Residual Reinforcement Learning with Stable Priors

Stabile Prior für Residual Reinforcement Learning
Master thesis by Jascha Hellwig
Date of submission: January 15, 2023

1. Review: João Carvalho
2. Review: Julen Urain De Jesus
3. Review: Jan Peters
Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 und § 23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Jascha Hellwig, die vorliegende Masterarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß § 23 Abs. 7 APB überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 15. Januar 2023

J. Hellwig

Abstract

Although learning under contacts is crucial for real-world tasks in robotics, it is still very challenging to solve contact-rich tasks because they are incredibly difficult to model. Reinforcement learning (RL) is a promising approach for such tasks, as it allows learning a behavior by receiving rewards through interactions with an environment. The need to explicitly model contacts is eliminated, as the learned policy only needs to find a way to deal with these contacts, which is still challenging. Many methods have been established to learn contact-rich tasks, such as assembly or peg-in-hole tasks, but there is still much research on how to improve the state of the art. Recent approaches use a combination of a nominal and a residual policy, where the nominal policy guides the residual policy, and the residual policy learns to perform minor corrections to the nominal policy. This idea is not only efficient, but also provides much room for introducing advanced structures for either the residual or the nominal policy to improve the performance on a given task. We believe that appropriate priors for the nominal policy can drastically increase the performance of such methods.

For this reason, this work proposes the use of manifold stable vector fields (MSVFs) as learnable stable priors for residual reinforcement learning. Not only do MSVFs provide a suitable method to deal with orientations by using Lie groups, but they are also stable beyond the space in which they are trained in. This means that when used as a prior for learning, they do provide a stable signal toward a specified goal pose, and furthermore, this signal is robust to perturbations. As our experiments show, this prior not only speeds up learning for insertion tasks, but the residual approach also allows us to efficiently extend the method to perform variable impedance control as well. We evaluate its performance in a variety of environments, including simulating a 7-DOF Franka robot arm inserting either a box or a more complex Ubongo object.

Zusammenfassung

Wenn Roboter in einer realen Umgebung eingesetzt werden sollen, treten fast immer Kontaktkräfte auf, die das Lernen erschweren. Diese Kräfte sind sehr schwer zu modellieren und erschweren somit das Lösen verschiedenster Aufgaben. Ein vielversprechender Ansatz zur Lösung solcher Aufgaben ist das Reinforcement Learning (RL), bei dem ein Verhalten erlernt werden kann, indem Belohnungen bei der Interaktion mit einer Umgebung generiert werden. Das erlernte Verhalten muss nur einen Weg finden, die erhaltenen Belohnungen zu maximieren, wofür es aber nicht zwangsläufig lernen muss, Kontaktkräfte darzustellen. Es gibt zwar bereits Ansätze, die Aufgaben mit Kontaktkräften erfolgreich erlenen, aber es gibt weiterhin viel Forschung, die versucht den aktuellen Stand der Technik weiter zu verbessern. Einige Ansätze verwenden eine Kombination aus einer Nominalen und einer Residualen Policy. Die Idee dabei ist, dass die Nominale Policy die Residuale Policy leitet und die Residuale Policy lernt, das Verhalten durch kleine Anpassungen zu optimieren. Dieser Ansatz ist nicht nur besonders effizient, sondern bietet auch reichlich Spielraum, um erweiterte Strukturen in die Nominale oder Residuale Policy zu integrieren, die die Leistung noch weiter verbessern können. Wir glauben, dass geeignete Prior für die Nominale Policy zu einer signifikanten Leistungssteigerung führen.

Aus diesem Grund stellen wir einen Ansatz vor, der Manifold Stable Vector Fields (MSVFs) als stabile Prior für Residual Reinforcement Learning verwendet. Ein MSVF ermöglicht nicht nur eine effiziente Handhabung von Rotationen durch die Verwendung von Lie Gruppen, sondern ist auch über den Raum hinaus stabil, in dem es trainiert wurde. Das bedeutet, wenn sie als Prior zum Lernen eingesetzt werden, liefern sie ein stabiles Signal, das zu einer gewissen Zielposition führt. Außerdem sind MSVFs besonders robust gegenüber externen Störungen. Wie unsere Experimente zeigen, beschleunigt dieser Prior nicht nur das Lernen von Einfügeaufgaben, sondern der residuale Ansatz erlaubt es uns auch, unsere Methode mit variabler Impedanzkontrolle zu ergänzen. Wir evaluieren unsere Methode in einer Vielzahl von Umgebungen, einschließich in einer simulierten Umgebung, in der ein Franka-Roboterarm einen Würfel oder ein komplexeres Ubongo-Objekt einfügen muss.

Contents

1. Introduction

Humans have long dreamed of robots with human-like intelligence and abilities. Science fiction often features robots that cannot even be distinguished from a human being, and modern research and technologies are achieving great success when it comes to artificial intelligence. We are already able to perform speech recognition [1], autonomous driving [2], and image classification [3] at a high level of performance, but we are still far away from human-like intelligence and abilities, especially in robotics [4, 5, 6, 7]. We dream of robots autonomously producing goods, transporting them, or even providing healthcare, but this is either not yet possible or can only be achieved in highly structured and specialized environments, such as automotive assembly.

To deal with uncertain and unstructured environments, which applies to almost every real-world scenario that humans can solve, we need not only robots with a high degree of autonomy, but also robots that can learn how to interact safely with their environment by perceiving the world using specialized sensors. While perception using sensors is getting better, there are still things we will probably never be able to model perfectly, such as contact forces, because they depend on too many parameters to estimate. Still, it is inevitable that contact will occur when manipulating objects in the real world. While it would be possible to handle contacts with an incredibly stiff and therefore high force robot, this is not a desirable approach for several reasons. A high force robot could damage objects or even humans in its environment or destroy itself if it applies too much force during contacts. Therefore, we desire a robot that is compliant and robust to disturbances from its environment to enable safe human-robot interaction.

A common task for evaluating robot performance with contacts are assembly or peg-in-hole tasks, as they represent the core of the problem in a way that is easy to set up in simulations or real-world experiments. Peg-in-hole tasks require the robot to learn how to insert a peg into a hole with a specified clearance; the narrower the hole, the more difficult the task. There are two common ways of learning a peg-in-hole task: either by learning from demonstrations or by learning through interaction with an environment [8]. In learning

from demonstrations, a robot attempts to imitate and perhaps even improve upon given demonstrations. The demonstrations are often provided by a human performing the desired behavior, and are given in the form of sensor data collected during the demonstration process. A major problem with this approach is the general need for demonstrations, since while it may be easy to provide them for a peg-in-hole task, it becomes more complicated as the task becomes more specific. In addition, collecting data for such demonstrations is very expensive, as they must be created by experts. On the other hand, simulations of such complex environments are getting better as computational power increases. The ability to describe tasks in simulation allows learning by interacting with a simulated environment without taking a risk. Ultimately, there is still the challenge of transferring behavior learned in simulation to the real world, but there is already much research aimed at achieving this [9].

A typical approach to learning by interaction with an environment is reinforcement learning (RL), in which the robot receives a reward for each action it performs in a given state of an environment [10]. The idea is that the robot should learn to maximize the expected reward for a given time horizon, leading it to perform actions with higher and higher rewards as learning progresses. For a peg-in-hole or general insertion task, the reward can be defined by the distance to the target pose that describes a correct insertion. The closer the robot gets to successfully inserting the object, the higher the reward should be. While there are several RL methods capable of inserting objects, and some of them are very powerful [11, 12], many of them use some kind of guidance for learning [13, 14, 15]. A special family of these methods that use guidance is called residual reinforcement learning [16], which has been successfully applied to insertion tasks [17, 18, 19]. In these methods, guidance is provided by a nominal policy, which can be represented, for example, by a hand-crafted expert policy or a simple heuristic. In addition, there is another policy, the residual policy, which learns to build on the behavior of the nominal policy and learns, for example, to generate minor corrections to the output to increase performance. These methods are not only very good in terms of performance, but also provide an elegant way to partition a maybe highly biased tuned behavior and learning to improve that behavior. The nominal policy establishes a baseline for performance, and when properly trained, the residual learns to improve that performance by adjusting the output.

The residual learning approach allows it to give more structure to both policies to make them better suited to their specific tasks of guiding and optimizing. For example, such a structure can be an appropriate internal representation of orientations that allows better guidance by the nominal policy when learning to rotate an object. Moreover, such a structure can be biased to provide a stable signal to ensure that the system is driven

toward a desired area. We believe that well-suited priors included in the nominal policy are key to improving the performance of residual approaches.

For this reason, in this work we aim to evaluate whether we are able to improve the performance of residual policies by providing a better structured nominal policy. We address the problem of efficiently learning orientations due to ambiguities in common representations [20], by applying manifold stable vector fields (MSVFs) [21], originally used in behavioral cloning, to RL. An MSVF uses Lie groups to efficiently learn to output velocities not only in terms of position, but also in terms of orientation. It is able to do so by defining such velocities in terms of manifolds that are particularly well suited for this task, i.e., $SO(3)$, $SE(3)$. Moreover, their learned policy is stable and therefore guaranteed to lead to a given stable point, which makes them particularly advantageous for insertion tasks. For this reason, we evaluate their performance when used independently of other policies or when used as a learnable stable prior for residual learning by using them as a nominal policy. We show that while a pure MSVF has problems with RL, the residual MSVF approach not only provides an increase in performance compared to a non-learnable hand-crafted nominal policy, but also performs incredibly well on various insertion tasks, including a simulated 7-DOF Franka robot arm performing the insertion. Furthermore, we propose several adaptations to our approach that can additionally perform variable impedance control or incorporate force sensor observations into the policy.

2. Background

In this section, we will provide an overview of the necessary foundations before explaining the actual approach in Section 4. For this reason, we will first introduce the general concept of reinforcement learning in Section 2.1, which will allow us to explain the methods we use to train our models. Then, in Section 2.2, we will define the foundations of robotics that are necessary to understand how we control and interact with the robotic system in our experiments. Next, in Section 2.3, we need to introduce the basics of manifold theory to introduce Lie groups such as $SO(N)$ and understand the basics of diffeomorphisms. Finally, we will explain the concepts of stable vector fields in Section 2.4 and neural ordinary differential equations in Section 2.5, since both concepts will play a fundamental role in our approach.

2.1. Reinforcement Learning

Reinforcement learning (RL) is about learning a specific behavior through reinforcement. In the context of Machine Learning (ML), we are typically concerned with solving decision making problems in which we attempt to reinforce optimal decisions [22]. Typically, a reinforcement learning task involves an agent that performs actions and an environment that returns observations corresponding to the actions performed by the agent. This means that an agent starts in a certain state, then performs an action and observes a new state and a reward. Thus, through repeated interaction with the environment, the agent can learn which action in which state leads to a positive reward. This learned decision making function is called a policy. An RL algorithm can therefore be described as an algorithm that aims to find a policy that, for any given state, provides an action that maximizes the sum of future (discounted) rewards.

2.1.1. Markov Decision Processes

A common method for mathematical modeling of decision making problems are Markov Decision Processes (MDPs) [23]. An MDP can be defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \mu_0, \gamma)$. This representation encodes what state $\boldsymbol{s}_t \in \mathcal{S}$ the environment is in at a given time t, and what actions $\boldsymbol{a}_t \in \mathcal{A}$ the agent can perform. The reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ defines a reward r_t that an agent receives when it performs an action $\boldsymbol{a}_t$ in a certain state $\boldsymbol{s}_t$. The state $\boldsymbol{s}_{t+1}$ that follows the execution of an action $\boldsymbol{a}_t$ in a previous state $\boldsymbol{s}_t$ is defined by the transition probability $\boldsymbol{s}_{t+1} \sim \mathcal{P}(\cdot|\boldsymbol{s}_t, \boldsymbol{a}_t)$. The initial state distribution $\mu_0(\boldsymbol{s})$ defines the distribution over the initial states $\boldsymbol{s}_0 \in \mathcal{S}_0$ in which the agent can start. Finally, the discount factor $\gamma \in [0, 1)$ is used to regularize (discount) the sum of future rewards.

In an MDP, the reward function describes the problem to be solved and hence the desired solution. A common way to define a reward function is to bind it such that $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to (-\infty, 0]$ and so the maximum discounted reward is not ∞ but 0. So if we want an agent to find the exit of a maze as fast as possible, we can simply define $\mathcal{R}(\boldsymbol{s}, \boldsymbol{a}) = -1$. Each time, the agent starts in an initial state $\boldsymbol{s}_0$ and has a maximum number of time steps $t \in 0, ..., N$ to reach the terminal state $\boldsymbol{s}_T$ of the episode. Then, when the agent tries to maximize the sum of discounted rewards $r_0 + \gamma r_1 + \gamma^2 r_2 + ... + \gamma^t r_t$, the learned policy $\pi : \mathcal{S} \to \mathcal{A}$ minimizes the number of transitions required to reach the terminal state $\boldsymbol{s}_T$ and therefore tries to find a faster and faster path through the maze. To compute a well-defined reward using only the current action and state, MDPs are defined to satisfy the Markov property. That is, the current action $\boldsymbol{a}_t$ and the current state $\boldsymbol{s}_t$ contain all the information needed to determine the next state $\mathcal{P}(\boldsymbol{s}_{t+1}|\boldsymbol{a}_t, \boldsymbol{s}_t) = \mathcal{P}(\boldsymbol{s}_{t+1}|\boldsymbol{a}_t, \boldsymbol{s}_t, \boldsymbol{a}_{t-1}, \boldsymbol{s}_{t-1}, ..., \boldsymbol{a}_0, \boldsymbol{s}_0)$ [22].

In general, MDPs can be divided into several subcategories, e.g., the case we introduced above including a finite horizon of $0 \leq t < N$ are finite MDPs, but there are also cases in which $t \to \infty$. Also, action selection can be deterministic $\boldsymbol{a}_t = \pi(\boldsymbol{s}_t)$ or stochastic $\boldsymbol{a}_t \sim \pi(\boldsymbol{a}|\boldsymbol{s}_t)$. Finally, the state and action spaces $\mathcal{S}, \mathcal{A}$ can be either discrete or continuous.

