as an additional input because it provides an optimal reference for predicting stable pushing directions, although not directly observable in real-world scenarios. In practice, the policy should learn to approximate this information from geometry alone.

Finally, the goal position defines the target for the manipulation, while the hand state encodes the robot's current configuration.

Action Space The action space of the robot arm is defined by a residual pose, consisting of a translational offset $\Delta\mathbf{t} \in \mathbb{R}^3$ and a rotational component $\Delta\mathbf{r} \in \mathbb{R}^3$ in axis–angle format, both expressed in the world frame, describing the end-effector pose. In total the action space at time step t is,

$$\mathbf{a}_t = (\Delta\mathbf{t}, \Delta\mathbf{r}) \in \mathcal{A} \subset \mathbb{R}^3 \times \mathbb{R}^3.$$

Similar to CORN, we use a variable-impedance controller to control the robot. In contrast to CORN, however, we keep the joint-space gains k_p and damping coefficients ρ fixed to default values provided by the simulator (they are not part of the action space).

Let ΔT_{ee} denote the residual transformation of the end-effector pose, i.e. a small translational and rotational offset applied relative to its current configuration. Given the residual transform

ΔT_{ee}, we compute the joint-position target

$$q_{\text{target}} = q_t + IK(\Delta T_{\text{ee}}),$$

where $IK(\cdot)$ denotes inverse kinematics with damped least squares [64].

The resulting torque command is computed as

$$\tau = k_p(q_{\text{target}} - q_t) - k_d\,\dot{q}_t, \quad \text{with } k_d = \rho \cdot \sqrt{k_p},$$

where q_t and $\dot{q}_t$ denote the current joint positions and velocities.

Details of the exact computation of the actions based on the keypoints are given in the next section on the data generation procedure.

4.3 Data Generation

To learn keypoints from demonstrations, we collect expert trajectories for planar pushing using an automated pipeline. These trajectories are simulated in Isaac Gym [65] using a 7-DoF Franka Emika Panda manipulator. Isaac Gym is a GPU-accelerated physics simulator originally developed for RL, enabling fast and large-scale data generation by keeping both simulation and computation on the GPU. The robot operates on a flat table workspace.

Expert pushing procedure Expert trajectories are generated by identifying optimal keypoints and executing a push motion along them. The keypoints are computed analytically.

Let $r_{\text{obj}} \in \mathbb{R}$ denote the object radius, $c_m \in \mathbb{R}^3$ the center of mass, and $\mathbf{g} \in \mathbb{R}^2$ the planar goal position. We define $c_{m,xy}$ as the projection of c_m onto the xy-plane. The push action is then obtained through a deterministic mapping

$$f : \mathbb{R} \times \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}^6, \qquad (r_{\text{obj}}, c_{m,xy}, g) \mapsto a_t. \tag{4.1}$$

Thus, the action at time t is given by

$$\mathbf{a}_t = f(r_{\text{obj}}, \mathbf{c}_{m,xy}, \mathbf{g}),$$

which represents the optimal push computed from the object radius, the center of mass, and the goal position.

To calculate the push, as a first step, we determine the pushing direction of the object.

The push direction is:

$$\hat{d} = \frac{g - c_{m,xy}}{\|g - c_{m,xy}\|}.$$

We use the push direction to compute the contact point where the robot should apply the push.

Using the object radius r_{obj}, we compute the contact point as follows:

$$p_{\text{target}} = c_m + r_{\text{obj}}\,\hat{d}.$$

Figure 4.2 illustrates the computed push direction, the resulting contact point, and the goal position.

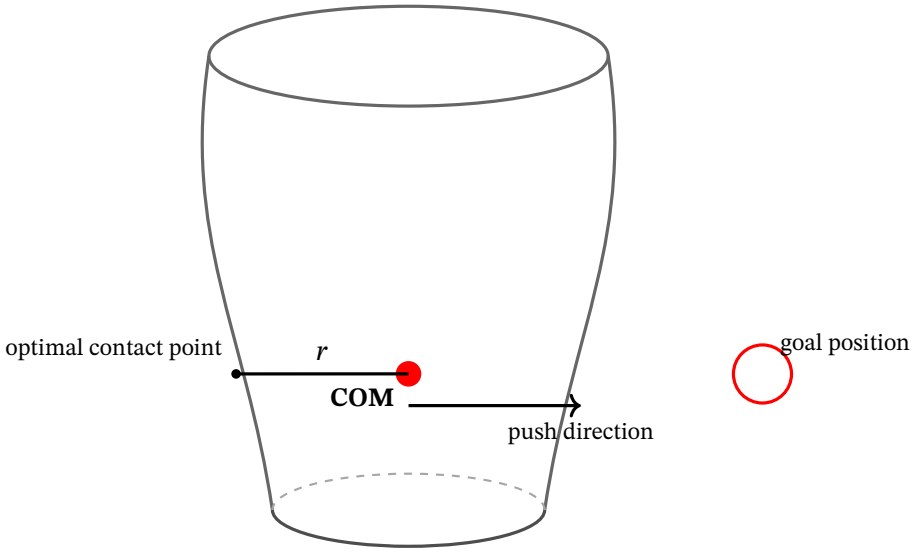


Figure 4.2: The center of mass (COM) is located at the lower middle of the object (red point). r denotes the object radius, and the push direction is calculated based on the COM, radius, and goal position. This yields the optimal contact point.

To push the object towards the goal, we orient the end-effector such that it faces the goal direction while being at a fixed angle of 60° relative to the table plane. This angle was chosen because in some positions 45° cannot be reached by the end-effector due to mechanical limits, while a near-horizontal orientation would reduce contact stability during pushing.

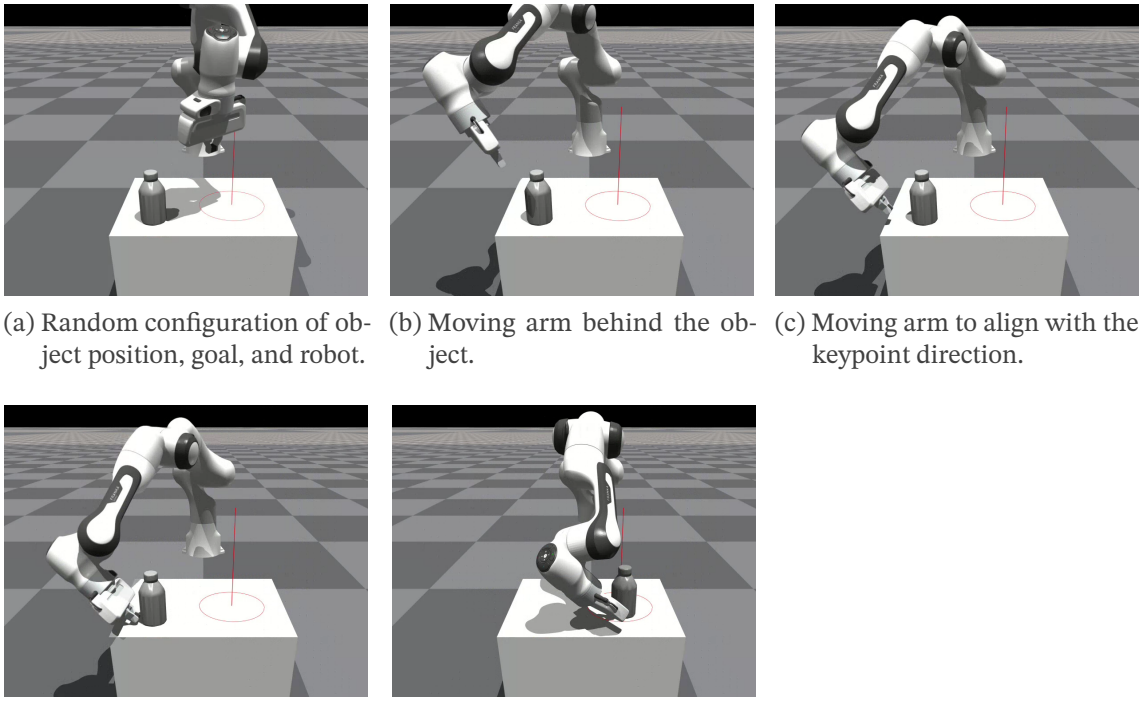
An example of the push procedure of the expert can be seen in Figure 4.3.

Our data collection consists of four steps. First, we perform sampling. For each episode and each fixed training object, we sample:

- the object position,
- the robot arm initial pose,
- and the goal position.

In the second step, the end effector is moving behind the object, on the line of the 2 dimensional hyperplane defined by $c_{m,xy}$ and the direction $\hat{d}$. In step 3 the end effector moves at a fixed height of 1cm above the table.

Afterwards in step 4 the end effector moves to p_{target}, establishes contact, and then applies a straight push towards the goal. Here, the contact point serves as the target keypoint in the training procedure. During the push, the end effector continuously corrects its direction to stay aligned with the target keypoint, compensating for object slippage.



(a) Random configuration of object position, goal, and robot.

(b) Moving arm behind the object.

(c) Moving arm to align with the keypoint direction.

(d) Establish contact and push towards red goal.

(e) Success push.

Figure 4.3: Expert sequence with the Franka Emika Panda robot [8], from (a) to (e), pushing the object toward the target (red circle).

Since the environment is subject to dynamics, the actual contact point deviates slightly from the optimal one, as illustrated in Figure 4.4. This deviation can be understood as noise introduced by physical effects such as friction. To mitigate this effect, the end-effector periodically reorients towards the optimal contact point after a fixed number of timesteps, as described earlier. This strategy helps to reduce the accumulated error and stabilize the pushing behavior. Nevertheless, the noise can not be completely avoided, as the robot is always influenced by stochastic variations in the environment and imperfect control execution.

Action Computation In the following, we will give details about the computation and implementation to get the desired actions for the data generation.

The translational offset is computed as the difference between the target location $\mathbf{p}_{\text{target}}$ (e.g., contact point) and the current end effector position $\mathbf{p}_{\text{ee}}$,

$$\Delta \mathbf{t} = \mathbf{p}_{\text{target}} - \mathbf{p}_{\text{ee}}.$$

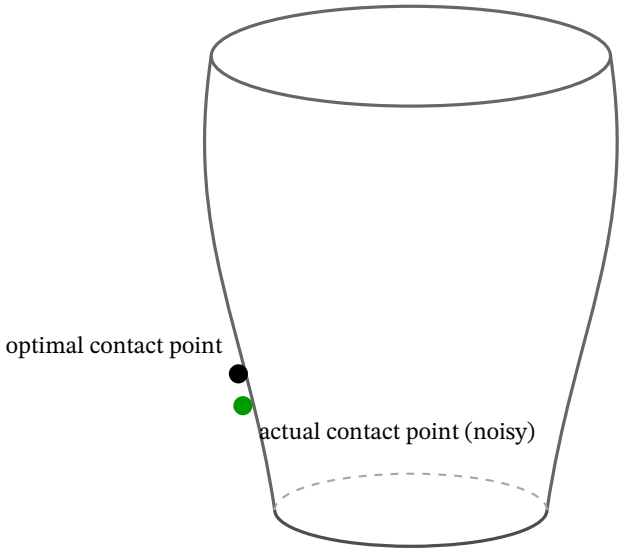


Figure 4.4: Glass with only the optimal contact point (black) and the actual contact point (green) reached by the end-effector. The deviation of the actual contact point is caused by noise from environmental dynamics, such as friction.

