Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

MASTERTHESIS

Data Augmentation in Offline Reinforcement Learning for Robotic Dough Shaping

vorgelegt von

Fabian Hendrik Wieczorek

MIN-Fakultät

Fachbereich Informatik

Technische Aspekte Multimodaler Systeme

Studiengang: Informatik

Matrikelnummer: 6911629

Erstgutachter: Prof. Dr. Jianwei Zhang

Zweitgutachter: Dr. Norman Hendrich

Betreuer: M.Sc. Michael Görner

Abstract

Robotic manipulation of elastoplastic material has recently attracted much attention with applications in e.g. health care services, medical surgeries, and food processing. The absence of rigidity poses unique challenges regarding perception, modelling, and control which are barely addressed in previous research. While the use of simulations and learned world models is prevalent in recent approaches, this thesis investigates the applicability of the offline reinforcement learning paradigm in this domain. In an elastoplastic manipulation task, the agents are trained to map actions from pixels that form a piece of dough into a target shape. However, acquiring large datasets that meet the standards of offline reinforcement learning seems infeasible for this task. Instead, a limited set of human demonstrations is recorded and a model-based data augmentation approach, leveraging conditioned U-Nets and diffusion-based re-projection, is applied. As this stresses the need for high-quality demonstrations, the teleoperation setup was gamified with the idea of motivating users to spend more effort on the task. The trained policies are evaluated in 8 tasks focusing on key actions from the original tasks. Though no clear correlation between augmentations and performance was identified in this setting, the results hint that the combination of augmentations can lead to more careful behaviour. Additionally, the qualitative analysis points out various difficulties that were not sufficiently covered by the training data which could be addressed in the future.

Zusammenfassung

Das Formen von elastoplastischem Material durch Roboter hat in letzter Zeit an Popularität gewonnen. Anwendungen gibt es z.B. in der Gesundheitsfürsorge, bei medizinischen Eingriffen und in der Lebensmittelverarbeitung. Flexible Objekte stellen besondere Herausforderungen an die Wahrnehmung, Modellierung und Steuerung dar, welche in bisheriger Forschung kaum behandelt wurden. In neueren Ansätzen werden häufig Simulationen und gelernte Weltmodelle genutzt, diese Arbeit untersucht jedoch ob Offline Reinforcement Learning in diesem Bereich verwendet werden kann. Agenten werden mit einer dafür konzipierten Aufgabe trainiert, Befehle anhand von Bildern so zu wählen, dass ein Stück Teig in eine Zielform überführt wird. Allerdings scheint es unmachbar, üblich große Datensätze für Offline Reinforcement Learning Ansätze zu sammeln. Stattdessen werden begrenzt viele menschliche Demonstrationen gesammelt und erweitert, wofür konditionierte U-Nets und diffusionsbasierte Re-Projektion eingesetzt werden. Da dies hochwertige Demonstrationen voraussetzt, wurde eine gamifizierte Robotersteuerung entwickelt. Die Idee dahinter ist, die Nutzer spielerisch

zu motivieren sich engagiert mit der Aufgabe zu befassen. Die trainierten Agenten wurden in 8 verschiedenen Aufgaben evaluiert, welche sich auf die Schlüsselaktionen der eigentlichen Aufgaben fokussieren. Trotz keiner eindeutigen Korrelation zwischen den Datenerweiterungen und der Leistung in diesem Szenario, gibt es Hinweise darauf, dass eine Kombination der Datenerweiterungen zu vorsichtigerem Verhalten führen könnte. Darüber hinaus weist die qualitative Analyse auf verschiedene Schwierigkeiten hin, die nicht genug in den Trainingsdaten abgebildet sind, und somit noch zukünftig behandelt werden könnten.

List of abbreviations

AWR	advantage-weighted regression	8
CNN	convolutional neural network	9
CURL	Contrastive Unsupervised Representations for Reinforcement Learning	17
DMC	DeepMind control suite	16
DQN	deep Q-Learning	15
EMA	exponentially moving average	8
GNN	graph neural network	14
IOU	intersection over unit	23
IQL	implicit Q-Learning	7
LLM	large language model	15
MDP	Markov decision process	5
MLP	multilayer perceptron	29
MSE	mean squared error	7
OOD	out-of-distribution	7
PPO	proximal policy optimization	16
RL	reinforcement learning	5
ROS	Robot Operating System	21
SAC	soft actor-critic	16
SUS	system usability scale	37
ERM	empirical risk minimization	6
TD	temporal difference	7
NASA-TLX	NASA Task Load Index	37

Contents

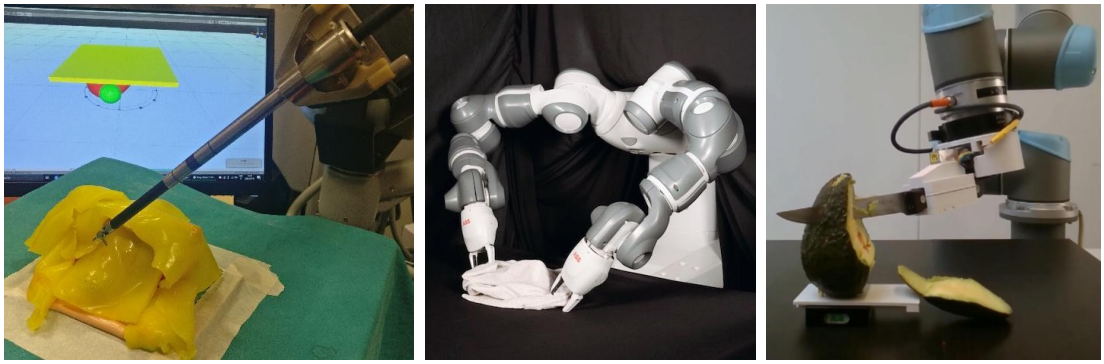
1. Introduction

In robotics, object manipulation is a crucial skill enabling robots to meaningfully interact with the physical world in order to achieve their goals. Existing research commonly assumes that the object is rigid [49] simplifying problems related to modelling, perception, or control. However, in the real world, this assumption does not hold for every object or material specifically for those where softness or non-rigidity is a primary trait including e.g. clothes, cables, plasticine, or silicone. Even jointed objects that are made up of rigid materials, or liquids fall into this category. Being able to manipulate such materials would allow for the use of robots in many additional applications. These include some in the fields of manufacturing, service and health care, gardening and agriculture, medical surgeries, and food processing [49, 45, 11] (see Figure 2.1). Despite the variety of deformable materials and their applications, this work will focus on the manipulation of elastoplastic materials such as plasticine, ground beef, or dough.

Dealing with non-rigid objects poses unique challenges that are not addressed by previous research. The inherently high dimensional state space presents a major difficulty as it makes deformable objects hard to perceive and to model [11]. Additionally, it increases the complexity of the dynamics which can vary between different materials. Recent advancements in deep reinforcement learning, learned world models, and differentiable physics simulators provide new perspectives on overcoming these problems. Hence, deformable object manipulation has grown in popularity in the past few years mostly utilizing data-driven methods. The goal of this thesis is to have a real robot solve a deformable object manipulation task (see section 4.2) by learning a pixel-to-action mapping from data only.

When working with robots directly, data-driven methods can be problematic as data collection on real hardware is usually costly in terms of time and effort. Therefore, simulations are popular in research as they can be used to e.g. train a reinforcement learning agent in a fraction of the time. But often the simulated performance cannot be achieved in the real environment due to the complexity and uncertainty of our physical world. Furthermore, in some scenarios, there might be no simulation available stressing the need for more data efficient methods. A recently developed paradigm is offline reinforcement learning where a policy is trained from using exclusively recorded experiences without any environmental interaction. Since this work refrains from using simulations, offline reinforcement learning presents an appealing choice to learn behaviours as costly policy training in the real world can be omitted. Instead, the data can be acquired independently of the learned agent allowing for the utilization of human demonstrations that are collected more safely and with a high quality.

In general, there is a growing demand for data acquisition used not only to learn decision-making engines. A recent, large-scale experiment utilized *gamification* to tackle a multiple alignment task concerning human ribosomal RNA sequences [34]. Gamification is a well-known phenomenon where game design elements are used in non-game contexts to motivate and increase user activity [8]. By designing the alignment task as a mini-game inside the pop-



(a) Robot Surgery [43]. (b) Garment-Folding [1] (c) Food-processing [44]

Figure 1.1.: Applications of deformable object manipulation.

ular mass-market game *Borderlands 3*, the authors collected over 135 million puzzle solutions from over 4 million players. Since this work also involves acquiring user demonstrations, the deformable object manipulation task will be gamified as well (see section 5.1), but with more simplistic game interface design patterns. However, due to the necessity of running the task on a real robot, it is foreseeable that only a few players can attend the gamified system.

To compensate for a low density in collected data distributions, a common technique is to transform existing data points into new samples which is known as data augmentation. Specifically for images, data augmentation techniques have been thoroughly applied in computer vision tasks. While traditional augmentations help to keep trained models from over-fitting, it is not clear whether they help agents to generalize in the presence of comparably few data samples. Hence, this work also aims to develop a learned world model producing advanced data augmentations that affect the context of the image samples (see section 5.2). Both, traditional and model-based augmentations will be compared regarding their effect on the agent's performance.

In summary, the goal of this thesis is to investigate the applicability of offline reinforcement learning to train an agent that autonomously solves a deformable object manipulation task. The training is implemented without the use of simulations but using robotic hardware directly. Human demonstrations are collected using a gamified version of the task and the collected dataset is enlarged with traditional and model-based data augmentations. The impact of the different augmentation techniques is compared by training multiple policies and measuring their performance in real-world tasks.

To fully understand the details of this thesis, chapter 2 explains common concepts and techniques used in related work and in later detailed methods. Chapter 3 takes a broader look at similar work in previous research regarding deformable object manipulation, pixel-based reinforcement learning, and offline reinforcement learning. In chapter 4, the overall robotic setup, the control implementation, and the deformable object task are detailed. The methods of this work are explained in chapter 5 which is divided into three parts. The first part describes the gamification of the setup, the second part is concerned with the augmentation techniques, and the third part explains the training of the policies. Chapter 6 goes into

detail about how gamification, the model-based augmentation, and the policies are evaluated and offers an explanation of the results. After that, chapter 7 discusses the results giving hypotheses for some of the findings. Lastly, chapter 8 briefly summarizes this work's most important aspects giving also an outlook on what could be improved in the future.

2. Fundamentals

This chapter briefly explains the fundamental concepts used for this work. Starting with the basics of reinforcement learning and offline RL, a concrete offline RL implementation is provided as it is used in this thesis. Then, contrastive learning is described as it is used for pre-training convolutional encoders and lastly, the workings of diffusion models are provided to better understand how the model-based augmentations function.

2.1. Reinforcement Learning

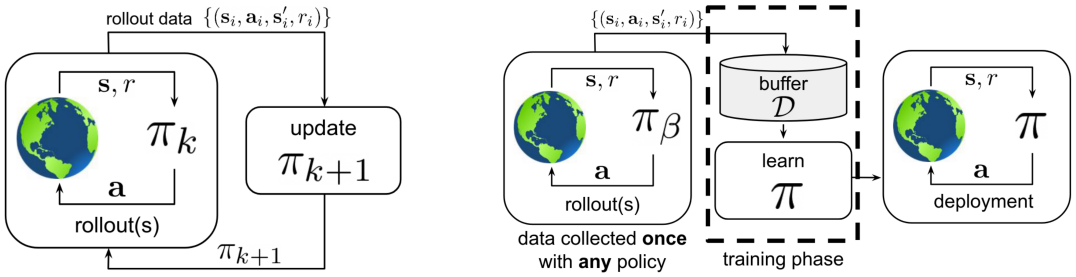
Reinforcement learning (RL) is a class of machine learning algorithms inspired by natural learning in humans and animals which is different from supervised and unsupervised learning. In essence, an RL agent repeatedly perceives some information about its environment and then performs an action interacting with it (see Figure 2.1a). Along the way, it tries to improve a numerical reward signal which evaluates the agent's behaviour.

Formally, reinforcement learning can be described as a Markov decision process (MDP) with states $s \in S$, actions $a \in A$, rewards $r \in R$, and transition probabilities $p(s'|s,a)$ [42]. The agent draws its actions from a policy π based on the current state. The objective of the agent is to learn a policy that maximizes the expected return $J(\pi)$ which is the sum of discounted future rewards:

$$J(\pi) = \sum_{t=0}^{H} \gamma^t r(s_t, a_t) \tag{2.1}$$

where $\gamma \in (0,1]$ is the discount factor, $r(s_t, a_t)$ returns the reward for choosing action a_t in state s_t, and H is the time horizon indicating the maximum number of subsequent steps. This means that *good* policies do not always choose actions greedily, yielding a high immediate reward, but also consider rewards that are to be received in the future.

Nonetheless, due to the nature of online learning, reinforcement learning is hard to apply in environments where policy interaction is either expensive (robotics, healthcare) or dangerous (autonomous driving, healthcare) [24]. While self-supervised setups have been proposed for learning rigid object manipulation tasks on real robots [23], doing so for elastoplastic material is much harder to realize as the setup would need to autonomously manipulate the object back into a desired start shape. Therefore, in this thesis, the focus is on offline RL where policy interaction with the environment is not required. For further reading on reinforcement learning, a good introduction on this topic was written by Richard S. Sutton and Andrew G. Barto [42].



(a) Online reinforcement learning (b) Offline reinforcement learning

Figure 2.1.: Visualization of online and offline reinforcement learning [24]. In online RL, an agent π_k directly interacts with its environment allowing for exploration to maximize rewards. In offline RL, the data collection part is separate from the learning process and can be done by any policy π_b. However, during the training phase, exploration is not possible anymore as the policy π learns from only the collected data.

2.2. Offline Reinforcement Learning

In offline reinforcement learning, the goal remains to find a policy that maximizes the expected return. Online RL agents can learn this objective by exploring different actions since these immediately affect the environment. However, an offline RL algorithm must do so, without any environmental interaction [24]. Instead, a static dataset $\mathcal{D}$ of past interactions $(s_t^i, a_t^i, s_{t+1}^i, r_t^i) \in \mathcal{D}$ with the MDP is provided allowing no further exploration (see Figure 2.1b). The behaviour policy π_b that generated the data may be unknown. It can be for example random sampling, human demonstrations, or a previously learned policy. What makes offline RL different from imitation learning, is the presence of the reward function. In imitation learning, the policy is trained to mimic the behaviour found in the demonstration data. Thus, the policy's performance is limited to the expert's performance. The reward function in offline RL enables the policy to learn more independent behaviours for example by avoiding actions that result in penalties. Generally, both imitation learning and offline RL allow for learning policies when many environmental interactions are costly or dangerous.

The difficulty of learning from only a static set of experiences is that the state/action distribution induced by π_b is different from the environmental distributions in non-trivial scenarios, yet the policy is tested against the latter. This is because the distributions cannot be assumed to be i.i.d as future states depend on the policy's past actions. Additionally, the policy should ideally be better than π_b meaning that it should learn something different from what is directly observable in the provided data. However, diverging too much can impose errors which cause the agent to end up in unseen states which unavoidably leads to more severe errors. This shift in distributions is well studied and one of the fundamental problems in both imitation learning and offline RL. Practically, this means that simple empirical risk minimization (ERM) algorithms, which are popular with supervised and unsupervised problems, are unsuitable. It has been shown that policies trained with ERM are at least quadratically more erroneous

with respect to the time horizon H than the behavioural policy from the dataset, even with optimal actions given [32]. Targeting these issues, researchers have come up with many different offline RL algorithms with different strengths and weaknesses.

For further reading on offline reinforcement learning, an introduction by Sergey Levine, Aviral Kumar, G. Tucker, and Justin Fu [24] can be referred to.

2.3. Implicit Q-Learning

Implicit Q-Learning (IQL) is an offline RL algorithm based on approximate dynamic programming [20]. It is used in this as the only offline RL algorithm which is why is it explained in detail. IQL minimizes a modified Temporal Difference loss and aims to approximate the policy improvement without evaluating out-of-distribution (OOD) actions. Temporal difference (TD) [42] is used to evaluate a policy by finding a state-value or action-value function $V(S_t)$. TD updates its value estimations using its own estimation for future states S_{t+1}:

$$V(S_t) \leftarrow V(S_t) + \alpha[r(s_{t+1}, a_{t+1}) + \gamma V(S_{t+1}) - V(S_t)] \tag{2.2}$$

This way, the trajectory does not need to finish in order to take gradient steps. The TD loss with respect to the action-value function parametrized by θ can be implemented with the mean squared error (MSE):

$$L(\theta) = \mathbb{E}_{(s,a,s',a')\sim\mathcal{D}}[(r(s,a) + \gamma Q_{\hat{\theta}}(s',a') - Q_\theta(s,a))^2] \tag{2.3}$$

where the difference between the predicted value of the current state-action $Q_\theta(s,a)$ and the immediate reward plus the discounted target network prediction of the next state-action $r(s,a) + \gamma Q_{\hat{\theta}}(s',a')$ is minimized. A drawback of using the MSE is that the action-value model will predict the mean return for a given state-action. Hence, it can occur that the presence of few high returns might be outweighed by many low returns for the same state-action. IQL incorporates expectile regression into the TD loss, meaning that the action-value function averages more towards optimism or pessimism depending on τ:

$$L_2^\tau(u) = |\tau - \mathbb{1}(u < 0)|u^2 \tag{2.4}$$

A visualization of the asymmetric expectile loss is given in Figure 2.2. Skewing the target values towards optimism (selecting $\tau > 0.5$) approximates maximum Q-values for a state over actions from the dataset:

$$L(\theta) = \mathbb{E}_{(s,a,s')\sim\mathcal{D}}[(r(s,a) + \gamma \max_{\substack{a'\in A \\ \text{s.t.}\pi_\beta(a'|s')>0}} Q_{\hat{\theta}}(s',a') - Q_\theta(s,a))^2] \tag{2.5}$$

which satisfies the Bellman equation used in dynamic programming approaches [42]. This allows for *stitching* parts from suboptimal trajectories in order to form a better one. This feature seems crucial to succeeding in the deformable object manipulation task in this thesis (see section 4.2) because it could allow for learning the effect of certain actions on the dough

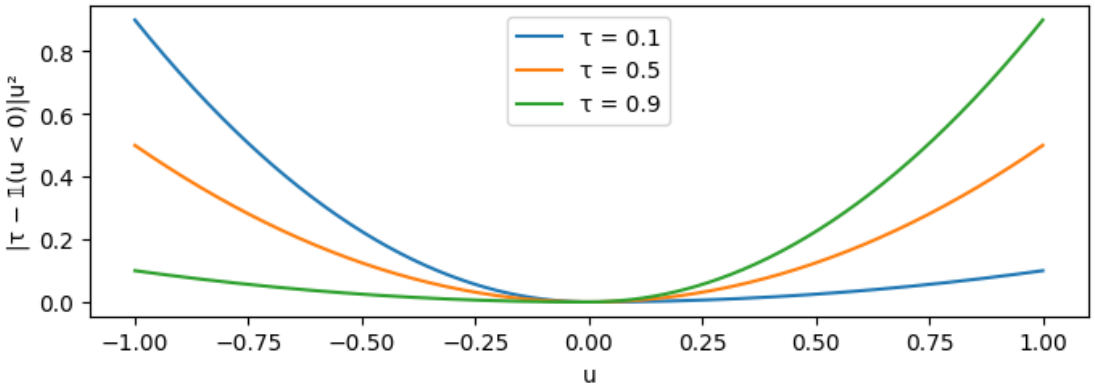


Figure 2.2.: The asymmetric expectile loss $L_2^\tau(u)$. Selecting $\tau = 0.5$ results in a symmetric L_2 loss. In IQL, choosing $\tau > 0.5$ penalizes pessimistic value predictions more than optimistic ones.

in order to produce an action sequence that leads to suitable deformations. The modified TD loss with expectile regression is defined as follows:

$$L(\theta) = \mathbb{E}_{(s,a,s',a')\sim\mathcal{D}}[L_2^\tau(r(s,a) + \gamma Q_{\hat{\theta}}(s',a') - Q_\theta(s,a))] \tag{2.6}$$

Compared to Equation 2.3 only the MSE was substituted with the expectile loss. A drawback of this loss is excessive optimism. This means, that single high values, that might only be encountered due to transition probabilities, have a large impact on the action-value model. To mitigate this issue, the expectile regression is only applied to the state-value function V so that the value integrates over all actions $a \sim \mathcal{D}$:

$$L_V(\psi) = \mathbb{E}_{(s,a)\sim\mathcal{D}}[L_2^\tau(Q_{\hat{\theta}}(s,a) - V_\psi(s))] \tag{2.7}$$

Using the MSE in the action-value model together with the state-value estimates $V_\psi(s')$ helps to reduce the excessive optimism:

$$L_Q(\theta) = \mathbb{E}_{(s,a,s')\sim\mathcal{D}}[(r(s,a) + \gamma V_\psi(s') - Q_\theta(s,a))^2] \tag{2.8}$$

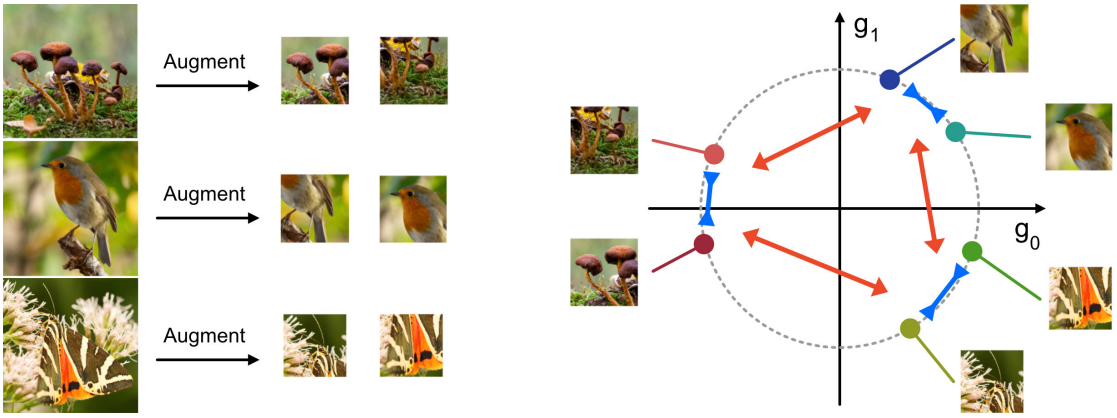
Since IQL relies on a target Q-network with soft updates, the parameters $\hat{\theta}$ need to be updated via exponentially moving average (EMA):

$$\hat{\theta} \leftarrow (1-\alpha)\hat{\theta} + \alpha\theta \tag{2.9}$$

Lastly, after fitting V_ψ, Q_θ, and $Q_{\hat{\theta}}$ the policy is extracted with advantage-weighted regression (AWR) [30] by minimizing the following loss:

$$L_\pi(\phi) = \mathbb{E}_{(s,a)\sim\mathcal{D}}[\exp(\beta(Q_{\hat{\theta}}(s,a) - V_\psi(s)))\log\pi_\phi(a|s)] \tag{2.10}$$

where $Q_{\hat{\theta}}(s,a) - V_\psi(s)$ represents the *advantage* which is exponentiated with the hyperparameter $\beta \in [0,\infty)$ as inverse temperature.