In summary, an RL algorithm attempts to find an optimal solution to a decision making problem described by an MDP by learning an optimal policy that maximizes the sum of discounted rewards. To solve this problem, the policy seeks to maximize the expected sum of discounted rewards, which is [22] defined as follows

$$J_\pi = \mathbb{E}\left[\sum_{t=0}^{N} \gamma^t \mathcal{R}(\boldsymbol{s}_t, \boldsymbol{a}_t) \middle| \boldsymbol{s}_0 \sim \mu_0(\boldsymbol{s}), \boldsymbol{a}_t \sim \pi(\boldsymbol{a}|\boldsymbol{s}_t), \boldsymbol{s}_{t+1} \sim \mathcal{P}(\boldsymbol{s}|\boldsymbol{s}_t, \boldsymbol{a}_t)\right]. \tag{2.1}$$

A fundamental approach to learning such a policy with respect to Equation 2.1 is to learn a value for each state and then transition to the states with the highest value. These approaches are called value function approaches and are presented in the next section.

2.1.2. Value Function

Value function methods reduce the problem of modeling the expected sum of rewards, as described in Equation 2.1 to modeling the expected sum of rewards in a given state [22,24]. The value function describes the expected sum of discounted rewards of a state $\boldsymbol{s}$ following a given policy π from that state

$$V_\pi(\boldsymbol{s}) = \mathbb{E}\left[\sum_{t=0}^{N} \gamma^t \mathcal{R}(\boldsymbol{s}_t, \boldsymbol{a}_t) \middle| \boldsymbol{s}_0 = \boldsymbol{s}, \boldsymbol{a}_t \sim \pi(\boldsymbol{a}|\boldsymbol{s}_t), \boldsymbol{s}_{t+1} \sim \mathcal{P}(\boldsymbol{s}'|\boldsymbol{s}_t, \boldsymbol{a}_t)\right].$$

A common alternative formulation is the state-action value function (Q-Function), which defines not only the value of a state, but also the value of performing a particular action $\boldsymbol{a}$ in a state $\boldsymbol{s}$. This formulation allows to find an optimal policy by greedily selecting actions for each state $\forall \boldsymbol{s} \in \mathcal{S} : \pi(\boldsymbol{s}) = \arg\max_{\boldsymbol{a}} Q_\pi(\boldsymbol{s}, \boldsymbol{a})$ [22].

In general, it has been shown that for discrete spaces and known transition probabilities, value function methods can be solved by Dynamic Programming (DP) [25]. When the transition dynamics are unknown, two other families of algorithms have been shown to be successful: Monte Carlo [26] and Temporal Difference Methods [27]. For very large or continuous state and action spaces, it is not possible to represent the value function for each state and action, so function approximators are used to represent the value function.

However, there are also methods that attempt to avoid explicitly modeling a value function, called Policy Search methods. These methods assume that the policy π is parameterized by some parameters $\theta \in \Theta$ and intend to find the optimal policy by directly searching in the parameter space [28]. One way to perform such a search is to use gradient ascent, which leads us to Policy Gradient (PG) methods [29] explained in the next section.

2.1.3. Policy Gradient Methods

In general, Policy Search (PS) methods do not explicitly model the value function, but guide the policy search by defining an objective that is used to learn the parameters of the policy. Typically, they attempt to model the expected reward for a given state $\boldsymbol{s}$ and a given action $\boldsymbol{a}$ through

$$J_{\pi_\theta} = \mathbb{E}_{\boldsymbol{s}\sim\mu_0,\boldsymbol{a}\sim\pi_\theta(\cdot|\boldsymbol{s})}[\mathcal{R}(\boldsymbol{s},\boldsymbol{a})] \tag{2.2}$$

which allows to select the states and actions with the highest expected reward.

To optimize the PS objective given in Equation 2.2, Policy Gradient (PG) methods have been shown to not only perform well in high-dimensional state-action spaces, but also enable appropriate learning of stochastic policies [22, 29, 30]. However, PG methods assume that the policy is differentiable, since they use an estimate of the gradient of the objective function to update the parameters of the policy using gradient ascent

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\nabla_{\boldsymbol{\theta}} J_{\pi_\theta} \tag{2.3}$$

where α is a learning rate that defines the step size for updating the parameters. The use of the gradient allows to define a policy update towards a local optimum of the target. Therefore, a major drawback and an important optimization aspect is to direct the policy not only towards a local optimum, but towards a global optimum.

Another problem is to estimate the value of $\nabla_{\boldsymbol{\theta}} J_{\pi_\theta}$ for the update described in Equation 2.3. The Policy Gradient Theorem introduces an estimate of the policy gradient by

$$\nabla_{\boldsymbol{\theta}} J_{\pi_\theta} \propto \mathbb{E}_{\boldsymbol{s}\sim\mu_0,\boldsymbol{a}\sim\pi_\theta(\cdot|\boldsymbol{s})}[\mathcal{R}(\boldsymbol{s},\boldsymbol{a})\nabla_\theta \log \pi_\theta(\boldsymbol{a}|\boldsymbol{s})] \tag{2.4}$$

where the log derivative trick is used to formulate the policy gradient in terms of the reward and the log derivative of the policy [29]. In practice, the reward is often modeled by the expected sum of discounted rewards and thus is often represented by the state-action value function Q_{π_θ}.

Since both value methods and policy search methods, such as the policy gradient, have their advantages and disadvantages, a joint approach called Actor-Critic algorithms has been introduced [31]. These algorithms use a critic that estimates the value function and an actor that updates the policy in the direction suggested by the critic, as explained in the next section.

2.1.4. Actor Critic Approaches

The idea of Actor-Critic methods is to combine the advantages of value methods and policy search methods in a single approach. The policy search methods represented by the actor allow to compute actions in the continuous domain without relying on the direct optimization of the value function. On the other hand, the value part represented by the critic allows measuring the current performance of the actor with low variance. Since this performance measure is initially quite biased due to inaccurate estimates of the critic at the beginning of learning, actor-critic methods suffer from a larger bias at the beginning of learning. However, these methods have generally been shown to have good convergence properties [31, 32].

Using the objective introduced in Equation 2.4 and employing the state-action value function as an estimate, we obtain the objective and update for the actor π_θ of the Q-Actor-Critic approach as follows:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J_{\pi_\theta} &\propto \mathbb{E}_{\boldsymbol{s}\sim\mu_0,\boldsymbol{a}\sim\pi_\theta(\cdot|\boldsymbol{s})}[Q_\omega(\boldsymbol{s},\boldsymbol{a})\nabla_\theta \log \pi_\theta(\boldsymbol{a}|\boldsymbol{s})] \\ \boldsymbol{\theta} &\leftarrow \boldsymbol{\theta} + \alpha_\theta \nabla_{\boldsymbol{\theta}} J_{\pi_\theta}\end{aligned}$$

where $\boldsymbol{\theta}$ are the actor's parameters, $\boldsymbol{\omega}$ are the critic's parameters and α_θ is the learning rate of the actor. To update the critic's parameters, we need to compute a correction of the state-action value function. To compute this correction, after sampling and executing an action $\boldsymbol{a}_t \sim \pi_\theta(\cdot|\boldsymbol{s}_t)$, we receive a reward r_t and a next state $\boldsymbol{s}_{t+1} \sim \mathcal{P}(\cdot|\boldsymbol{s},\boldsymbol{a})$. Now we again sample an action from the next state $\boldsymbol{a}_{t+1} \sim \pi_\theta(\cdot|\boldsymbol{s}_{t+1})$. Therefore, the critic's objective and update for the Q-Actor-Critic can be computed as

$$\begin{aligned}\delta_t &= r_t + \gamma Q_\omega(\boldsymbol{s}_{t+1},\boldsymbol{a}_{t+1}) - Q_\omega(\boldsymbol{s}_t,\boldsymbol{a}_t) \\ \boldsymbol{\omega} &\leftarrow \boldsymbol{\omega} + \alpha_\omega \delta_t \nabla_\omega Q_\omega\end{aligned} \tag{2.5}$$

where α_ω is the learning rate of the critic. In practice the so called advantage function defined as

$$A(\boldsymbol{s},\boldsymbol{a}) = V(\boldsymbol{s}) - Q(\boldsymbol{s},\boldsymbol{a}),$$

and is used to replace the state-action value function. It models the benefit of choosing an action $\boldsymbol{a}$ in a certain state $\boldsymbol{s}$ and is known to reduce the variance of the estimate.

Moreover, the update of the critic can be either on-policy or off-policy [33]. The on-policy case is the one we introduced in Equation 2.5. It is considered an on-policy method because the update is performed according to the agent's policy, since the next action is selected by the policy $\boldsymbol{a}_{t+1} \sim \pi_\theta(\cdot|\boldsymbol{s}_{t+1})$. On the other hand, if the update is off-policy, the next action

used to update the critic is selected according to a different policy. The simplest case for the off-policy setting is to use a greedy policy for the update $\boldsymbol{a}_{t+1} = \arg\max_{\boldsymbol{a}} Q(\boldsymbol{s}_{t+1}, \boldsymbol{a})$. In general, off-policy methods provide a better exploration because the exploration policy can be chosen to be different from the target policy, in addition they have also been shown to be used successfully with data from human demonstrations. Furthermore, they can be considered more consistent, since in the on-policy case the initial policy can be arbitrary bad, while in the off-policy case a reasonable exploration policy can be defined. Nevertheless, on-policy methods are still widely used, since training off-policy methods is known to be a very hard problem [22, 33].

2.1.5. Model-free and Model-based Reinforcement Learning

So far, we have always assumed that the state transition probabilities $\mathcal{P}$ are known. However, this is often not the case, especially for environments in the context of robotics. In practice, the agent learns not by assuming a known state transition probability, but by gaining experience through interaction with the environment. This means that an agent collects information in the form of tuples $(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1})$.

To learn a policy based on the collected experience, one can either attempt to learn a model of the state transition probabilities (model-based RL) or not model them at all (model-free RL) [34]. An important motivation for model-based methods is that sampling data from a real environment is very expensive. Collecting data is very time consuming, and if a model can be learned, the data can be sampled efficiently using the model rather than the real system. However, learning a model is quite difficult, and even the smallest error in the model can lead to a large error in the estimates. Model-free methods, while quite sample inefficient, are generally easier to learn and do not suffer from the biases introduced by collecting samples from an error-prone model [34, 35].

2.1.6. Proximal Policy Optimization

After discussing different RL methods such as actor-critic approaches as well as on-policy and off-policy methods or model-based and model-free approaches, we will now to introduce Proximal Policy Optimization (PPO) [36], which we will use as an optimization method throughout this thesis. PPO can be defined as an on-policy and model-free method that combines Policy Gradient methods [30] with Trust Region Policy Optimization (TRPO) [37] into a joint actor-critic approach.

In TRPO, a so-called surrogate objective is maximized in order to learn the parameters of the policy. This surrogate objective is formulated as a constrained optimization problem that considers the probability ratio of the policy before and after an update, constrained by its Kullback-Leibler (KL) divergence. TRPO formulates this problem as follows

$$\begin{aligned} &\arg\max_\theta \mathbb{E}_t \left[\frac{\pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t)}{\pi_{\theta_{\text{old}}}(\boldsymbol{a}_t|\boldsymbol{s}_t)} A(\boldsymbol{s}_t, \boldsymbol{a}_t) \right] \\ &\text{subject to } \mathbb{E}_t \left[\text{KL}[\pi_{\theta_{\text{old}}}(\cdot|\boldsymbol{s}_t), \pi_\theta(\cdot|\boldsymbol{s}_t)] \right] \leq \delta \end{aligned} \tag{2.6}$$

where $\boldsymbol{\theta}_{\text{old}}$ are the policy parameters before the update. By using the conjugate gradient algorithm, this problem can be efficiently approximated [37]. Instead of solving the constrained optimization problem given in Equation 2.6, it is also possible to formulate the object in terms of an unconstrained optimization problem

$$\arg\max_\theta \mathbb{E}_t \left[\frac{\pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t)}{\pi_{\theta_{\text{old}}}(\boldsymbol{a}_t|\boldsymbol{s}_t)} A(\boldsymbol{s}_t, \boldsymbol{a}_t) - \beta \text{KL}[\pi_{\theta_{\text{old}}}(\cdot|\boldsymbol{s}_t), \pi_\theta(\cdot|\boldsymbol{s}_t)] \right] \tag{2.7}$$

where β is a penalty coefficient. However, there are additional modifications needed to solve this unconstrained optimization problem, since simply choosing a fixed value for β does neither perform well across different problems, nor within a single one.

Proximal Policy Optimization (PPO) combines the idea of the unconstrained optimization problem following TRPO (Equation 2.7) with policy gradient methods. For each training iteration, the policy $\pi_{\theta_{\text{old}}}$ is run for $t = 0, 1, ..., N$ time steps and for each state-action pair $(\boldsymbol{s}_t, \boldsymbol{a}_t)$ the advantage function $\hat{A}_t = A(\boldsymbol{s}_t, \boldsymbol{a}_t)$ is computed. The collected data can then be used to perform Stochastic Gradient Descent (SGD) for the optimization problem formulated in Equation 2.7. Interestingly, the authors found that using a different objective that does not use the KL divergence is superior. The new objective is given by

$$\arg\max_\theta \mathbb{E}_t \left[\min \left(\frac{\pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t)}{\pi_{\theta_{\text{old}}}(\boldsymbol{a}_t|\boldsymbol{s}_t)} \hat{A}_t, \text{clip} \left(\frac{\pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t)}{\pi_{\theta_{\text{old}}}(\boldsymbol{a}_t|\boldsymbol{s}_t)}, 1-\epsilon, 1+\epsilon \right) \hat{A}_t \right) \right] \tag{2.8}$$

where ϵ is a hyper-parameter. This objective formulates a lower bound on the unclipped objective and can be extended by adding an entropy term to ensure sufficient exploration [36]. In the actor-critic style, PPO updates the critic and the actor independently. The critic represents the advantage function and can be updated as previously described, and the actor is updated by using SGD to optimize the objective in Equation 2.8. This method has proven to be simple yet powerful in most scenarios.