We now present the detailed computation of the goal orientation for the end-effector. The formulation is based on standard representations of rotations and quaternions, as described by Murray *et al.* [53].

The goal orientation of the end effector is computed by constructing a local coordinate frame aligned with the push direction. First, the z-axis is defined as the normalized vector from the current end effector position $\mathbf{p}_{\text{start}}$ to the goal position $\mathbf{p}_{\text{goal}}$,

$$\hat{\mathbf{z}} \;=\; \frac{\mathbf{p}_{\text{goal}} - \mathbf{p}_{\text{start}}}{\left\|\mathbf{p}_{\text{goal}} - \mathbf{p}_{\text{start}}\right\|}.$$

To ensure a horizontal orientation, we take the vector $\mathbf{u} = (0, 0, 1)$ and define

$$\hat{\mathbf{x}} \;=\; \frac{\mathbf{u} \times \hat{\mathbf{z}}}{\|\mathbf{u} \times \hat{\mathbf{z}}\|}, \qquad \hat{\mathbf{y}} \;=\; \hat{\mathbf{z}} \times \hat{\mathbf{x}},$$

where $\times$ denotes the cross product.

A fixed downward pitch of $\theta = -60°$ is applied by rotating $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ about the horizontal axis

$$\mathbf{a}_{\text{pitch}} \;=\; \frac{\hat{\mathbf{z}} \times \mathbf{u}}{\|\hat{\mathbf{z}} \times \mathbf{u}\|}, \qquad q_{\text{pitch}} = (\,\mathbf{a}_{\text{pitch}} \sin\tfrac{\theta}{2},\; \cos\tfrac{\theta}{2}\,),$$

so that

$$\hat{\mathbf{x}} \;\leftarrow\; q_{\text{pitch}}\, \hat{\mathbf{x}}\, q_{\text{pitch}}^{-1}, \qquad \hat{\mathbf{y}} \;\leftarrow\; q_{\text{pitch}}\, \hat{\mathbf{y}}\, q_{\text{pitch}}^{-1}.$$

The resulting rotation matrix is

$$R = [\,\hat{\mathbf{x}}\ \ \hat{\mathbf{y}}\ \ \hat{\mathbf{z}}\,],$$

which is converted into a quaternion q_{goal}. To maintain continuity, the sign of q_{goal} is flipped if $\langle q_{\text{goal}}, q_{\text{prev}} \rangle < 0$. Here, q_{prev} denotes the end-effector quaternion from the previous time step.

Finally, the residual orientation that brings the current end effector quaternion q_{ee} to the goal quaternion q_{goal} is computed as

$$q_{\Delta} \;=\; q_{\text{goal}}\, q_{\text{ee}}^{-1}.$$

Writing $q_{\Delta} = (\mathbf{v}, w)$ with $\mathbf{v} \in \mathbb{R}^3$, the axis-angle representation is obtained by

$$\phi = 2\arccos(w), \qquad \hat{\mathbf{a}} = \frac{\mathbf{v}}{\sin(\phi/2) + \varepsilon}, \qquad \Delta\mathbf{r} = \phi\,\hat{\mathbf{a}}.$$

The vector $\Delta\mathbf{r} \in \mathbb{R}^3$ serves as the residual rotation command in the controller.

Reset Heuristic The preexperiments revealed a consistent drop in accuracy as the object neared its target. Here, the error from the keypoint prediction to the true prediction increased from 2cm (end-effector is far away from goal during the push) to up to 15cm (end-effector is close to the goal during the push). Overall the network could not properly learn the distribution. So in general there are two problems to solve, referring to the OOD problem in BC [24].

1. mitigating large prediction errors near the goal,
2. addressing the out-of-distribution (OOD) problem in behavior cloning (BC) [24].

In the first problem (see 1), the policy kept the wrist in persistent contact with the object while pushing. Small pose errors accumulated, which caused the arm to move into configurations unseen during training. This effect is illustrated in Figure 4.5, which shows an example trajectory of the expert compared to the agent. While the expert follows a smooth and stable path toward the goal, the agent starts off well but gradually deviates. These deviations, though small at each step, accumulate over time and eventually cause the wrist to leave the distribution of expert trajectories. As a result, the agent encounters configurations for which it has seen no data during training, leading to failure cases such as unstable contacts or incorrect pushing directions.

In the second problem (see 2), the robot arm again deviates from the demonstrated path. This issue is illustrated in the same Figure 4.5. Here, a small initial error before the push begins causes the arm to drift slightly in the wrong direction. This, in turn, increases the deviation even further over time.

To illustrate drift in practice, Figure 4.6 shows two snapshots from the same rollout. In the top image, the robot wants to establish contact close to the intended point. In the bottom image, a

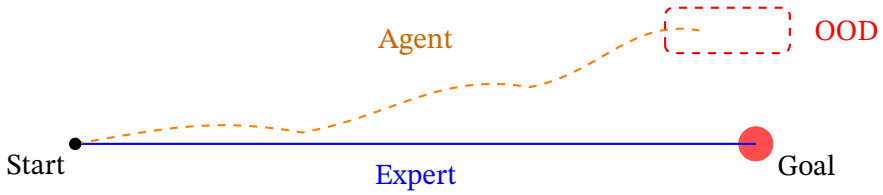


Figure 4.5: Example trajectories of the expert (blue) and the agent (orange). The agent deviates over time due to small pose errors, leading to an OOD state.

small bias in the predicted keypoint accumulated over time and by previously unseen states leads to a big drift to the right.

Therefore, we introduced a simple reset heuristic. After each push segment, every 20 controller iterations, the gripper breaks contact and retreats to a point positioned 5cm behind the current target contact point. As in Figure 4.3 illustrated, this means that one moves from (d) again to (c) at certain iterations. Since we would still stay on the hyperplane, we would add noise to get outside of the hyperplane, for OOD scenarios, as illustrated in Figure 4.7. This point is chosen by independently adding offsets drawn from the uniform distribution $\mathcal{U}(-5\text{cm}, 5\text{cm})$ along both the x- and y-axes. During training, by repeatedly resetting and perturbing the gripper's approach direction, the policy sees a broader range of configurations. This improves the model to better learn the structure of the distribution.

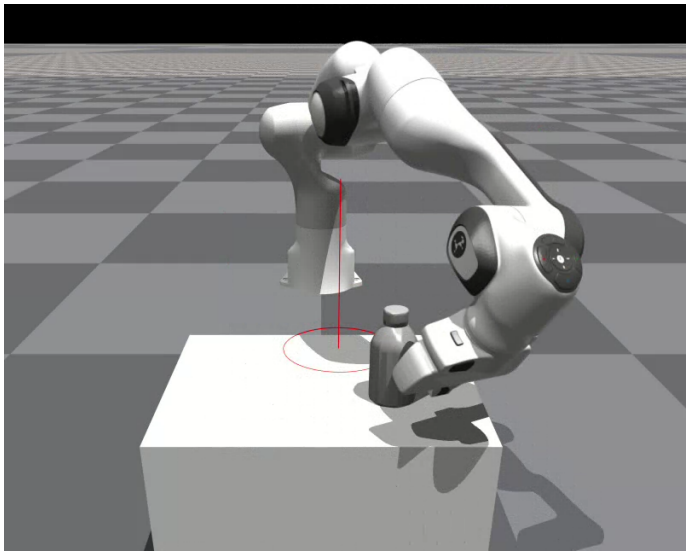
Distributional shift We now summarize how our data generation procedure addresses different types of distributional shifts in order to promote robust generalization. As outlined in Chapter 2, relevant shifts include:

- Covariate shift,
- Selection bias,
- Domain shift,
- Imbalanced data,
- Prior probability shift.

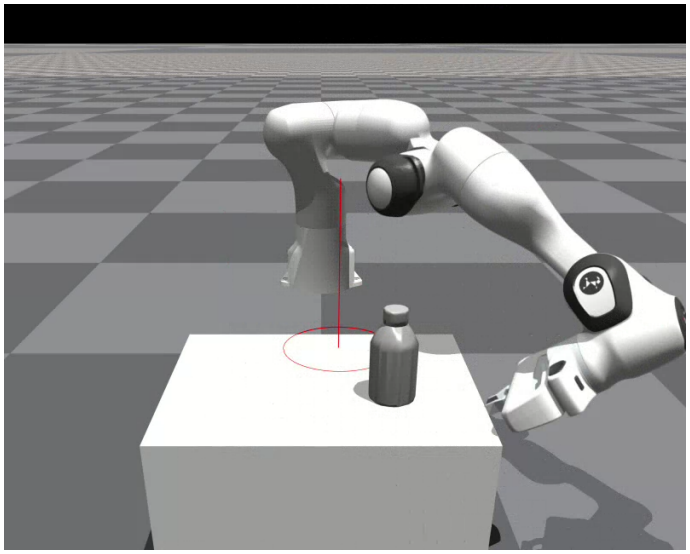
Covariate shift is mitigated through the reset heuristic, which ensures that the robot does not repeatedly explore the same local regions of the state space. This prevents trajectories from drifting into unrepresentative parts of the distribution and reduces error accumulation.

Selection bias is minimized by generating data across a diverse set of objects and initial configurations, rather than restricting to a subset of shapes or poses.

Domain shift is not explicitly addressed here, as we focus on simulation only. Bridging the gap to real-world deployment would require additional strategies such as domain randomization or adaptation.



(a) Almost initial contact close to the intended point (no visible drift).



(b) Later in the same rollout: pronounced rightward drift due to a small keypoint bias.

Figure 4.6: Drift emerging within a single rollout. The top frame shows a correct move to initial alignment. The bottom frame shows how a small prediction error in the keypoint accumulates over time and, especially in previously unseen states, induces a systematic drift to the right.

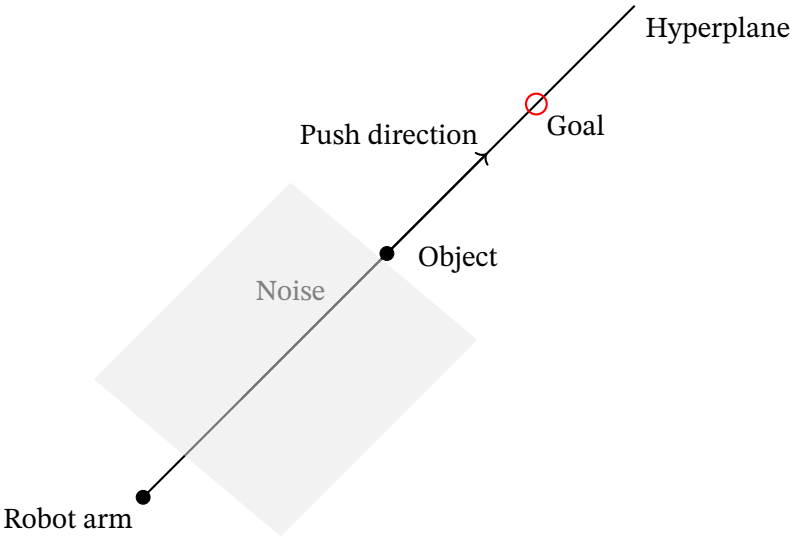


Figure 4.7: Illustration of the optimal push along the hyperplane from object to goal. Noise is applied to move outside of the hyperplane, simulating OOD scenarios.