(a) Create positive pairs with data augmentation.

(b) Maximize agreement between positive samples.

Figure 2.3.: Visualization of contrastive representation learning. First, positive pairs are created with data augmentation (a). In this example, random-crop is used. Then, the agreement between positive samples is maximized using the contrastive loss (b). Here, g_0 and g_1 are example dimensions in the contrastive space.

2.4. Contrastive learning

Contrastive Learning is a type of representation learning that can be used in an unsupervised fashion. The idea is to use augmentations of data points to create similar samples, encoding them, and then maximise the agreement of the embeddings using the contrastive loss [5].

After sampling a minibatch x, stochastic data augmentation transformations are applied to obtain two positive samples x_i, x_j for each data sample resulting in $2N$ samples (see Figure 2.3a). When dealing with images, this can be e.g. random-resized-crop, random colour distortions, or adding random noise. The choice of augmentation depends on the context as not every method works equally well. The remaining pairs from the batch are considered negative samples. The contrastive loss is defined as

$$L_{i,j} = -\log \frac{\exp(\mathsf{sim}(z_i, z_j)/\beta)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\mathsf{sim}(z_i, z_k)/\beta)} \tag{2.11}$$

where z_i and z_j are pairwise encodings, sim is the cosine similarity, and β is a temperature parameter. A model trained with this loss encodes positive samples into similar embedding contrary to negative samples (see Figure 2.3b). However, using the encoder $f(\cdot)$ alone gives suboptimal results. Instead, the authors of [5] suggest adding a small non-linear MLP $g(\cdot)$ with one hidden layer after the encoder $z_k = g(f(x_k))$. After training, this projection head is discarded. Nevertheless, the embeddings need to be normalized otherwise they could be pushed apart arbitrarily. Similar to the work of Zhan et al. [48], contrastive representation learning is used to pre-train the convolutional neural networks (CNNs) in this thesis in an unsupervised manner.

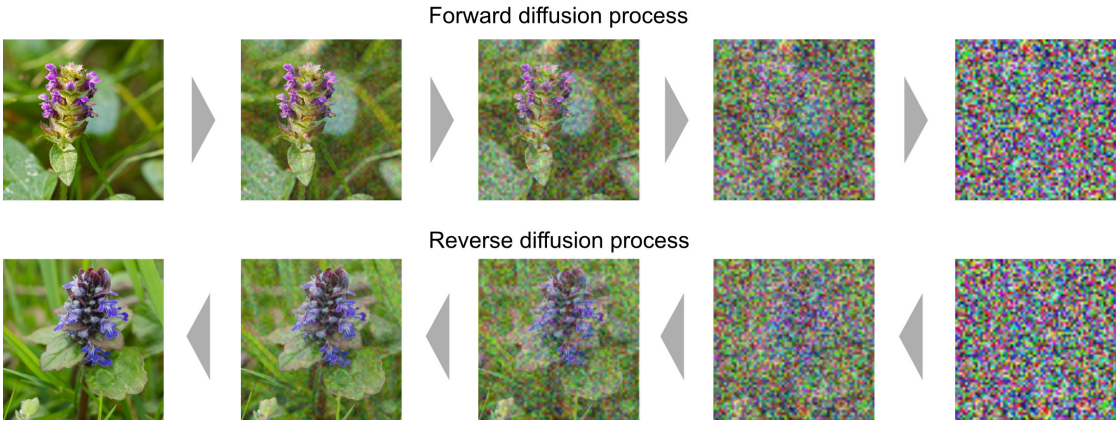


Figure 2.4.: Example of the diffusion processes. During the forward process, the image is stepwise destroyed with noise. During the reverse process, a model predicts the added noise which will be subtracted from the image generating new samples.

2.5. Diffusion Models

Denoising Probabilistic Diffusion models [17, 27] are a class of generative models where a model repeatedly denoises an input until it follows a learned distribution. The idea is to have a forward process destroy information of a sample x_0 and then restore this during the reverse process. During the forward process, Gaussian noise $\epsilon \sim \mathcal{N}(0,1)$ is added on x_0 T times. During the reverse process, the model $\epsilon_\theta(x_t, t)$ predicts ϵ to subtract it from x_t. This way, new images can be synthesized (see Figure 2.4).

In this work, a diffusion model is utilized to re-project a flawed sample back into its original distribution. To understand how that works, it is required to know how the t coefficient works: First, the loss function $L(\theta)$ is defined as:

$$L(\theta) = \mathbb{E}_{t,x_0,\epsilon}[\|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)\|^2] \tag{2.12}$$

where t is uniformly sampled from $1, ..., T$ and $\epsilon \sim \mathcal{N}(0,1)$. $\bar{\alpha}$ corresponds to a (non-) linear schedule determining how much noise is added at timestep t. As ϵ_θ predicts ϵ, no image synthesis is done during training. Overall, t determines the level of noise added during the forward process and gives that information to the network. Thus, ϵ_θ learns to predict ϵ given x_t with *any* level of noise.

The reverse diffusion process handles generating images by querying ϵ_θ T times to denoise x_t which initially is $x_T \sim \mathcal{N}(0,1)$:

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)) + \sqrt{\beta_t}z \tag{2.13}$$

where at each step, $z \sim \mathcal{N}(0,1)$ and β_t describes the forward process variances, in this case a t-depended constant, which is used to derive $\alpha_t := 1 - \beta_t$ and $\bar{\alpha}_t := \prod_{s=1}^{t} a_s$. These

constants simply ensure that the right amount of (predicted) noise is added or subtracted resulting in a synthesized, fully denoised image.

However, in this thesis, the objective is not to synthesize new images but rather *improve* flawed ones. To do so, the forward and reverse process is not repeated T times but for $K < T$ steps. This has the effect, that x_K is not fully noisy meaning that the image content is still present to some extent. During the reverse process, image details are restored pushing the sample back into the learned distribution. An example is given in Figure 2.5.

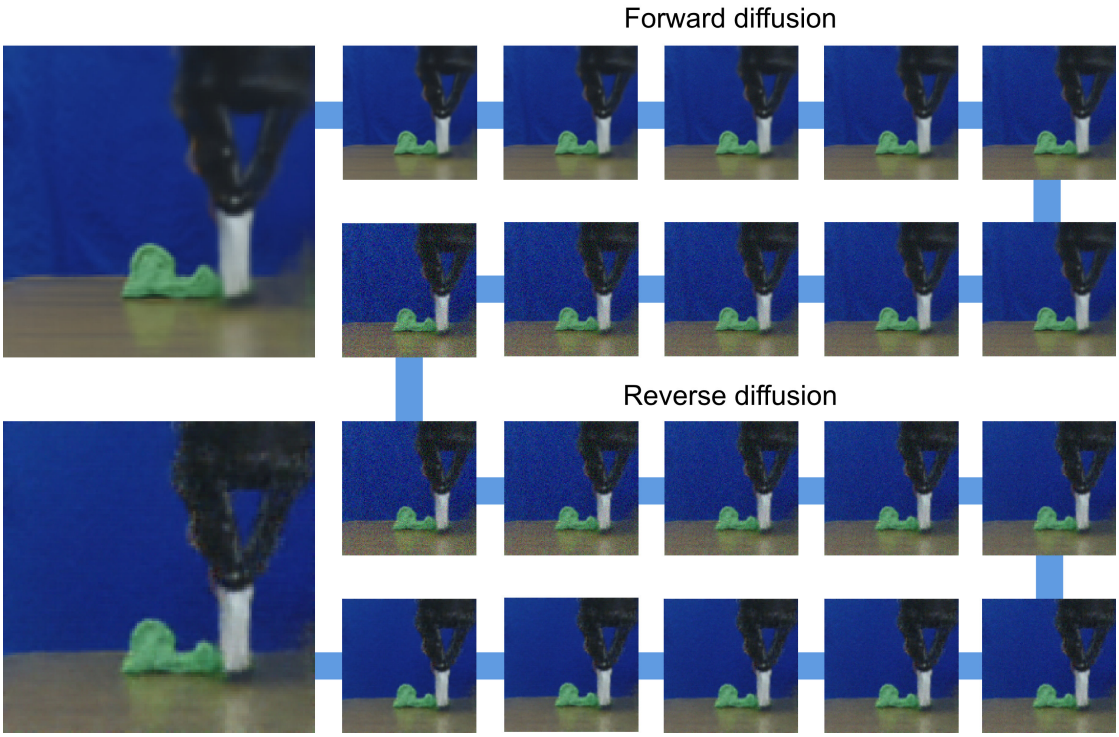


Figure 2.5.: Visualization of the re-projection. On a flawed image (top left), noise is added for $K < T$ steps (forward diffusion). A learned diffusion model denoises the sample (reverse diffusion) restoring some image of the details (bottom left). This prevents compounding errors when predicting longer trajectories.

3. Related Work

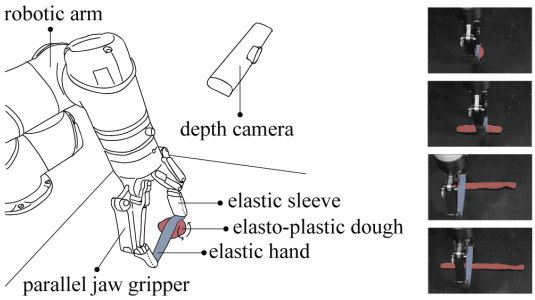
In this chapter, related work to this thesis is listed. It is divided into three sections: section 3.1 details other deformable object manipulation tasks and the corresponding approaches, section 3.2 features methods that aim to solve pixel-based reinforcement learning problems, and section 3.3 outlines research about offline reinforcement learning used in different contexts.

3.1. Deformable Object Manipulation

Cherubini et al. [6] use visual servoing to deform kinetic sand with push and pat actions (see Figure 3.1). Two heuristic methods and one learning approach were investigated for selecting the pushing action. With contour detection, the pushing path is determined using either the maximum distance of two contour points (maximum) or along the contours centroids (average). The learning approach used a three-layer MLP with user demonstrations collected from an earlier study. The metric consists of the mutual information error over the images. In the experiments, the three pushing methods were compared against each other doing three different shapes. Overall, the average strategy showed the lowest performance while the learning and the maximum approach were similar. Though the latter showed the overall best results in the long run.

Figure 3.1.: The shape servoing task from [6]. The left image depicts the setup with the robot actively moulding the material with the tool (blue). The top right image shows the initial shape of the sand while a goal shape is depicted in the bottom right image.

(a) In [29], three types of deformable materials are tested (left image). The task was to roll a piece of material into a target shape (right image).



(b) The setup of [25]. A tape-like hand is used to roll a piece of dough into a desired length. The deformation can be seen on the right.

Matl and Bajcsy [25] used model-based reinforcement learning to roll dough with an elastic end-effector into an elongated shape (see Figure 3.2b). The model was learned with random samples and then used to train a policy that performed better than heuristic and random methods. The dough shape is detected with a depth camera and represented by a bounding box and the highest point. The action space is defined as a 5D vector comprising the 3D position, the elastic band length, and the motion distance.

In the work of Ondras et al. [29], three different types of deformable material were deformed using a rolling pin mounted on a robotic arm (see Figure 3.2a). The scene was captured using an RGBD camera and a tactile sensor was attached to the arm, measuring the stiffness of the material. The task involves flattening plasticine, Play-Doh, and kinetic sand from a ball to fill a given circular shape. Given the target shape and the detected shape, a start and end point were determined with different conventional methods. Actions include flattening as well as shrinking and the performance is measured with the intersection over unit. Similarly, the goals in this thesis are 2D as well hence the IOU metric will also be used here.

In the system called RoboCraft by Shi et al. [38], the goal is to automatically manipulate dough with a two-finger gripper into an alphabet letter (see Figure 3.3). The setup captures the dough state with four RGBD cameras and reconstructs a mesh from the point cloud data to resample a fixed number of points from the geometry. A conditioned graph neural network (GNN), which was trained with 6000 random samples, is used as a model to predict the gripper actions' impact on the dough. Using the model, several approaches to select actions, including RL and gradient-based methods, were tested against each other. Due to the differentiability of the GNN, the gradient-based models yielded the best performance even compared to human actions. While RoboCraft processes the dough in 3D, the target shapes are actually 2D letters. Similarly, this thesis looks at 2D goal shapes but omits to process 3D data by restricting motion to only two axes. Hence, the setup could be simplified to only one RGB-camera allowing to utilize existing pixel-based (offline) RL implementations.

RoboCook by Shi et al. [37] is the follow-up version of RoboCraft which makes use of 15 different tools to make dumplings. The setup is similar to RoboCraft but one GNN is trained for each tool. The desired tool is selected based on PointNet and a policy is trained in a self-supervised manner.

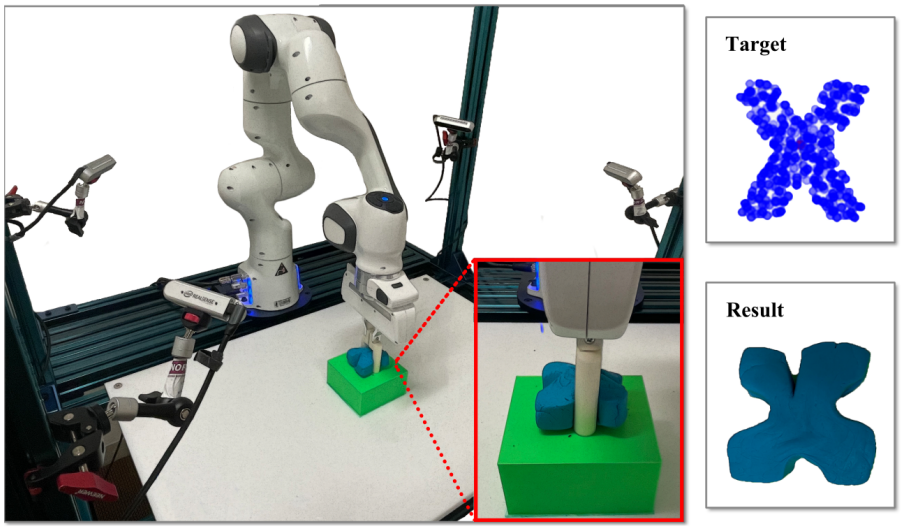


Figure 3.3.: The RoboCraft setup [38]. The dough is manipulated with a two-finger gripper to form a squared sough shape into alphabet letters. The system uses four RGBD cameras to reconstruct the dough shape in order to select actions.

You et al. [46] use a large language model (LLM) for zero-shot dough manipulation. By defining tools and their use for the LLM, it outputs a multi-step plan indicating what tool to use for the geometry of the intermediate point cloud via Python code. A differentiable physics module then predicts the actions given the tool, initial and target point cloud via gradient descent on the Earth Mover Distance. The experiments include making a doughnut, a baguette, and two pancakes multiple times.

Bartsch et al. [2] divide their point cloud into 64 clusters to use a pre-trained variational auto-encoder from Point-BERT to obtain discrete point tokens for each cluster. A simplistic physics-dynamics approximator predicts the positions of the 64 centroids which are then passed, along with the point tokens, into a dynamic graph CNN to obtain the predicted next states point tokens. To restore the predicted point cloud, the tokens are fed into the decoder from Point-BERT along with the predicted centroids. This dynamics pipeline is used with model predictive control to sample actions forming the dough into a target shape. The dynamics model was trained on random samples as well as human demonstrations with the overall performance being better when trained on the latter.

3.2. Pixel-based Reinforcement Learning

An early pixel-based reinforcement learning approach, namely deep Q-Learning (DQN), was presented by Mnih et al. [26] in 2013 to play atari games. A 2-convolution layer network was employed to approximate an optimal Q-function in an off-policy style. Since the action space is discrete, the probabilities for all possible actions for a given state could be estimated. DQN performed well on all seven tested atari games and even surpassed human performance on

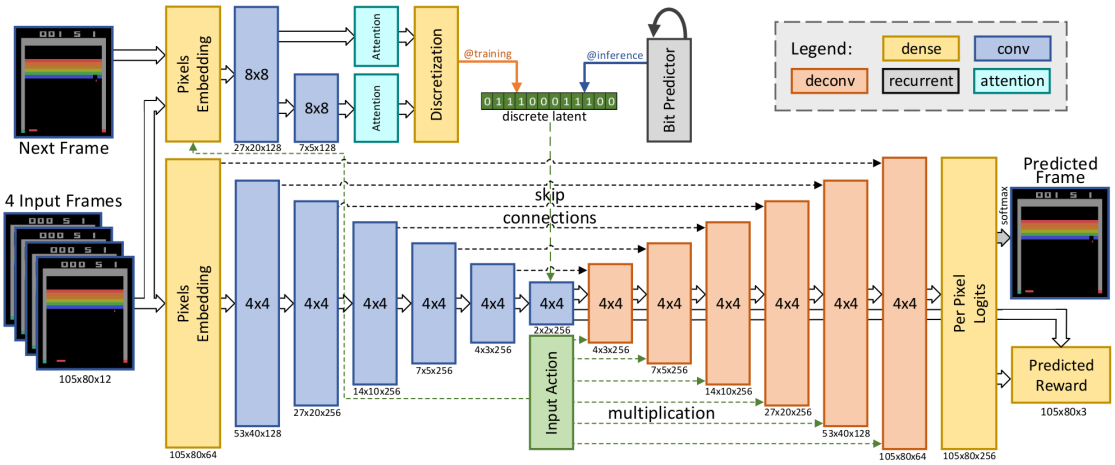


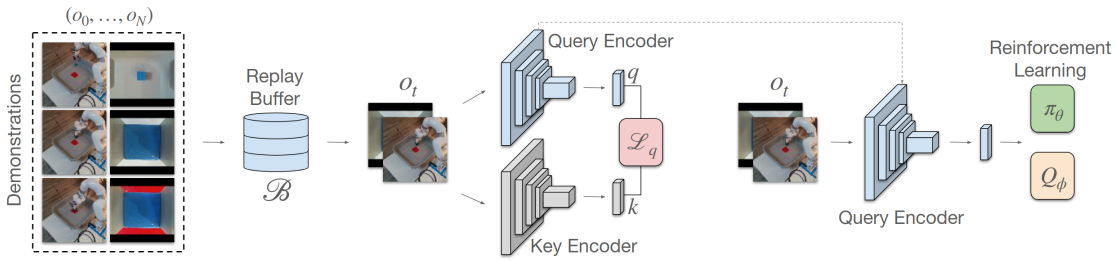
Figure 3.4.: The dynamics model approach from [18]. Roughly, the bottom part is a conditioned U-Net predicting next the frames similar to this work. The top part trains a recurrent inference network that produces a latent that will be, together with the input action, used to condition the U-Net.

three of them.

Since then, more different approaches have been explored. Hafner et al. [14] used a state latent dynamics model to train their policy. This requires an auto-encoder to compress the images into more compact representations. A recurrent state-space model is learned to predict the environment dynamics, including rewards and q-values, in the latent space. Experiments with the DeepMind control suite (DMC) have shown that the approach, dubbed Dreamer, outperforms off-policy algorithms specifically when the number of environment steps is limited. This thesis aims to learn a world model as well but cannot be trained on millions of transitions. To compensate for this, the actions used for state prediction are restricted to staying close to the data distribution relaxing the need for strong generalization capacities of the dynamics model.