How exactly the critic or actor is represented is a particularly important design choice. With the rise of neural networks (NN) in ML, they have been shown to be incredibly well suited to representing such parameterized policies and training them via SGD [38]. Also, in the case of RL, there are many state-of-the-art approaches that use NNs to represent their policies [39]. This is also the reason why in this work we focus on using NNs to represent the actor and the critic and train them using PPO.

2.2. Robotics

Robotics deals with robots of all kinds, which usually consist of rigid bodies connected by joints. While it is not necessary for the links between joints to be rigid, and in practice the joints themselves may not even be completely rigid, we will ignore those effects in the remainder of this work, as they are particularly difficult to model whilst having only minor influences [40].

2.2.1. Representation of a Robot

We can describe the configuration of a robot by the state of its joints. This means that if we know the dimensions of the links and joints, the type of joints (revolute or prismatic) and the way they are connected, we can describe the complete configuration of each point of the robot in terms of the configuration space. Thus, if our robot has N_{DOF} joints, the configuration space has N_{DOF} dimensions and each dimension represents the state of the corresponding joint. Usually, the joint configuration of a robot is denoted as $\boldsymbol{q}$, the joint velocity as $\dot{\boldsymbol{q}}$ and the joint acceleration as $\ddot{\boldsymbol{q}}$.

Even if the configuration space is able to describe the robot configuration, we often use robots that are equipped with some kind of gripper, hand or similar. In these cases, we are often more interested in the position of the robot's end-effector, rather than in all of its joints. We therefore call the space of positions and orientations of the end-effector the task space. Here the current position $\boldsymbol{p}$ and rotation $\boldsymbol{q}$ can be represented by an n-dimensional vector $\boldsymbol{x}$ and again the velocities as $\dot{\boldsymbol{x}}$ and the accelerations as $\ddot{\boldsymbol{x}}$. The number of dimensions of $\boldsymbol{x}$ depends on how the orientation is represented. Positions are usually represented in Euclidean space and are therefore 3-dimensional. However, there are various representations for rotations such as quaternions, rotation matrices or Euler angles, all of which have their advantages and disadvantages, which we will further elaborate throughout this thesis.

2.2.2. Kinematics and Dynamics

In robotics, we often need to describe the kinematics and dynamics of a robot [40]. The kinematics describe the relationship between the position and orientation of the end-effector and the joint positions. The forward kinematics

$$f(\boldsymbol{q}) = \boldsymbol{x} \tag{2.9}$$

describes a mapping from the configuration space to the task space. In this context, the Jacobian $\boldsymbol{J}$ is particularly interesting, since it allows to formulate the differential forward kinematics as

$$\dot{\boldsymbol{x}} = \frac{\partial f(\boldsymbol{q})}{\partial \boldsymbol{q}} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}},$$

which describes a mapping to the velocities $\dot{\boldsymbol{x}}$ in the task space. It is important to notice that the Jacobian $\boldsymbol{J}(\boldsymbol{q})$ is therefore a mapping from configuration space velocities to task space velocities.

On the other hand, the dynamics describes the relationship between the configuration space and the resulting torques. This is, the dynamics takes into account the current joint configuration $(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})$ and maps it to the joint torques $\boldsymbol{\tau}$. In general, the dynamics of a robot can be formulated as

$$\boldsymbol{\tau} = \boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{g}(\boldsymbol{q}) \tag{2.10}$$

where $\boldsymbol{M}$ represents the mass of the robot, $\boldsymbol{C}$ represents the Coriolis and centripetal forces, and $\boldsymbol{g}$ is the gravitational force. Theoretically, there would also be a term modeling joint friction and other factors. This additional term is often not considered in practice due to its complexity and negligible influence, also the term $\boldsymbol{C}$ is often not explicitly modeled. By rewriting Equation 2.10, we can formulate the forward dynamics as follows

$$\begin{aligned}\ddot{\boldsymbol{q}} &= f(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{u}) \\ &= \boldsymbol{M}^{-1}(\boldsymbol{q})\big(\boldsymbol{u} - \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) - \boldsymbol{g}(\boldsymbol{q})\big)\end{aligned}$$

which maps a given control signal $\boldsymbol{u}$ in torques and robot configuration to the resulting joint acceleration.

2.2.3. Robot Control

In order to use control signals to control a robot, a controller is used to compute these signals. One of the simplest forms of such a controller is a P-controller [40] which is based on the position error in the joints

$$\boldsymbol{u} = \boldsymbol{K}_P(\boldsymbol{q}_{des} - \boldsymbol{q})$$

where $\boldsymbol{q}_{des}$ are the desired joint values and the matrix $\boldsymbol{K}_P$ is a gain matrix indicating the influence of the position error on the control signals.

The P-controller often suffers from overshoot and oscillation because it only considers the current position and not the current velocity in the joints. To dampen the robot and counteract the overshoot, PD-controllers have proven to be very useful and can be formulated as

$$\boldsymbol{u} = \boldsymbol{K}_P(\boldsymbol{q}_{des} - \boldsymbol{q}) + \boldsymbol{K}_D(\dot{\boldsymbol{q}}_{des} - \dot{\boldsymbol{q}}) \tag{2.11}$$

where $\dot{\boldsymbol{q}}_{des}$ are the desired joint velocities and the matrix $\boldsymbol{K}_D$ now defines the influence of the velocity error on the control signals. Often a desired velocity of $\dot{\boldsymbol{q}}_{des} = 0$ is chosen to guide the system to a stable point with zero velocity. Also, $\boldsymbol{K}_D, \boldsymbol{K}_P$ must be a positive definite matrix to ensure a stable system [40].

2.2.4. Jacobian Transpose Method

A fundamental problem in robot control is that there are multiple configurations in the configuration space that lead to a single configuration in the task space [40, 41]. Thus, if we want to drive our robot end-effector to a specific pose $\boldsymbol{x}_{des}$, there are multiple solutions $f^{-1}(\boldsymbol{x}_{des}) = \boldsymbol{q}_{des}$ for the inverse of the forward kinematics introduced in Equation 2.9.

The Jacobian Transpose method [41] approaches the problem by minimizing the squared task space error

$$E = \frac{1}{2}(\boldsymbol{x}_{des} - f(\boldsymbol{q}))^T(\boldsymbol{x}_{des} - f(\boldsymbol{q}))$$

and uses its gradient to formulate a gradient descent update for the desired joint velocity

$$\begin{aligned}\nabla_{\boldsymbol{q}} E &= (\boldsymbol{x}_{des} - f(\boldsymbol{q}))^T \frac{\partial f(\boldsymbol{q})}{\partial \boldsymbol{q}} = (\boldsymbol{x}_{des} - f(\boldsymbol{q}))^T \boldsymbol{J}(\boldsymbol{q}) \\ \boldsymbol{q}_{des} &= \boldsymbol{q} - \alpha(\nabla_{\boldsymbol{q}} E)^T \\ &= \boldsymbol{q} - \alpha\big((\boldsymbol{x}_{des} - f(\boldsymbol{q}))^T \boldsymbol{J}(\boldsymbol{q})\big)^T \\ &= \boldsymbol{q} - \alpha \boldsymbol{J}^T(\boldsymbol{q})(\boldsymbol{x}_{des} - f(\boldsymbol{q})),\end{aligned}$$

where α controls the step size for updating the desired joint configuration. Using this method, a desired joint configuration $\boldsymbol{q}_{des}$ can be computed for a desired pose $\boldsymbol{x}_{des}$ in the task space. This desired joint configuration can be passed to a PD-controller as introduced in Equation 2.11 and results in minimizing the error in the task space.

2.2.5. Operational Space Control

In classical robotics, stiff high-gain control has often been used to maximize the accuracy when following given trajectories. While this type of control can be used in mostly static environments without interference from humans, it is very dangerous to use such stiff high-gain robots in dynamic environments. For this reason, modern research focuses on compliant, redundant robots that do not require such high gains, but are controlled with a reasonable amount of force. This is also the motivation behind Operational Space Control (OSC), which attempts to achieve compliant behavior while defining a desired trajectory in task space [42, 43, 44].

Although OSC defines the desired pose in the task space, just like the the Jacobian Transpose does, OSC models this problem in a different way. It attempts to define a mapping from end-effector forces to the corresponding control signal. For this reason, we can use the Jacobian transpose with respect to the end-effector $\boldsymbol{J}^T(\boldsymbol{q})$ to formulate this mapping, which leads to

$$\boldsymbol{u} = \boldsymbol{J}^T(\boldsymbol{q})\boldsymbol{F}_{ee},$$

where $\boldsymbol{F}_{ee}$ is the end-effector force. To model the forces at the end-effector, OSC rewrites the force in terms of an OSC-specific mass matrix $\boldsymbol{M}_{osc}$ and the desired end-effector acceleration $\ddot{\boldsymbol{x}}_{des}$, which results in

$$\boldsymbol{u} = \boldsymbol{J}^T(\boldsymbol{q})\boldsymbol{M}_{osc}(\boldsymbol{q})\ddot{\boldsymbol{x}}_{des}$$

bringing us to the problem of modeling $\boldsymbol{M}_{osc}$. It has been shown that this mass matrix can be modeled as follows

$$\boldsymbol{M}_{osc}(\boldsymbol{q}) = [\boldsymbol{J}^T(\boldsymbol{q})\boldsymbol{M}^{-1}(\boldsymbol{q})\boldsymbol{J}^T(\boldsymbol{q})]^{-1} \tag{2.12}$$

where $\boldsymbol{M}$ is the mass matrix of the robot [43]. In order to define the desired acceleration, a typical PD-controller can be used

$$\ddot{\boldsymbol{x}}_{des} = \boldsymbol{K}_P(\boldsymbol{x}_{des} - \boldsymbol{x}) + \boldsymbol{K}_D(\dot{\boldsymbol{x}}_{des} - \dot{\boldsymbol{x}})$$

and if we then add also gravity compensation, we can formulate the full OSC control by

$$\boldsymbol{u} = \boldsymbol{J}^T(\boldsymbol{q})\boldsymbol{M}_{osc}(\boldsymbol{q})[\boldsymbol{K}_P(\boldsymbol{x}_{des} - \boldsymbol{x}) + \boldsymbol{K}_D(\dot{\boldsymbol{x}}_{des} - \dot{\boldsymbol{x}})] + \boldsymbol{g}(\boldsymbol{q}) \tag{2.13}$$

where the desired velocity is often set to zero $\dot{\boldsymbol{x}}_{des} = 0$ to end up with a stable pose.

2.2.6. Null Space Control

Complex robots usually have more degrees of freedom than the dimensions of their task space. This means that the desired pose in the task space does not completely constrain the pose of the robot and therefore there are multiple joint configurations for the same pose in the task space. This characteristic makes it possible to define a secondary task that drives the robot to a specific rest position. In order not to interfere the primary objective of the controller, the secondary controller operates in the null space of the first controller, which is known as Null Space Control [40].

If we want to adjust OSC to combine the control signal $\boldsymbol{u}$ defined in Equation 2.13 and the control signal $\boldsymbol{u}_{null}$ from the secondary controller, we can define

$$\boldsymbol{u}_{total} = \boldsymbol{u} + (\boldsymbol{I} - \boldsymbol{J}^T(\boldsymbol{q})\boldsymbol{J}^{T\dagger}(\boldsymbol{q}))\boldsymbol{u}_{null}$$

where $J^{T\dagger}(\boldsymbol{q})$ is the pseudo-inverse of $\boldsymbol{J}^T(\boldsymbol{q})$. It is important to choose this pseudo-inverse such that it cancels the signal $\boldsymbol{u}_{null}$ everywhere except in the null space. This behavior has been shown to be achieved by setting

$$\boldsymbol{J}^{T\dagger}(\boldsymbol{q}) = \boldsymbol{M}_{osc}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q})\boldsymbol{M}^{-1}(\boldsymbol{q})$$

where $\boldsymbol{M}_{osc}(\boldsymbol{q})$ is given by Equation 2.12 [42]. To allow for stabilized null-space movements, the control signal of the secondary controller can be defined as

$$\boldsymbol{u}_{null} = \boldsymbol{M}(\boldsymbol{q})[\boldsymbol{K}_{P_{null}}(\boldsymbol{q}_{null} - \boldsymbol{q}) - \boldsymbol{K}_{D_{null}}(\dot{\boldsymbol{q}})]$$

where $\boldsymbol{q}_{null}$ is the desired rest position and $\boldsymbol{K}_{P_{null}}, \boldsymbol{K}_{D_{null}}$ are specific gains for the secondary controller [45].

2.3. Manifold Theory

In general, Euclidean space and its geometry are well known. It is easy to define distances or to draw triangles in $\mathbb{R}^2$ because we know that the angles in a triangle in Euclidean space must sum up to $180°$. Although we consider the Euclidean space $\mathbb{R}^3$ as the space we live in, for certain tasks it is fundamental to consider other spaces as well. Suppose one has the task of drawing a triangle on the surface of a sphere. Do the general rules we know from the Euclidean space then also apply to the space defined by the surface of the sphere? No, because the angles of a triangle on the surface of a sphere do not add up to $180°$, and we cannot use the Euclidean distance metric to measure the distance between two points on that surface. For this reason, manifolds are incredibly important, especially for describing rotations in robotics, as they give us exactly the tools we need to deal with, for example, the space described by the surface of a sphere.

2.3.1. Manifolds

An n-manifold $\mathcal{M}$ (more precisely: topological manifold) is a topological space that is locally homeomorphic to a Euclidean space, e.g. a sphere is a 3-dimensional manifold that locally looks like the 2-dimensional Euclidean space. Specifically, this means that every point must have a neighborhood and for that neighborhood there must exist a homeomorphism that maps the neighborhood to a Euclidean space $\mathbb{R}^n$, called a chart of the manifold. In this case, an atlas defines a particular collection of charts covering the manifold, and a transition map is a function that allows transition between charts in overlapping parts of the atlas's charts [46].

Such a manifold is called differentiable if it has a globally defined differential structure [47], which means that the transition maps are differentiable functions on the corresponding vector space. For such differentiable manifolds $\mathcal{M}$, one can attach to each point $\boldsymbol{x} \in \mathcal{M}$ a tangent space $\mathcal{T}_{\boldsymbol{x}}\mathcal{M}$. This tangent space is a real vector space in $\mathbb{R}^n$ containing all possible vectors that are tangential at $\boldsymbol{x}$ [48]. Moreover, a smooth manifold describes a differentiable manifold with infinitely differentiable transition maps [48].