With respect to imbalanced data, we will balance the number of episodes per object so that each category contributes equally, which reduces the risk of biased training.

Finally, prior probability shift is partially controlled by balancing objects during training, although effective priors may still differ at evaluation, as we want to test it.

4.4 Architecture

In our approach, we will use CORN embeddings as a basis. Figure 4.8 shows the overall architecture for keypoint prediction. We adapt CORN's architecture by retaining its encoder while modifying the output layers. The point cloud and hand state is first processed by the CORN encoder, producing a set of latent features that capture local and global geometric properties of the object.

In the cross-attention module, the embeddings serve as keys and values, while the goal position and optionally the center of mass are processed by an MLP to produce query tokens. This design allows the network to focus on those parts of the scene that are most relevant for reaching the goal.

The actor–critic structure is replaced with an MLP that predicts a single keypoint representing an optimal push region. Here, we use the output of the cross-attention and concatenate it with the remaining inputs (goal position and, optionally, the center of mass).

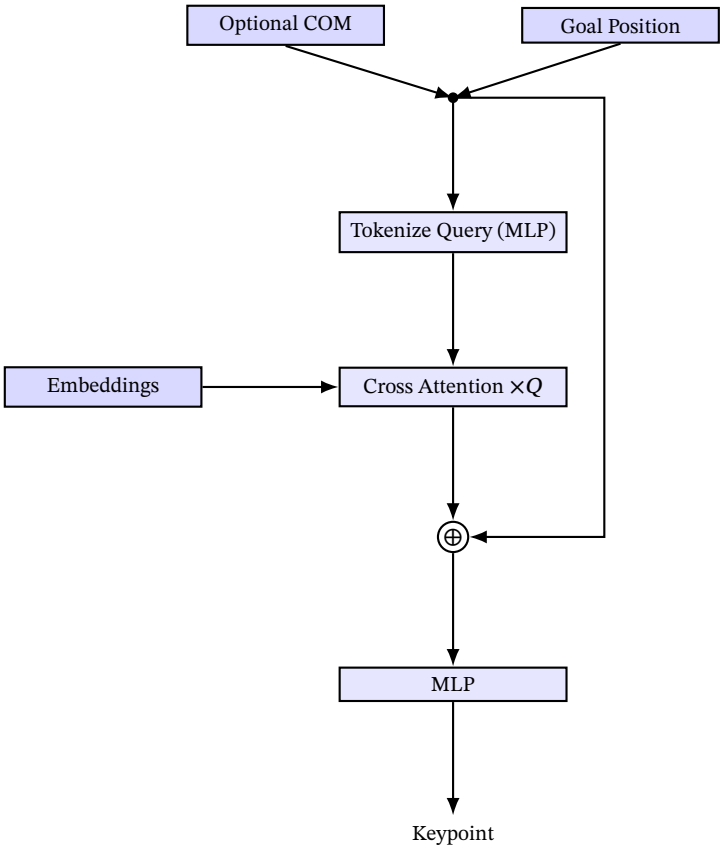


Figure 4.8: Model architecture: we build on CORN embeddings [7], which encode the structure of the input point cloud and hand state. In the cross-attention module, the embeddings serve as keys and values. The goal position and optionally the center of mass are processed by a Tokenize-Query MLP to form queries for the cross-attention module. The attended features are then combined with the original input via a skip connection and passed through a final MLP, which outputs a single 3-D keypoint representing the optimal pushing contact.

Hyperparameters The main architectural hyperparameters considered in our work are:

- **Patch size:** determines how many points are grouped into a local patch. Smaller patches capture local details, while larger patches provide more global context.
- **Encoder layers:** specify the depth of the encoder, i.e. how many layers are used to process the point cloud features. Increasing the number of layers typically enhances model capacity but also increases complexity.
- **Attention heads:** denote the number of parallel heads in the multi-head attention mechanism. Multiple heads allow the model to attend to different geometric aspects of the input simultaneously.

We will further investigate the influence of these hyperparameters in Section 5.2. Beyond these, many other hyperparameters can influence performance as well, such as learning rate schedules, optimizer choice, batch size, or weight decay.

Loss function We choose the ℓ_2 loss since it directly penalizes Euclidean deviations between predicted and ground-truth keypoints. This aligns with the geometric nature of the task. The overall optimization problem can be formulated as

$$\min_{\theta} \mathcal{L}(\theta) = \min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \left\| \hat{\mathbf{k}}_i(\theta) - \mathbf{k}_i \right\|_2^2$$

where θ denotes the model parameters, $\hat{\mathbf{k}}_i(\theta)$ the predicted keypoint for the sample i, and $\mathbf{k}_i$ the corresponding keypoint.

Push motion After predicting the keypoints, we apply a motion push as described in Section 4.3, but without the reset heuristic.

Given the predicted keypoint $\hat{\mathbf{k}}_t \in \mathbb{R}^3$ at time t and the goal position $g \in \mathbb{R}^2$, we compute the push action via a deterministic mapping

$$f : \mathbb{R}^3 \times \mathbb{R}^2 \to \mathbb{R}^6, \qquad \left(\hat{\mathbf{k}}_t, g\right) \mapsto a_t. \tag{4.2}$$

Thus, the action is $a_t = f\left(\hat{\mathbf{k}}_t, g\right)$. We now describe the closed-form definition of f that generates the push command.

The center of mass is replaced by the predicted keypoint to compute the push direction. Let $\hat{\mathbf{k}}_t$ denote the keypoint prediction of our architecture at time step t. We define $\hat{\mathbf{k}}_{t,xy}$ as the projection of the predicted keypoint (at timestep t) onto the xy-plane. As described before, we define $g \in \mathbb{R}^2$ as the goal position. The push direction is then given as:

$$\hat{d} = \frac{g - \hat{\mathbf{k}}_{t,xy}}{\|g - \hat{\mathbf{k}}_{t,xy}\|}.$$

We also use the predicted keypoint as the contact point, so that:

$$p_{\text{target}} = \hat{\mathbf{k}}_t.$$

Here, the object radius is unknown. The rest of the procedure is the same as in the data generation for a pushing motion (without the reset heuristic). Again, noise occurs in the actual contact points relative to the calculated keypoint due to environment dynamics.

Motivation The intuitive idea of our architecture is the following. Given the information of the goal position and center of mass, we have enough information to predict on which hyperplane the keypoint is lying. The point cloud embeddings produced by the CORN encoder provide information about how close the end-effector is to specific regions of the object. Therefore, we would move closely along the hyperplane in an optimal scenario, while having information about whether we are touching or how close we are. Problems would occur if we move outside the hyperplane, and make wrong predictions, so we are moving outside from this hyperplane and make a failed push. However, this scenario is tried to be solved with the reset heuristic.

4.5 Limitations

We will now discuss the main limitations of our approach. Our approach assumes access to point cloud observations of the scene and, for best performance, ground-truth information about the object's center of mass. While these quantities are readily available in simulation, in real-world settings they are difficult to obtain directly. In particular, the center of mass is not observable without additional modeling.

Furthermore, the exact contact point between the robot and the object is available in simulation and can be used as ground-truth supervision for training. In a real-world setting, however, obtaining such precise contact locations would require good object tracking, which is often infeasible. Also, we do not have any constraints, that the keypoint must lie on the object surface, potentially leading to wrong predictions.

Another limitation of our approach is that the encoder–decoder architecture requires a large amount of training data and diverse object instances to generalize effectively.

Furthermore, since we only predict keypoints, we do not take environmental noise into account in the push motion, as it is computed analytically.

Finally, during data generation we rely on a reset heuristic, where the end-effector is occasionally moved back during a push to increase data diversity. This behavior is specific to simulation and would not occur naturally in a real environment, and is therefore a limitation of the dataset rather than of the policy itself.

4.6 Summary

Our approach learns to predict keypoints that serve as contact targets for planar pushing. Given point cloud observations, the goal position, center of mass and the current hand state, our architecture maps the input to a single 3-D keypoint. We build on CORN's architecture, to

make use of its pretrained encoder and adapt it for use within an IL setup. To address the OOD issue during data generation, we incorporated a simple reset heuristic.

We now evaluate the learned keypoint predictor by integrating it into an push planner and testing its generalization to novel object geometries and shape.

5 Evaluation

To evaluate our approach in terms of reliability and generalization, we use a set of 32 distinct objects. Sixteen of these objects are used for training, while the remaining sixteen are held out entirely and used only for testing. We follow the same procedure as described in Section 4.3. However, since the robot can not always reach the object at certain positions, such as corners, we only consider a smaller part of the table, omitting places that are difficult to access.

The setup consists of a flat table, the Franka Emika Panda robot, the object to be pushed, and a goal position, as illustrated in Figure 4.3. For testing out architecture we will use 50 episodes and average the success score.

We compare our approach with a BC variant that uses the same architecture as ours, but where the output of the keypoint prediction is replaced by direct actions.

An overview of this chapter is given below:

- *Experimental Setup:* In Section 5.1, we describe the randomization parameters used in the simulation.
- *Ablation Study:* In Section 5.2, we mention and analyze the effect of different hyperparameters and select the best performing model for evaluation.
- *ID Generalization:* In Section 5.3, we evaluate in-distribution generalization by varying object sizes and testing on objects similar to the training set.
- *OOD Generalization:* In Section 5.4, we evaluate out-of-distribution generalization by introducing novel objects and by shifting the center of mass.
- *Qualitative Examples:* In Section 5.5, we illustrate selected successful and failure cases to provide deeper insights into the behavior of the learned policy.
- *Summary:* In Section 5.6, we provide an overall summary of the evaluation results, limitations and for future improvements.

5.1 Experimental Setup

The experiments are performed in simulation using the Franka Emika Panda robot. At the beginning of each episode, the robot arm is initialized in a randomized pose above the table to avoid deterministic starting configurations. Objects are placed uniformly at random on the table surface. To prevent trivial success, the goal position is sampled at least 0.1m away from the object's initial position.

Figure 5.1: Selected subset of 16 objects for training.

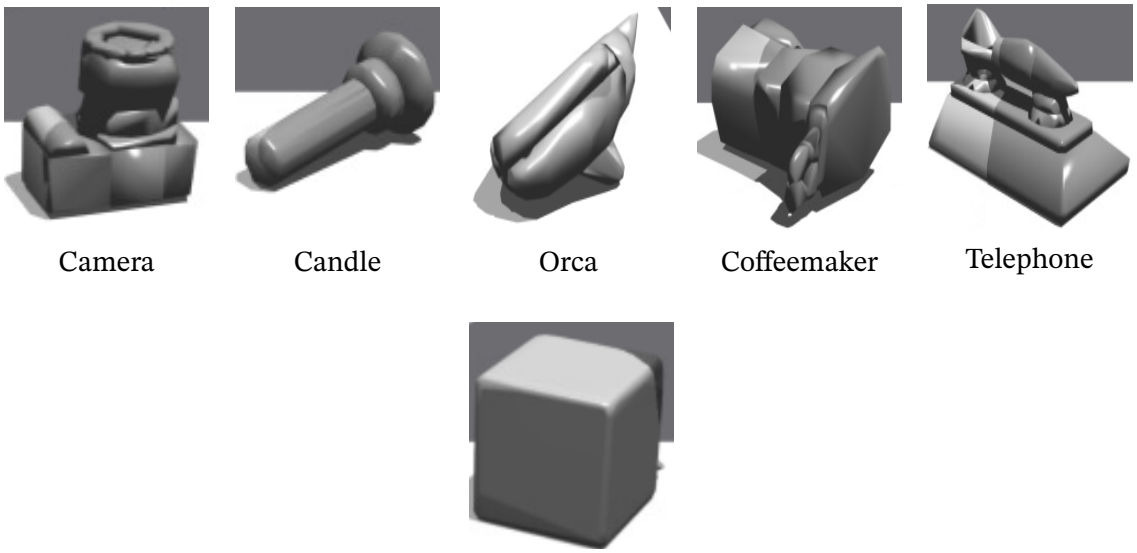


Figure 5.2: Selected subset of 6 objects for testing.