Similarly, Kaiser et al. [18] trained a model to predict atari game dynamics with discrete latent states, though the model predicts observations directly (see Figure 3.4). Due to imperfections of the model, each trajectory was cut short to 50 steps. The policy was trained using proximal policy optimization (PPO) with 100K true environment steps and 15M simulated steps. The experiments were carried out on 26 atari games from the Atari Learning Environment. The dynamics architecture of this thesis is inspired a lot by the architecture from this work. Though, the inference network has been left out in favour of a more simplistic implementation.

A model-free approach to pixel-based reinforcement learning was proposed by Srivanas et al. [41] utilizing contrastive learning. The discrimination objective is based on random-crop data augmentation and two encoders transform the cropped images into representations. Only one encoder is trained jointly with the reinforcement learning task whereas the other one is updated using the EMA. The policy is trained with either soft actor-critic (SAC) or Rain-



(a) Collect 10 human demonstrations

(b) Initialize CNN encoder with contrastive pre-training

(c) Continue training with data-augmented RL

Figure 3.5.: The CoDER architecture [48]. It consists of three steps to quickly learn control tasks with RL on a real robot. Around 10 demonstrations are collected (a) which are used to train a CNN encoder via contrastive learning (b). Lastly, the pre-trained encoder is used to train an online RL agent utilizing random-crop image augmentations (c).

bowDQN using the learned representations. The approach, namely Contrastive Unsupervised Representations for Reinforcement Learning (CURL), was tested on the DMC as well as the same 26 atari games from [18].

Concentrating on image augmentations, Laskin et al. [22] used SAC and PPO without altering the loss function outperforming Dreamer and CURL on the DMC. In detail, they have shown how different image augmentation techniques, and their combinations, affect the trained policies. Specifically, random-crop alone has shown to be most effective as it increases the agent's resilience towards pixel translations.

Combining contrastive pre-training and data augmentation for visual robotic control, Zhan et al. [48] learned control policies on real robots in less than an hour of real-world training time. Building upon the works of [41] and [22], the architecture named CoDER (see Figure 3.5) requires around 10 human demonstrations for a given task. These are used to pre-train a CNN-based key and query encoder via contrastive learning and EMA respectively. The key-encoder is then used to train a SAC agent utilizing not only collected data by the policy but also the human demonstrations. Throughout the pre-training and the online training, random-crop augmentations are utilized. The CoDER architecture can be seen as a basis for this thesis, though transferred into the offline RL setting but with more demonstrations as the deformable object task is inherently more complex.

3.3. Offline Reinforcement Learning

Data augmentation in offline reinforcement learning was investigated by Sinha et al. [40]. Working with virtual environments only, their augmentations affect the true underlying state, e.g., the robot's simulated joint positions/velocities, and improve the smoothness of the Q-function and policy performance on several virtual benchmarks. The authors tested their approach on D4RL [9], Meta-World [47], and robosuite [50]. The datasets were either provided or came from a learned SAC agent. While S4RL could sometimes even outperform the

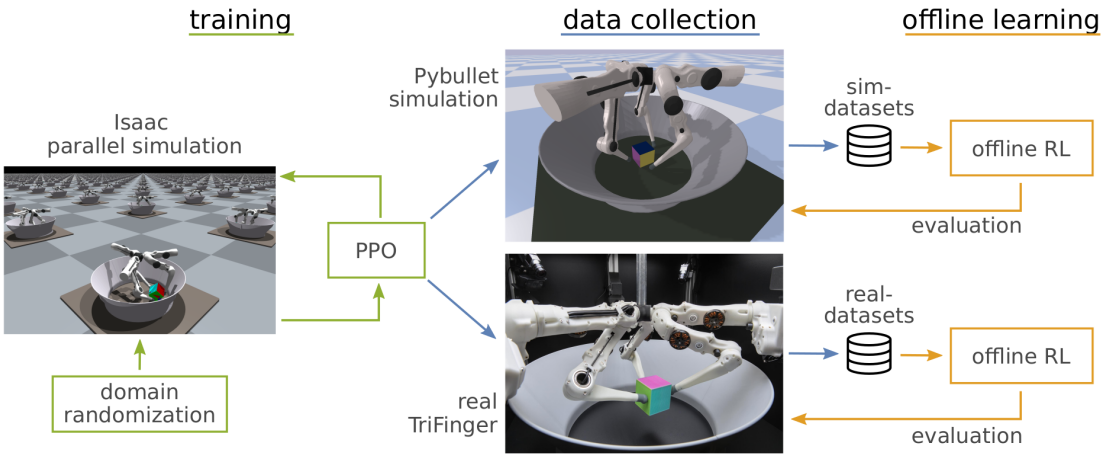


Figure 3.6.: The procedure of benchmarking offline RL. A PPO agent is trained in simulation on the two tasks. With zero-shot transfer, the trained policy is used to infer actions in a simulation and on the real platform generating two datasets. The resulting datasets are used to benchmark five offline RL algorithms that are evaluated in the respective environments.

behaviour policy, in this thesis, directly perturbing the states is impossible as only real-world data is collected. However, by learning the dynamics model, trajectories can be recomputed with slightly altered actions similar to the zero-mean Gaussian noise augmentations from S4RL ideally causing similar effects.

Seno and Imai [36] developed d3rlpy, an offline RL benchmark offering tested implementations for many popular offline RL algorithms that can be trained not only offline but fine-tuned online as well. The algorithms interface with the `MDPDataset` class during offline training or with environments from OpenAI Gym [3] during online training. The plug-and-play API allows for customizing the experiment e.g. by providing custom encoders. This way, complex or pre-trained encoders can be utilized in order to assemble derivates of algorithms. This is a vital feature for this thesis, as the encoders are pre-trained with contrastive learning.

Gürtler et al. [13] proposed a benchmark for offline reinforcement learning on real hardware using a three-finger robot and two cube-handling tasks. On the TriFinger platform from the Real Robot Challenge 2022 [12], an expert policy was trained in simulation using an already developed, specialized version of PPO. With the trained policy, a simulation dataset and a real-world dataset were created by running zero-shot inference in the respective environments (see Figure 3.6). Using d3rlpy [36], five popular offline RL algorithms were compared regarding their performance on the two tasks in the simulated and real domains. Generally, the performance in the real world is worse than in simulation for most algorithms. Additionally, a systematic hyperparameter optimization for all algorithms was performed which is of interest for this thesis as the optimized hyperparameters for IQL can be adopted.

4. Setup and Task

This chapter outlines the robotic setup and the deformable object task developed for this work. Details about the control implementation as well as the workings of the task are provided.

4.1. Robotic Setup

The teleoperation setup consist of a wall-mounted UR5 robot equipped with a Robotiq 3-finger hand (see Figure 4.1). Additionally, three custom fingertips were 3D printed to prevent the dough from damaging the hardware (see Figure 4.2). The fingertips are designed in a way that, when the gripper closes, the three parts align forming one tool that can be used to push or dent the dough. When the gripper is opened, the fingers offer a large surface area in between, allowing for gripping or squeezing the dough. Examples of how the fingers can be utilized are provided in the appendix (section B, Figure 1). The dough is placed below the gripper on a table. To increase friction with the table surface, a foil is affixed preventing the dough from slipping easily when pushing it from the side. One RGB camera with a resolution of 1280x720 is placed in front of the gripper, providing side view of the scene. A schematic overview of the setup components is presented in Figure 4.3

4.1.1. Control

The idea is to move the gripper only along the y- and z-axis relative to the base-link (see Figure 4.3, green/blue arrows). Additionally, the gripper can be rotated around its z-axis and can be opened and closed. Though, it is a 3-finger gripper, the two outer fingers are permanently in pinch mode meaning they act as one larger finger when opening or closing the gripper. The camera provides the primary observation source for the user. The control signals can come from either humans or policies. In this chapter, only the technical control interface is mentioned. The human interface will later be explained in section 5.1.

Robot commands should be intuitive for the user which is why the translation commands are defined relative to the camera image. This means that normalized pixel coordinates $\langle x, y \rangle$, with values in $[0, 1]$, are used to determine the end-effector position in Cartesian space. These need to be transformed to $\langle y, z \rangle$ coordinates relative to the base-link. The axis differences arise due to the different frame orientations visible in Figure 4.3: While the end-effector's y-axis points towards the base-link, the same happens for the camera's x-axis. This might lead to confusion but generally, $\langle x, y \rangle$ is used in the context of input commands while $\langle y, z \rangle$ is only used to compute the 3D end-effector position.

The grippers orientation is restricted to the three intuitive modes $-90°$, $0°$, and $+90°$ which is expressed by $\gamma \in \{-1, 0, 1\}$ respectively. Lastly, the opening width is determined by

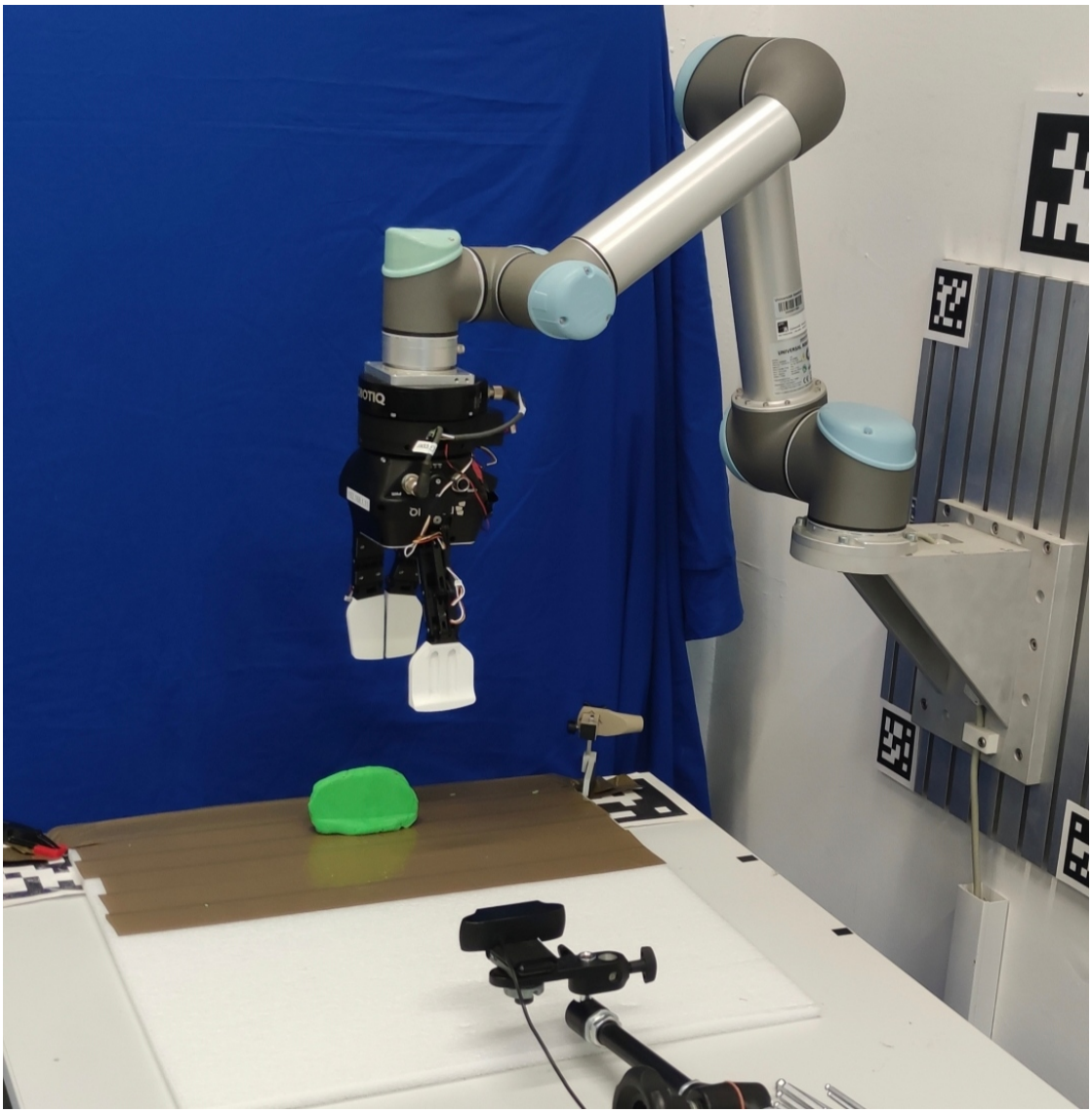
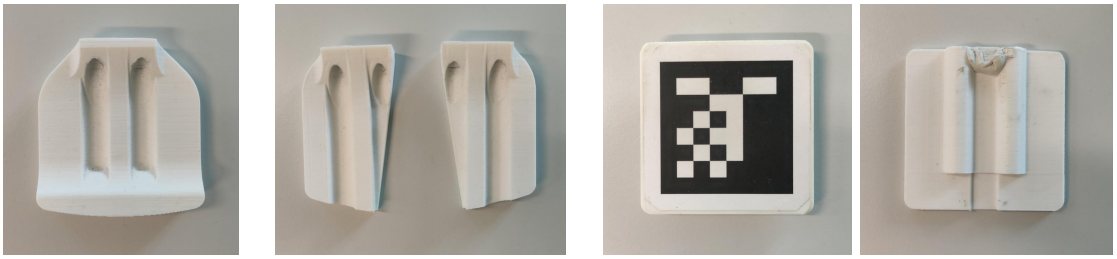


Figure 4.1.: The setup consists of a UR5 equipped with a Robotiq 3-finger gripper. Custom fingertips were 3D-printed to avoid hardware damage and to facilitate better dough manipulation for the given tasks. While the material mostly prevents sticking to the dough, a foil is fixed to the working surface minimizing side-slipping of the dough due to the increased friction. The RGB-camera is placed pointing at the scene with the end-effector being restricted to only move on the orthogonal axis. A blue curtain serves as a background with few distractions. The colour of the dough (green) is chosen to easily detect the contours.



(a) (b) (c)

Figure 4.2.: Custom 3D printed fingertips and AprilTag [28] extension. The fingertip for the middle finger is depicted in (a) having a thicker bottom side to have more surface that could make dents in the dough. The fingertips for the outer finger (b) are designed so that when those fingers pinch, the two parts construct a bigger fingertip similar to the middle one. The fingertips are also used to mount the AprilTag for calibration (c). The front side (c, left) shows the tag itself while the back side (c, right) has a convex relief allowing for quick attachment on the middle fingertip with a putty-like adhesive.

d with values in $[0, 1]$ where 0 means fully closed and 1 fully opened. In summary, this leads to a 4-valued control interface $\langle x, y, \gamma, d \rangle$:

Parameter	Values	Description
x	$0 \leq x \leq 1$	End-effector position along x-axis
y	$0 \leq y \leq 1$	End-effector position along y-axis
γ	$\gamma \in \{-1, 0, 1\}$	End-effector orientation
d	$0 \leq d \leq 1$	Gripper opening width

The control implementation is realized using the Robot Operating System (ROS) [31]. A ROS-node was designed to receive the control commands $\langle x, y, \gamma, d \rangle$ from either policies or the human interface. In the RL context this can be referred to as the *action*. For the actual motor control, the high-level motion planning framework *MoveIt* [7] is leveraged.

First, the mapping from the $\langle x, y \rangle$ input coordinates to the Cartesian end-effector coordinates on the $\langle y, z \rangle$ plane is described. This mapping can be described as a line-plane intersection where the plane is defined by the y/z-axis of the base-link while the line is defined by every point in Cartesian space that would be projected onto the image coordinates $\langle x, y \rangle$. However, this requires the transform from the base-link to the camera (BC) to be known (see Figure 4.3, BC). Effectively, only the transform from the base-link to the end-effector (BA) is known as it can be computed with forward kinematics. In order to estimate BC, an AprilTag [28] is attached to the end-effector (see Figure 4.2c). This allows for an estimated transform from the camera to the AprilTag. Since the transform from the AprilTag to the end-effector is known, the transform from the camera to the end-effector (CA) can be inferred. Knowing CA and BA, BC can finally be computed with $BC = BA - CA$.

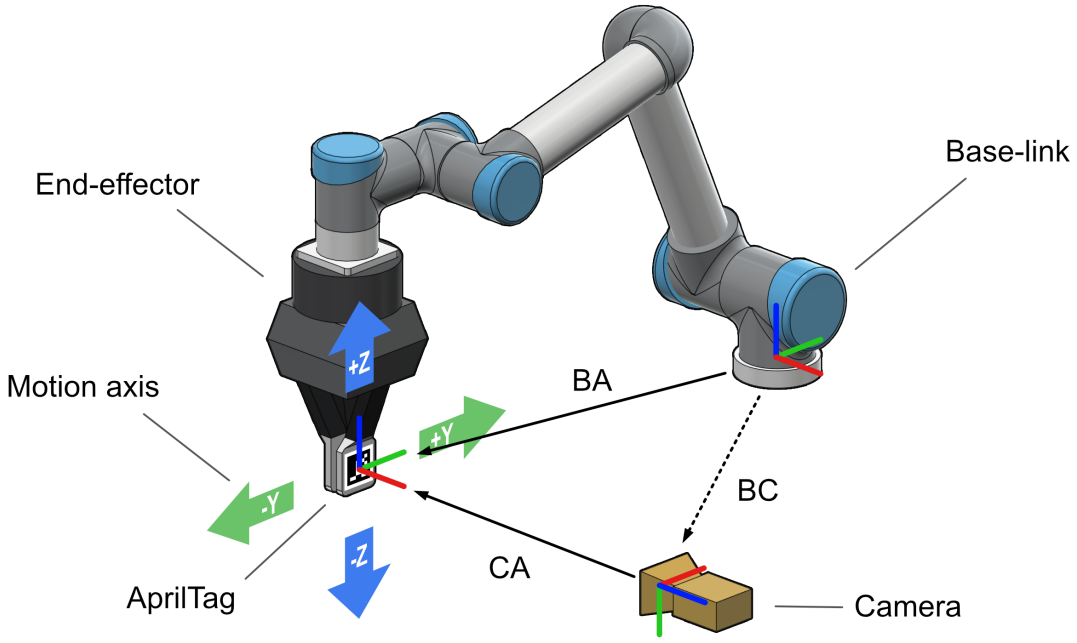


Figure 4.3.: Schematic overview of the robot control. The position of the camera relative to the base-link (BC) has to be determined. An AprilTag can be mounted to the end-effector which allows for the estimation of CA. BA can computed with forward kinematics providing all transforms necessary to estimate BC. Once it is known, the robot can be controlled to only move along the y/z-axis of the base-link (large, green/blue arrows) given 2D camera-image coordinates.

Given the cameras transform relative to the base-link along with the command coordinates $\langle x, y \rangle$, a ray can be computed with the `PinholeCameraModel` class that goes through $\langle x, y \rangle$. The y,z-plane intersects with the base-link allowing to compute the intersection with the ray. The intersection lies on the y,z-plane and aligns with the $\langle x, y \rangle$ coordinates on the image and effectively determines the end-effector position.

To move the end-effector with MoveIt, the new end-effector pose needs to be submitted to an instantiated `MoveGroupCommander`. The framework then takes care of planning and executing the trajectory. With the target end-effector position available, only the orientation has to be determined. Here, the γ parameter from the command is used to set the right angle around the z-axis. The angles for the remaining axis are constant as the end-effector should always face down. Lastly, the gripper's opening width needs to be adjusted based on d. The gripper state can be directly set using the `Robotiq3FGripperRobotOutput` class to send a message to the respective node. As the `MoveGroupCommander` is instructed at the same time, all robot motion happens simultaneously.

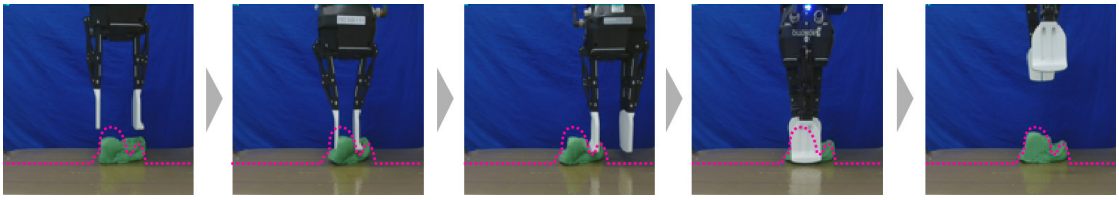


Figure 4.4.: The deformable object manipulation task. The goal is to provide a sequence of actions that deform the dough (green) into the target shape (magenta dashed line) using the robot.

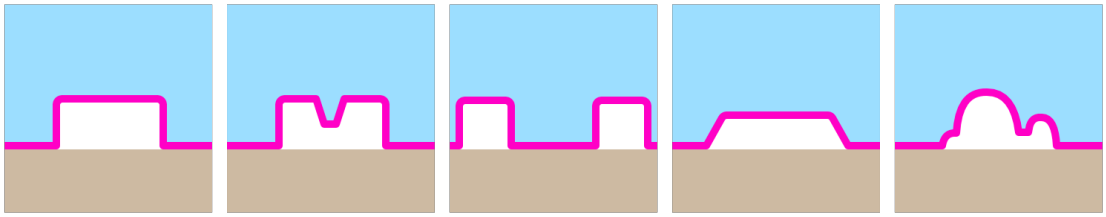


Figure 4.5.: The target shapes of the different tasks. The goal is to manipulate the dough to match the white area as closely as possible. Ideally, the tasks are completed in order as the shapes build up on each other.