It is also possible to define a diffeomorphism [47] between two smooth manifolds $\mathcal{M}, \mathcal{N}$. This diffeomorphism is a differentiable map $\phi : \mathcal{M} \to \mathcal{N}$, which is not only a bijection (each element in $\mathcal{M}$ maps to exactly one element in $\mathcal{N}$), but also its inverse $\phi^{-1} : \mathcal{N} \to \mathcal{M}$ is differentiable.

2.3.2. Lie Groups

We call a mathematical group $\mathcal{G}$ a Lie group [49] if it is also a differentiable manifold. Associated with each Lie group there is a Lie algebra $\mathfrak{g}$, which is a vector space that is the tangent space of the Lie group at its identity element.

To connect a Lie group and its Lie algebra, two important concepts can be defined: the exponential map and its inverse, the logarithmic map. In a Lie group $\mathcal{G}$, the exponential map is a function $\text{ExpMap} : \mathfrak{g} \to \mathcal{G}$ that maps an element in the Lie algebra back to the Lie group. Its inverse, the logarithmic map, is a function $\text{LogMap} : \mathcal{G} \to \mathfrak{g}$, which maps an element in the Lie group to an element in its Lie algebra [49].

An important class of Lie groups is the special orthogonal group [49] denoted as $SO(N)$ whose Lie algebra is called $\mathfrak{so}(N)$. This group consists of all orthogonal matrices that have a determinant of 1. This class is so important because, for example, $SO(2)$ represents rotation around a point in 2 dimensions and $SO(3)$ represents rotation around a line in 3 dimensions. So with the Lie groups $SO(2), SO(3)$ we can not only represent rotations, but also transform rotations into the tangent space and thus define a metric that moves smoothly in the direction of a given rotation. This is also the reason why we want to use the structure of the special orthogonal groups to reason about rotations in our approach.

2.4. Stable Vector Fields

In order to introduce stable vector fields (SVF), we will first give an introduction to ordinary differential equations to also define the meaning of stability. This will then allow us to apply these ideas to the concept of vector fields.

2.4.1. Differential Equations

Systems of ordinary differential equations (ODEs) [50] are widely used in mathematics and physics. More precisely, since the laws of physics hold at all time, so-called autonomous differential equations of the form

$$\frac{d}{dt}\boldsymbol{x}(t) = f(\boldsymbol{x}(t))$$

are very common in physics and robotics because they depend only on the current state of the system $\boldsymbol{x}$. In this work, we will focus on autonomous differential equations and therefore simplify the notation by using $\boldsymbol{x}(t) = \boldsymbol{x}$ and similarly $\dot{\boldsymbol{x}}$ represents the first derivative of $\boldsymbol{x}$ with respect to time t.

This allows us to write an example of an autonomous differential equation as

$$\dot{\boldsymbol{x}} = -\boldsymbol{x} \tag{2.14}$$

which states that the velocity of the system $\dot{\boldsymbol{x}}$ is equal to its negative position $\boldsymbol{x}$. This equation has an equilibrium point $\bar{\boldsymbol{x}} = 0$ because once $\boldsymbol{x} = 0$, it follows that $\dot{\boldsymbol{x}} = 0$ and the system will never leave the state of $\boldsymbol{x} = \dot{\boldsymbol{x}} = 0$ by itself. Moreover, this is also a stable equilibrium because even if there are external influences that cause the system to leave the equilibrium point, its formulation will always drive it back to that point.

This stable behavior to an equilibrium point is highly desirable in many robot tasks. Imagine a robot is asked to push a button. Would it not be beneficial to be able to guarantee that the robot will end up pushing the button regardless of small disturbances? The Equation 2.14 is not only a stable autonomous differential equation, but also defines a so-called stable vector field, as will be explained in the next section.

2.4.2. Stable Vector Fields

A vector field [51] is a function $V : S \to \mathbb{R}^N$ that assigns a vector to each point in a given space S. For example, a vector field can describe the relationship between position and velocity of an object. Suppose we have an object position $\boldsymbol{x} \in \mathbb{R}^3$ and define its behavior by Equation 2.14. Then we define a vector field $V : \mathbb{R}^3 \to \mathbb{R}^3$, which maps each position of the object to its velocity by

$$\dot{\boldsymbol{x}} = V(\boldsymbol{x}) = -\boldsymbol{x}$$

and not only follows the same properties as Equation 2.14, but is exactly the same. This means that the given autonomous differential equation is the vector field itself. This also means that the defined vector field is again stable to the equilibrium point $\bar{\boldsymbol{x}} = 0$.

Vector fields cannot be defined only for mappings from one Euclidean space to another Euclidean space. Considering a differentiable manifold $\mathcal{M}$, then a vector field on $\mathcal{M}$ associates a tangent vector for each point in $\mathcal{M}$ [52]. This is particularly important since it allows us to define stable vector fields on, for example, the Lie groups $SO(2)$ and $SO(3)$.

2.5. Neural Ordinary Differential Equations

The concept of neural ordinary differential equations (NODEs) [53] is relatively new, but already has attracted considerable interest. A NODE is in some ways similar to a residual neural network (ResNet), a classical ResNet composes a sequence of hidden states as

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \boldsymbol{H}_{\boldsymbol{\theta}}(\boldsymbol{x}_t),$$

where $\boldsymbol{H}_{\boldsymbol{\theta}}$ is the ResNet and $\boldsymbol{x}_t$ is the hidden state at a given time t. While NODEs follow the same concept, their intent is to approach this problem similarly to an ordinary differential equation. Therefore, they define a NN that learns to map a state to its derivative $\boldsymbol{H}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) = \dot{\boldsymbol{x}}_t$. This means that we can then use an ODE solver to find the next state $\boldsymbol{x}_{t+1}$, for example we can define

$$\boldsymbol{x}_{t_1} = \boldsymbol{x}_{t_0} + \int_{t_0}^{t_1} \boldsymbol{H}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)dt$$

and approximate the integral using, for example, the forward Euler method. This may be a simple modification, but it provides a powerful method allowing for variable depths. In addition, NODEs can be run backwards to easily invert the mapping using

$$\boldsymbol{x}_{t_0} = \boldsymbol{x}_{t_1} - \int_{t_0}^{t_1} \boldsymbol{H}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)dt,$$

which is often a desirable but difficult to achieve property of a neural network. Furthermore, it has been shown how to perform efficient backpropagation by avoiding backpropagation through the ODE solver and instead using the adjoint sensitivities method [53].

3. Related Work

Since the goal of this work is to provide a learnable stable prior for residual reinforcement learning in insertion tasks, we will summarize related approaches below. First, in Section 3.1, we will introduce not only alternative residual approaches, but learning-based approaches for insertion or assembly tasks in general. Then, we will discuss some methods that provide stable policies for robot tasks in Section 3.2.

3.1. Robot Learning for Insertion Tasks

Assembly or insertion tasks are the focus of a wide range of research [8, 54]. In general, these tasks require the control of a robot in a contact-rich environment in order to assemble or insert an object. This task is special because it combines several challenges that must be overcome. First, the control must be very accurate not only for the position, but also for the orientation. Second, the approach must also be able to be robust against contact forces that occur when the object is inserted. Finally, it is highly desirable to achieve a compliant behavior so as not to generate excessive forces that could harm objects or even humans in the vicinity of the robot.

One group of work focuses on learning such tasks using demonstrations, often provided by human experts [55, 56, 57]. A general framework for teaching robot peg-in-hole tasks using human demonstrations was presented in [56]. They propose to mimic the human behavior by using Gaussian Mixture Regression in combination with a Dimension Reduction and Recovery method. In another work that also uses Gaussian Mixture Regression [57], admittance gains are learned that define how large a corrective velocity should be according to a measured contact wrench. In [55], a Gaussian process regression is used to assemble large, heavy components. The proposed method predicts and compensates for object deformation in order to perform assembly on hard-to-measure, easy-deformation

tasks. While learning from demonstrations is a powerful way to imitate a behavior, it requires demonstrations which are particular expensive when created by human experts.

For this reason, there are many works that do not learn from demonstrations but use RL methods [11, 12, 13, 14]. In [11], a high precision peg-in-hole task is solved by training a recurrent neural network with RL using force and position sensors that provide observations for learning. The method proposed in [14] combines an actor-critic RL approach with supervised learning and initially guides the policy using trajectory optimization methods. In [12], self-supervision is used to learn a compact representation of the sensory inputs, which improves the sample efficiency. They do not only use force information, but combine it with additional visual observations. To increase the learning performance for the insertion task, [13] uses CAD data to create a geometric motion plan that serves as a prior to guide the policy while learning.

Other methods also use RL, but focus on compliant behavior [58,59,60]. In [58], a dynamic movement primitive is trained using position and force trajectories from a demonstration of a peg-in-hole task. This is then combined with a control policy that adjusts the impedance parameters in an online fashion, resulting in higher robustness and performance. The uncertainty of hole's position is tackled in [59] and the approach is successfully transferred to a real robot system by using sim2real and domain randomization techniques. Deep RL approaches are also able to handle a mixture of deformable and rigid objects, as [60] shows, again by using force-torque information from force sensors.

Finally, there are also methods that combine multiple policies to increase insertion performance [15, 16, 17, 18]. GUAPO [15] combines a model-based method to drive the system to an uncertainty area, and when this is reached, a reinforcement learning policy, learned through image and velocity information, takes over. In residual policy learning [16] a nominal non-differentiable policy is combined with a model-free deep RL policy. The second policy learns a residual on the first policy and therefore has the purpose of improving the performance of the nominal policy or being guided by the nominal policy to learn more efficiently. In [17], this strategy is used to control a real-world robot performing an assembly task. The approach proposed in [18] also uses a residual policy but uses a dynamic movement primitive learned through demonstrations as its nominal policy.

We believe that these multiple policy approaches, where each policy attempts to solve a particular part of the task, are a very promising approach for modeling insertion tasks. Not only have they already proven to be very successful, but they also allow the task to be clearly partitioned and provide specialized structures for each policy, which could be key to further improving performance.

3.2. Learnable Stable Policies for Robotics

As we have presented many approaches that solve various robot tasks, none of them guarantees stability. This is not because stability is not a desirable property, but because it is very difficult to achieve. However, stability is important if we want to transfer robot learning to real-world scenarios. In real-world tasks, stability allows us to guarantee that a robotic system behaves as we expect it to, which is especially important when we need to consider safety.

Recent work also attempt to learn robot tasks while guaranteeing stability [21, 61, 62]. In [61], variable impedance control is combined with a novel Evolution Strategy policy inspired by the Cross-Entropy Method. It guarantees stability by ensuring that the sample distribution can only generate stable parameter samples. The approach is evaluated on several RL insertion tasks in simulation and also in the real world. A major drawback of this approach is that it is only capable of using Euler angles. Another approach to guarantee stability is proposed in [62], where stable normalizing-flows are learned using RL to control a robot. Here, stability is not guaranteed during exploration, but the learning process produces a deterministic controller with provable stability in the end. Again, there is to mention a drawback in terms of handling orientations, as these are not learned at all, but the authors claim that it is possible to extend their approach to also include orientations. An approach that is capable of providing stability and using efficient representations for orientations are manifold stable vector fields (MSVFs) [21]. In this approach, a latent stable vector field is learned on Lie groups to efficiently learn a stable policy that can handle orientations. Moreover, this approach shows that it is very robust to perturbations, but it has not yet been applied to RL since it was proposed using learning from demonstrations. Still we believe that it is possible to apply MSVFs to RL and use them efficiently as a learnable stable prior for residual reinforcement learning, which we will show throughout the next chapters.

4. Learnable Stable Priors for Residual Reinforcement Learning

For the application of robotics to real-world tasks, it is very important to learn to control under contact, which is why contact-rich tasks are becoming increasingly important in current research [6, 7, 8, 54]. The challenge in these tasks is that contact is very hard to model and new strategies need to be developed to overcome this issue. One group of tasks that are easy to construct, but still difficult to solve, are peg-in-hole tasks or, more general, insertion tasks [8]. To solve such tasks, several strategies have proven to be successful. Some of them achieved very high precision [11], others used a guidance to improve the learning process [15, 63]. There is also a group of tasks that use residual policy learning [16] in order to train a residual policy that is combined with a nominal policy [17, 18, 64].

Since these residual methods are already very effective and achieve decent performance, we believe that a particularly important choice for increasing performance is the nominal policy. Typically, the residual policy is a type of NN that learns to provide minor corrections to the nominal policy. The nominal policy usually guides the learning and is often handcrafted or learned a priori, but is still a crucial optimization component. We aim to focus on the nominal policy and evaluate which representations for the nominal policy provide an appropriate prior for insertion tasks. In order to do so, we will propose a novel method that applies manifold stable vector fields (MSVFs) [21] to reinforcement learning and combines them with a residual policy to achieve an even better performance than classical residual learning with a fixed nominal policy. We choose MSVFs as they are not only a learnable stable policy, but also use Lie groups to efficiently represent rotations [21], resulting in an overall compliant and globally stable policy.

We will define our residual MSVF approach by learning the mean of a Gaussian distribution $\boldsymbol{a} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$ where the $\boldsymbol{\sigma}$ is the learned variance and $\boldsymbol{\mu}$ is the mean given by the residual approach $\boldsymbol{\mu} = \pi(\boldsymbol{x}) = \pi_{nominal}(\boldsymbol{x}) + \pi_{res}(\boldsymbol{x})$ and $\boldsymbol{a}$ is the sampled action to be performed.