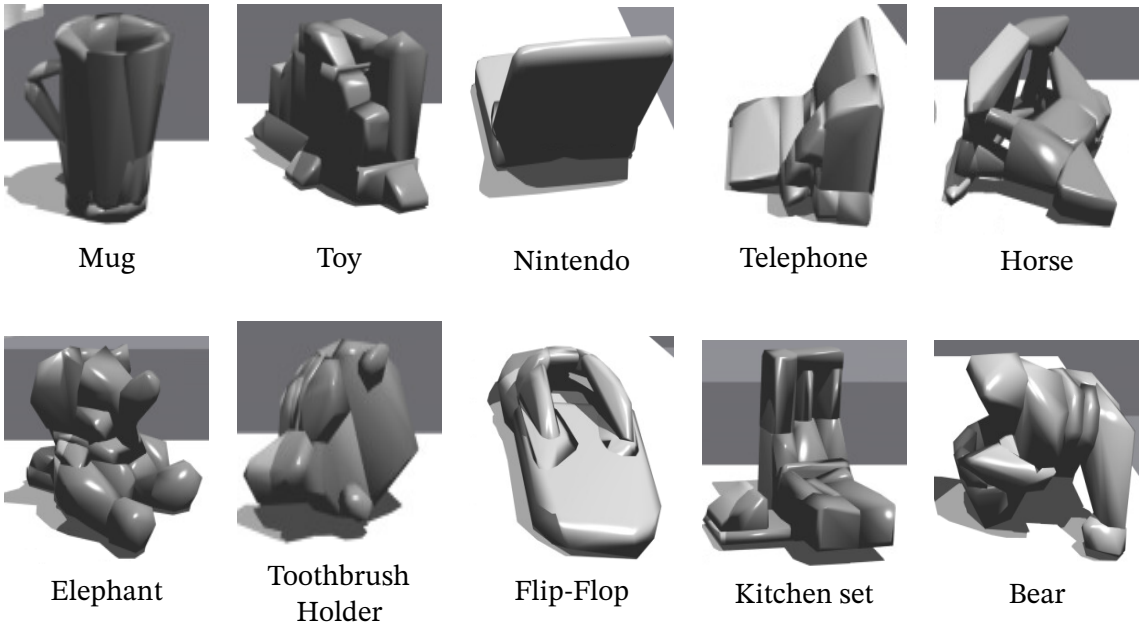


Figure 5.3: Selected subset of 10 objects for testing.

To better reflect real-world scenarios, we inject noise into the point cloud observations, sampled from a normal distribution $\mathcal{N}(0, 0.005)$. In addition, several environment parameters are randomized to increase robustness and generalization of the learned policy.

Table 5.1 summarizes the main values and randomized quantities including their ranges. Some parameters are fixed to ensure stable contact dynamics for pushing, such as the object mass (0.1kg) or scale (0.1m). We use the same randomization for both, data generation and testing.

In the experiments, each object is evaluated over 50 episodes in simulation.

Parameter	**Value / Range**
Object mass (kg)	0.1
Object scale (m)	0.1
Object friction	0.7
Table friction	0.4
Gripper friction	1.0
Torque noise	$\mathcal{N}(0,\ 0.03)$
Point cloud noise	$\mathcal{N}(0,\ 0.005)$

Table 5.1: Randomized / fixed parameters used in the simulation environment. $\mathcal{N}(\mu, \sigma)$ denotes a normal distribution.

Success measure The maximum length of an episode is 300 timesteps. If the goal is not reached within this limit, or if the object falls from the table, the attempt is considered a failure. An episode is counted as a success only if the object ends within 5 cm of the goal and remains upright as in the beginning, as described in Section 4.1.

5.2 Ablation Study

We first evaluate the performance of our approach under different hyperparameter settings. For this purpose, 12 objects are used for training and 4 for validation, with 900 episodes per object, resulting in a total of 2,034,900 samples of state keypoint pairs. This split provides a sufficiently large training set while still allowing validation on unseen objects to measure performance. Both the network input and the target values are normalized to the range $[-1, 1]$ for all models. Training is performed for 20 epochs using the AdamW optimizer [66] with a scheduled learning rate, starting at 4.00×10^{-4} and halved every 4 epochs. After selecting the best-performing model, we retrain it on the full set of 16 objects for the final evaluation.

Hyperparameters Table 5.2 summarizes the network hyperparameters of our architecture. The hidden dimension is fixed to 128 and used consistently across all modules. The Tokenize MLP processes the goal position (2-D) and optionally the center of mass (3-D), resulting in an

input of either 2 or 5 dimensions. Encoder related parameters such as patch size, number of layers, and number of attention heads are varied to analyze their effect on performance.

Hyperparameter	**Value**
Hidden dimension	128
Dropout probability	0.1
Tokenize MLP	(2/5, 128, 128, 128)
Cross Attention heads	8 / 16 (ablation)
Keypoint MLP	(130 / 133, 128, 64, 3)
Encoder embedding dim.	128
Encoder patch size	16 / 32 (ablation)
Encoder layers	2 / 3 (ablation)
Batch Size	64
Activation Function	ReLU
Layer Normalization	Yes
Batch Normalization	No
Optimizer	AdamW
Learning rate (initial)	4.00×10^{-4}
Learning rate schedule	decay by 50% every 4 epochs
Training epochs	20
Loss function	MSE
Normalization	Into the range [-1,1]

Table 5.2: Network hyperparameters of our architecture. Encoder parameters (patch size, encoder layers, attention heads) are varied in the ablation study.

Results In the first step, we train our encoder (see Figure 4.8). Here we use changing patch sizes (32 and 12) and a different number of encoder layers (2 and 3). The keypoint network is tested with varying numbers of attention heads (8 and 16). We further investigate the effect of the number of epochs on our model.

We evaluate all settings in two different variants. One that includes the center of mass as input and one that excludes it. In Table 5.3, we report the performance when including the center of mass as an input.

All models achieve a training loss close to zero, indicating that they fit the training set well. The validation error is significantly larger, ranging from about 0.08 to 0.15, which suggests limited generalization. Nevertheless, perfect accuracy is not required. While predictions must be accurate along the z-axis, in the x, y-plane it is sufficient if the keypoint lies on the correct hyperplane, as illustrated in Figure 4.7.

The results also indicate, that increasing the number of encoder layers generally improves performance. The effect of the number of attention heads, depends on the encoder depth. At layer 2, fewer heads perform better, while at layer 3, additional heads yield improved results.

This suggests that the benefit of multi-head attention emerges only in deeper encoders, where richer representations are available.

Reducing the patch size also improves performance, as the model attends to smaller regions of the point cloud, leading to more accurate keypoint localization.

Model	Encoder Layer	Patch Size	Attention Head	Train error	Validation error
Model 1	2	32	8	0.0678×10^{-6}	0.1034
Model 2	2	32	16	0.0620×10^{-6}	0.1494
Model 3	2	16	8	0.0748×10^{-6}	0.0838
Model 4	2	16	16	0.0622×10^{-6}	0.1017
Model 5	3	32	8	0.0639×10^{-6}	0.1185
Model 6	3	32	16	0.0400×10^{-6}	0.0960
Model 7	3	16	8	0.0688×10^{-6}	0.1052
Model 8	3	16	16	0.0490×10^{-6}	0.0821

Table 5.3: Performance with different hyperparameters, including the center of mass as input.

Similar results can be observed in Table 5.4 without using the center of mass as input, regarding training and validation errors. However, the absence of the center of mass leads to a much lower peak validation performance.

Compared to the experiments with the center of mass, using deeper encoders without the center of mass led to worse results. This difference can be explained by the fact that the center of mass is only available to the keypoint network, not during pretraining. With the center of mass, the keypoint network can interpret and exploit the richer encoder representations provided by additional layers. Without the center of mass, the keypoint network must rely only on the encoder features, where increased depth adds complexity without providing useful context.

A smaller patch size improves performance when the center of mass is included as input. In contrast, without the center of mass we have the opposite effect, apart from one outlier (from Model 6 to 8). This suggests that the center of mass provides the necessary global reference, which allows the model to benefit from finer local patch information. Without it, reducing the patch size mostly increases complexity without gains.

For the number of attention heads, we observe another effect compared to the experiments with the center of mass. Performance generally decreases as the number of heads increases, although we observe one outlier (from Model 1 to 2). This effect may be due to the fact that, in the case with the center of mass, the network can directly use this input to better interpret the encoder features relative to the object's center of mass. Without the center of mass as input, additional attention heads may instead introduce redundancy and noise, thereby harming performance.

Model	Encoder Layer	Patch Size	Attention Head	Train error	Validation error
Model 1	2	32	8	0.3045×10^{-6}	0.1585
Model 2	2	32	16	0.2055×10^{-6}	0.1375
Model 3	2	16	8	2.2314×10^{-6}	0.4317
Model 4	2	16	16	1.2106×10^{-6}	0.2522
Model 5	3	32	8	2.1039×10^{-6}	0.3065
Model 6	3	32	16	1.7595×10^{-6}	0.4241
Model 7	3	16	8	1.5396×10^{-6}	0.3566
Model 8	3	16	16	1.3708×10^{-6}	0.2854

Table 5.4: Performance with different hyperparameters, excluding the center of mass as input.

Training Epochs Figure 5.4 shows that during training, the error rapidly drops close to zero after the first epoch, indicating that the model quickly adapts to the task. When including the center of mass in the input, the validation error continues to decrease and reaches a stable level after approximately 15 epochs. In contrast, when excluding the center of mass, stabilization occurs faster, around epoch 10. This suggests that the model without the center of mass converges more quickly, but the additional input ultimately leads to more stable and reliable performance. The difference in convergence speed may indicate that incorporating the center of mass requires more training iterations to exploit the additional information effectively.

The complete results of both experiments are provided in the appendix (Appendix A, Table A.1, Table A.2), including the training and validation errors per epoch as well as the learning rate.