4.2. Deformable Object Manipulation Task

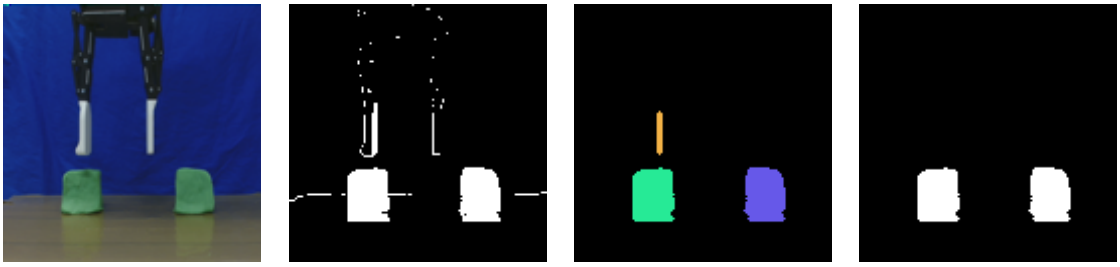
Since this work focuses on the manipulation of elastoplastic objects, the choice of material has to be carefully made as it will affect the agents training and behaviour. In prior research, different materials like kinetic sand [29], plasticine [29], and dough/Play-Doh [38, 25, 29] were explored. Play-Doh appears to be the most often used material which is more soft compared to plasticine while not drying out too fast making it a good choice for this work. Throughout the entire time, the dough was regularly moistened to maintain a steady softness.

The deformable object manipulation task of this work consists of forming a piece of dough into a given target shape by providing a sequence of actions (see Figure 4.4). A total of five shapes are defined (see Figure 4.5) which are ideally completed in order. There is no strictly defined start state. The better the shape matches, the higher will be the reward.

The goal shape is provided as a 1D vector defining the relief with length equal to the observed image's width. Hence, it varies depending on whether the human interface is used or a policy processes a state. To make the task less repetitive, the goal shape is randomized along the x-axis while always staying between well-defined bounds.

4.2.1. Reward

To measure how much the dough matches the shape, the intersection over unit (IOU) is used. This requires the target shape to be an image as well which is not immediately the case. Instead, the 1D vector is rasterized onto an otherwise black image with a constant



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

Figure 4.6.: Dough masking process used to determine rewards. Given a regular observation (a), a colour threshold in the HSV space is applied (b). Erosion and dilation are used to remove noise and connected regions are labelled (c). If a region fails to have a certain amount of pixels, it is deleted (d).

y-offset. Furthermore, the dough has to be masked in the observation image. The masking process is shown in Figure 4.6. The image is first converted into the HSV colour space to apply simple colour thresholding (see Figure 4.6b). Using erosion and dilation, smaller false positives are eliminated leaving only larger regions (see Figure 4.6c). The area of all regions is measured in order to eliminate the ones that fall below a certain threshold (see Figure 4.6d). A mask remains that can be used to compute the IOU together with the rasterized goal shape:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{4.1}$$

where A and B are both masks. Therefore the reward values will be in range $[0, 1]$.

5. Methods

This chapter looks at the methods used in this work to obtain policies, that are compared later concerning performance. Overall, it is split up into three sub-components:

1. The **gamification of the setup** concerns the implementation of the user interface for the collection of human demonstrations.
2. The **data augmentations** aim to meaningfully increase the amount of collected data without any further environmental interaction.
3. The **policy training** uses the produced data in order to train a policy with the offline reinforcement learning algorithm IQL.

5.1. Gamification of the Setup

A gamified user interface for the teleoperation setup described in chapter 4 is developed to motivate users to put in an effort during the demonstrations (see Figure **??**). The objective remains to manipulate the dough into a target shape, with a total of 5 goals to achieve. Additionally, there is a free mode that allows users to practice without any specific goal. Users define actions by configuring the orientation γ and gripper opening width d of the robot and then clicking on the image to specify the end-effector position $\langle x, y \rangle$. The mouse click ultimately commands the action to the teleoperation setup. A running score is displayed during the game to inform the users about their current performance. Once satisfied with their score, users can take on the next goal, adding the current score to the total score. Finally, the total score is used to compile a leader-board. Each state/action pair is stored and utilized to create a dataset for offline training.

5.1.1. Visual Elements

The top bar shows four elements: The current turn, the running score, the total score, and a menu button. The turn increases with each action giving the user feedback on how many commands were issued already for the current goal. The running score updates every second recomputing the reward each time. The total score updates when the user proceeds to the next goal and the menu button opens the menu.

In the centre, the camera image is presented depicting the scene. A yellow-magenta line shows the current goal. The image is cropped to a resolution of 960x720 as 1280x720 would be too wide. During reward computation (see subsection 4.2.1), the image is scaled down to 128x128 to match the conditions during policy training which uses the same image size.

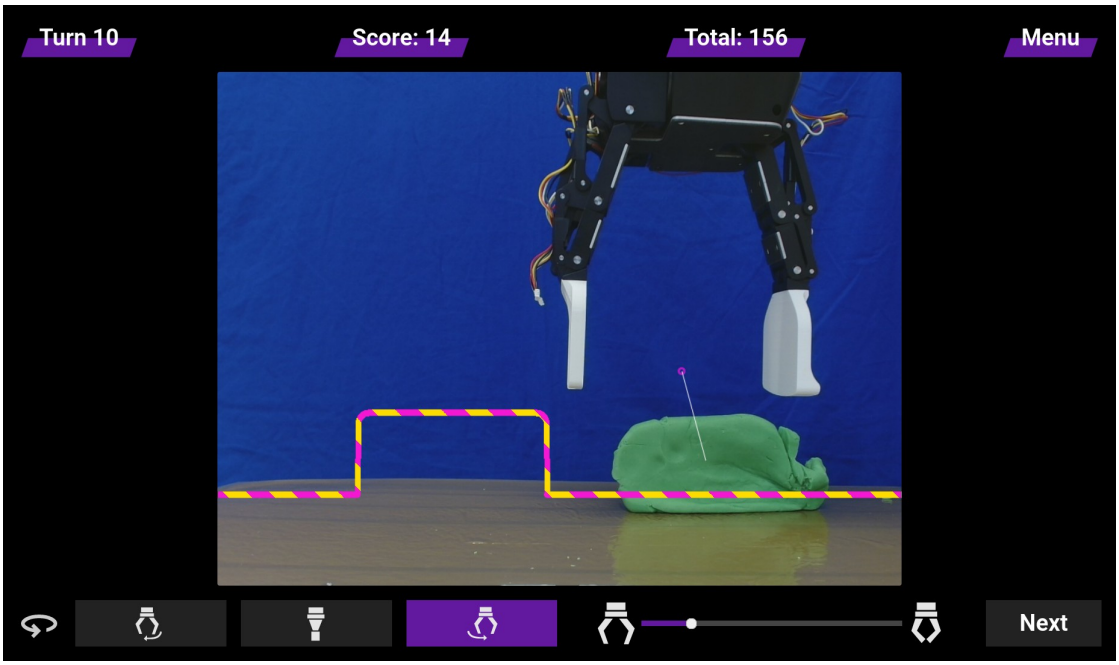


Figure 5.1.: The gamified user interface of the teleoperation system. The controls are located at the bottom while a task score and a total score are displayed at the top. The robot can be moved by clicking inside the camera image after adjusting the orientation with the three bottom-left buttons and the opening width with the bottom-right slider. Once the user is satisfied, clicking *Next* will switch to the next of the 5 tasks.

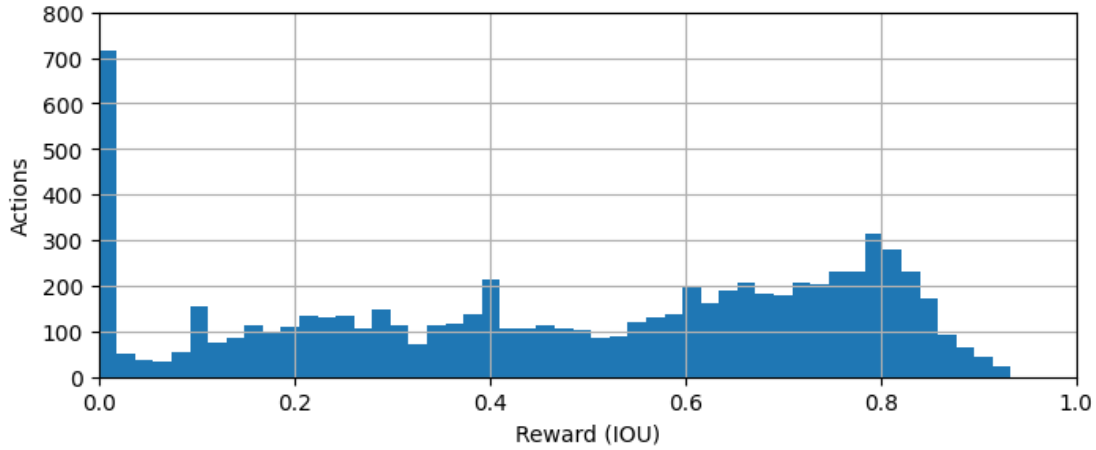


Figure 5.2.: Reward distribution of the collected actions. The majority of rewards is 0 as every new goal is placed randomly causing a reset in the reward as well.

When the user clicks somewhere on the image, the selected $\langle x, y \rangle$ coordinates are used to issue a command which moves the robot. To simplify control for the user, the last issued position is indicated by a purple circle connected with a white line to the current mouse cursor position. This line indicates the path the end-effector will move.

At the bottom, the controls for orientation as well as opening width are displayed including the button for the next goal. While for the orientation, one of the three states can be selected, the opening width is controlled with a slider. Changing the state with the bottom controls does not immediately affect the robot. Instead, the robot's new state is computed only when the user clicks on the camera image, assembling a command with the image coordinates along with the preferences from the bottom controls. After that, the command is forwarded to the control ultimately moving the robot.

5.1.2. Dataset Analysis

Overall, 12 people attempted the game playing for around 30-45 min per session. As a result, a total of 7250 actions were collected in the process. Since the policies will be trained by this data without further environmental interaction, the recorded rewards present an indicator of how well the learned policies could perform.

The reward distribution is shown in Figure 5.2. It can be seen that 10% of all rewards have a value of close to 0. This is probably due to the random x-offset that replaces the goal each time a new goal is presented. Other than that, there is a slow increase towards 0.8 forming the second highest number of rewards. This could indicate that most users aimed to increase their running score to a certain level before continuing with the next task. After the 0.8, rewards fall rapidly due to the complexity of shaping the dough *pixel-perfect*. No one achieved a perfect match however, as the reward is computed from observations as well, it can be assumed that this was likely due to imperfect dough masking.

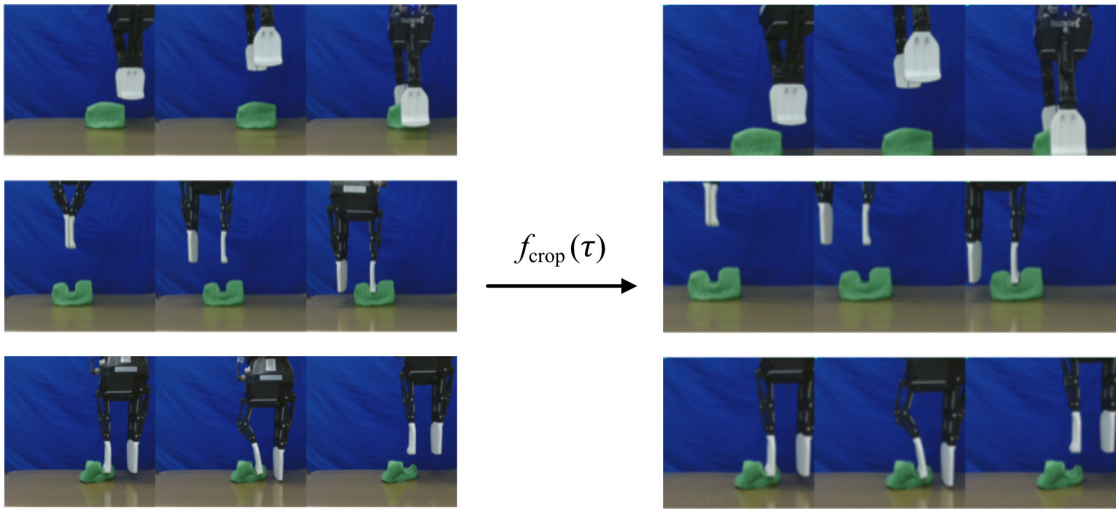


Figure 5.3.: Three, with f_{crop} augmented trajectories. The original samples are displayed on the left, and the augmented ones on the right. The randomized parameters $\langle x, y, size \rangle$ are kept constant for a given trajectory to preserve translational information.

5.1.3. Action and Reward Pre-processing

So far, the recorded data contains *absolute* actions and rewards. This means that the same action will put the robot into the same configuration independent of its current state. Rewards are still defined by the raw IOU. However, during this thesis, it was empirically found that *relative* actions and rewards allowed for better training. Therefore, in the following sections relative values are assumed unless stated otherwise. The rewards now indicate an improvement compared to the last state if positive, or a deterioration if the value is negative. The value range of the actions slightly differs from before (see subsection 4.1.1):

Parameter	Values	Description
x	$-1 \leq x \leq 1$	End-effector translation along x-axis
y	$-1 \leq y \leq 1$	End-effector translation along y-axis
γ	$\gamma \in \{-1, 0, 1\}$	Relative end-effector orientation
d	$-1 \leq d \leq 1$	Relative gripper opening width

5.2. Data Augmentations

Data augmentation has been shown to improve performance in image-based online reinforcement learning [22, 21]. In this work, two types of data augmentations are explored: traditional resize-crop and model-based augmentations. Each type is used to increase the number of samples in the buffer by up to 10x as much. In this work, a transformation f is applied to

trajectories τ yielding randomly augmented versions $\hat{\tau}$:

$$f : \mathcal{D} \times \mathcal{T} \to \mathcal{D} \tag{5.1}$$

where $\mathcal{T}$ is the parameter space for the respective transformation. Since augmentations can alter states, actions and rewards it is important to keep the parameters consistent over the trajectory.

5.2.1. Traditional Augmentations

Traditional data augmentation methods regarding images have been thoroughly studied [39]. Regarding pixel-based reinforcement learning, traditional image augmentations help in terms of learning efficiency and policy performance. Specifically, random-resize-crop is effective for certain tasks where pixel translation invariance helps to generalize to the task [22, 21]. Since the robot moves along the image axes in this work, random cropping could lead to similar improvements with learning, though prior experiments have been mostly done in simulation indicating there might still be a gap compared to this work.

Nevertheless, the first of two augmentations f_{crop} is chosen to perform random-resize-crop on the trajectories. It is important to point out, that the randomly chosen parameters stay constant for one trajectory as shown in Figure 5.3. Otherwise, task-related translation information would be destroyed. Additionally, since the relative translation of an action $\langle x, y \rangle$ is normalized to image coordinates, a crop will skew the length of this translation. Hence, it has to be scaled accordingly: Given a random scale $u \in [0.5, 1.0]$ then the adjusted relative translation $\hat{a}_{xy} = \frac{a_{xy}}{u}$. The reward stays unchanged as it is assumed the dough shape is not altered by the crop even if it might be cut off.

5.2.2. Model-based Augmentations

With random-resize-crop, the depicted scene remains unchanged, only the visuals are affected. Due to the small buffer size, image augmentations changing the context might enable significantly better offline learning.

To acquire context alterations, the recorded data is utilized to train a dynamics model U_θ predicting the next image state given the current one in combination with a desired action. A conditional U-Net with skip connections is designed similarly to [18] (see Figure 5.4). The network consists of ResNet blocks [16] operating on the images combined with strided convolutional layers. To condition the feature layers, the action is position-encoded and passed to a multilayer perceptron (MLP). The number of output neurons matches the number of features of the corresponding layer so that each feature map is multiplied by one output neuron. Skip connections between the feature layers allow for faster training. Overall, the network consists of around 4M parameters.

The objective is simply to minimize the MSE between the next state s' and the predicted state $U_\theta(s, a)$:

$$L_{\text{dyna}}(\theta) = \mathbb{E}_{(s,a,s') \sim \mathcal{D}}[(s' - U_\theta(s, a))^2] \tag{5.2}$$

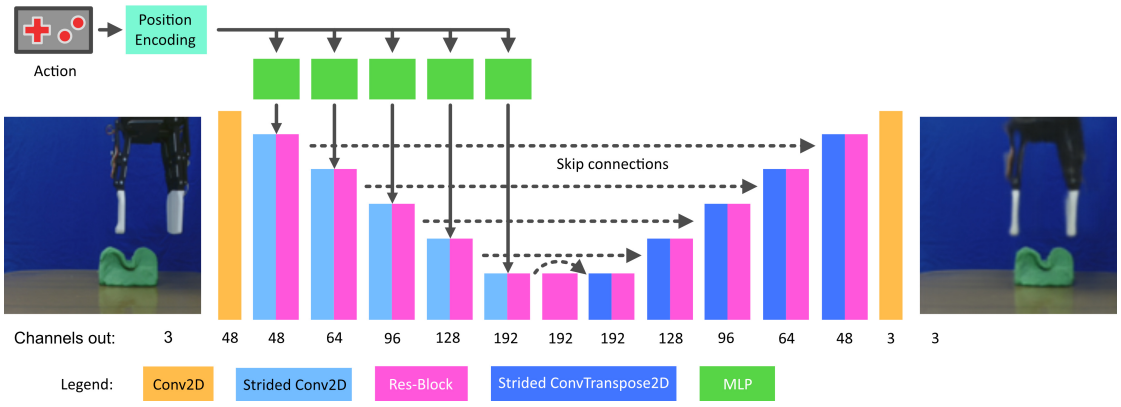


Figure 5.4.: Visualization of the Conditional U-Net architecture. The input is an image and an action. The output is an image only. The action is first transformed with sinusoidal positional encoding and then forwarded into 2-layer MLPs. The MLPs outputs condition the feature maps with scalar multiplication. Skip connections connect the convolutional encoding blocks with the decoding blocks to accelerate the training process.

Dynamics model training

The training data was split into 20 test episodes and 96 training episodes corresponding to 1044 and 6206 samples respectively. The model was trained for 1000 epochs with a batch size of 32 resulting in approximately 193000 gradient updates. To increase robustness, the random-resize-crop augmentation was applied to the transition analogous to subsection 5.2.1. Overall, training U_θ took around 2 hours on a NVIDIA RTX 4090. The training loss converged at around 1000 episodes while the test loss converged after 300 episodes already (see Figure 5.5). The losses diverge likely due to the difficulty of predicting details in unseen test samples whereas for the training sample, the details can slowly be memorized by the network.

After the training, U_θ can be used to predict sequences. An example is given in Figure 5.6: A trajectory with 11 actions is predicted entirely given only s_0. However, compounding errors deteriorate the visual quality with increasing t. At step $t = 11$, several visual errors can be identified: The left fingertip is divided, the end-effector's appearance turned round and smooth, and white artefacts emerge in the top-left corner as well as below the dough. Furthermore, the dynamics of the dough show flaws visible in $t = 10$, where the dent from $t < 10$ suddenly vanishes. A positive result is the reasonable preservation of the dough shape and lighting from $t = 0$ to $t = 7$, where no manipulation took place. While the imperfections are difficult to avoid completely, utilizing U_θ alone could only allow for generating trajectories limited to $H < 10$.

Re-projection with Diffusions Models

The compounding errors by U_θ let the state predictions diverge more with each step from the state distribution. To mitigate this issue, an unconditional diffusion model is used. Diffusion

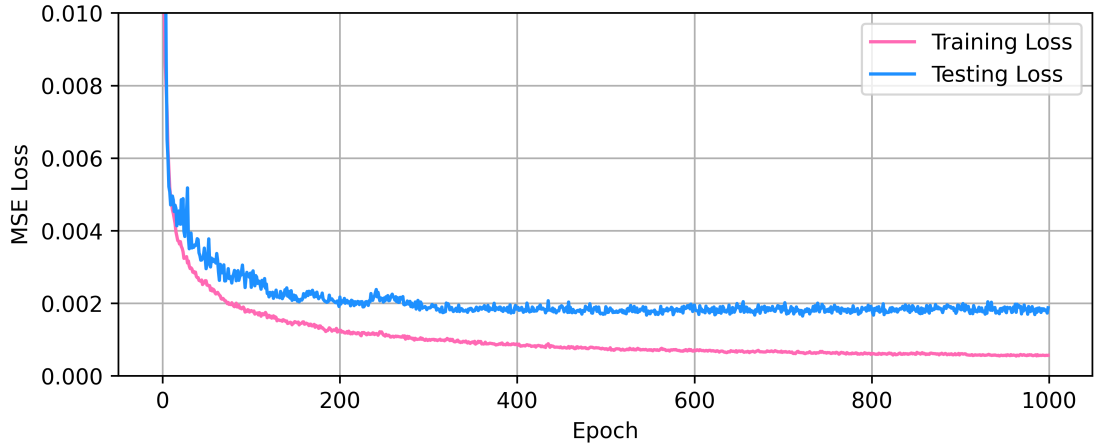


Figure 5.5.: Training and testing losses of the dynamics model U_θ. The testing loss already converges after 300 episodes while training loss still improves slightly. Overall, training and testing loss diverge a bit.

models have shown to be exceptional at learning image distributions. The reverse diffusion process allows not only to generate new samples given only Gaussian noise. Instead, an existing image can serve as intermediate result when noise is applied only to a certain degree. This way, details can be restored, while the overall image context is preserved (see Figure 2.5).