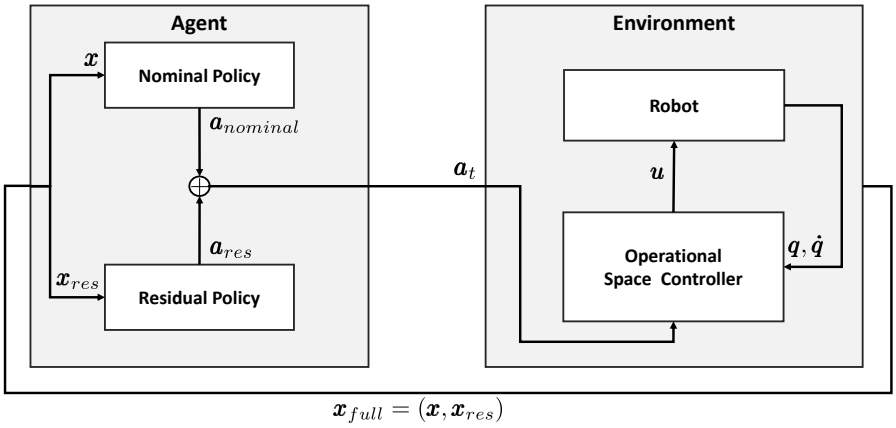

Figure 4.1.: **General control scheme** - This figure illustrates the control loop for generating a control signal u from the observation x_{full}. First, the observation from the environment is split into x, which is passed to the nominal policy, and x_{res}, which is passed to the residual policy. Both policies then provide an output with respect to their input, which is combined into an action a_t that is passed to the environment. The operational space controller takes this action, and in combination with the current joint states $q, \dot{q}$, generates a control signal u, which is then used to control the robot's joints and a new observation is generated.

We will evaluate our approach in multiple environments representing different insertion tasks and use operational space control to control a robot performing the insertion. The general control scheme is described in Figure 4.1 and we will present residual learning in Section 4.1, MSVFs in Section 4.2 and other considered adaptations to the approach in Sections 4.3 and 4.4.

4.1. Residual Learning with a Nominal Policy

The idea of residual policy learning [16] is to improve policies with model-free reinforcement learning. This not only allows for improvement over using one or the other, but also for learning tasks with long time horizons and sparse rewards that are particularly difficult to solve with pure reinforcement learning [16].

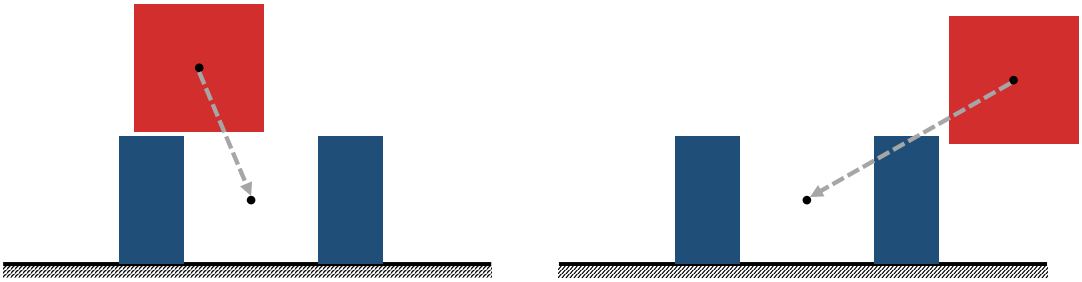

Figure 4.2.: **Problems of a naive nominal policy** - This figure illustrates the problem of a naive PD controller used as a nominal policy. While on the left side this nominal policy provides reasonable behavior, on the right side we can see that it causes the agent to get stuck.

The concept of residual policy learning is quite simple, the output of the policy for a given state $\boldsymbol{x}$ is defined by

$$\pi(\boldsymbol{x}) = \pi_{nominal}(\boldsymbol{x}) + \pi_{res}(\boldsymbol{x})$$

where $\pi_{nominal}$ is a non-differentiable policy (called nominal policy in the latter) and π_{res} is the residual policy. The important difference between the nominal and residual policy is that the residual policy is parameterized by some parameters $\boldsymbol{\theta}$. We can train these parameters by using gradient-based methods because the gradient with respect to the parameters is not affected by the nominal policy $\nabla_\theta \pi(\boldsymbol{x}) = \nabla_\theta \pi_{res}(\boldsymbol{x})$.

The purpose of using a nominal policy in addition to the initial random residual policy can be divided into two extremes

1. The nominal policy delivers acceptable performance on its own, and the residual policy learns to make slight corrections to optimize the performance.
2. The nominal policy is far from solving the task by itself, but provides reasonable exploration behavior so that the residual policy can learn the task.

Regardless of which side of the extreme the problem description is on, it is often beneficial to leverage the influences of the two policies. This can be done by using a term λ that either reduces or increases the influence of the residual policy

$$\pi(\boldsymbol{x}, \lambda) = \pi_{nominal}(\boldsymbol{x}) + \lambda \pi_{res}(\boldsymbol{x})$$

where the leverage term can be, for example, a constant $\lambda = 1.5$, or a function with respect to time $\lambda(t) = (1 - 0.95^t)$ or even with respect to the state, to formulate all kinds

of relations. For the course of this work we will assume $\lambda = 1$ and therefore use equal influence for both policies. Adapting this leverage term in order to even further improve the performance of the proposed approach is left for future work.

Although, classical residual policy learning can be very effective, it has one major issue: it needs a handcrafted nominal policy. In many cases, it is very difficult to create a near-optimal handcrafted policy to use the residual policy only for minor corrections and optimization. On the other hand, a very easy to create nominal policy can actually hinder learning. Imagine we want to solve an insertion task and we define a nominal policy as a simple PD-controller that generates movements towards the insertion position. While this nominal policy provides great behavior when it starts above the insertion position, it could get stuck after starting in some difficult initial configurations, as can be seen in Figure 4.2. To avoid these issues, we want to provide an approach that does not have the need for hand-crafted nominal policies and instead uses a learned stable policy as a nominal policy, which is explained in the next section.

4.2. Learnable Stable Vector Fields in Latent Manifolds - An Expressive Prior for Residual Reinforcement Learning

As we have already presented the drawbacks of a non-learnable, hand-crafted nominal policy, we will now introduce manifold stable vector fields (MSVFs) [21] and explain why we believe that they are an expressive prior for residual reinforcement learning.

First of all, MSVFs are learnable and therefore, when used as nominal policy, allow for learning and adapting depending on the seen data. Nevertheless, so far they have only been used for behavioral cloning [21], and we want to provide an approach to use them for RL and in this context also as nominal policy. One could argue that also other approaches that have already been applied to RL could also be used, but there are several properties of MSVFs that we believe make them the optimal candidate.

Although dynamic movement primitives (DMPs) [65] have been successfully used for residual learning in insertion tasks [18], they are phase dependent. MSVFs do not have a phase dependency like DMPs, they only depend on the current state of the system. This property makes them inherently reactive to disturbances, and they have been shown to provide a very compliant and also robust policy [21].

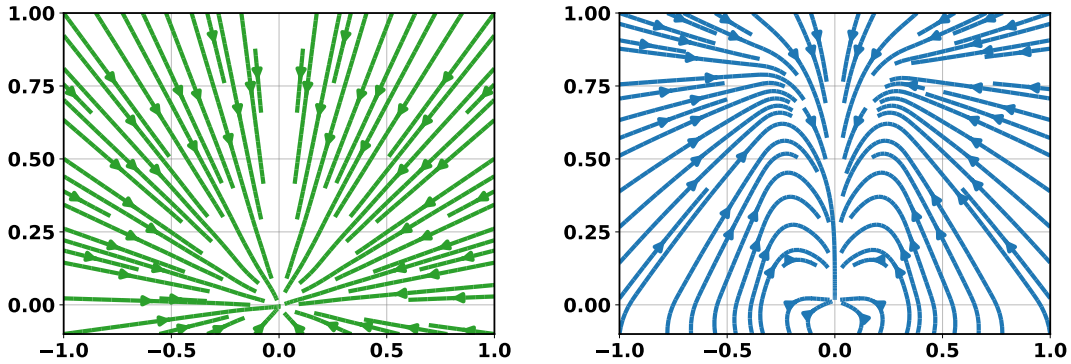

Figure 4.3.: **Comparison of vector fields** - This figure shows the difference between a hand-crafted stable vector field $\dot{\boldsymbol{x}} = -\boldsymbol{x}$ (left) and a stable vector field that can be learned with an MSVF (right).

Moreover, MSVFs are stable beyond the space in which they are trained, since they provide a diffeomorphism between manifolds in which the observation manifold is mapped to a latent stable vector field [21]. We assume that this stability is crucial to provide a reasonable signal for the residual network to learn. If the nominal policy is learned but not stable, its motion could also be random and when combined with the residual network, the motions could be even worse than when training only a single network. However, if the nominal policy is stable, it will always lead to a particular goal pose. Thus, the motions are very likely to lead to that goal pose despite the interference from the other learned policy, especially when trained with a dense reward. We consider the stability towards a goal pose as a prior for the learning process that accelerates learning with two policies.

Furthermore, MSVFs are defined using manifolds, which can be successfully used to learn orientations [66]. The necessity of manifolds in representing orientations is that while it is easy to define a distance metric in Euclidean space, this is not true for orientations. In addition, many orientation representations such as Euler angles or quaternions suffer from ambiguities [20]. Rotation matrices do not suffer from such ambiguities, but are difficult to learn due to their dimensionality. Manifolds and, in particular, Lie groups allow to use rotation matrices in an elegant way and define velocities in the tangent space [21].

Finally, MSVFs allow the definition of almost arbitrary stable vector fields. For example, if one learns only the impedance gains and controls a system by a non learned stable vector field, it is not possible to change the direction of the vectors more than $90°$, since the

impedance gains have to be strictly positive. This means that the solution space is much more constrained, but an MSVF can learn to deform the space in many ways and produce a variety of different motions, as can be seen in Figure 4.3. For this reason, in this thesis we will propose a residual MSVF approach and evaluate its benefits and costs for RL in insertion tasks, but first we will explain how MSVFs achieve these properties and how we use them in our work.

4.2.1. Stable Vector Fields in Latent Manifolds

An MSVF can be modeled by a pipeline of three main constructs: a parameterized diffeomorphism Φ, the latent dynamics g, and the pullback operator $d\Phi^*$ [21]. Although we will explain each construct in greater detail, we will first give a brief description of each component in the following

1. The parameterized diffeomorphism learns a diffeomorphic mapping $\Phi : \mathcal{M} \to \mathcal{N}$ from a manifold $\mathcal{M}$ to a latent manifold $\mathcal{N}$ by using a NODE to describe the mapping between the two manifolds.
2. The latent dynamics g defines a stable vector field in the latent manifold $\mathcal{N}$ to produce a stable latent velocity signal. In practice, an additional NN is used to first normalize the latent velocities and then rescale them according to its learned parameters.
3. The pullback operator reverses the mapping to the latent manifold and therefore describes a mapping $d\Phi^* : \mathcal{N} \to \mathcal{M}$. This velocity signal is a deformed version of our stable velocity signal, where the deformation is related to the applied diffeomorphism.

Thus, if we receive an observation $\boldsymbol{x}$ and want to process it into a velocity signal according to the MSVF policy, we use

$$\dot{\boldsymbol{x}} = d\Phi^* \circ g \circ \Phi(\boldsymbol{x})$$

which describes the full dynamics of the MSVF policy as depicted in Figure 4.4.

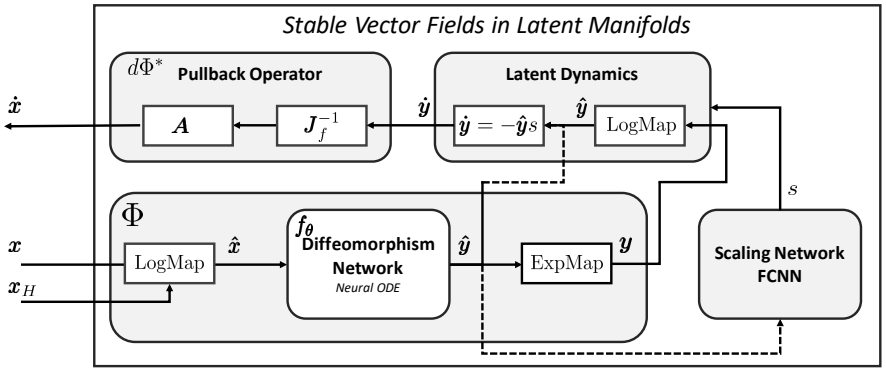

Figure 4.4.: **MSVF pipeline** - First, the LogMap transforms a given state (position and/or rotation) $\boldsymbol{x}$ and a target state $\boldsymbol{x}_H$ to a state $\hat{\boldsymbol{x}} \in \mathcal{T}_{x_H}\mathcal{M}$. Then a NODE representing a diffeomorphism between the observation manifold $\mathcal{M}$ and the latent manifold $\mathcal{N}$, is used to transform the given state in the observation tangent space to the latent tangent space $\hat{\boldsymbol{y}} = \boldsymbol{F}_{\boldsymbol{\theta}}(\hat{\boldsymbol{x}})$. In practice, we can consider a shortcut to avoid computing the ExpMap and LogMap, since both are computed in the same origin frame. The state in the latent tangent space $\hat{\boldsymbol{y}}$ is passed to the scaling network to generate a scaling term s, which is then used in the dynamics to generate a latent velocity $\dot{\boldsymbol{y}} = -\hat{\boldsymbol{y}}s$. Now, to represent this latent velocity in the observation manifold, two additional operations are required. Therefore, the pullback operator is defined as $d\Phi^* = \boldsymbol{A} \circ \boldsymbol{J}_{\boldsymbol{F}_{\boldsymbol{\theta}}}^{-1}$, where the Jacobian $\boldsymbol{J}_{\boldsymbol{F}_{\boldsymbol{\theta}}}^{-1} : \mathcal{T}_{y_H}\mathcal{N} \to \mathcal{T}_{x_H}\mathcal{M}$ reverses the mapping of the NODE and the adjoint $\boldsymbol{A} : \mathcal{T}_{x_H}\mathcal{M} \to \mathcal{T}_x\mathcal{M}$ transforms the signal to the tangent centered in $\boldsymbol{x}$.