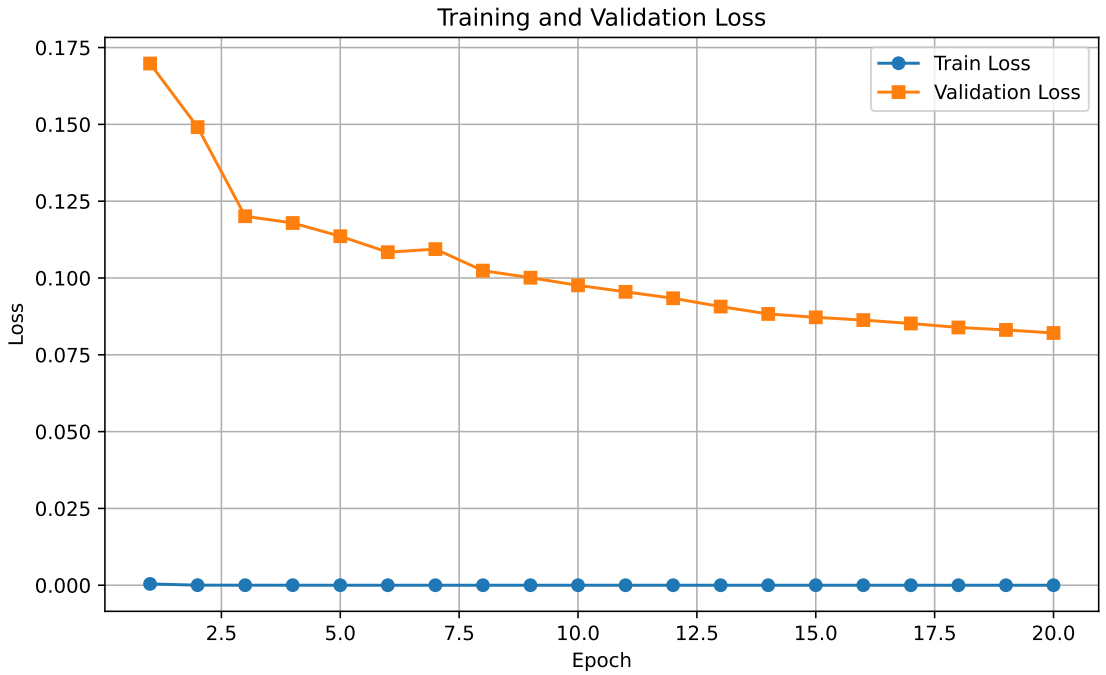
For the final evaluation, we use the hyperparameters of Model 8 from Table 5.3 and Model 2 from Table 5.4, and retrain them on the full set of 16 objects.

5.3 ID Generalization

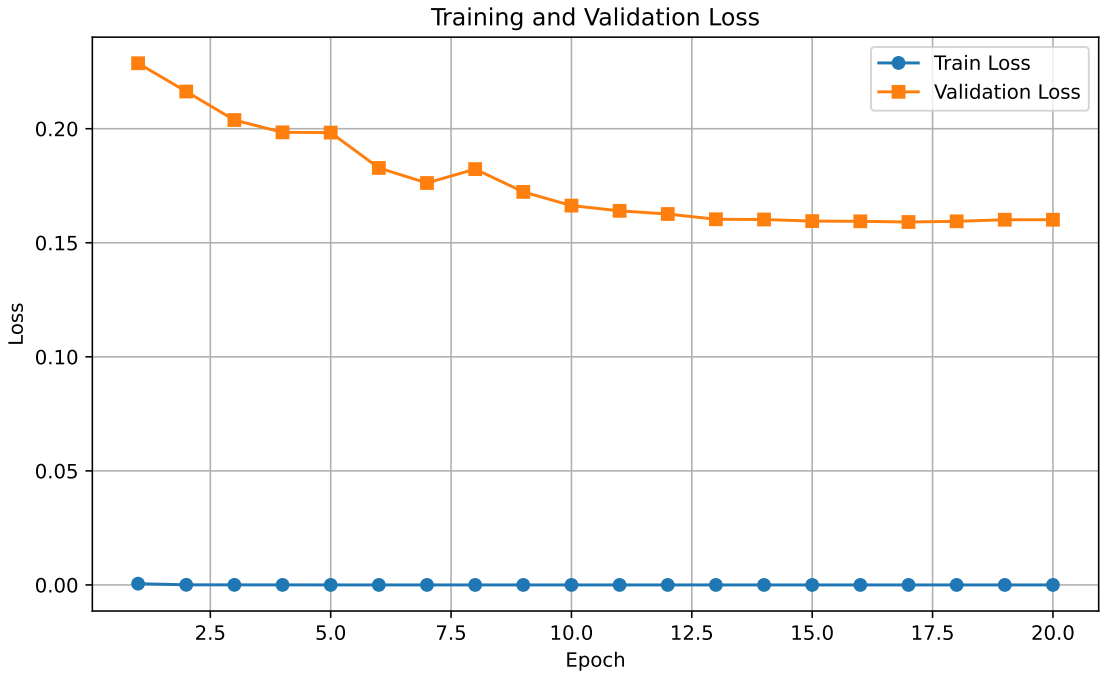
We evaluate ID generalization by training and testing on the same object instances while varying their scale. In addition, we assess how well the model handles size variations of the same objects. Finally, we include objects that are similar to those in the training set to further evaluate ID generalization.

In Table 5.5, we report the performance under different object size variations. Specifically, we test on objects that are 12.5% and 25% smaller or larger than in training. Overall, our architecture is relatively robust to moderate variations in size, while the baseline performs way worse. This indicates that representing the task via keypoint prediction provides better ID generalization than direct action prediction. Furthermore, excluding the center of mass (COM) from the input leads to a significant reduction in success rate, suggesting that this feature is essential for accurate prediction.

In addition, as shown in Table 5.6, our method maintains its performance also when evaluated on novel objects, demonstrating its ability to transfer beyond the training set. However, the



(a) Training and validation loss over 20 epochs of the best model (Model 8), including the center of mass in the input.



(b) Training and validation loss over 20 epochs of the best model (Model 2), excluding the center of mass in the input.

Figure 5.4: Comparison of training and validation error curves with and without the center of mass as input.

performance on novel objects does not differ much from that on the training objects. This is because the novel objects are similar to those seen during training. Also the block object does bias the performance.

Object size variation	Ours (with COM)	Ours (w/o COM)	BC
Original	87.1%	50.3%	40.5%
−12.5%	86.4%	44.7%	35.5%
−25%	80.9%	42.7%	32.9%
+12.5%	87.0%	48.9%	37.8%
+25%	83.5%	49.5%	38.1%

Table 5.5: Average success rate under size variations of the 16 training objects, evaluated over 50 episodes per object.

Object size variation	Ours (with COM)	Ours (w/o COM)	BC
Original	82.7%	48.9%	36.5%
−12.5%	77.4%	43.7%	32.5%
−25%	74.8%	41.1%	30.3%
+12.5%	85.6%	55.5%	38.8%
+25%	90.3%	57.6%	42.1%

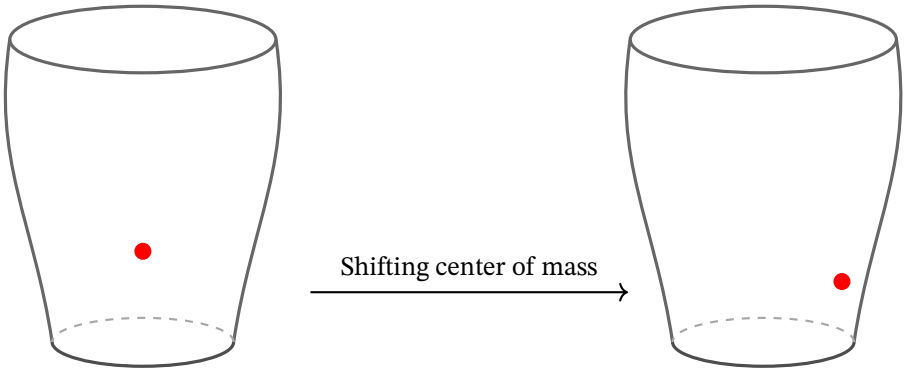
Table 5.6: Average success rate under size variations of 6 novel objects similar to the training set (ID), evaluated over 50 episodes per object.

5.4 OOD Generalization

For testing OOD generalization, we apply two methods. First, we evaluate on novel objects outside the training set and varying sizes of those objects. Second, we test robustness to shifts in the center of mass. This is done for novel objects, as illustrated in Figure 5.5, where the true center of mass is located in the middle (e.g. at the bottom origin of the glass), while in the abnormal case the center of mass is shifted far outward. By this, we obtain a completely different test set distribution compared to the training set.

Tests on novel objects In Table 5.7, we evaluate the performance on novel objects under size variations. Compared to the ID case, the success rates are lower, reflecting the increased difficulty of generalizing to unseen shapes. Still, our method with center of mass input achieves consistently higher performance than the baseline across all size changes. Excluding the center of mass reduces performance again. These results shows both the challenge of OOD generalization and the importance of including physical object properties such as the center of mass for reliable predictions.

Tests on objects with shifted center of mass Table 5.8 reports the performance for objects with and without a shifted center of mass. We observe again, that the performance drops



(a) The center of mass is located at the lower middle of the object (red point).

(b) The center of mass is located at the lower right (red point).

Figure 5.5: Glass illustrations with center of mass (red points) at different positions. Subfigure 5.5a shows the original center of mass of the object, while Subfigure 5.5b depicts the shifted center of mass for abnormal objects.

Object size variation	**Ours (with COM)**	**Ours (w/o COM)**	**BC**
Original	59.3%	34.4%	28.2%
−12.5%	62.0%	42.1%	31.0%
−25%	56.6%	34.5%	28.7%
+12.5%	61.3%	37.3%	34.5%
+25%	59.9%	41.9%	32.9%

Table 5.7: Average success rate under size variations of 10 novel objects (OOD), evaluated over 50 episodes per object.

once the center of mass is moved. A shifted mass distribution makes the object more unstable during pushing, which requires higher precision of the keypoint network. As a result, the network often fails to adapt to this change, and the success rate decreases significantly across all methods.

The problem is more severe when the center of mass is not given as input. In this case, the model can not even detect whether a shift has occurred. This results in a worse performance and results close to pure BC.

Performance also depends on the stability of the object itself. The block and the telephone achieve the highest success rates in the shifted settings. These objects are geometrically simple and less likely to tip over, which reduces the need for very precise control. In contrast, objects like the mug or bottle are more unstable and sensitive to small errors. Shifting the center of mass in such objects increases this difficulty, resulting in poor performance.

In summary, the experiments highlight two main aspects. First, a shift in the center of mass poses a clear generalization challenge, since the policy must act more precisely and compensate for instabilities. Second, providing the center of mass as an explicit input is critical to handle such variations.

Object	Ours (with COM)	Ours (w/o COM)	BC
Block	88.9%	63.0%	54.5%
Shifted	53.0%	25.6%	20.4%
Bottle	86.7%	60.0%	46.3%
Shifted	28.8%	13.5%	19.4%
Elephant	91.7%	50.0%	41.0%
Shifted	30.3%	14.9%	17.2%
Telephone	77.8%	33.3%	22.9%
Shifted	42.4%	19.1%	15.1%
Mug	92.6%	40.7%	25.6%
Shifted	28.8%	20.2%	13.8%

Table 5.8: Average success rate per object with and without a shifted center of mass (1.5 cm), evaluated over 50 episodes per object.

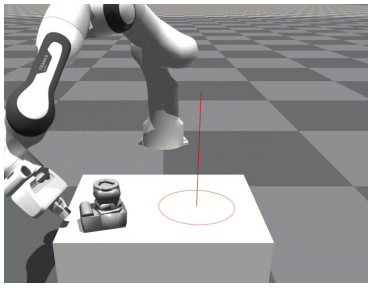
5.5 Qualitative Examples

While the quantitative results provide an overall measure of performance, we now analyze how the learned policy behaves in individual rollouts. In this section, we illustrate several examples from the best performing policy, including both successful and failed trials. These examples highlight typical behaviors and strengths of the approach, as well as common failure modes.