Utilizing the states from $\mathcal{D}$, a diffusion model U_ψ based on the same conditional U-Net with skip connection is trained to restore erroneous visuals of the output of U_θ while keeping the image context as close as possible. Following [17], the time step parameter t (see section 2.5) was used to condition the feature maps. Compared to U_θ, only around 2M parameters were used due to the necessity of running the network multiple of times for one reverse diffusion process. For the same reason, T is set to 200, sacrificing quality for a 5x speedup during reverse diffusion. U_ψ was trained for 500 episodes with the test loss converging after around 200 episodes (see Figure 5.8). The batch size was set to 64 and the overall training took only about one hour on the NVIDIA RTX 4090. After training U_θ and U_ψ, the model-based augmentation technique f_{model} can be defined as in Algorithm 1.

When applying $K = 20$ noise steps to the output of U_θ, and then do the reverse diffusion process with U_ψ, the image quality changes significantly (see Figure 5.7, $t = 11$). However, when inspecting the trajectory closely, the dough shape is less invariable over consecutive steps where no deformation occurred, as visible $t = 8$ where the dent almost completely vanished without physical cause. Also the lighting changes unreasonably which can be observed from $t = 0$ to $t = 4$. This trade-off between dynamics accuracy and visual quality has to be considered when choosing a K for the re-projection.

Since both models are only trained on 6206 samples, their generalization capabilities might not be sufficient to produce new trajectories based on entirely new actions. Instead, given a trajectory τ, the transformation is done with s_0 and all a_t from that trajectory where the translation, as well as the opening width from a_t is altered with Gaussian noise parametrized

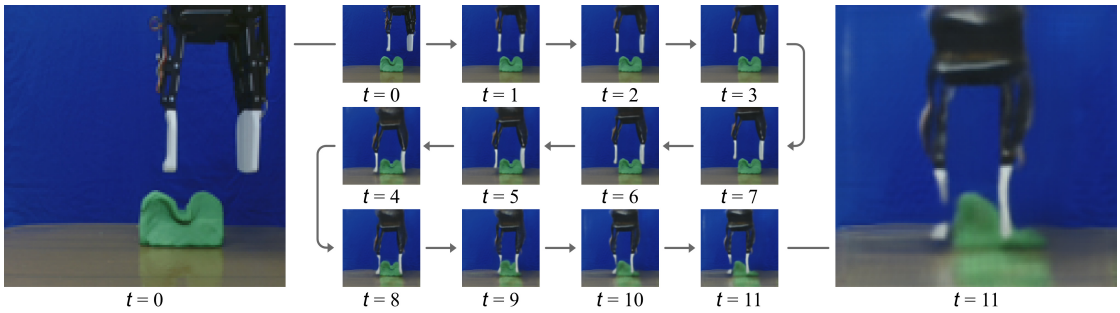


Figure 5.6.: Inference of only the dynamics model U_θ for 11 steps. The dough manipulations are rather good but compounding errors in visuals deteriorate the image quality after a few steps. The result at $t = 11$ shows a blurry image with severe deformations in the robots shape and white artefacts in the corner and at the bottom.

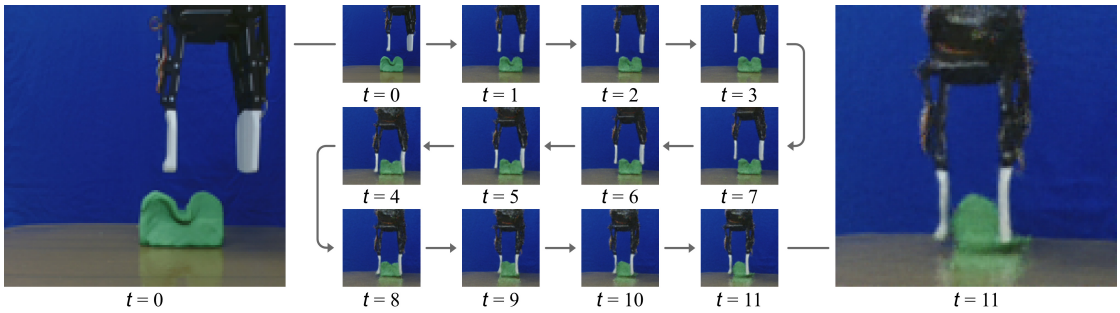


Figure 5.7.: Inference of the dynamics model U_θ with re-projection using the diffusion model U_ψ for 11 steps. Compared to Figure 5.6 the visual quality improved and allows for predicting longer trajectories. However, the dynamics are less accurate.

Algorithm 1 The model-based augmentation procedure f_{model}

$\hat{s}_0 \leftarrow s_0 \in \tau$ ▷ Initialize
for all $a_t \in \tau$ **do**
  $\hat{a}_t \leftarrow a_t + \epsilon_t$ with $\epsilon_t \sim \mathcal{N}(0, \sigma)$ ▷ Augment action
  $\hat{s}_{t+1} \leftarrow U_\theta(\hat{s}_t, \hat{a})$ ▷ Predict dynamics
  $\hat{s}_{t+1} \leftarrow \sqrt{\bar{\alpha}_K}\hat{s}_{t+1} + \sqrt{1-\bar{\alpha}_K}\epsilon$ with $\epsilon \sim \mathcal{N}(0,1)$ ▷ Forward diffusion
  for $i \leftarrow K, 1$ **do** ▷ Reverse diffusion
    $\hat{s}_{t+1} \leftarrow \frac{1}{\sqrt{\alpha_i}}(\hat{s}_{t+1} - \frac{1-\alpha_i}{\sqrt{1-\bar{\alpha}_i}}U_\psi(\hat{s}, i)) + \sqrt{\beta_t}z$ with $z \sim \mathcal{N}(0,1)$
  end for
  $\hat{r}_t \leftarrow r(\hat{s}, \hat{a})$ ▷ Determine reward
  yield $(\hat{s}_t, \hat{a}_t, \hat{s}_{t+1}, \hat{r}_t)$
end for

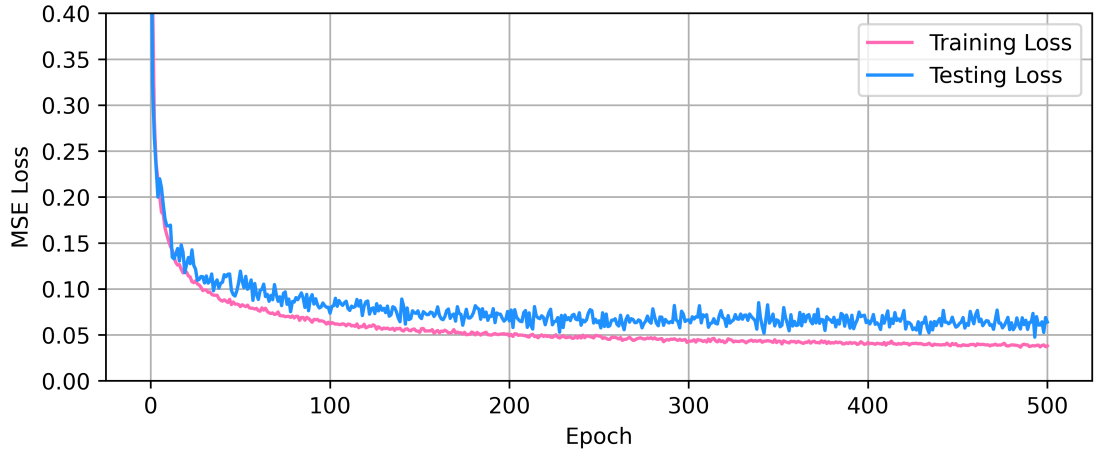


Figure 5.8.: Training and testing loss of the diffusion model U_ψ. The testing loss converges after 200 episodes while training loss still improves slightly. Overall, both losses do not seem to diverge too much.

by σ (see Algorithm 1, line 3). Keeping the augmented states close to the training distribution like that reduces the amount of necessary extrapolation and hence, the need for strong generalization capabilities.

5.3. Policy Training

The goal is to learn a policy with offline reinforcement learning given only a dataset. Because of limited data in the recorded dataset $\mathcal{D}_{\text{src}}$, two different augmentation techniques are applied to increase the buffer size without further environmental interaction. In order to compare the effects of the augmentations, 7 different dataset variations are created in addition to $\mathcal{D}_{\text{src}}$ (see Table 5.1). While $\mathcal{D}_{\text{crop1}}$ - $\mathcal{D}_{\text{crop9}}$ use only augmentations from f_{crop}, $\mathcal{D}_{\text{model1}}$ - $\mathcal{D}_{\text{model9}}$ use augmentations from f_{model} in addition to $\mathcal{D}_{\text{src}}$ ranging from 2x to 10x as many samples. Finally, $\mathcal{D}_{\text{all}}$ is a combination of $\mathcal{D}_{\text{crop9}}$ and $\mathcal{D}_{\text{model9}}$ creating the largest dataset with both augmentation techniques.

The policies consist of a convolutional encoder conditioned on the current goal shape with 16 feature neurons (see Figure 5.9). The input is the camera image scaled down to 128x128 pixels. The output of the policy is 4 neurons directly forming the relative action command (see subsection 5.1.3). The same architecture is also used for the action-value and the state-value model which only output one neuron representing the estimated return.

Each dataset is used to train one policy for 1M gradient steps. The training progress is monitored using the mean-absolute error over actions of a test set unseen by the agent. It contains one trajectory for each goal and a hand-picked set of keyframes (see appendix, Figure 5). A keyframe is a special frame where the action is semantically very clear for example by making a dent or cutting the dough in half. Additionally, these keyframes are also augmented with random-resize-crop to observe whether the policy generalizes to pixel

Dataset	Number of transitions			
	Source	Traditional	Model-based	Total
$\mathcal{D}_{\text{src}}$	7250	0	0	7250
$\mathcal{D}_{\text{crop1}}$	7250	7250	0	14500
$\mathcal{D}_{\text{crop3}}$	7250	21750	0	29000
$\mathcal{D}_{\text{crop9}}$	7250	65250	0	72500
$\mathcal{D}_{\text{model1}}$	7250	0	7250	14500
$\mathcal{D}_{\text{model3}}$	7250	0	21750	29000
$\mathcal{D}_{\text{model9}}$	7250	0	65250	72500
$\mathcal{D}_{\text{all}}$	7250	65250	65250	137750

Table 5.1.: Table showing the eight different dataset variations. They are chosen to evaluate the effects when increasing the amount of the respective augmentations. D_{all} combined the full amount of both augmentations. All datasets contain the full unmodified source trajectories.

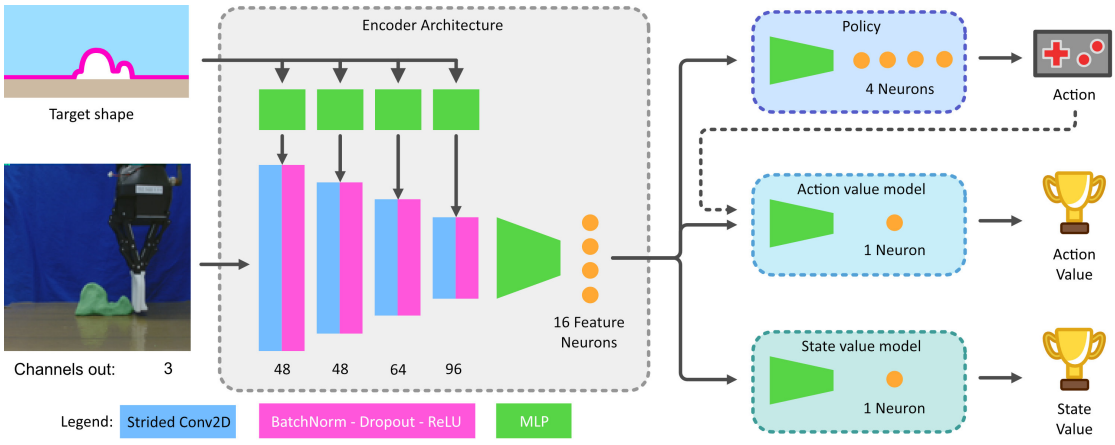


Figure 5.9.: The encoder architecture. Depending on the algorithm, the d3rlpy framework automatically creates policies, action-value and state-value models based on the same encoder architecture. In this work, the encoder consists of convolutional blocks conditioned on the goal shape.

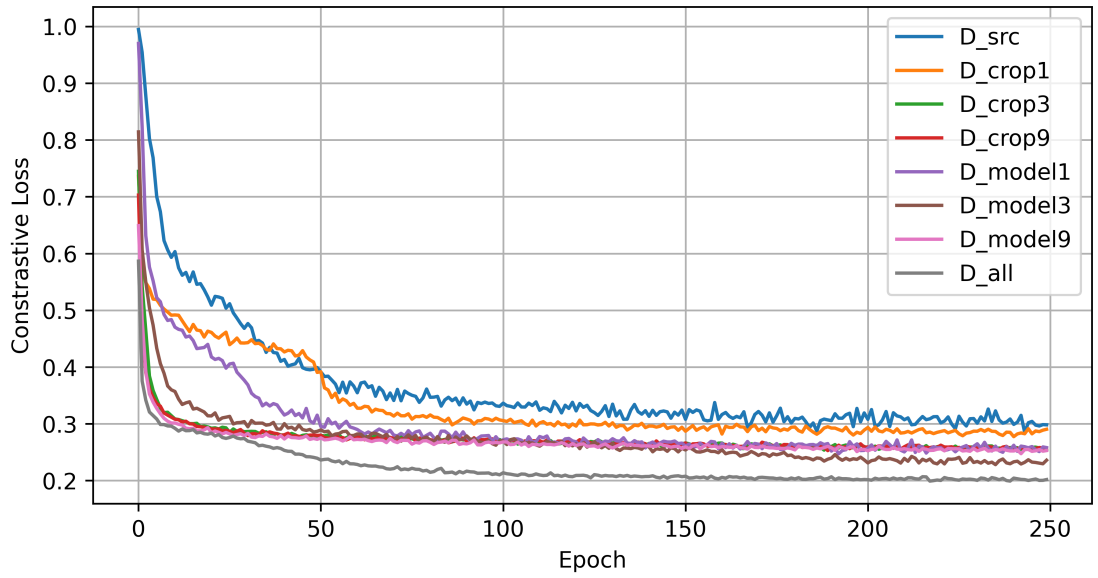


Figure 5.10.: The contrastive pre-training losses across all datasets. All losses converge after around 150 epochs. D_{src} appears as an upper bound while D_{all} as a lower bound. In between there seems to be no pattern followed by the remaining datasets.

translations.

Before the policies are trained, the encoders are pre-trained with contrastive learning on the respective dataset for 250 episodes (see Figure 5.10). The 16 output neurons from the encoders are transformed into an 8-dimensional contrastive space via a non-linear projection head (see section 2.4). The augmentations for the positive embeddings are obtained using the `AutoAugment` implementation from PyTorch. Similarly, the Triplet Loss [35] was used as it is already implemented in PyTorch as well.

Finally, the policies are trained with IQL based on the implementation from *d3rlpy*. For all training runs, the expectile parameter is set to $\tau = 0.9$, the target network synchronization coefficient to $\alpha = 0.00005$, the temperature coefficient of AWR to $\beta = 5$, and the batch size to 64 (see section 2.3 for all parameter definitions). The d3rlpy framework allows to split the total number of training steps into epochs which in this case is set to 200 steps resulting in 5000 epochs. Training one policy took about 8h 15min each on the NVIDIA RTX 4090, with 45min spent on the pre-training. Despite empirical attempts to tune the hyperparameters, the critics would in all cases have exploding losses with the actor receiving large rewards (see appendix, Figure 4, Figure 3). However, the mean-absolute-error with regards to the test set indicated no severe behaviour changes in the policies (see Figure 5.11).

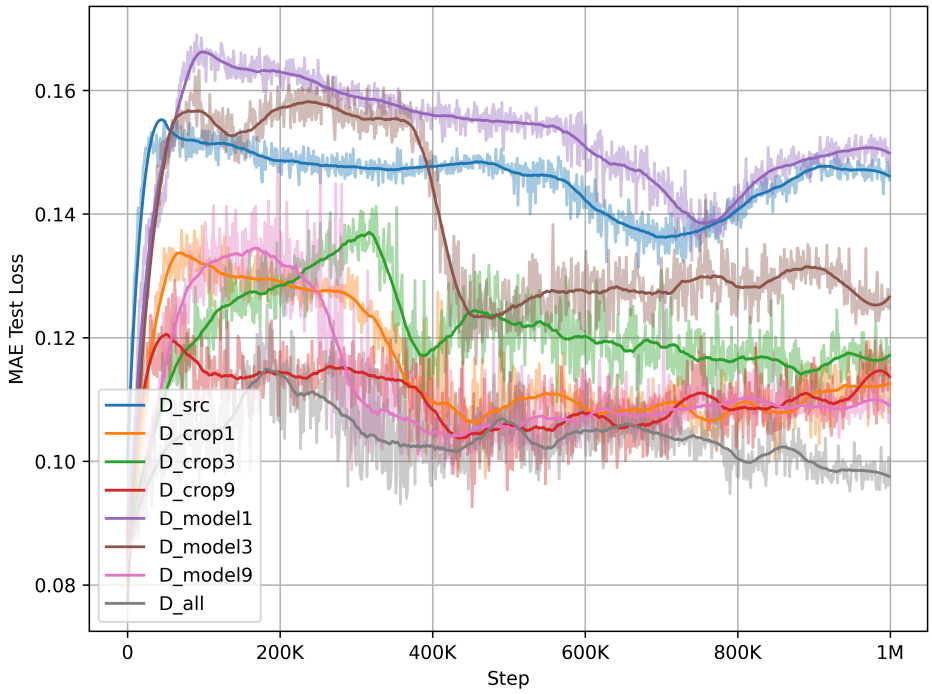


Figure 5.11.: Mean absolute error over the predicted actions compared to the test dataset during training with IQL. The values of one relative action $\langle x, y, \gamma, d \rangle$ range in $[-1, 1]$, hence an error of 0.1 for x means that the end-effector was $1/10$ of the image width off. At Step $= 0$ the error is rather low as the values for $\langle x, y, \gamma, d \rangle$ are all close to 0. Since the test set actions are relative, on average 0 is a good estimate. However, since IQL was trained with $\tau = 0.9$, the policy action distribution does not necessarily equal the test action distribution.

6. Evaluation

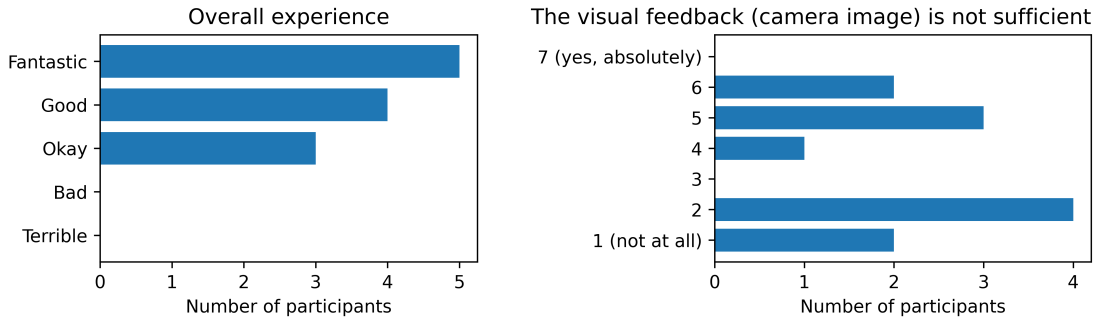
This chapter contains experiments for the gamified setup, the model-based augmentations, and the trained policies. Furthermore, all results are listed and briefly reviewed.

6.1. Survey for Gamified Setup

The idea behind the gamification of the teleoperation setup is to motivate users to put in an effort in order to acquire high-quality demonstrations. Hence, a survey was created to qualitatively evaluate the setup. The survey consists of 25 questions including the NASA Task Load Index (NASA-TLX) [15], the system usability scale (SUS) [4], and questions related to the teleoperation setup. Twelve people operated the robot for 30-45 minutes and then participated in the survey. The entire survey can be viewed in the appendix (section A).

Generally, the system was well received (see Figure 6.1a). In Figure 6.2, the responses to questions specifically about the system are shown. Overall, the game instructions were mostly clear and people quickly learned to control the robot. However, there were technical issues for some participants including one longer-lasting outage tarnishing the experience of one individual. Nevertheless, the gamification did not fail in its purpose as almost everyone felt encouraged to achieve a high score which should hypothetically produce good demonstrations. Interestingly, when asking whether people found the single RGB camera to be sufficient for the task, the opinions were split as shown in Figure 6.1b. No one absolutely disagreed with the single camera, but half of the responses were more negative while the other half was more positive leaving a gap in between.