LogMap & Expmap

In order to transform observations into velocities, we can interpret our observation space as a manifold $\mathcal{M}$ and the velocities as its tangent space $\mathcal{T}$. Then we can use the LogMap to transform a given observation $\boldsymbol{x} \in \mathcal{M}$ into an observation in the tangent space $\hat{\boldsymbol{x}} \in \mathcal{T}$. The relation of $\boldsymbol{x}, \hat{\boldsymbol{x}}$ is that if we integrate $\hat{\boldsymbol{x}}$ for one second, we reach the position $\boldsymbol{x}$. To use this formulation for our purpose, we perform two steps:

1. Since we want to move toward a desired goal $\boldsymbol{x}_H$, we first change the reference frame from the given observation $\boldsymbol{x}$ to $\boldsymbol{x}_H$.
2. We apply the LogMap in the reference frame of $\boldsymbol{x}_H$ to obtain an observation $\hat{\boldsymbol{x}}$ in the tangent space $\mathcal{TM}_{x_H}$ centered in $\boldsymbol{x}_H$ which, when integrated for one second starting in $\boldsymbol{x}_H$ yields the given observation $\boldsymbol{x}$.

For our purpose, we consider 4 cases for the observation manifold $\mathcal{M} \in \{\mathbb{R}^2, \mathbb{R}^2 + SO(2), \mathbb{R}^3, \mathbb{R}^3 + SO(3)\}$. Since some of these cases need to be treated differently, we will explain the different methods we used in the following.

In the case of $\boldsymbol{x} \in \mathbb{R}^2, \mathbb{R}^3$ we have no orientation but only translation. In order to transform the given observation in the world frame $\boldsymbol{x}$ to the observation with respect to the goal $\boldsymbol{x}^{\text{GOAL}}$ we can simply subtract the observation in the world frame from the goal position in the world frame $\boldsymbol{x}_H$, which gives us $\boldsymbol{x}^{\text{GOAL}} = \boldsymbol{x}_H - \boldsymbol{x}$. Since the LogMap in Euclidean space is the identity, we can set $\hat{\boldsymbol{x}} = \text{LogMap}(\boldsymbol{x}^{\text{GOAL}}) = \boldsymbol{x}^{\text{GOAL}}$ as the observation in tangent space with respect to the goal.

Now, if we use rotations and have the two-dimensional case $\boldsymbol{x} \in \mathbb{R}^2 + SO(2)$, we can treat the translation and orientation parts separately. We therefore split the observation $\boldsymbol{x} = (x, y, \theta)^T$ into its translation part $\boldsymbol{p} = (x, y)^T$ and its orientation part θ given in radians. We process the position as described previously to obtain the position in the tangent space $\hat{\boldsymbol{p}}$. To transform the orientation into the tangent space, we can again use the same procedure, since we are only dealing with single angle. We transform the orientation from the world frame to the goal frame with $\theta^{\text{GOAL}} = \theta_H - \theta$ and then apply the identity as LogMap which gives us $\hat{\theta} = \text{LogMap}(\theta^{\text{GOAL}}) = \theta^{\text{GOAL}}$. The observation in the tangent space is then given as $\hat{\boldsymbol{x}} = (\hat{\boldsymbol{p}}, \hat{\theta})^T$.

If we now want to perform the LogMap for the three-dimensional case $\boldsymbol{x} \in \mathbb{R}^3 + SO(3)$, we can again treat the translation part separately from the orientation. We obtain $\boldsymbol{x} = (\boldsymbol{p}, \boldsymbol{q})^T$, where $\boldsymbol{p} = (x, y, z)^T$ is the translation and $\boldsymbol{q}$ is the rotation given as a quaternion. The process for the translation part is again the same and yields the translation in the tangent space $\hat{\boldsymbol{p}}$. However, the LogMap for the orientation part is now different than for two dimensions. First we transform the given quaternion $\boldsymbol{q}$ to a rotation matrix $\boldsymbol{R}$. Then we have to transform the rotation matrix into the goal frame, by setting $\boldsymbol{R}^{\text{GOAL}} = \boldsymbol{R}_H^{-1}\boldsymbol{R}$ where $\boldsymbol{R}_H$ is the goal orientation. Then, we can set the rotation in tangent space $\hat{\boldsymbol{r}} \in \mathfrak{so}(3)$ using $\hat{\boldsymbol{r}} = \text{LogMap}(\boldsymbol{R}^{\text{GOAL}})$. Therefore, the observation in the tangent space with respect to the goal is given as $\hat{\boldsymbol{x}} = (\hat{\boldsymbol{p}}, \hat{\boldsymbol{r}})^T$

Diffeomorphism Network

The diffeomorphism network $\boldsymbol{F}_{\boldsymbol{\theta}}$ as introduced in [21], models a diffeomorphism between two differential manifolds. The motivation for this network is the idea of mapping the observation in the tangent space to a latent tangent space in which a stable vector field is defined. This stable vector field can then be used to generate motions towards a specific

stable point $\boldsymbol{y}_H$ in the latent manifold, and through the characteristics of a diffeomorphism we can then map these vectors back to the observation tangent space. This inverse mapping leads to motions toward the point $\boldsymbol{x}_H$ in the observation manifold, which is the stable point in the latent manifold [21].

In order to model such a diffeomorphism, a neural ODE is particularly well suited due to its invertibility [21]. We can interpret the represented ODE as a system that takes a point $\hat{\boldsymbol{x}}$ in the tangent space of the observation manifold $\mathcal{TM}$ at time $t = 0$ and transforms it to a point $\boldsymbol{y}$ in the tangent space of the latent manifold $\mathcal{TN}$ at time $t = 1$. In detail, the neural ODE representing the diffeomorphism network $\boldsymbol{F_\theta}$ can be modeled as

$$\boldsymbol{y} = \boldsymbol{F_\theta}(\hat{\boldsymbol{x}}) = \hat{\boldsymbol{x}} + \int_0^1 \boldsymbol{H_\theta}(\hat{\boldsymbol{x}}, t)dt \tag{4.1}$$

where $\boldsymbol{H_\theta}$ is a neural network and $\hat{\boldsymbol{x}}$ is the observation in the tangent space. It is important to understand that this neural ODE does not represent the dynamics of the policy, but just the mapping from the observation manifold to the latent manifold. The reason for using a dynamical system represented by a NODE is that it allows us to formulate the desired diffeomorphism.

One problem we have not yet considered, is that the topology of the Lie groups and their Lie algebra is not the same. Therefore, formulating a single diffeomorphic function for all points in the Lie algebra is not possible [21]. Imagine a 1-sphere like shown in Figure 4.5, here it is easy to define motions to a given point $\boldsymbol{x}$, in general the shortest path would be rotating either in π or $-\pi$ direction. However, for the antipodal point $\boldsymbol{x}_{\text{ANTIPODE}}$, there is no unique shortest path, and here lies the problem of defining a single diffeomorphic mapping. A solution to this problem is to define a second diffeomorphism which is simply the identity, resulting in

$$\boldsymbol{y} = \begin{cases} \boldsymbol{F_\theta}(\text{LogMap}(\boldsymbol{x})) & \text{if } \boldsymbol{x} \in \mathcal{U}_\mathcal{M} \\ \boldsymbol{x} & \text{if } \boldsymbol{x} \in \mathcal{M} \setminus \mathcal{U}_\mathcal{M} \end{cases},$$

where $\mathcal{U}_\mathcal{M}$ is the manifold without the antipodal point and $\mathcal{M}$ is the complete manifold, and respectively $\mathcal{M} \setminus \mathcal{U}_\mathcal{M}$ is the antipodal point.

Even though both cases are diffeomorphic, this does not mean that their composition is also a diffeomorphism. To guarantee this, one must show that the composition is continuous and differentiable in the boundaries between $\mathcal{U}_\mathcal{M}$ and $\mathcal{M} \setminus \mathcal{U}_\mathcal{M}$. In order to do so, we can define a distance term $d(\boldsymbol{x})$ that defines the distance to the boundaries of a point $\boldsymbol{x} \in \mathcal{U}_\mathcal{M}$.

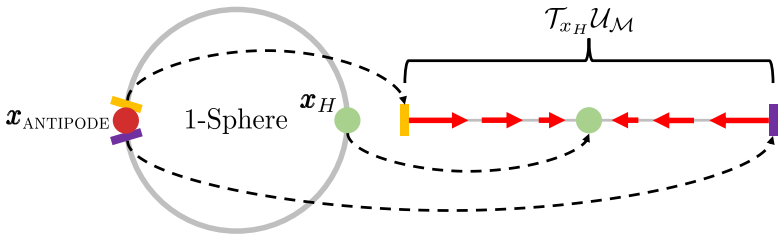

Figure 4.5.: **1-Sphere tangent space** - This figure illustrates why boundaries are important to consider for the diffeomorphic mapping. There is no clear shortest path from the antipodal point $\boldsymbol{x}_{\text{ANTIPODE}}$ to the center $\boldsymbol{x}_H$. Therefore the tangent space $\mathcal{T}_{x_H}\mathcal{U}_\mathcal{M}$ is defined only for the manifold $\mathcal{U}_\mathcal{M}$ and the antipodal point is in the manifold $\mathcal{M} \setminus \mathcal{U}_\mathcal{M}$, which is handled separately.

Then we can change Equation 4.1 to

$$\boldsymbol{F_\theta}(\hat{\boldsymbol{x}}) = \hat{\boldsymbol{x}} + \int_0^1 d(\boldsymbol{x})\boldsymbol{H_\theta}(\hat{\boldsymbol{x}}, t)dt$$

where multiplying $d(\boldsymbol{x})$ leads the output velocity of the NODE $\boldsymbol{H_\theta}$ to converge smoothly to 0 as the boundaries are approached, because $d(\boldsymbol{x})$ converges to 0. This means that the mapping represented by $\boldsymbol{F_\theta}$ converges to the identity at the boundaries.

Latent Dynamics

A fundamental characteristic of this approach is to define a latent stable vector field that is stable to a given point $\boldsymbol{y}_H$ in the latent manifold $\mathcal{N}$. This means that we create a dynamical system that maps every point in the latent manifold $\boldsymbol{y} \in \mathcal{N}$ to a velocity $\dot{\boldsymbol{y}}$ defined in the tangent space $\mathcal{T}_{y_H}\mathcal{N}$. For this reason we can define

$$\dot{\boldsymbol{y}} = -\boldsymbol{y}, \tag{4.2}$$

which induces a stable dynamical system in the latent manifold generating linear trajectories to the equilibrium point $\boldsymbol{y}_H = 0$.

However, since we have defined the diffeomorphism for two separate cases, we must do the same for the dynamics. For this reason, we need to separate how we process points $\boldsymbol{y} \in \mathcal{U}_{\mathcal{N}}$ and points $\boldsymbol{y} \in \mathcal{N} \setminus \mathcal{U}_{\mathcal{N}}$ and define

$$\dot{\boldsymbol{y}} = \begin{cases} -\boldsymbol{y} & \text{if } \boldsymbol{y} \in \mathcal{U}_{\mathcal{N}} \\ 0 & \text{if } \boldsymbol{y} \in \mathcal{N} \setminus \mathcal{U}_{\mathcal{N}} \end{cases}$$

which generates a stable point $\boldsymbol{y}_H = 0$ for all points $\boldsymbol{y} \in \mathcal{U}_{\mathcal{N}}$ [21].

Scaling Network

Because of the definition of the latent stable vector field introduced in Equation 4.2, the velocities are directly related to the distance from $\boldsymbol{y}$ to the equilibrium point $\boldsymbol{y}_H$. This means that if we want our policy to drive our agent to $\boldsymbol{y}_H$, this will lead to smaller and smaller vectors in the latent stable vector field, which could be limiting. To avoid this behavior, we normalize all velocities in the latent stable vector field so that $\|-\boldsymbol{y}\| = 1$ and learn a scaling term s to allow for variations in latent velocities [21]. The scaling term is learned by a neural network $\boldsymbol{S}_{\boldsymbol{\psi}}(\boldsymbol{y}) = s$ and forced to a positive value using the softplus function

$$\textsc{Softplus}(s) = \log(1 + \exp(s))$$

to avoid affecting the stability of the vector field by changing direction of vectors.

Pullback Operator

All of the previously explained parts of the latent stable vector field, had the purpose of transforming a pose $\boldsymbol{x}$ represented in an observation manifold $\mathcal{M}$ into a latent velocity signal $\dot{\boldsymbol{y}}$ given by a stable vector field defined in a latent manifold $\mathcal{N}$. However, this latent velocity $\dot{\boldsymbol{y}}$ has no meaning for our agent since it is defined in a different manifold. Even if the manifolds are the same, this does not mean that our diffeomorphism is the identity, so in any case it is necessary to pull back the velocity in the latent manifold to a representation in our observation manifold, which is the purpose of the pullback operator [21]. The pullback operator unrolls all the steps taken to transform the state in our observation manifold to the latent manifold. It does this using two fundamental components, the Jacobian $\boldsymbol{J}_{\boldsymbol{F}_{\boldsymbol{\theta}}}$ of the diffeomorphism network $\boldsymbol{F}_{\boldsymbol{\theta}}$ and the adjoint $\boldsymbol{A}$.

The Jacobian $\boldsymbol{J}_{\boldsymbol{F}_{\boldsymbol{\theta}}}$ tracks the change in coordinates as we transform from the tangent space of the observation manifold $\mathcal{T}_{\boldsymbol{x}_H}\mathcal{M}$ to the tangent space of the latent manifold $\mathcal{TN}$ by using our diffeomorphism $\boldsymbol{F}_{\boldsymbol{\theta}}$. This means that the inverse of the Jacobian reverses this mapping and therefore we can use it to map from the latent manifold to our observation manifold

$$\dot{\boldsymbol{z}} = \boldsymbol{J}_{\boldsymbol{F}_{\boldsymbol{\theta}}}^{-1}\dot{\boldsymbol{y}}$$

where $\dot{\boldsymbol{z}} \in \mathcal{T}_{\boldsymbol{x}_H}\mathcal{M}$. It is important to notice that we have not yet completely reversed the mapping. We still need to reverse the change of reference frame, our current $\dot{\boldsymbol{z}}$ is represented with respect to the tangent space centered in $\boldsymbol{x}_H$ which is also the stable point of the latent stable vector field if $\boldsymbol{y}_H = \boldsymbol{F}_{\boldsymbol{\theta}}(\boldsymbol{x}_H)$. In order to reverse this mapping to define $\dot{\boldsymbol{x}}$ in the tangent space in the current pose $\mathcal{T}_{\boldsymbol{x}}\mathcal{M}$ we need to use the adjoint.