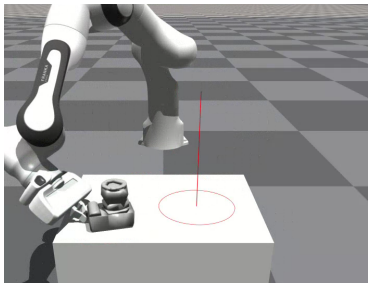
In order to conduct this analysis, we use three examples of successful pushing sequences and three examples of unsuccessful ones. For both success and failure cases, we consider the same set of objects: a candle, a camera, and a toy object with bumps. The camera represents a challenging case due to its geometry, yet it remains relatively stable; here, we aim to investigate how the robot handles objects that are geometrically complex but not easily tipped over. In contrast, the candle object is positioned in an unstable manner, as it is thin and prone to rolling. This allows us to observe how the robot behaves when dealing with objects that are difficult to stabilize. Finally, we include the toy object with bumps, which poses an additional challenge, as successful pushing requires precise contact points. This case provides insights into how the agent behaves when local irregularities are present and what typical failure modes emerge.

Successful pushing sequences Figure 5.6 illustrates an example of a successful pushing sequence with a camera as an object. The robot first moves behind the object to establish a favorable position for pushing. Afterwards, it aligns its motion direction with the intended keypoint orientation. A particularly interesting aspect of this sequence is that the robot does not push the object directly in the center. Instead, it makes contact at a corner of the camera object and uses this point to apply a controlled push. This strategy prevents the object from rotating and ensures a straight motion toward the target. This example highlights the ability of the learned policy to recognize and exploit small geometric details, such as object corners, even though they represent relatively small features in the scene. For objects with irregular

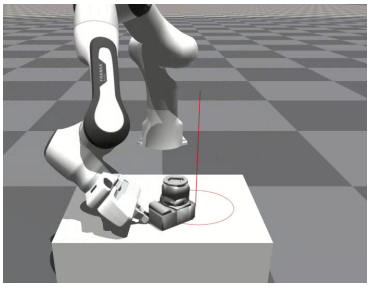
shapes, identifying such precise contact points is essential to maintain stability during pushing. This demonstrates that the policy not only captures the global movement direction but is also capable of fine grained local reasoning about contact strategies. As a result, the robot can reliably move the object to the target, even under conditions that require high precision.



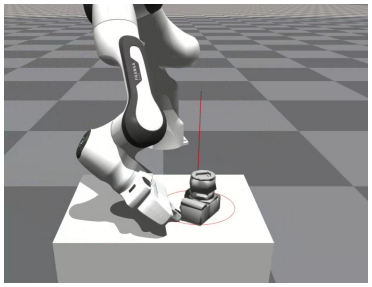
(a) Robot moves behind the object.



(b) Targeted contact at the object corner.



(c) Push from the corner toward the goal.

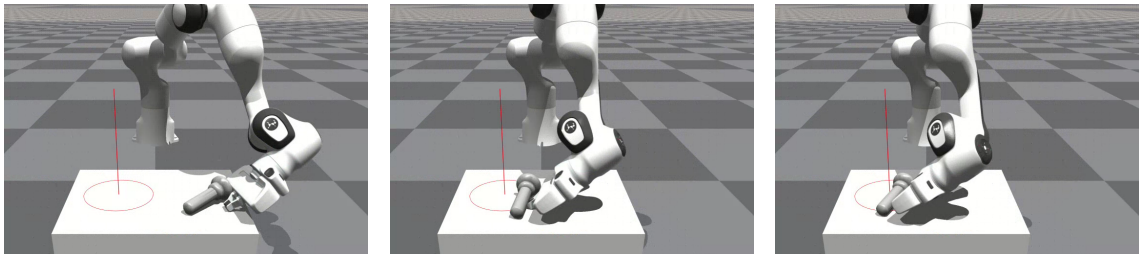


(d) Successful Push.

Figure 5.6: Example of a successful pushing sequence with the Franka Emika Panda robot [8], where the object (camera) is moved toward the red goal circle.

Figure 5.7 shows a successful push of a candle. In the beginning, the robot does not immediately establish contact with the object using the end-effector. Instead, the interaction is initiated through the hand state. This causes a slight rotation. While rotations often lead to unstable or failed outcomes, in this case the effect is beneficial. The object turns in a more favorable orientation to align with the goal direction. Once the object is properly aligned, the robot manages to establish a stable contact with the end-effector and continues to push the object directly toward the target.

In this example (Figure 5.8), the robot successfully completes the pushing task, with a toy as an object which has multiple bumps. The push is a success, even though the initial contact is not established at the corner of the object. Instead, the robot approaches from the left and makes contact with the left side surface. This deviation from the expected contact point results in a rotation of the object. In this case it helps to realign the object toward the goal region. As the robot continues to push from the side, the object reorients and moves closer to the target, until it fully reaches the goal.



(a) Robot approaches from the side, initial contact made with the hand state rather than the end-effector.

(b) Push from the middle.

(c) Successful push.

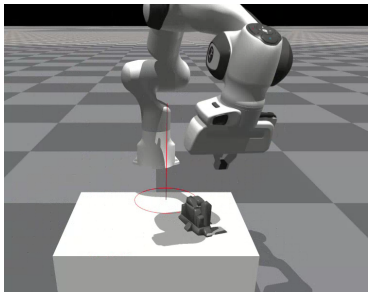
Figure 5.7: Successful pushing sequence with the Franka Emika Panda robot, where an initial side contact with the hand state allows the object to rotate into a better orientation, enabling the end-effector to complete the push into the target.

This example illustrates that successful outcomes do not always require precise contact at predefined spots such as corners or edges. The policy is able to exploit alternative contact configurations, here turning a side push into a stable trajectory. Such flexibility highlights the robustness of the learned strategy, as the robot can recover from small variations in contact point while still achieving the overall task objective.

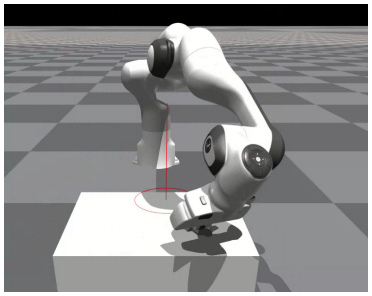
Unsuccessful pushing sequences Figure 5.9 shows an example of an unsuccessful pushing sequence with a camera as an object. In this case, the robot establishes contact with the object at its center, which at first appears to be a stable configuration. However, due to uncertainty in the prediction at the end, the arm drifts slightly to the right during the pushing motion. This small deviation makes the object go off the optimal path. Since the robot is unable to recover from this early error, the object continues moving in the wrong direction. Such failure cases illustrate how even minor inaccuracies in contact placement or trajectory estimation can accumulate over time and result in unsuccessful outcomes. Unlike the successful example, where the policy leveraged small features such as object corners, here the policy lacked a corrective mechanism once the deviation occurred.

Figure 5.10 shows an unsuccessful pushing attempt with the candle object. The robot initially starts well and tries to push from the tip of the object. However, it approaches slightly from the side, which creates an unstable contact. When the robot attempts to correct its motion, the object rolls off the table and is pushed in the wrong direction. This example highlights how even small errors in the contact point can quickly cause failure.

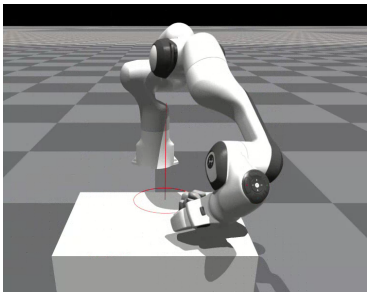
Figure 5.11 shows an unsuccessful pushing attempt with a toy that has multiple bumps. The robot starts well and establishes contact in the center of the object. However, as it pushes, the end-effector hits one of the protruding bumps, which causes the object to rotate unexpectedly. Since the policy cannot clearly distinguish between the flat surface and the raised bump, the



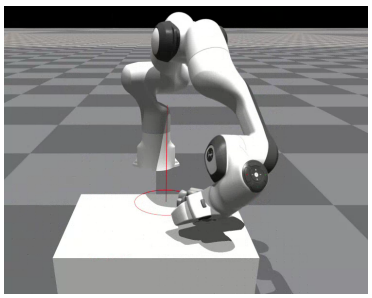
(a) Robot approaches the object from behind.



(b) Contact is made with the left side instead of the corner.

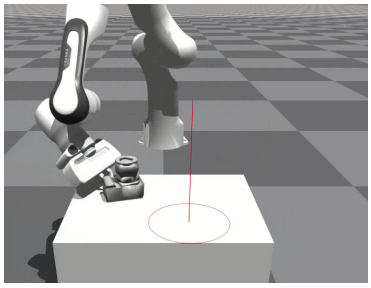


(c) The object rotates while being pushed sideways.

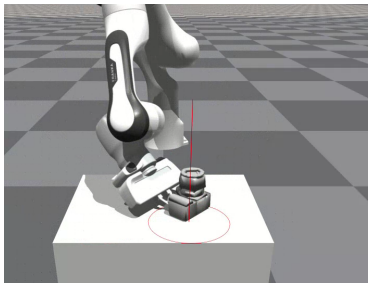


(d) Final alignment and successful push to the goal.

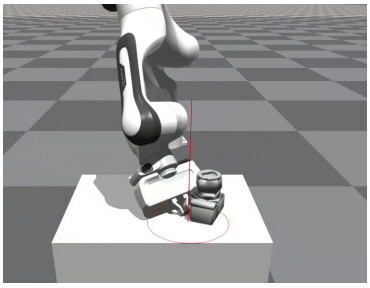
Figure 5.8: Successful pushing sequence with the Franka Emika Panda robot. Instead of pushing at the corner, the robot establishes contact on the side surface, which induces a rotation that helps to align the object toward the goal.



(a) Initial contact established at the center of the object.

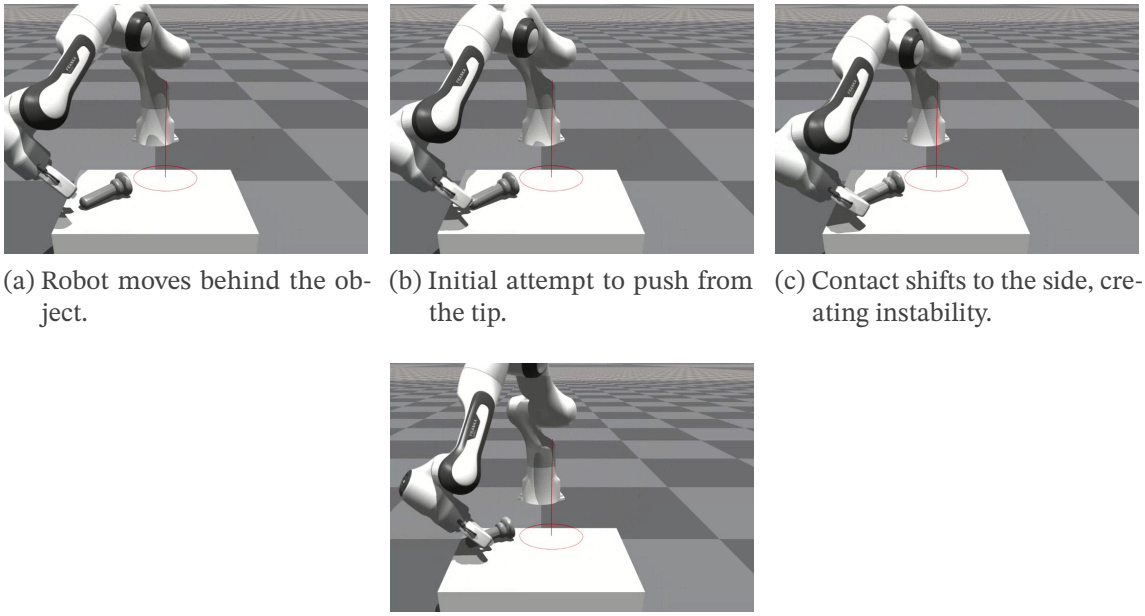


(b) Arm drifts slightly to the right due to prediction uncertainty.