The NASA Task Load Index (NASA-TLX) is a questionnaire used to subjectively rate the



(a) (b)

Figure 6.1.: Two bar charts showing the response distribution of two task-specific questions.

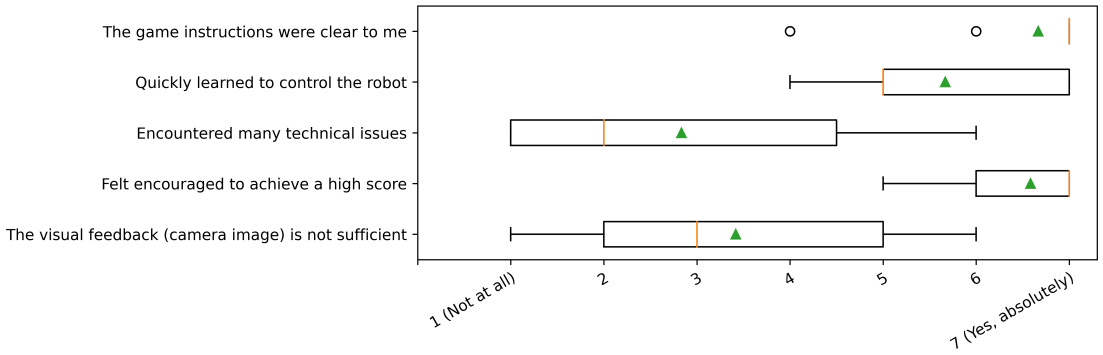


Figure 6.2.: Box-plot of the task-specific survey questions. The orange line indicates the median while the green triangle shows the mean. Circles correspond to outliers while the whiskers are drawn within the interquartile range of 1.5.

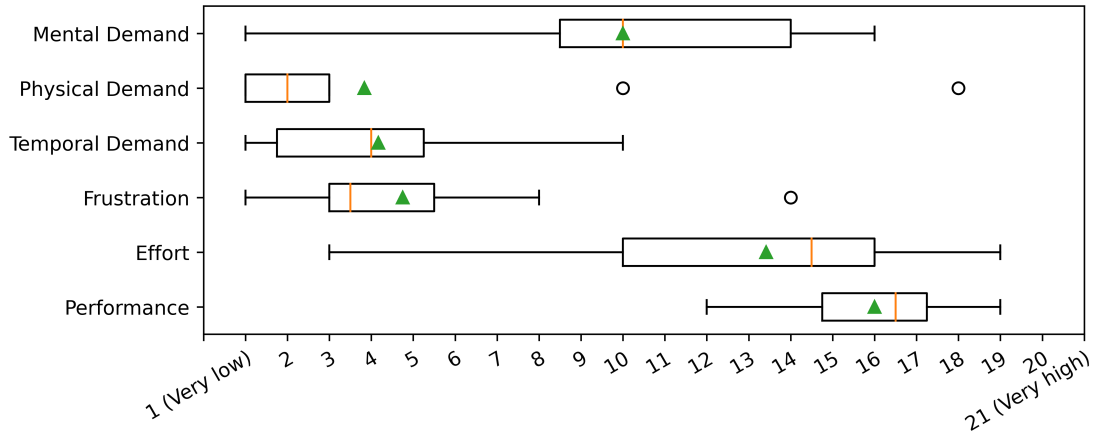
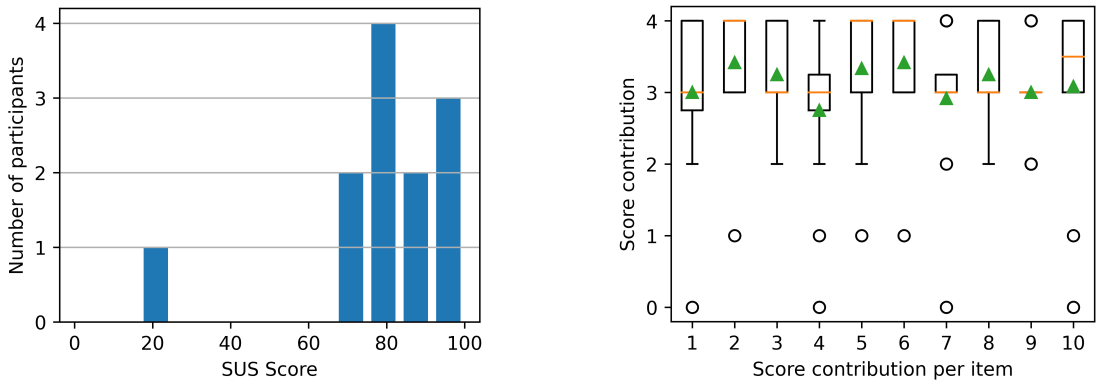


Figure 6.3.: Responses for the NASA-TLX.

perceived workload of various applications [15]. The responses are shown in Figure 6.3. It reveals that people found this task to be rather mentally demanding. As the task was carried out in front of a computer, most people felt not physically strained except for two participants. Furthermore, the participants agreed that the task would not put them under time pressure. Similarly, the task was also not perceived as frustrating except by one person. However, the amount of effort spent varies a lot between participants with a tendency that the task requires to work hard. Nevertheless, all participants were rather satisfied with their accomplishments indicating that the method is suitable to collect demonstrations.

The system usability scale (SUS) is a ten-item scale subjectively assessing the general usability of a system [4]. Figure 6.4a shows the distribution of individual scores by each participant. Most scores lie within the upper third of possible scores indicating that there might still be some potential to improve the system's usability in order to obtain better or more demonstrations. Figure 6.4b focuses on the score contribution of each questionnaire



(a) (b)

Figure 6.4.: The scores of the System Usability Scale.

item. When looking at the means (green triangles), four scales fail to reach a value of at least 3:

1. I think that I would like to use this system frequently. The participants indicate that the system lacks appeal to be frequented more. This could be due to the nature of a robotic system that made participants suspect a complicated, time-consuming setup for a game that might not be worth the effort. Another cause could be the technical complications some participants experienced or that the game is simply not rewarding enough (e.g. with sounds, visuals, or in-game progression).

4. I think that I would need the support of a technical person to be able to use this system. The participants indicate a certain dependency on a technical person in order to play the game. This is the case as the hardware setup is not explained to the participants and thus could not be done without help. And again, the technical errors could not be fixed without knowledge of the system. While in this work, independent starting of the game was not in the scope, this has to be considered when further unsupervised data collection is intended.

7. I would imagine that most people would learn to use this system very quickly. The participants indicate at least some parts of the game are hard to learn. This could mean that a certain level of technical experience is required to learn to control the robot, or that the control itself lacks simplicity. Another reason might be that certain actions are not intuitive, for example, that the dough shape can be enlarged by repeatedly opening and closing the gripper facing the viewer with the dough in between.

9. I felt very confident using the system. The participants indicate a lack of confidence while using the system. This could be because it was each participant's first attempt at the game. However, confidence might cease with a lack of control. This could be due to uncertainties like low end-effector precision or bad calibration which limits the player's ability to improve over time.

6.2. Model Experiments

In this section, the models used in f_{model} are evaluated regarding their performance. The aim of f_{model} is to accurately predict the dynamics of the dough shaping task given modified actions. However, when leaving the actions as they are, the predicted trajectories should not deviate much from the ground truth. Therefore, the performance of the models is measured by predicting the test set of 20 trajectories given only s_0 and a_t. As a metric, the intersection over unit (IOU) regarding the detected dough shape is used.

Table 6.1 shows the overall results. Four statistics are shown for all predicted states as well as only the last predicted state. In both cases the median is higher than the mean, meaning that there are more low-score outliers. The average scores of the last step are generally worse than the average of all scores as the states diverge less at the beginning and more towards the end. There are states in which the IOU is 0.0 due to dough occlusion which can be seen later. The overall max is not 1.0 as s_0, which does not have to be predicted, is not considered in this evaluation.

Figure 6.5 shows 4 exemplary trajectories with 5 steps depicted. In 6.5a a good example is shown with the IOU falling notably below 0.8 only once and then recovering above 0.8. This is likely due to the shorter length of the trajectory and the comparably simple actions. As discussed in subsection 5.2.2, the lighting conditions of the dough change the most. Also, the shape is not preserved ideally, which is explained later on.

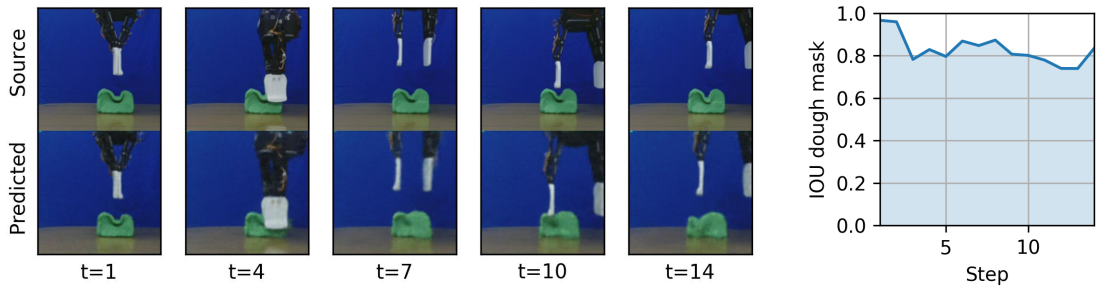
6.5b depicts a bad example where the trajectory length is similarly short like 6.5a, but the dynamics are more complex as the dough is cut in half and dragged for a longer distance. Specifically after the cut, the IOU falls quickly below 0.4 since the dough was predicted to stay in one piece contrary to the true data.

6.5c shows a rather long trajectory with 27 steps. While the IOU stays consistent roughly over 0.7, a growing visual deviation of the end effector size can be identified with the largest difference in $t = 27$. The reason lies in the data augmentation with random-resize-crop during the training of U_θ and U_ψ and is discussed in more detail later on.

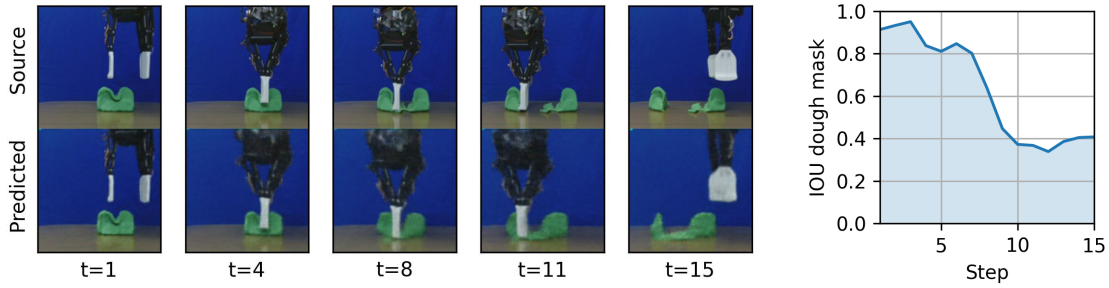
Finally, in 6.5d reveals a dough occlusion at $t = 17$. Due to the missing area where dough could be detected, the IOU suddenly drops to zero explaining the minimum value over all steps in Table 6.1 and the smaller mean compared to the median.

	Min	Mean	Median	Max
IOU all states	0.0	0.7284	0.7553	0.9731
IOU last state	0.3436	0.6485	0.6653	0.8349

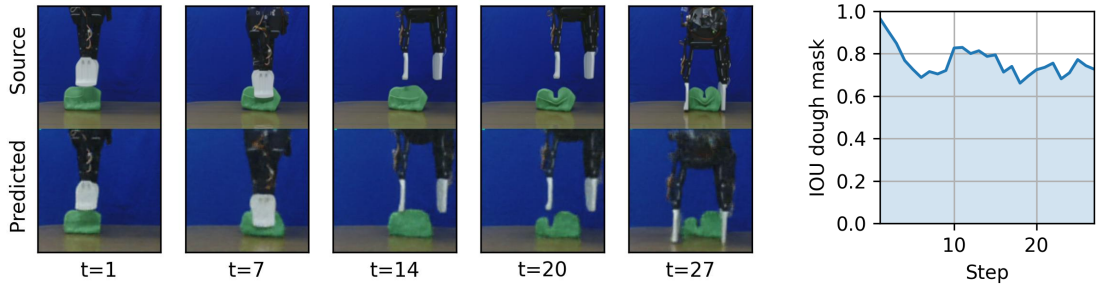
Table 6.1.: Results of recreating test trajectories with f_{model} measured by the intersection over unit of the dough.



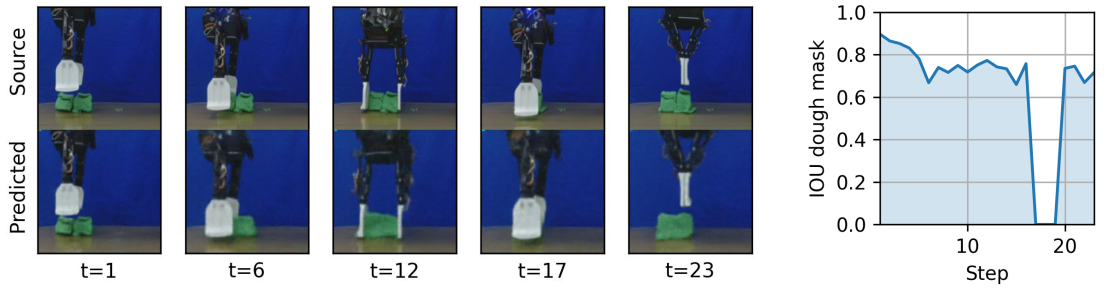
(a) Positive example



(b) Negative example



(c) Medium but lengthy example, growing end effector due to random-crops



(d) Dough occlusion example

Figure 6.5.: Selection with 4 of 20 reconstructed trajectories. On the left, extracts of the trajectories are shown, while on the right the IOU progression over all steps is depicted.

6.3. Policy Evaluation

Each policy was tested on 8 different tasks three times. They resemble the original tasks but focus more on specific actions. This promotes dough contact as the end-effector can be placed close to the dough. Additionally, a random policy π_{random} was implemented serving as a simple baseline. π_{random} chooses actions by sampling an action $a_t^i \in \mathcal{D}_{\text{src}}$ randomly. Since the policies were trained without stopping condition, each attempt has a limited number of steps. An overview of the tasks including their number of steps is given in Table 6.2.

Every single task attempt was manually prepared with an evaluation script, allowing for similar starting conditions. Because of the dough's plasticity, it is hard to form identical figures with complex shapes. Hence, the script captures a starting state for each test which is used as a reference when setting up a new attempt. The reference state is then continuously overlayed with the current setup when starting an attempt. A visualization of each start state is given in Figure 6.6. The script also records the pixel differences between the reference and the actual start state allowing for low deviations concerning dough shape.

The mean pixel differences of the start states for each task are shown in Figure 6.7. Generally, the error ranges from 5 to 13 and it can be observed that for tasks with less end-effector presence (Place L, Merge, Snail) the error is also smaller on average compared to tasks with full end-effector presence (Cut, Push). Another aspect is that outliers are consistently towards higher errors. This might be due to external influences like lighting conditions, hardware calibration, or drapes in the background curtain as the experiments could not be all done in one session. Another reason might be human fatigue towards the end of a session.

After an attempt finished, the gripper was moved away from the dough to ensure no occlusion was present on the last frame. However, since the policies were trained without stopping condition, using only the last frame as performance measurement might be insufficient, as the IOU improvement might be higher during episodes and then lowered towards the end again. To also consider intermediate results, the best IOU improvement of an episode is measured as well. This leaves two metrics: 1. the IOU improvement of s_H, and 2. the best IOU

Nr.	Task	Steps H	Description
1	Place R	10	Pick and place to the right
2	Place L	10	Pick and place to the left
3	Dent	10	Lower gripper to make a dent
4	Cut	10	Move gripper left to cut the dough
5	Smooth	10	Lower gripper to flatten surface elevation
6	Push	10	Push sideways to shrink dough
7	Merge	50	Merge two pieces
8	Snail	50	Form rectangular shape into snail shape

Table 6.2.: The policy evaluation tasks including a description of the crucial actions needed.

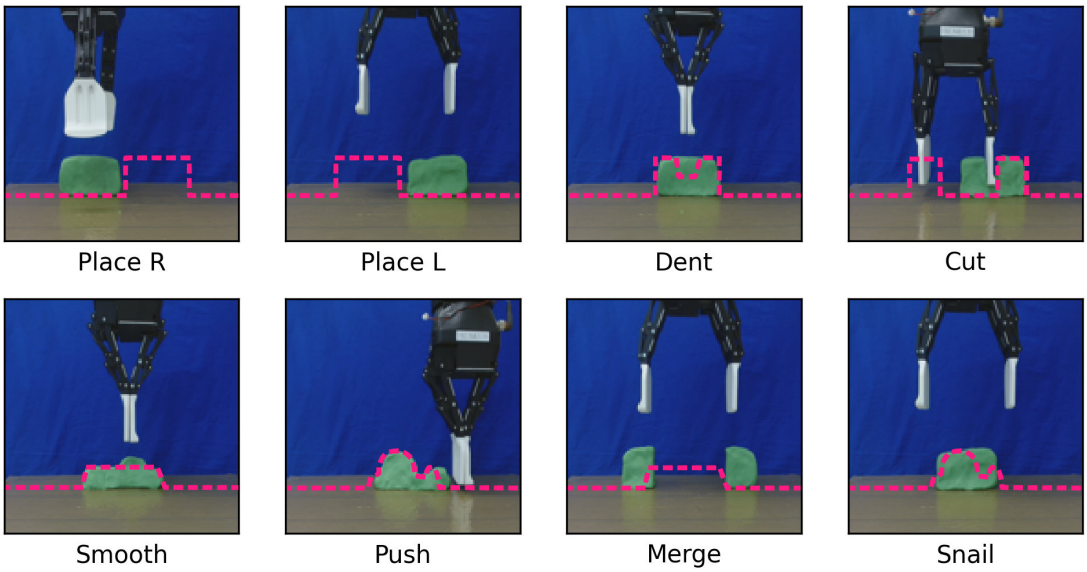


Figure 6.6.: Each policy evaluation task. The state is s_0 and the goal shape is visualized (magenta, dashed line).

improvement compared to s_0.

The final results of the s_H improvement can be seen in Table 6.3. Generally, the trend is to have negative improvements, meaning that the dough in the last state matches less than in the starting state. π_{crop1} and π_{all} are similar in overall performance. While π_{crop1} has the best results in 4/8 cases and the best sum, it shows large negative results in the remaining tasks. π_{all} overall less deteriorating behaviour, but also improved the IOU only in small quantities. In almost every task, π_{random} behaves significantly worse than all other policies. Interestingly, π_{src} performs better than most augmented policies (except π_{crop1} and π_{all}). There is also no clear pattern with increasing amounts of augmentations. The table also reveals differences in difficulty across the tasks themselves: Tasks like *Dent* or *Push* show significantly more deterioration than *Place R/L*, *Merge* or *Smooth*.

Compared to the best intermediate IOU improvement in Table 6.4, similarities and differences can be found. π_{crop1} still has the best results, but this time, followed by π_{model1}. π_{all} still scores highest on two tasks but with a lower margin towards the next best policy in the overall score. Furthermore, a monotonically falling overall score can be observed with an increasing amount of observations for π_{crop} and π_{model}. Also π_{random} is almost on the same level as π_{src}.

Finally, Figure 6.8 shows the IOU improvement over time for each policy on each task. The lines show the mean IOU over the three attempts while the ranges indicate the smallest IOU on the lower bound and the largest IOU on the upper bound. A larger range thus indicates less consistency in performance. It can be observed, that certain tasks (Place R/L, Cut, Merge) are less prone to IOU variance than others (Smooth, Push, Snail).

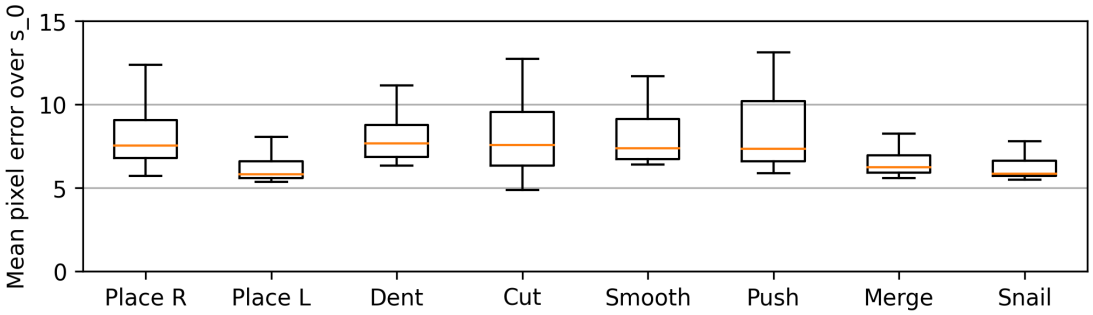


Figure 6.7.: Mean pixel error over s_0 for the different attempts compared to the reference state. The error is lower when less of the end-effector is seen. Here, the whiskers denote the minimum and maximum value.