The adjoint is a linear mapping $\boldsymbol{A} : \mathcal{T}_{\boldsymbol{x}_H}\mathcal{M} \rightarrow \mathcal{T}_{\boldsymbol{x}}\mathcal{M}$ and therefore allows to change the reference frame of the velocity vector by

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\dot{\boldsymbol{z}}$$

resulting in our final velocity signal $\dot{\boldsymbol{x}} \in \mathcal{T}_{\boldsymbol{x}}\mathcal{M}$. Therefore, the full pullback operator is given by $d\Phi^* = \boldsymbol{A} \circ \boldsymbol{J}_{\boldsymbol{F}_{\boldsymbol{\theta}}}^{-1}$.

4.2.2. Fix-Center-Loss

Even though we know that our latent stable vector field is stable towards the defined equilibrium $\boldsymbol{y}_H = 0$, we need to make sure that we can define a desired stable point $\boldsymbol{x}_H$ in our observation manifold such that $\boldsymbol{y}_H = \boldsymbol{F}_{\boldsymbol{\theta}}(\boldsymbol{x}_H)$. Otherwise, we would have a stable vector field in a latent manifold, but we would have no way to infer the stable point in the observation manifold. For this reason, a loss function is added to produce a signal that fixes the mapping from $\boldsymbol{F}_{\boldsymbol{\theta}}(\boldsymbol{x}_H)$ to the desired $\boldsymbol{y}_H = 0$. This can be interpreted as applying a penalty when the diffeomorphism does not map the center of the tangent space of the observation manifold $\mathcal{T}_{\boldsymbol{x}_H}\mathcal{M}$ to the center of the stable point of the latent stable vector field $\boldsymbol{y}_H = 0$.

Such a loss function can be created by applying a distance loss defined as

$$\text{Loss} = \|\boldsymbol{F}_{\boldsymbol{\theta}}(\boldsymbol{x}_H)\|_1$$

which uses a smooth L1 norm to define the distance from the center in the latent tangent space [21]. However, we found that when optimizing for multiple loss functions in RL,

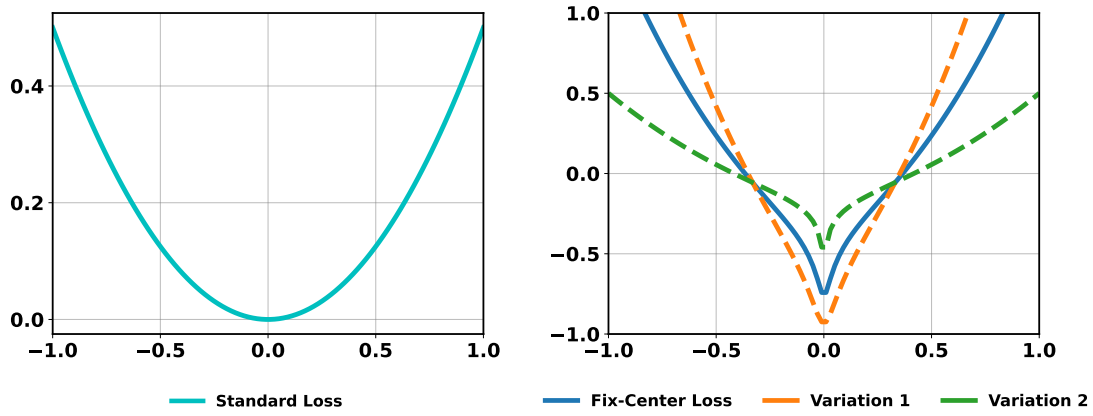

Figure 4.6.: **Fix-Center-Loss** - The figure on the left shows a loss function defined by using the distance to the target. The right figure depicts the fix-center-loss function along with two parameter variations. For the standard version (blue) the parameters were set to $\alpha = 1e-5, w = 1.5, v = 0.1$, the first variation (orange) uses the parameters $\alpha = 2e-3, w = 2.5, v = 0.15$ and the second variation (green) shows the fix-center-loss with the parameters $\alpha = 1e-6, w = 0.5, v = 0.05$.

this loss function is not sufficient to ensure that the center remains fixed while other losses are also affecting the learning. Therefore, we have adapted the fix-center-loss function to make it more expressive by using an alternative formulation

$$\textsc{Fix-Center-Loss} = w\|\boldsymbol{F}_{\boldsymbol{\theta}}(\boldsymbol{x}_H)\|_2^2 + v\log(\|\boldsymbol{F}_{\boldsymbol{\theta}}(\boldsymbol{x}_H)\|_2^2 + \alpha),$$

where w, v, α are hyper-parameters [67]. Unlike the distance-based loss, this formulation yields high gradients near the center, as can be seen in Figure 4.6. In practice, this loss function is able to fix the center very close to $\boldsymbol{F}_{\boldsymbol{\theta}}(\boldsymbol{x}_H) = \boldsymbol{y}_H = 0$, which is very important if we want to ensure stable behavior with respect a specific point.

4.3. Variable Impedance Control

Most of the time, research attempts to achieve better performance by changing the structure of a policy or by adjusting the observation space to provide more expressive information to the agent. However, fewer works focus on adapting the action space instead. A recent work called VICES [68] evaluated just such action space adjustments and found that variable impedance control in end-effector space is a highly beneficial action space for constrained, high-contact tasks. In VICES, they propose not only learning the desired end-effector pose, but also dynamically adjusting the impedance gains for each dimension. In general, there is much research that has shown that variable impedance adjustments can be successfully integrated into learning [69, 70, 71]. Additionally, some of these ideas have already been applied to insertion tasks [58, 59].

In this work, we stick to the idea of VICES and combine it with the proposed method of [58] to model the learning of the gains. Therefore, we let the residual part of the policy not only output the velocities in terms of position and orientation, but extend its output by the diagonal elements of the stiffness matrix $\boldsymbol{K}_P$. This means that for each joint the controlled robot has, a gain has to be estimated by the residual policy. To facilitate learning this mapping, we do not directly output the diagonal elements, but instead output values v_i in a range of $[-1, 1]$, where i is the number of joints. We do so by applying the hyperbolic tangent (Tanh) on the output of the residual network that defines to the diagonal elements of $\boldsymbol{K}_P$. Therefore, the output of the residual network can be described as $(\boldsymbol{a}_{pose}, \boldsymbol{a}_{gains})$, where $\boldsymbol{a}_{pose}$ is the action describing the desired velocities and is processed as explained in previous sections, and $\boldsymbol{a}_{gains}$ is the Tanh of the output of the residual network with respect to the gains. Since $\boldsymbol{a}_{gains}$ is in $[-1, 1]$, we need to define a mapping to suitable impedance gains. To do this, we set a minimum and maximum gain K_i^{min}, K_i^{max} for each dimension of the gain matrix $\boldsymbol{K}_P$. Then we apply a function g to $\boldsymbol{a}_{gains}$ that maps -1 to K_i^{min} and 1 to K_i^{max}, and all intermediate values are mapped linearly to a corresponding intermediate value. Now we can set $\boldsymbol{K}_P = \textsc{diag}(g(\boldsymbol{a}_{gains}))$ to define the gain matrix for the impedance. Finally, we set the damping gains $\boldsymbol{K}_D$ with respect to $\boldsymbol{K}_P$ in order to define a critically damped system, by using

$$\boldsymbol{K}_D = 2\sqrt{\boldsymbol{M}\boldsymbol{K}_P},$$

where $\boldsymbol{M}$ is mass matrix of the robot [58]. Therefore, the gain matrices used for the operational space controller are now learned by the residual network and can be adjusted for each state.

4.4. Observing Forces and Torques

In addition to adjusting the action space, as is done with variable impedance control, it is also possible to adjust the observation space to provide more information to the agent. It has been shown that information from force sensors is quite useful for a wide range of tasks. In [72], the authors use force information from demonstrations to learn a residual policy for insertion tasks. Another approach does not only use a force trajectory, but also performs variable impedance control [58]. The desired insertion force for a peg-in-hole task can also be modeled and used to achieve compliant behavior [59]. Finally, the use of force sensor information allows to achieve incredible precision when combined with Deep RL [11].

Since in all this work the force sensor information provided great benefits, we would like to propose an adaptation of our approach to include force information as well. We suggest that, again, it is most reasonable to not change the nominal policy, but to pass the additional force observation to the residual network. We therefore provide two observations $\boldsymbol{x}, \boldsymbol{x}_{res}$, where $\boldsymbol{x}$ is the observation with respect to the pose as previously introduced, and $\boldsymbol{x}_{res}$ is $\boldsymbol{x}$ extended by the force vectors at the end-effector normalized to $[0, 1]$. This formulation allows us to smoothly adjust the observation space and evaluate the influence of the force information for specific tasks.

5. Experiments

Throughout this section, we will not only present our experiments, but also make clear how we set up the environments and RL algorithms to generate our results. For this reason, we will first explain the experimental setup in Section 5.1. After that, we will list and discuss all of our experiments in Section 5.2 and draw a conclusion in Section 5.3.

5.1. Experimental Setup

In order to explain how we generated our results, we will first explain in Section 5.1.1 how we defined the environments in which we tested our approach. To also provide insight into how we set up the reinforcement learning methods we used, we will explain in more detail how we configured the reinforcement learning of our approach in Section 5.1.2. Finally, we explain how we evaluated the learning process of the compared approaches in Section 5.1.3, before presenting the actual results of our work in Section 5.2.

5.1.1. Environments

To evaluate our approach, we have created three environments that simulate an insertion task. We chose to model the insertion of a box rather than a peg because this makes learning the correct rotation even more difficult, since all three axes must to be rotated correctly to insert the object. To reduce the complexity at first, we tested our approach in a two-dimensional insertion task, called Box2D. After observing good performance, we started to increase the complexity by moving to a three-dimensional environment, the Box3D environment. To show that our approach works for more than just toy examples, we modified the Box3D environment to also include a simulated robot arm performing the insertion, called Box3D-Franka environment. All of these environments will be discussed

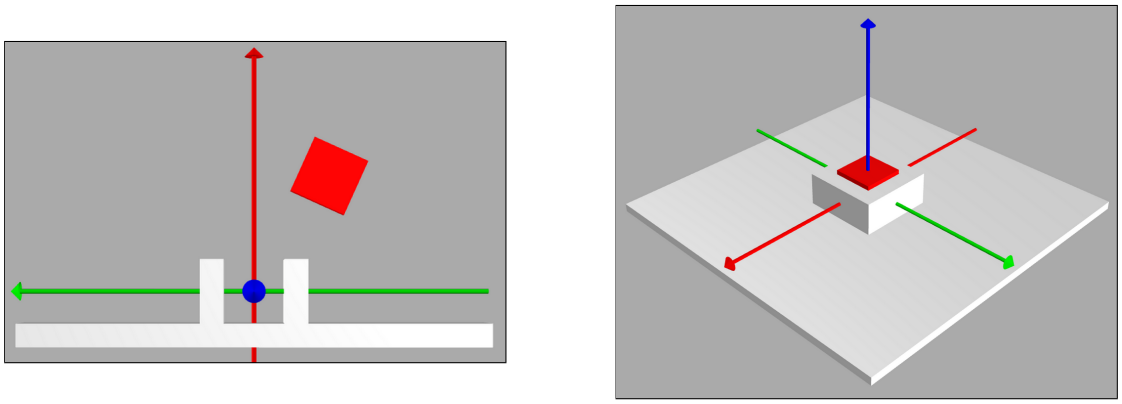

Figure 5.1.: **Box2D & Box3D environments** - This figure shows the Box2D-Environment on the left and the Box3D-Environment is shown on the right. The agent is visualized by the red box and its position and rotation can be controlled directly with respect to the world frame, where the x-axis is shown in red, the y-axis in green and the z-axis is blue. The point defining a fully inserted box is the center of the environment.

in more detail throughout the next sections, but first we will mention some common characteristics of the environments.

In general, the units are in meters and kilograms, and for simplicity, gravity compensation is provided by eliminating all gravitational forces acting on the robot. The friction coefficient of the insertion object and the table as well as the insertion box, is set to 0.4, which is a reasonable value for 3D printed materials [73]. The actions given to the environment are always specified in the world coordinate frame, and after the action $\boldsymbol{a}$ is processed, an observation $\boldsymbol{x}$ is returned to the agent. By default, the observation is the complete pose of the agent with respect to the number of dimensions. For each task, there is a maximum number of allowed steps, when reached, the episode must be terminated and the agent reset. The episode will end early if the object is inserted correctly.

Box2D Environment

The Box2D environment can be seen in Figure 5.1. It represents a two-dimensional insertion task where the agent is a rigid body of dimension 0.25×0.25. It is controlled by two prismatic joints defining its positions p_x, p_y in the two dimensions, and one revolute joint for its orientation θ, so $\boldsymbol{q} = (q_1, q_2, q_3)^T = (p_x, p_y, \theta)^T$. The mass of the rigid body is 1 kilogram and the inertia is set according to its mass and dimensions. The environment is

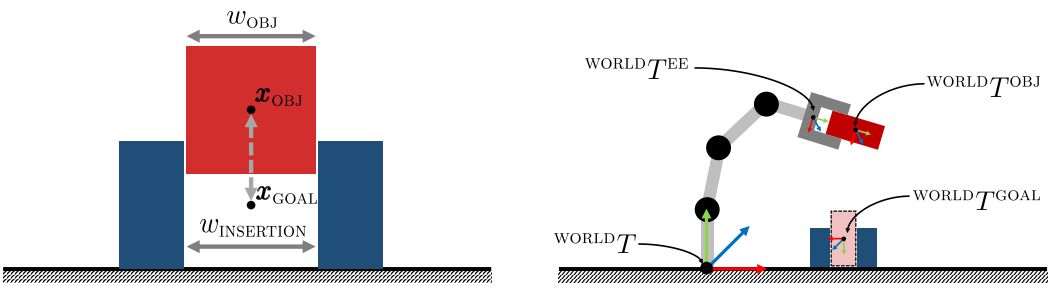

Figure 5.2.: **Setup of environments** - This figure illustrates the configuration of the environments. On the left, one can see the spacing of the insertion with respect to the insertion object. We use $w_{\text{INSERTION}} = w_{\text{OJB}} + 0.005$ and the reward is defined by the difference of the object pose $\boldsymbol{x}_{\text{OBJ}}$ to the goal pose $\boldsymbol{x}_{\text{GOAL}}$. The right figure shows the different reference frames for the task with the Franka-Robot, where ${}^{\text{WORLD}}\boldsymbol{T}^{\text{EE}}$ describes the transformation of the world coordinates into the reference frame of the end-effector.

bounded in $[-1, 1]$ for the x, y dimensions and the initial state for the pose $\boldsymbol{q}_0$ of the agent is uniformly sampled so that $p_{x_0} \in [0.8, 0.99], p_{y_0} \in [-0.99, 0.99], \theta_0 \in [-\pi/2, \pi/2]$. The insertion can be considered complete if the agent's position is $\boldsymbol{q} = (0, 0, 0)^T$. However, we let the episode terminate within some tolerance, if the norm of the position $\|(p_x, p_y)^T\|$ is less than 0.025, and a tolerance of $5°$ for the rotation error. This means that the episode will not termiante if the box has been inserted but does not have the desired rotation. The width of the insertion is equal to the width of the agent added with 0.005, as shown in Figure 5.2. An OSC controller is used to control the joints of the robot, and to limit the velocities that can be set by the policies, the actions are clipped before being passed to the controller.