(c) Robot fails to recover, object deviates from the goal.

Figure 5.9: Example of an unsuccessful pushing sequence with the Franka Emika Panda robot, where the camera object misses the red goal circle due to accumulated errors.



(a) Robot moves behind the object.

(b) Initial attempt to push from the tip.

(c) Contact shifts to the side, creating instability.

(d) Object rolls away as it is pushed in the wrong direction.

Figure 5.10: Example of an unsuccessful pushing sequence with the Franka Emika Panda robot, where the candle object rolls off and misses the goal.

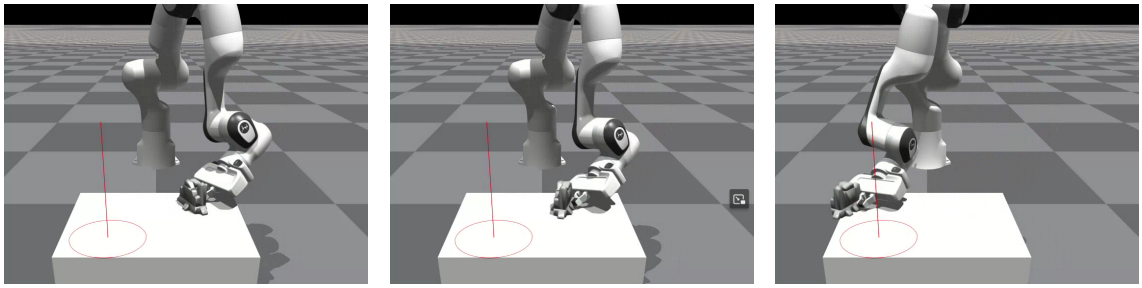
object is pushed off the table. This example demonstrates the difficulty of handling objects with irregular geometries.

5.6 Summary

In this chapter we evaluated our approach under different settings, including ablation studies, in-distribution (ID) and out-of-distribution (OOD) generalization. The ablation experiments showed that model depth and patch size influence performance, and that including the center of mass as input is a key factor for obtaining reliable results. Training and validation curves further show that both model variants converge quickly.

For ID generalization, our method demonstrated robustness to moderate variations in object size and maintained good performance on objects similar to the training set. This indicates that representing the task through keypoints supports generalization better than direct action prediction. The center of mass again proved to be critical, as excluding it consistently reduced performance.

For OOD generalization, we tested on unseen objects and shifts in the center of mass. As expected, performance was lower compared to ID settings, reflecting the higher difficulty of novel object shapes and changed dynamics. Shifting the center of mass strongly reduced success rates, since such changes make objects more unstable and require higher precision.



(a) Robot establishes contact in the center of the object.

(b) End-effector collides with a protruding bump, rotating the object.

(c) Object is pushed off the table while in contact with the bump.

Figure 5.11: Example of an unsuccessful pushing sequence with the Franka Emika Panda robot, where the toy-like object with protruding bumps is pushed in the wrong direction and falls off the table.

When the center of mass was not provided, the network could not recognize these changes and had sample performance of simple BC. The results also showed that stable objects such as blocks or telephones are handled more successfully, while unstable shapes like mugs or bottles are more prone to failure.

Overall, the experiments highlight two main conclusions. First, keypoint based representations enable stronger generalization compared to direct action prediction, both in ID and OOD scenarios. Second, explicitly providing the center of mass is important to handle shifts in the center of mass and to maintain robust performance.

6 Conclusion

This thesis addressed the challenge of generalization in non-prehensile manipulation, specifically pushing, by reformulating the task as a keypoint prediction problem. Rather than directly mapping observations to actions, the proposed method predicts a single 3-D contact point. This abstraction compresses high-dimensional inputs such as point clouds into a compact representation that emphasizes the regions of the object most relevant for manipulation. The architecture is built on CORN embeddings, which provide geometric features of the object. These embeddings serve as the input to a cross-attention module, where they are combined with query tokens derived from the goal position and optionally the center of mass. The attended features are then merged with a skip connection from the original input and passed through an MLP that outputs a single 3-D keypoint representing the predicted contact location.

The evaluation provided evidence that this representation is beneficial compared to direct behavior cloning on actions. In in-distribution (ID) experiments, the model maintained high success rates even when object sizes varied, demonstrating robustness to moderate perturbations of the training distribution. In the ID experiments, our approach also outperforms the baseline. In out-of-distribution (OOD) settings with novel objects or shifted mass distributions, the keypoint-based approach consistently outperformed the baseline as well. The performance naturally decreased on more challenging and irregular objects, especially when the center of mass is shifted. Also, including the center of mass as an input is very important for performance. Qualitative examples further illustrated the strengths of the approach, but also revealed its limitations. The policy can generate precise pushes by targeting local features, small errors in contact placement may accumulate over time and lead to failure, especially for unstable shapes.

Overall, keypoint prediction provides a more generalizable representation of pushing compared to direct action policies, showing that abstraction helps to reduce overfitting to specific training distributions. This insight may extend beyond pushing to other non-prehensile and prehensile manipulation tasks.

At the same time, the study also exposed several open challenges. The approach was validated exclusively in simulation, and the question of real-world transfer remains open. Bridging this gap will require handling sensor noise and imperfect perception in practice. Moreover, while the reset heuristic during data generation reduced some distributional drift, long-horizon stability and recovery from early mistakes are still unresolved problems. The reliance on oracle information, such as ground truth center of mass during training, further limits the direct

applicability of the method outside simulation. Also, although the experiments indicate the potential of keypoint based representations, scaling to more complex manipulation scenarios may require richer abstractions, or multiple keypoints.

In conclusion, this thesis demonstrates that reframing robotic pushing as a keypoint prediction task improves generalization across varied conditions. At the same time, the study also exposed open challenges.

6.1 Future Work

There are several directions for future research building on this work. First, the current keypoint network could be extended to directly serve as a policy. Instead of predicting only contact keypoints, the architecture could process them further to output actions. Both, the keypoint network and the policy network can be trained seperately. Second, an alternative line of research would be to jointly train both components, e.g. the keypoint predictor and the policy, within a single backpropagation framework. This joint optimization may allow better integration of keypoint information into the policy, potentially leading to improved performance.

To increase prediction accuracy, one could add a constraint enforcing that keypoints lie on the object's surface. Furthermore, by retraining the contact network and incorporating location information already in the encoder, the model could gain more accurate knowledge about the true position of points in 3-D space.

Another promising direction is to extend the framework beyond predicting a single keypoint. By predicting multiple candidate keypoints, the model could represent several possible contact regions on an object, allowing downstream components to select the most stable or task-relevant option. This would be particularly useful for objects with multiple affordances or when different pushing strategies are possible. Also we can extend the network, so it learns sequences of keypoints to guide a full pushing trajectory. Such an approach would reduce error accumulation.

To further improve performance, the data generation procedure could be made more diverse. For example, the end-effector could approach objects from different angles instead of always following the same path, thereby exposing the policy to a wider variety of contact situations. Additionally, the network could be trained on a larger and more diverse set of objects to improve robustness and facilitate sim-to-real transfer.

Finally, Inverse Reinforcement Learning (IRL) [14] could be applied. IRL has the advantage of inferring reward functions from demonstrations. By leveraging keypoints within this framework, the learned rewards could directly reflect object specific properties or task constraints tied to keypoint interactions. This may improve generalization and robustness beyond what

explicit supervision provides. Moreover, IRL could also offer an alternative to the reset heuristic, reducing the need for manually designed interventions during training. This is because we first learn a pushing relevant reward function and then apply RL, which allows the policy to explore states beyond those seen in the demonstrations, thereby reducing the problem of distributional shift in the data.

A Appendix

Epoch	Training Error	Validation Error	Learning Rate
1	423.8254×10^{-6}	0.1698	4.00×10^{-4}
2	23.9675×10^{-6}	0.1491	4.00×10^{-4}
3	12.3349×10^{-6}	0.1201	4.00×10^{-4}
4	7.8454×10^{-6}	0.1179	4.00×10^{-4}
5	1.8320×10^{-6}	0.1136	2.00×10^{-4}
6	1.6691×10^{-6}	0.1084	2.00×10^{-4}
7	1.5458×10^{-6}	0.1094	2.00×10^{-4}
8	1.4633×10^{-6}	0.1024	2.00×10^{-4}
9	0.4519×10^{-6}	0.1001	1.00×10^{-4}
10	0.4302×10^{-6}	0.0976	1.00×10^{-4}
11	0.4146×10^{-6}	0.0955	1.00×10^{-4}
12	0.4011×10^{-6}	0.0934	1.00×10^{-4}
13	0.1490×10^{-6}	0.0907	5.00×10^{-5}
14	0.1398×10^{-6}	0.0883	5.00×10^{-5}
15	0.1321×10^{-6}	0.0872	5.00×10^{-5}
16	0.1254×10^{-6}	0.0863	5.00×10^{-5}
17	0.0574×10^{-6}	0.0852	2.50×10^{-5}
18	0.0539×10^{-6}	0.0839	2.50×10^{-5}
19	0.0517×10^{-6}	0.0831	2.50×10^{-5}
20	0.0490×10^{-6}	0.0821	2.50×10^{-5}

Table A.1: Training and validation error of best Model 8 (including center of mass in the input) across epochs with corresponding learning rates.

Epoch	Training Error	Validation Error	Learning Rate
1	545.7587×10^{-6}	0.2287	4.00×10^{-4}
2	51.3762×10^{-6}	0.2163	4.00×10^{-4}
3	31.5225×10^{-6}	0.2038	4.00×10^{-4}
4	22.3477×10^{-6}	0.1984	4.00×10^{-4}
5	5.8093×10^{-6}	0.1983	2.00×10^{-4}
6	5.4314×10^{-6}	0.1828	2.00×10^{-4}
7	4.9038×10^{-6}	0.1762	2.00×10^{-4}
8	4.6085×10^{-6}	0.1823	2.00×10^{-4}
9	1.6191×10^{-6}	0.1723	1.00×10^{-4}
10	1.5197×10^{-6}	0.1663	1.00×10^{-4}
11	1.4240×10^{-6}	0.1640	1.00×10^{-4}
12	1.4041×10^{-6}	0.1626	1.00×10^{-4}
13	0.6343×10^{-6}	0.1603	5.00×10^{-5}
14	0.6128×10^{-6}	0.1602	5.00×10^{-5}
15	0.5896×10^{-6}	0.1595	5.00×10^{-5}
16	0.5646×10^{-6}	0.1594	5.00×10^{-5}
17	0.3371×10^{-6}	0.1591	2.50×10^{-5}
18	0.3249×10^{-6}	0.1594	2.50×10^{-5}
19	0.3132×10^{-6}	0.1601	2.50×10^{-5}
20	0.3045×10^{-6}	0.1601	2.50×10^{-5}

Table A.2: Training and validation error across epochs of best Model 2 (excluding center of mass in the input) with corresponding learning rates.