Policy	Place R	Place L	Dent	Cut	Smooth	Push	Merge	Snail	Sum
π_{src}	0.000	0.000	0.001	0.076	**-0.012**	-0.231	0.013	-0.317	-0.469
π_{crop1}	**0.073**	**0.227**	-0.160	-0.014	-0.170	-0.286	**0.249**	**-0.110**	**-0.191**
π_{crop3}	0.000	0.025	-0.086	**0.116**	-0.062	-0.413	-0.033	-0.263	-0.717
π_{crop9}	0.032	0.000	-0.115	-0.009	-0.425	-0.135	-0.032	-0.199	-0.883
π_{model1}	0.015	0.191	-0.455	0.007	-0.196	-0.180	-0.039	-0.591	-1.248
π_{model3}	0.000	0.020	-0.037	0.005	-0.123	-0.124	-0.016	-0.424	-0.700
π_{model9}	0.000	0.000	-0.017	-0.027	-0.343	-0.236	0.032	-0.236	-0.828
π_{all}	0.000	0.025	**0.075**	0.008	-0.083	**0.042**	-0.034	-0.247	-0.215
π_{random}	0.000	0.000	-0.403	-0.019	-0.475	-0.574	-0.047	-0.602	-2.120

Table 6.3.: Mean IOU improvement of the last state compared to s_0.

Policy	Place R	Place L	Dent	Cut	Smooth	Push	Merge	Snail	Sum
π_{src}	0.000	0.000	0.025	0.125	0.027	0.008	0.080	0.079	0.34
π_{crop1}	**0.073**	**0.227**	0.013	0.090	0.016	0.028	**0.303**	0.055	**0.81**
π_{crop3}	0.000	0.131	0.044	**0.139**	0.002	0.032	0.032	**0.082**	0.46
π_{crop9}	0.032	0.000	0.027	0.080	0.000	0.036	0.092	0.013	0.28
π_{model1}	0.017	0.191	0.072	0.081	0.002	0.023	0.160	0.016	0.56
π_{model3}	0.000	0.045	0.007	0.065	**0.027**	0.000	0.016	0.037	0.20
π_{model9}	0.000	0.000	0.028	0.002	0.015	0.005	0.069	0.055	0.17
π_{all}	0.009	0.036	**0.152**	0.069	0.009	**0.055**	0.145	0.057	0.53
π_{random}	0.027	0.000	0.001	0.112	0.001	0.035	0.075	0.002	0.25

Table 6.4.: Mean IOU improvement of the best states compared to s_0.

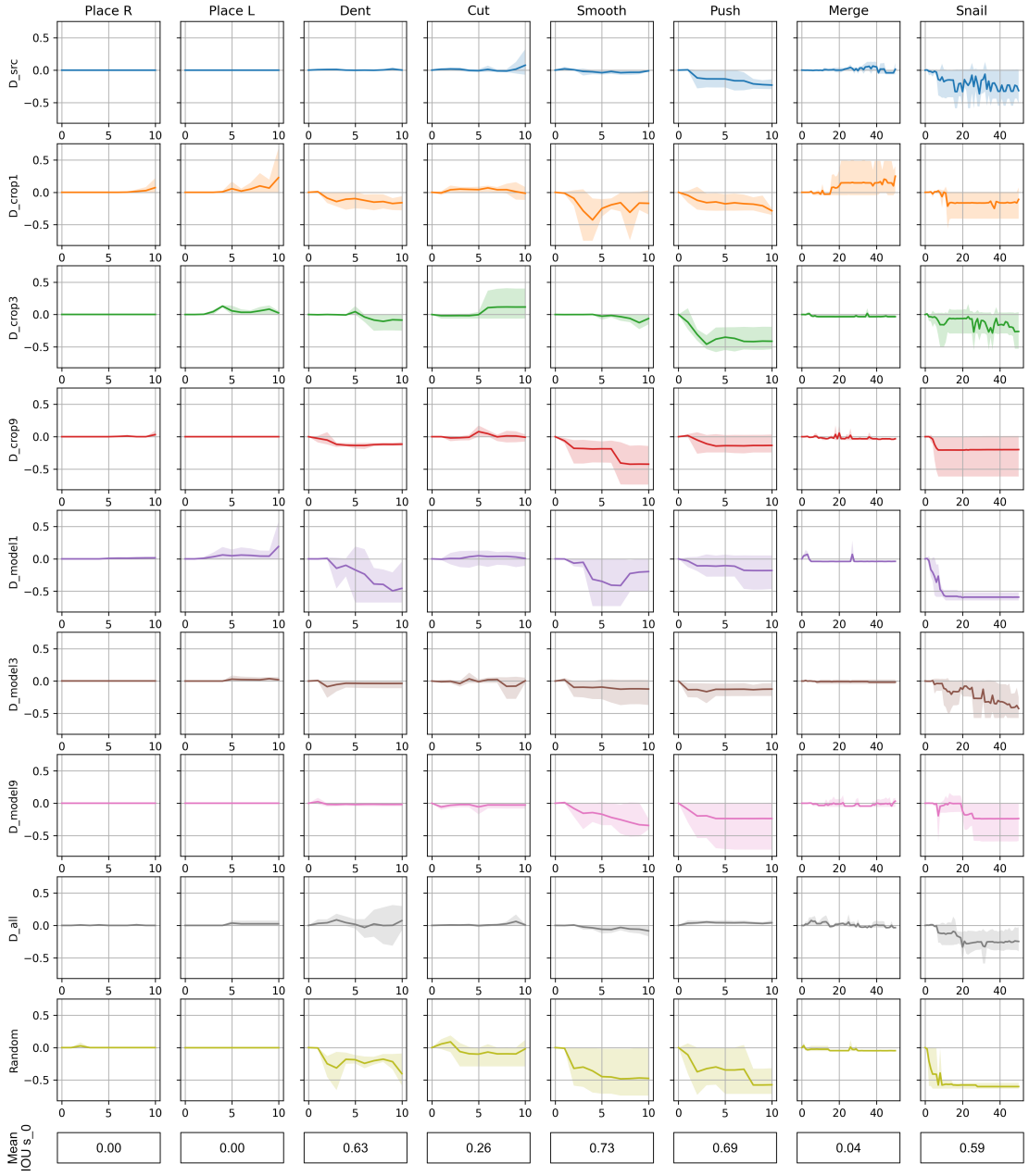


Figure 6.8.: IOU improvement relative to s_0 for all goals for all policies. Each goal was attempted 3 times. The range indicated the overall upper and lower bound. The line indicates the mean IOU. For each task, the mean IOU of s_0 across all attempts is given.

7. Discussion

In the previous chapter, the setup gamification, the model-based augmentation, and the policy have been evaluated in different aspects. This chapter discusses the results more in-depth.

7.1. Survey Results

The survey about the gamified setup asks 12 participants for their subjective perception of different aspects of the system. Although most parts were well received, some results indicate how the system could be changed in the future to improve data collection. A recurring point was the technical problems which are hard to completely eliminate in interactive robotic systems. However, this stresses that efforts have to be directed towards the system's robustness as it limits the quantity and quality of the player's demonstrations.

Half of the participants stated the single RGB camera is not sufficient for the game. This could have multiple causes, but it is most likely due to the inability to observe the scene along the z-axis. Certainly, it raises the question of whether introducing additional sensors (e.g. a second camera) to the user but not necessarily to the RL agent would be beneficial for data collection or would overwhelm the player with information.

The results of the NASA-TLX show that the mental demand and the effort required to spend are comparably high for this task. This might be due to the complexity of the task itself as dough manipulation without the use of our hands is quite hard. Additionally, the robot control is not optimized for precision, hence the tolerances are high meaning that it is hard to predict the outcome of the player's intended actions. This also correlates with the lack of confidence seen in the SUS results. Spending a lot of time and effort on the accuracy and reliability of the system was not prioritized in this work. However, the resulting limitations reveal that this does make a difference, not due to the (expected) imperfections in data, but due to user satisfaction in the collection process. Optimizing the system in these aspects could not only reduce mental demand and required effort but probably decrease frustration levels even further.

Limitations: While the survey provided insights into the participants perception, a study with two groups might be better suited to compare whether gamification positively impacts the data collection process. Providing two teleoperation systems, one with gamification, and one without, could give subjective insight with questionnaires as well as allow for numerical comparisons between the achieved rewards or participation time. Unfortunately, this was not in the scope of this work.

7.2. Model-based Augmentations

The idea of model-based augmentations is to alter the context of the state images. This requires a model of the transition dynamics of the MDP. Utilizing only U_θ for dynamics prediction trained with the MSE loss function results in compounding errors. This is due to the nature of the pixel-wise MSE as the resulting image becomes blurry where high variances occur like the edges of the dough or the robot [19, 10]. To counteract blurriness, a diffusion model U_ψ was trained to re-project erroneous images back into the training distribution. Even though the diffusion model is trained with MSE as well, the combination of predicting noise rather than the denoised image and repeating this step multiple times result in better image quality. Utilizing U_ψ, details like sharp edges can be restored allowing for the prediction of longer trajectories – at the cost of dynamics accuracy. This is because the diffusion model is trained without considering the current state resulting in erroneous reconstructions in e.g. shape and light. This leads to a tradeoff between dynamics accuracy and long-term predictability that has to be borne in mind. Ultimately, the augmented trajectories are used to train a policy with offline reinforcement learning, so the effects of using different diffusion factors on the policy could be further studied.

Another noticeable behaviour of the predicted trajectories is the growing end-effector. This can be explained by the training process of the diffusion model. Since random-resize-crop augmentations were utilized to regularize U_ψ, the model learned a different distribution in which the average scene appears *magnified* compared to the ones from the original data distribution. This leads to a subtle but steady magnification of the whole scene with each prediction. Hence, in longer trajectories, this effect is more apparent than in shorter ones. Interestingly, the magnification of the dough seems less severe, which could be due to its deformable property allowing for a more diversity of shapes.

One other aspect is the occlusion of dough through the end-effector removing information about the dough shape. This can be problematic as drastic shape changes can occur without physical reason which leads to wrongful assumptions on behalf of the offline RL algorithm about the MDP. This issue could be mitigated by keeping more information about the past e.g. using recurrent neural networks or stacking the last N states as input.

Overall, the combination of the two models U_θ and U_ψ is able to reconstruct unseen trajectories with an IOU of around 0.65 over the dough shape in the final state. This result itself leaves room for improvement but is also not surprising, as less than 6000 training samples were used to train a total of around 6M parameters. While longer trajectories tend to have lower IOU because of compounding errors, the complexity of the manipulation actions plays a much more important role in determining the quality.

7.3. Policy Results

The policies were evaluated with eight, more specialized tasks focusing on *key-actions* that are crucial to receiving good rewards. This promotes dough contact rather than having policies move the robot aimlessly above the dough. Each of the 8 trained policies and one random policy were evaluated three times on the 8 tasks. Generally, none of the policies excelled

at all the tasks. At the mean IOU improvement over the last state, π_{crop1} showed the best overall performance followed by π_{all}. Looking at the increase of augmentations of π_{crop} and π_{model}, there seems to be no correlation to the performance. One thing to notice is that π_{all} shows the most careful behaviour whereas the other policies significantly fail in at least two tasks. This might be due to the more dense training distribution as more actions showing destructive behaviour can be learned to avoid.

However, as the policies were trained without stop condition, evaluating just the last state is insufficient. When inspecting the best improvement during the trajectories, it can be observed that for all tasks there is at least one policy achieving at least 5% improvement except for the *Smooth* task. This is likely due to the fact that there is no dough with such a distinct bump in the training data, though attempts to smooth smaller bumps in general are present.

7.3.1. Qualitative Analysis

The quantitative analysis shows information about the general performance with regard to the reward. However, it lacks details about the actual behaviours of the policies. This section discusses qualitative findings about the policies. Here, π_{random} is ignored as the random behaviour requires no further explanation.

The first aspect that reduces performance severely is the dough falling over. Recovering the dough from a lying position is not only hard for humans, but also underrepresented in the collected data. Hence it is unlikely to be learned by any of the policies. In 204 trajectories, the dough fell over 26 times meaning that around every 8th trajectory was impossible to recover. Most of them can be described with one of four observed error-classes (see Figure 7.1):

(a) Gripper rotation. Rotating the gripper while being close to the dough causes a physical impact along the scene's z-axis knocking the dough over.

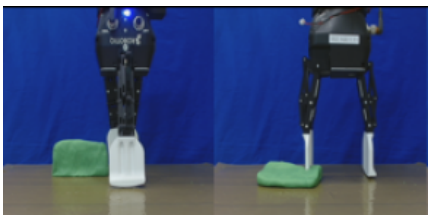
(b) Passive joints. The spring-equipped, passive joints of the gripper may misalign during the manipulation building up undesired forces potentially pushing the dough over.

(c) Rapid gripper relocation. Long-distance translations of the gripper can cause the dough to fall over when the robot collides during the motion with the dough. This is because longer motion trajectories allow for more acceleration and thus higher velocities.

(d) Dropping the dough. Occasionally, policies would grip the dough and then drop it from an elevated position resulting in an abrupt, instable landing.

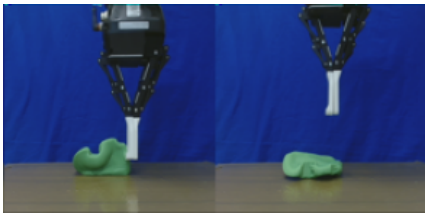
Another aspect that can be observed is utilizing key-action from a different task. Specifically, the denting and cutting action can be seen in several cases where it is inappropriate (see Figure 7.2). This indicates that the conditioning on the task using the goal shape is insufficient.

At last, an undesirable behaviour that occurred was to translate the dough while performing a key-action (see Figure 7.3). This was not due to sudden movements but rather small displacements over multiple steps. This could be explained by training data where the dough is intentionally displaced in small steps towards the target to prevent overshooting. However,

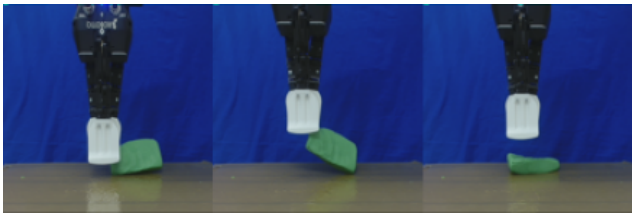


(a) Gripper rotation

(b) Passive joints

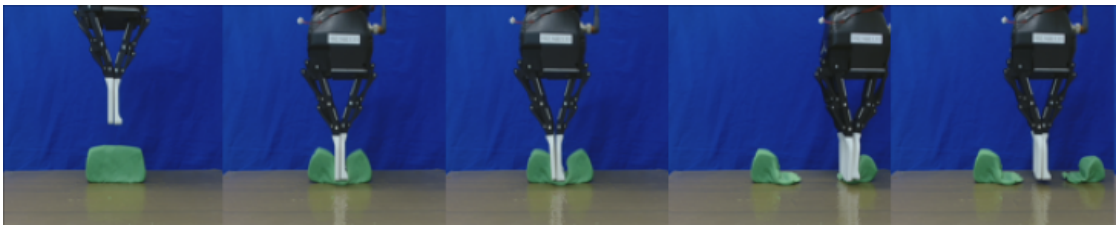


(c) Rapid gripper relocation

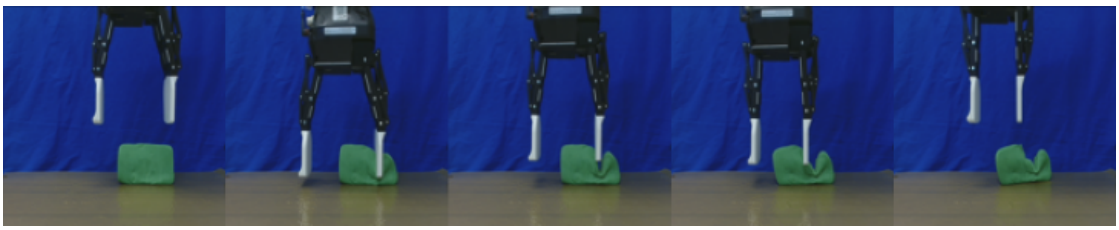


(d) Dropping the dough

Figure 7.1.: Four reoccurring reasons why the dough fell over were identified (a-d). In around every 8th trajectory the dough was pushed over.

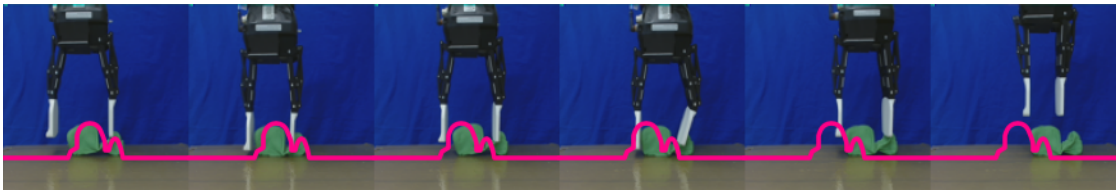


(a) A cutting action was performed during the *Dent* task by π_{crop9}.

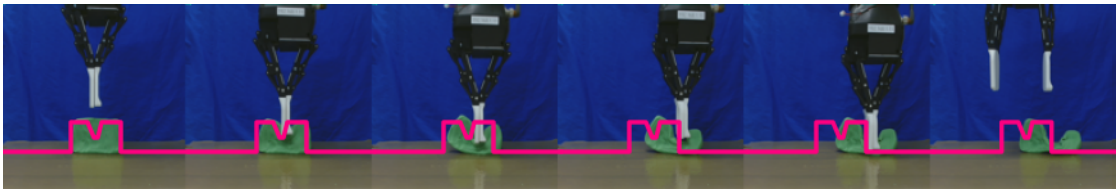


(b) A denting action was performed during the *Place L* task by π_{all}.

Figure 7.2.: Two exemplary trajectory extracts were key-actions from different tasks were performed.



(a) A well shaped snail received low reward due to the faulty dough translation from π_{model1}.



(b) The key-action was performed with translation by π_{crop1} resulting in a malformed dent. Furthermore, it caused to dough to end up in the wrong position yielding a low reward.

Figure 7.3.: Two exemplary trajectory extracts were an undesired translation of the dough caused a low reward despite the presence of correct key-actions. The red line indicates the target goal.

the opposite case where small steps reduce overlap is much less represented causing a lack of negative feedback from such actions during training.

8. Conclusion

This work looks at the utilization of offline reinforcement learning to carry out a deformable object manipulation task on a real robot with the help of different augmentation techniques. Various aspects are examined, starting from the implementation of the robotic system, over the data collection process, to the details of the model-based augmentations and the evaluation of the trained policies on the real system.

The setup was gamified with game design patterns to motivate users to achieve a high score leading to high-quality demonstrations. The survey revealed that almost everyone felt encouraged to do so. Despite being a mentally demanding task, the frustration levels stayed rather low indicating that the gamification might have a positive effect on the data collection process. However, technical failures have to be prevented as they negatively affect user satisfaction.

A model was developed to predict the environment dynamics which alone could not produce longer trajectories due to the compounding errors. By leveraging an unconditioned diffusion model, the predicted states could be re-projected so that the visual errors were mitigated allowing for advanced data augmentation. Together with the traditional random-size-crop augmentation, 7 additional datasets with up to 20x as much data were generated.

For each dataset, a policy was trained with the IQL offline reinforcement learning algorithm under the same conditions. While a numerical correlation between increasing augmentations and performance could not be clearly identified, several observations were made. The policy trained on all augmentations made the fewest errors, though it did not lead in performance. This hints that a combination of augmentations might help to avoid selecting destructive actions. The qualitative analysis explained several bad behaviours leading to low rewards. Several types of pushing the dough over were identified and also undesired translations while performing key-actions could be observed. At last, the analysis revealed that the task conditioning might be insufficient as key-actions were performed in the wrong task leading to good shapes but in the wrong context.

8.1. Future Work

During the work of this thesis, many aspects came up that could be further investigated, improved, and evaluated. While there are common facets e.g. hyperparameters, network architectures, or reward signals that are underexplored, this work also reveals specific limitations that could be improved in the future:

Dataset improvement A problem encountered in the policy evaluation (see subsection 7.3.1) was repeating destructive behaviour of different policies by either pushing the dough

over or translating the dough incorrectly. This indicates that policies tend to encounter states uncovered by the training data that trigger destructive actions. This problem has already been studied by Ross et al. [33] suggesting the use of learned policies for incremental dataset generation whereas the expert (in this case humans) corrects the actions of the encountered states. This could be done by utilizing the game interface to show states visited by the policy and letting the user decide the best next action generating new datasets.

Single-Task offline RL Since the policies sometimes performed the right actions but for the wrong task, the conditioning seems to be insufficient. Hence it could be investigated, whether the augmentations have a more apparent effect when training the policies for only one task.

Random-resize-crop in diffusion model The regularization using random-resize-crop during the training of the diffusion model leads to a step-wise magnification of the scene in the trajectory predictions. Further investigations could show whether the use of other regularization techniques helps to reduce this problem.

Improvement of setup The teleoperation setup was developed without high precision in mind. This potentially leads to a lack of confidence during gameplay. To improve the control accuracy, a different calibration method might be preferred. This could be realized by e.g. using a fixed setup where the positions of all hardware components are known and thus the end-effector pose can be computed more precisely.

IQL parameter tuning Since it was not possible to find hyperparameters that ensured convergence of the critics during the policy training with IQL, more time could be spent to understand why the algorithm failed in this setting. Lastly, the d3rlpy framework seems to continuously extend its selection of offline RL algorithms giving the possibility to evaluate different approaches beyond IQL.

Bibliography

[1] Yahav Avigal et al. "SpeedFolding: Learning Efficient Bimanual Folding of Garments". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2022), pp. 1–8.

[2] Alison Bartsch, Charlotte Avra, and Amir Barati Farimani. "SculptBot: Pre-Trained Models for 3D Deformable Object Manipulation". In: *arXiv preprint arXiv:2309.08728* abs/2309.08728 (2023).

[3] Greg Brockman et al. "OpenAI Gym". In: *ArXiv* abs/1606.01540 (2016).

[4] John Brooke et al. "SUS-A quick and dirty usability scale". In: *Usability evaluation in industry* 189.194 (1996), pp. 4–7.

[5] Ting Chen et al. "A Simple Framework for Contrastive Learning of Visual Representations". In: *ArXiv* abs/2002.05709 (2020).

[6] Andrea Cherubini et al. "Model-free vision-based shaping of deformable plastic materials". In: *The International Journal of Robotics Research* 39 (2020), pp. 1739–1759.

[7] D.M. Coleman et al. "Reducing the Barrier to Entry of Complex Robotic Software: a MoveIt! Case Study". In: *ArXiv* abs/1404.3785 (2014).

[8] Sebastian Deterding et al. "From game design elements to gamefulness: defining "gamification"". In: *International Conference on Entertainment and Media in the Ubiquitous Era*. 2011.

[9] Justin Fu et al. "D4RL: Datasets for Deep Data-Driven Reinforcement Learning". In: *ArXiv* abs/2004.07219 (2020).

[10] Vahid Ghodrati et al. "MR image reconstruction using deep learning: evaluation of network structure and loss functions." In: *Quantitative imaging in medicine and surgery* 9 9 (2019), pp. 1516–1527.

[11] Feida Gu et al. "A Survey on Robotic Manipulation of Deformable Objects: Recent Advances, Open Challenges and New Frontiers". In: *ArXiv* abs/2312.10419 (2023).

[12] Nicolas Gurtler et al. "Real Robot Challenge 2022: Learning Dexterous Manipulation from Offline Data in the Real World". In: *Neural Information Processing Systems*. 2023.

[13] Nico Gürtler et al. "Benchmarking Offline Reinforcement Learning on Real-Robot Hardware". In: *ArXiv* abs/2307.15690 (2023).

[14] Danijar Hafner et al. "Dream to Control: Learning Behaviors by Latent Imagination". In: *ArXiv* abs/1912.01603 (2019).

[15] Sandra G Hart and Lowell E Staveland. "Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research". In: *Advances in psychology*. Vol. 52. Elsevier, 1988, pp. 139–183.

[16] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 770–778.

[17] Jonathan Ho, Ajay Jain, and P. Abbeel. "Denoising Diffusion Probabilistic Models". In: *ArXiv* abs/2006.11239 (2020).

[18] Lukasz Kaiser et al. "Model-Based Reinforcement Learning for Atari". In: *ArXiv* abs/1903.00374 (2019).

[19] Nishant Khare et al. "Analysis of Loss Functions for Image Reconstruction Using Convolutional Autoencoder". In: *International Conference on Computer Vision and Image Processing*. 2021.

[20] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. "Offline Reinforcement Learning with Implicit Q-Learning". In: *ArXiv* abs/2110.06169 (2021).

[21] Ilya Kostrikov, Denis Yarats, and Rob Fergus. "Image Augmentation Is All You Need: Regularizing Deep Reinforcement Learning from Pixels". In: *ArXiv* abs/2004.13649 (2020).

[22] Michael Laskin et al. "Reinforcement Learning with Augmented Data". In: *ArXiv* abs/2004.14990 (2020).

[23] Sergey Levine et al. "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection". In: *The International Journal of Robotics Research* 37 (2016), pp. 421–436.

[24] Sergey Levine et al. "Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems". In: *ArXiv* abs/2005.01643 (2020).

[25] Carolyn Matl and Ruzena Bajcsy. "Deformable Elasto-Plastic Object Shaping using an Elastic Hand and Model-Based Reinforcement Learning". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2021), pp. 3955–3962.

[26] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: *ArXiv* abs/1312.5602 (2013).

[27] Alex Nichol and Prafulla Dhariwal. "Improved Denoising Diffusion Probabilistic Models". In: *ArXiv* abs/2102.09672 (2021).

[28] Edwin Olson. "AprilTag: A robust and flexible visual fiducial system". In: *2011 IEEE International Conference on Robotics and Automation* (2011), pp. 3400–3407.

[29] Jan Ondras et al. "Robotic Dough Shaping". In: *2022 22nd International Conference on Control, Automation and Systems (ICCAS)* (2022), pp. 300–307.

[30] Xue Bin Peng et al. "Advantage-Weighted Regression: Simple and Scalable Off-Policy Reinforcement Learning". In: *ArXiv* abs/1910.00177 (2019).

[31] Morgan Quigley. "ROS: an open-source Robot Operating System". In: *IEEE International Conference on Robotics and Automation*. 2009.

[32] Stéphane Ross and Drew Bagnell. "Efficient Reductions for Imitation Learning". In: *International Conference on Artificial Intelligence and Statistics*. 2010.

[33] Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning". In: *International Conference on Artificial Intelligence and Statistics*. 2010.

[34] Roman Sarrazin-Gendron et al. "Improving microbial phylogeny with citizen science within a mass-market video game." In: *Nature biotechnology* (2024).

[35] Florian Schroff, Dmitry Kalenichenko, and James Philbin. "FaceNet: A unified embedding for face recognition and clustering". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 815–823.

[36] Takuma Seno and Michita Imai. "d3rlpy: An Offline Deep Reinforcement Learning Library". In: *J. Mach. Learn. Res.* 23 (2021), 315:1–315:20.

[37] Haochen Shi et al. "RoboCook: Long-Horizon Elasto-Plastic Object Manipulation with Diverse Tools". In: *ArXiv* abs/2306.14447 (2023).

[38] Haochen Shi et al. "RoboCraft: Learning to See, Simulate, and Shape Elasto-Plastic Objects with Graph Networks". In: *ArXiv* abs/2205.02909 (2022).

[39] Connor Shorten and Taghi M. Khoshgoftaar. "A survey on Image Data Augmentation for Deep Learning". In: *Journal of Big Data* 6 (2019), pp. 1–48.

[40] Samarth Sinha and Animesh Garg. "S4RL: Surprisingly Simple Self-Supervision for Offline Reinforcement Learning". In: *ArXiv* abs/2103.06326 (2021).

[41] A. Srinivas, Michael Laskin, and P. Abbeel. "CURL: Contrastive Unsupervised Representations for Reinforcement Learning". In: *ArXiv* abs/2004.04136 (2020).

[42] Richard S. Sutton and Andrew G. Barto. "Reinforcement Learning: An Introduction". In: *IEEE Trans. Neural Networks* 9 (1998), pp. 1054–1054.

[43] Eleonora Tagliabue et al. "Soft Tissue Simulation Environment to Learn Manipulation Tasks in Autonomous Robotic Surgery". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, pp. 3261–3266.

[44] Zhenjia Xu et al. "RoboNinja: Learning an Adaptive Cutting Policy for Multi-Material Objects". In: *ArXiv* abs/2302.11553 (2023).

[45] Hang Yin, Anastasia Varava, and Danica Kragic. "Modeling, learning, perception, and control methods for deformable object manipulation". In: *Science Robotics* 6 (2021).

[46] Yang You et al. "Make a Donut: Language-Guided Hierarchical EMD-Space Planning for Zero-shot Deformable Object Manipulation". In: *ArXiv* abs/2311.02787 (2023).

[47] Tianhe Yu et al. "Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning". In: *ArXiv* abs/1910.10897 (2019).

[48] Albert Zhan et al. "Learning Visual Robotic Control Efficiently with Contrastive Pre-training and Data Augmentation". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2020), pp. 4040–4047.

[49] Jihong Zhu et al. "Challenges and Outlook in Robotic Manipulation of Deformable Objects". In: *IEEE Robotics & Automation Magazine* 29 (2021), pp. 67–77.

[50] Yuke Zhu et al. "robosuite: A Modular Simulation Framework and Benchmark for Robot Learning". In: *ArXiv* abs/2009.12293 (2020).

Appendix

A. Questionnaire

In the following, all questions of the questionnaire regarding the gamified teleoperation setup are listed.

Introduction

Which gender do you feel you belong to?

Choose one of the following answers. Please choose only one of the following:
☐ woman ☐ man ☐ non-binary ☐ prefer not to disclose ☐ Other

What is your age?

Choose one of the following answers. Please choose only one of the following:
☐ 18-24 ☐ 25-34 ☐ 35-44 ☐ 45-54 ☐ 55-64 ☐ 65 or higher

Experience

How would you rate your overall experience using the system?

Please choose the appropriate response for each item:
☐ Terrible ☐ Bad ☐ Okay ☐ Good ☐ Fantastic

How strongly do you agree or disagree with the following statements?

Please choose the appropriate response for each item:

The game instructions were clear to me
☐ 1 (not at all) ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 (yes, absolutely)

I encountered many technical issues
☐ 1 (not at all) ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 (yes, absolutely)

I quickly learned to control the robot
☐ 1 (not at all) ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 (yes, absolutely)

I felt encouraged to achieve a high score
☐ 1 (not at all) ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 (yes, absolutely)

The visual feedback (camera image) is not sufficient for the game
☐ 1 (not at all) ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 (yes, absolutely)

(Optional) How could the system be improved to enchance your experience?

_____

Workload

How mentally demanding was the task?

Very Low — Very High

Mental Demand ○

How physically demanding was the task?

Very Low — Very High

Physical Demand ○

How hurried or rushed was the pace of the task?

Very Low — Very High

Temporal Demand ○

How successful were you in accomplishing what you were asked to do?

Perfect — Failure

Performance ○

How hard did you have to work to accomplish your level of performance?

Very Low — Very High

Effort ○

How insecure, discouraged, irritated, stressed, and annoyed were you?

Very Low — Very High

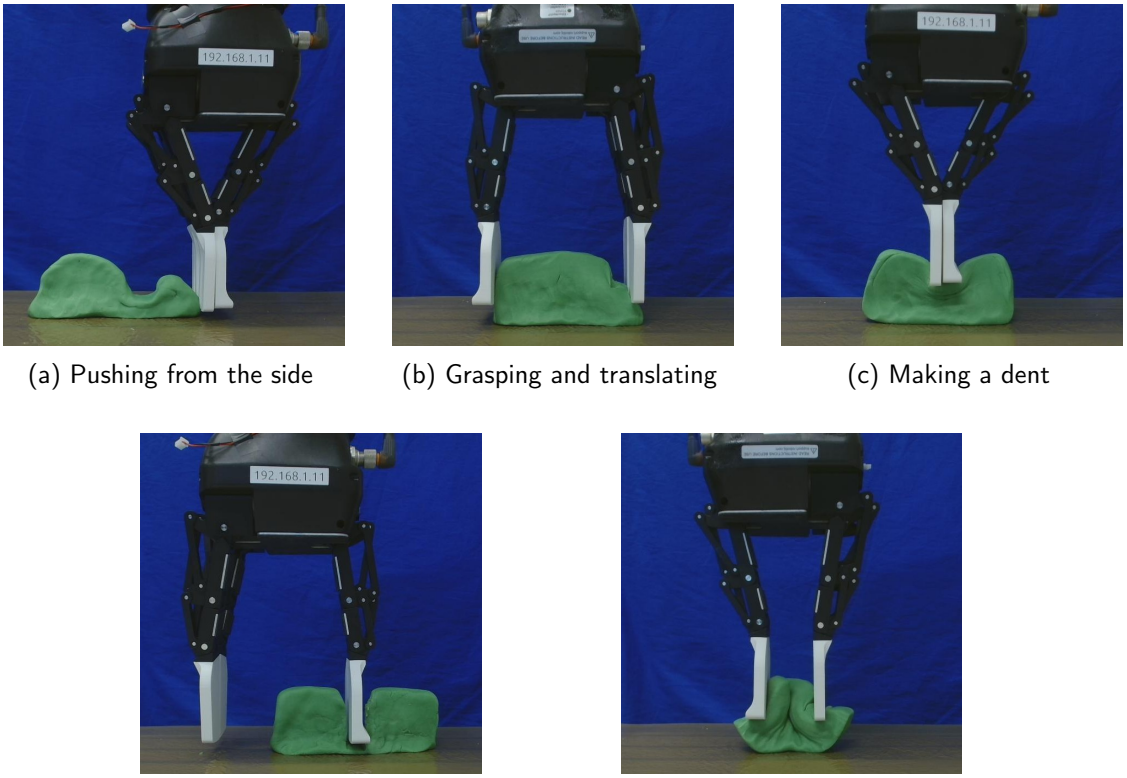
Frustration ○

Usability

Please rate the system's usability
Please choose the appropriate response for each item:

	1 - Strongly Disagree	2	3	4	5 - Strongly Agree
1. I think that I would like to use this system again	◯	◯	◯	◯	◯
2. I found the system unnecessarily complex	◯	◯	◯	◯	◯
3. I thought the system was easy to use	◯	◯	◯	◯	◯
4. I think that I would need the support of a technical person to be able to use this system	◯	◯	◯	◯	◯
5. I found the various functions in this system were well integrated	◯	◯	◯	◯	◯
6. I thought there was too much inconsistency in this system	◯	◯	◯	◯	◯
7. I would imagine that most people would learn to use this system very quickly	◯	◯	◯	◯	◯
8. I found the system very awkward to use	◯	◯	◯	◯	◯
9. I felt very confident using the system	◯	◯	◯	◯	◯
10. I needed to learn a lot of things before I could get going with this system	◯	◯	◯	◯	◯

B. Additional Images



(a) Pushing from the side (b) Grasping and translating (c) Making a dent

(d) Cutting the dough (e) Squeezing the dough

Figure 1.: Exemplary actions that utilize the custom fingertips in different ways.

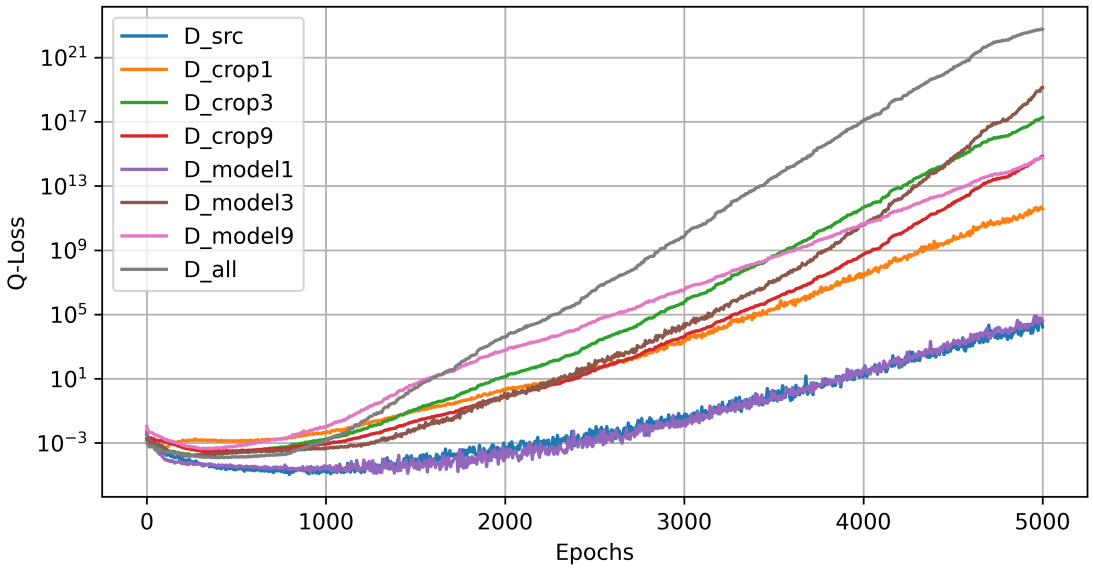


Figure 2.: The action-value losses of the IQL training.

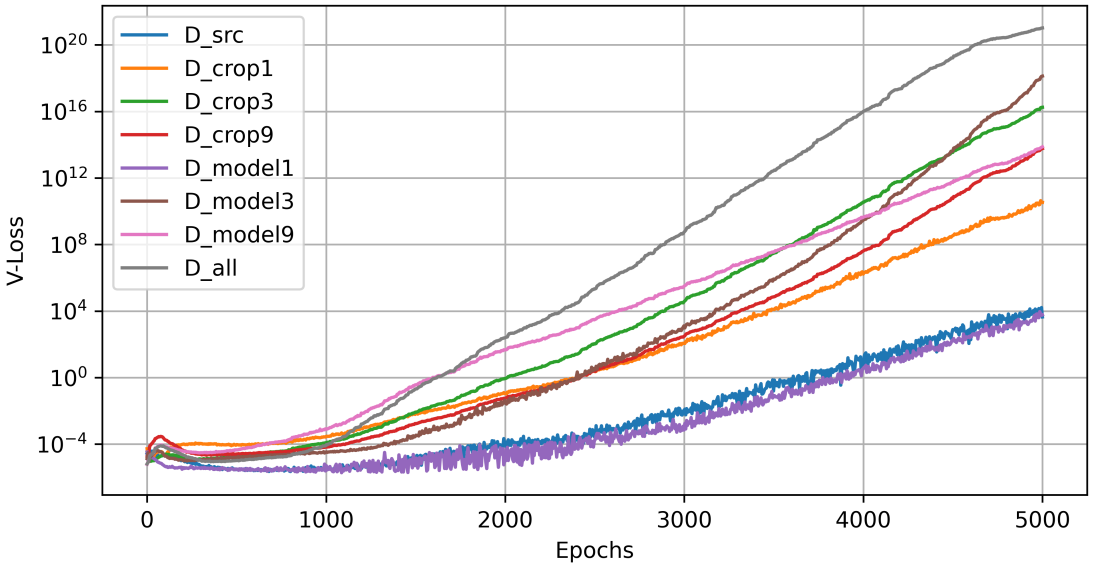


Figure 3.: The state-value losses of the IQL training.

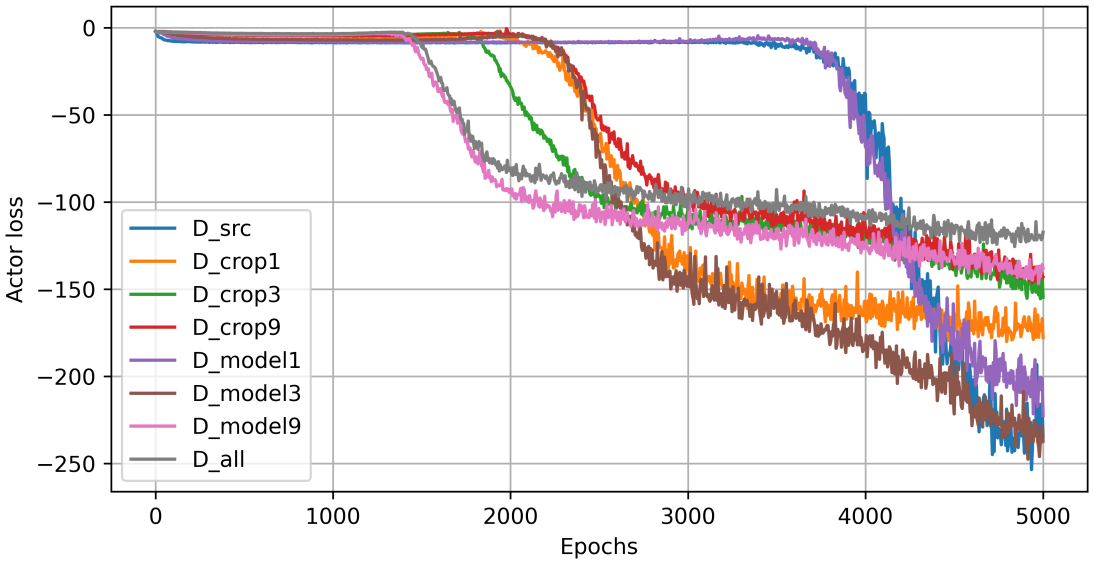


Figure 4.: The actor losses of the IQL training.

Figure 5.: Example test trajectory of the IQL training. The policies selected action for the state is drawn on the image. It is displayed as a white bar with the center being the $\langle x, y \rangle$ coordinates, the coloured bar filling expresses the opening width d and the colour itself the orientation γ where red $:= -90°$, blue $:= 0°$, and green $:= +90°$. The magenta line indicates the goal shape.

Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Masterstudiengang Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel — insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen — benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe.

Hamburg, den 14.08.2024 — Fabian Wieczorek

Veröffentlichung

Ich stimme der Einstellung der Arbeit in die Bibliothek des Fachbereichs Informatik zu.

Hamburg, den 14.08.2024 — Fabian Wieczorek