List of Figures

List of Tables

List of References

[1] I. Mordatch, Z. Popović, and E. Todorov, "Contact-invariant optimization for hand manipulation," in *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, 2012, pp. 137–144.

[2] J. Moura, T. Stouraitis, and S. Vijayakumar, "Non-prehensile planar manipulation via trajectory optimization with complementarity constraints," in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022, pp. 8725–8731.

[3] F. Lin *et al.*, "Data scaling laws in imitation learning for robotic manipulation," *arXiv preprint arXiv:2410.18647*, 2024.

[4] E. Jang *et al.*, "Bc-z: Zero-shot task generalization with robotic imitation learning," in *Conference on Robot Learning*, PMLR, 2022.

[5] Y. Duan *et al.*, "Benchmarking deep reinforcement learning for continuous control," in *International Conference on Machine Learning*, PMLR, 2016.

[6] T. Z. Zhao *et al.*, "Aloha unleashed: A simple recipe for robot dexterity," *arXiv preprint arXiv:2410.13126*, 2024.

[7] Y. Cho *et al.*, "Corn: Contact-based object representation for nonprehensile manipulation of general unseen objects," *arXiv preprint arXiv:2403.10760*, 2024.

[8] S. Haddadin, "The franka emika robot: A standard platform in robotics research," *IEEE Robotics & Automation Magazine*, 2024.

[9] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison, "Rlbench: The robot learning benchmark & learning environment," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3019–3026, 2020. DOI: 10.1109/LRA.2020.2977263.

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

[11] K. Wolff, W. Förstner, and D. Cremers, "Point cloud noise and outlier removal for image-based 3d reconstruction," in *2016 Fourth International Conference on 3D Vision (3DV)*, IEEE, 2016, pp. 118–127.

[12] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2014, pp. 3104–3112.

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017.

[14] A. Y. Ng and S. Russell, "Algorithms for inverse reinforcement learning," in *Proceedings of the Seventeenth International Conference on Machine Learning (ICML)*, vol. 1, 2000.

[15] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[16] J. Liu, Z. Lin, T. Ding, Y. Liu, and J. Tang, "Towards out-of-distribution generalization: A survey," *arXiv preprint arXiv:2108.13624*, 2021.

[17] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge: MIT Press, 1998, vol. 1.

[18] J. Quiñonero-Candela, M. Sugiyama, A. Schwaighofer, and N. D. Lawrence, *Dataset Shift in Machine Learning*. Cambridge, MA: MIT Press, 2009.

[19] F. Provost, "Machine learning from imbalanced data sets 101," in *Proceedings of the AAAI'2000 Workshop on Imbalanced Data Sets*, vol. 68, AAAI Press, 2000.

[20] A. Mandlekar, D. Xu, S. Nasiriany, Y. Zhu, L. Fei-Fei, S. Savarese, D. Fox, and J. Bohg, "Human-in-the-loop task and motion planning for imitation learning," in *Proceedings of the Conference on Robot Learning (CoRL)*, PMLR, 2023, pp. –.

[21] V. N. Vapnik, *Statistical Learning Theory*. New York: Wiley, 1998.

[22] Q. Wang, L. Ma, S. Wang, Y. Yang, Y. Tian, and Y. Liu, "A comprehensive survey of loss functions in machine learning," *Annals of Data Science*, vol. 9, no. 2, pp. 187–212, 2022.

[23] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016, arXiv:1609.04747.

[24] S. Ross and J. A. Bagnell, "Efficient reductions for imitation learning," in *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010, pp. 661–668.

[25] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in Neural Information Processing Systems*, vol. 29, 2016.

[26] L. Harwin and P. Supriya, "Comparison of sarsa algorithm and temporal difference learning algorithm for robotic path planning for static obstacles," in *2019 Third International Conference on Inventive Systems and Control (ICISC)*, IEEE, 2019, pp. 730–734. DOI: 10.1109/ICISC44355.2019.9036442.

[27] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[28] H. Tan, "Reinforcement learning with deep deterministic policy gradient," in *2021 International Conference on Artificial Intelligence, Big Data and Algorithms (CAIBDA)*, IEEE, 2021, pp. 228–231. DOI: 10.1109/CAIBDA53623.2021.00057.

[29] A. Mucherino, P. J. Papajorgji, and P. M. Pardalos, "K-nearest neighbor classification," in *Data Mining in Agriculture*, Springer New York, 2009, pp. 83–106.

[30] K. Rana *et al.*, "Affordance-centric policy learning: Sample efficient and generalisable robot policy learning using affordance-centric task frames," *arXiv preprint arXiv:2410.12124*, 2024.

[31] J. Borja-Diaz *et al.*, "Affordance learning from play for sample-efficient policy learning," in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022.

[32] X. Zhang and A. Boularias, "One-shot imitation learning with invariance matching for robotic manipulation," *arXiv preprint arXiv:2405.13178*, 2024.

[33] L. Manuelli *et al.*, "Kpam: Keypoint affordances for category-level robotic manipulation," in *The International Symposium of Robotics Research*, Cham: Springer International Publishing, 2019.

[34] S. A. Mehta *et al.*, "Stable-bc: Controlling covariate shift with stable behavior cloning," *IEEE Robotics and Automation Letters*, 2025.

[35] Z. Wang *et al.*, "Reward-constrained behavior cloning," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2021.

[36] A. Jonnavittula, S. Parekh, and D. P. Losey, "View: Visual imitation learning with waypoints," *Autonomous Robots*, vol. 49, no. 1, pp. 1–26, 2025.

[37] Y. Zhu *et al.*, "Learning generalizable manipulation policies with object-centric 3d representations," *arXiv preprint arXiv:2310.14386*, 2023.

[38] L. Cong *et al.*, "Reinforcement learning with vision-proprioception model for robot planar pushing," *Frontiers in Neurorobotics*, vol. 16, p. 829 437, 2022.

[39] L. Bergmann *et al.*, "Precision-focused reinforcement learning model for robotic object pushing," *arXiv preprint arXiv:2411.08622*, 2024.

[40] D. Kalashnikov *et al.*, "Scalable deep reinforcement learning for vision-based robotic manipulation," *Conference on Robot Learning*, 2018.

[41] A. Al-Shanoon, H. Lang, Y. Wang, *et al.*, "Learn to grasp unknown objects in robotic manipulation," *Intel Serv Robotics*, vol. 14, pp. 571–582, 2021. DOI: 10.1007/s11370-021-00380-9.

[42] H. Lang and A. Al-Shanoon, "Learn to grasp unknown-adjacent objects for sequential robotic manipulation," *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 105, 2022. DOI: 10.1007/S10846-022-01702-4.

[43] A. Zeng *et al.*, "Learning synergies between pushing and grasping with self-supervised deep reinforcement learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018.

[44] X. Xie *et al.*, "Learning virtual grasp with failed demonstrations via bayesian inverse reinforcement learning," in *Proceedings of the 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2019.

[45] Z. Hu, Y. Zheng, and J. Pan, "Grasping living objects with adversarial behaviors using inverse reinforcement learning," *IEEE Transactions on Robotics*, vol. 39, no. 2, pp. 1151–1163, 2023.

[46] S. Arora, P. Doshi, and B. Banerjee, "Online inverse reinforcement learning with learned observation model," in *Proceedings of Machine Learning Research*, Presented on June 4, 2023, PMLR, 2023, pp. 1468–1477.

[47] Z. Wu *et al.*, "Efficient sampling-based maximum entropy inverse reinforcement learning with application to autonomous driving," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5355–5362, 2020.

[48] D. Brown *et al.*, "Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations," in *International Conference on Machine Learning*, PMLR, 2019.

[49] B. D. Ziebart *et al.*, "Maximum entropy inverse reinforcement learning," in *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, vol. 8, 2008.

[50] J. Fu, K. Luo, and S. Levine, "Learning robust rewards with adversarial inverse reinforcement learning," *arXiv preprint arXiv:1710.11248*, 2017.

[51] T. Ni *et al.*, "F-irl: Inverse reinforcement learning via state marginal matching," in *Conference on Robot Learning*, PMLR, 2021.

[52] A. Szot *et al.*, "Bc-irl: Learning generalizable reward functions from demonstrations," *arXiv preprint arXiv:2303.16194*, 2023.

[53] R. M. Murray, Z. Li, and S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 2017.

[54] L. P. Kaelbling and T. Lozano-Pérez, "Integrated task and motion planning in belief space," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1194–1227, 2013.

[55] Y. Maeda, H. Kijimoto, Y. Aiyama, and T. Arai, "Planning of graspless manipulation by multiple robot fingers," in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, IEEE, vol. 3, 2001, pp. 2474–2479.

[56] J. Liang, X. Cheng, and O. Kroemer, "Learning preconditions of hybrid force-velocity controllers for contact-rich manipulation," *arXiv preprint arXiv:2206.12728*, 2022.

[57] C. R. Qi, H. Su, K. Mo, and L. 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 (CVPR)*, 2017, pp. 652–660.

[58] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 5099–5108.

[59] S. Xie, J. Gu, D. Guo, C. R. Q. Li, L. J. Guibas, and O. Litany, "Pointcontrast: Unsupervised pre-training for 3d point cloud understanding," in *European Conference on Computer Vision*, Springer, 2020, pp. 574–591.

[60] H. Wang, Q. Liu, X. Chen, H. Chen, J. Yu, Y. Yang, and C. Wang, "Unsupervised point cloud pre-training via occlusion completion," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 9782–9792.

[61] X. Yu, W. Tang, Y. Rao, J. Huang, B. Zhou, J. Lu, and J. Zhou, "Point-bert: Pre-training 3d point cloud transformers with masked point modeling," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 19 313–19 322.

[62] A. Pang, W. Wang, F. E. Tay, W. Liu, Y. Tian, and L. Yuan, "Masked autoencoders for point cloud self-supervised learning," in *European Conference on Computer Vision*, Springer, 2022, pp. 604–621.

[63] X. Ying, "An overview of overfitting and its solutions," in *Journal of Physics: Conference Series*, IOP Publishing, vol. 1168, 2019, p. 022 022.

[64] S. R. Buss, "Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods," *IEEE Journal of Robotics and Automation*, vol. 17, no. 1-19, p. 16, 2004.

[65] V. Makoviychuk, A. Wawrzyniak, A. Mehta, D. Makoviichuk, Y. Lu, M. Macklin, G. State, F. Yoon, X. Yin, A. Jain, *et al.*, "Isaac gym: High performance gpu-based physics simulation for robot learning," *arXiv preprint arXiv:2108.10470*, 2021.

[66] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.