Box3D Environment

The Box3D environment is the three-dimensional version of the Box2D environment and can also be seen in Figure 5.1. It follows exactly the same structure as the Box2D environment, but in three dimensions. Therefore, there is an additional prismatic joint to model the translation in the z-axis, and two additional revolute joints to model the rotations around the x- and y-axes. The dimensions of the box are $0.25 \times 0.25 \times 0.25$ and the full pose of the agent is described by $\boldsymbol{q} = (q_1, q_2, q_3, q_4, q_5, q_6)^T = (p_x, p_y, p_z, \theta_x, \theta_y, \theta_z)^T$. Again, the mass of the robot is 1 kilogram and the inertia is updated with respect to

the new dimensions. The boundaries of the environment are correspondingly extended to three dimensions and the initial pose of the agent is uniformly sampled so that now $p_{x_0}, p_{y_0} \in [-0.99, 0.99], p_{z_0} \in [0.8, 0.99], \theta_{x_0}, \theta_{y_0}, \theta_{z_0} \in [-\pi/2, \pi/2]$. The termination rule remains the same, but now in three dimensions, allowing a maximum rotation error of 5° per axis. The precision required for insertion also remains the same but now in terms of the width and length of the insertion object. The used controller is again an OSC controller, where the actions are clipped in the norm to a suitable range before being given to the controller.

Box3D-Franka Environment

The Box3D-Franka environment is a more realistic version of the previously described Box3D environment. The general setup is actually identical, with a single but important difference: the agent is now not the insertion object itself, but the Franka Emika Panda robot arm [74]. We do not use the grippers of the Franka robot because we fixate the insertion object at a fixed distance from the end-effector. This is a reasonable simplification since it represents fixing an object on the Franka robot in the real world, and allows to not have the need to also learn to grasp the object. The controller is again an OSC controller, but now with an additional null space controller. The Franka robot itself is placed at $(-0.65, 0, 0)^T$ and all the joints are initialized in the rest position with a noise of 0.05 per joint. The rest position is chosen so that the end-effector of the Franka robot is approximately at $(0, 0, 0.4)^T$ with an orientation that aligns the insertion object with the negative z-axis of the world frame. For termination, the desired rotation is now changed as it is now with respect to the rotation of the end-effector in the world frame, as shown in Figure 5.2.

5.1.2. Reinforcement Learning Setup

For all experiments in the simulation, we use PPO in the actor critic setting to train all evaluated approaches. We chose PPO because it provides a solid baseline without requiring much approach-specific fine-tuning. In addition, we use IsaacGym [75] to speed up the learning by parallelizing the simulation on the GPU, as shown in Figure 5.3. This allows more data to be generated in less time, making the sampling of data from the environment even more efficient. We stop training a model after a certain number of data points have been used for training; this number is the same for all approaches we use in an specific experiment. Since PPO does not need so much fine-tuning, it allows us to

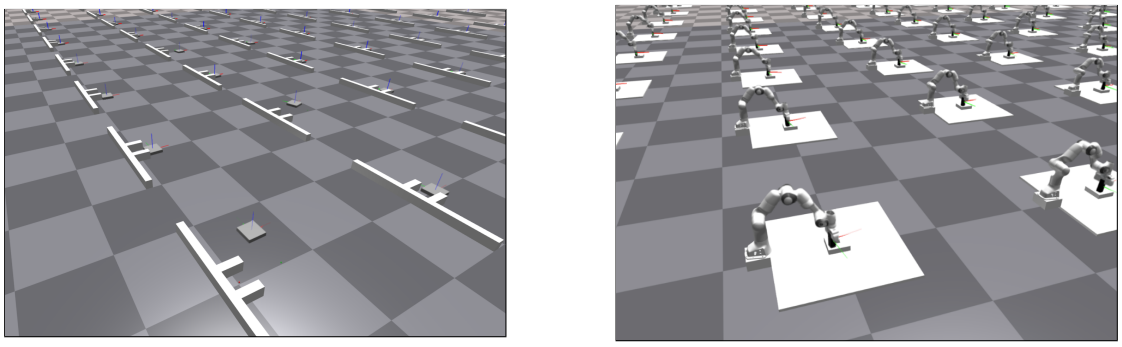

Figure 5.3.: **Simulation with IsaacGym** - This figure shows the parallel data generation using the GPU with IsaacGym. On the left, one can see the Box2D-Environment and on the right the Box3D-Environment where the box is manipulated by the Franka-Robot.

compare the different approaches with fairly similar hyper-parameters, allowing for a fair comparison in a similar setting. We fit the models every epoch, and a single epoch contains the information of 100 environments performing each 100 steps, therefore we perform each fit with 10000 data points from the simulation.

In general, we define a dense reward where the reward for each action is the distance to the goal pose in terms of position and orientation. In addition, the dense reward is normalized to a maximum of -1. In the case of orientations, our policy has a built-in advantage in that it is able to transform a quaternion suffering from the ambiguity $\boldsymbol{q} = -\boldsymbol{q}$ into a rotation matrix. For this reason, we decided to duplicate the batches so that for each observation $\boldsymbol{x} = (\boldsymbol{p}, \boldsymbol{q})^T$ there is a duplicated version $\boldsymbol{x} = (\boldsymbol{p}, -\boldsymbol{q})^T$ that allows a standard NN to more easily learn the relationship of $-\boldsymbol{q}$ and $\boldsymbol{q}$.

5.1.3. Evaluation Procedure

To empirically evaluate our experiments, we train each approach for a total of 15 random seeds. During training, we save a current representation of the model and its parameters every n epochs for later loading and evaluation of the model. After training all approaches in the given experiment for all seeds, we start the empirical evaluation, which is the same for each approach and experiment.

We load the specific configuration of the experiment and the model at a specific epoch. Then we run the policy 1000 times until it either terminates by inserting correctly or it

reaches the maximum number of steps for that experiment. During these runs, we record the number of steps required to insert the object and whether the insertion was successful or not. We do this for all seeds of the evaluated approach at that specific epoch. Then we can use the results across all seeds to calculate the mean and confidence interval for the total number of steps taken and the success rate.

5.2. Experimental Evaluation

In our experiments we will compare four approaches:

1. **MLP**: The policy is represented by a fully connected neural network $\pi(\boldsymbol{x}) = \mathcal{F}_\psi(\boldsymbol{x})$.
2. **MSVF**: The policy is represented by a manifold stable vector field $\pi(\boldsymbol{x}) = \Phi_\theta(\boldsymbol{x})$.
3. **Nominal (linear) + Residual**: The policy is represented by a combination of a non-learnable linear nominal policy and a residual policy represented by a fully connected neural network $\pi(\boldsymbol{x}, \boldsymbol{x}_{res}) = \pi_{nominal}(\boldsymbol{x}) + \mathcal{F}_\psi(\boldsymbol{x}_{res})$, where the inputs $\boldsymbol{x}, \boldsymbol{x}_{res}$ can be different for the two policies.
4. **Nominal (MSVF) + Residual**: The policy is represented by a combination of a manifold stable vector field and a residual policy represented by a fully connected neural network $\pi(\boldsymbol{x}, \boldsymbol{x}_{res}) = \Phi_\theta(\boldsymbol{x}) + \mathcal{F}_\psi(\boldsymbol{x}_{res})$.

The policy output is used to represent the mean of a Gaussian distribution from which we sample the actions $\boldsymbol{a} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$. As mentioned earlier, we use PPO to solve the tasks in the different environments (see Section 5.1.1) and evaluate the learning process of the model by storing and evaluating it every 50 epochs.

The motivation for our experiments is to answer several research questions. First, we want to investigate whether the MSVF approach is able to learn an insertion in the presented environments. Next, we will evaluate the performance of using an MSVF as a nominal policy for residual learning, as we believe it to be an appropriate prior. While there are tasks that require learning orientations, in the environments we present there may not be a need to learn the orientation in each state since we already know the desired final orientation. Therefore, we want to investigate whether there is an advantage to learning orientations, or if we could simply specify the desired final orientation and use a linear controller to achieve it. As we have presented the advantages of variable impedance control and force feedback, we also want to investigate how both adaptations affect the performance of the residual MSVF approach. Finally, we also want to analyze whether

the MSVF approach is able to learn more difficult insertion tasks by showing its ability to learn another task called Ubongo3D and perform insertions with less clearance.

5.2.1. Performance of an MSVF Policy in Classic Reinforcement Learning

In our first experiment, we want to evaluate the overall performance of an MSVF compared to an MLP. In this scenario, there is no residual policy, but the policy is completely defined by the MSVF or the MLP. The idea of this experiment is to show the general ability of the two models to learn the task independently and also to evaluate how their learning differs from each other. For this reason, we test the MSVF and the MLP in all three environments and discuss the results below.

As one can be seen in Figure 5.4, the MLP approach takes some time to learn, but steadily improves. In our experiments, we also evaluated longer training runs, where it can be seen that the MLP converges at the end. On the other hand, the MSVF approach is very unstable in learning and suffers from unlearning.

One might think that this is just a tuning problem, but we have done our best to eliminate this behavior. Not only have we tried several hyper-parameter configurations and used learning rate optimizers like ADAM [76], but we also avoid gradient problems caused by backpropagation through the ODE solver by using the adjoint method [53]. Moreover, there is an intuitive explanation for the reason of this unlearning. The MSVF itself constrains the problem to provide stability, which is done by using a diffeomorphism between manifolds. A main problem with this approach is that if we want to change the direction of only one vector, this will affect at least the vectors within a certain space around that vector, as the mapping is smooth. That is, if we want to change vectors at the sides of the insertion, we will also change the vectors pointing into the insertion. The main problem is that we do not learn to directly output a vector but to learn a mapping to a space that outputs the vectors. This means that we do not directly change our output, but need to change the mapping between the spaces to indirectly affect the output. Therefore, to achieve the desired behavior, i.e., not change the other vectors, the mapping between spaces must be torn apart, which means that large changes to the weights are required to achieve a small adjustment. While this is not such big problem in behavioral cloning (here the intermediate policies do not affect data collection for future learning), it is a problem in reinforcement learning to require large changes in the weights for small changes in the output. Imagine that a small desired change in an output requires many updates that could actually degrade performance at times. If intermediate updates really decrease the

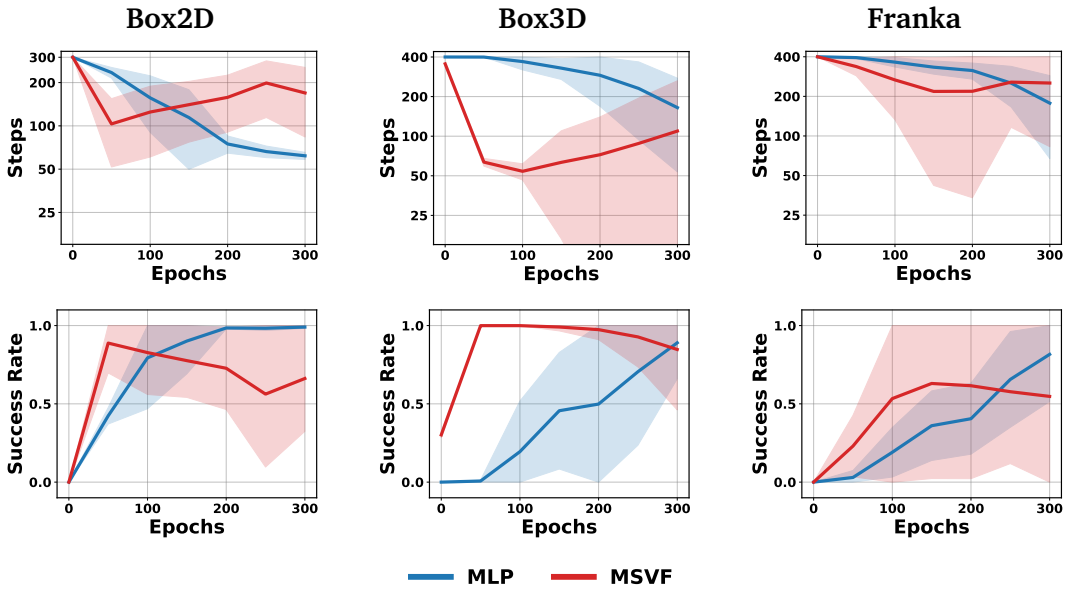

Figure 5.4.: **Performance of MLP and MSVF** - This figure shows the results for the different environments with respect to the the MSVF approach and the MLP approach. In this evaluation there is no residual learning and we stopped learning after 300 epochs as the MSVF diverges. With longer training, it can be seen that that the MLP approach converges. The solid line represents the mean of the result for the evaluated approach and epoch and the shaded area shows the 95% confidence interval. Further configurations of the approaches and the environment can be found in Appendix A.

reward, then this change gets very unlikely in RL and the agent is likely to be stuck in a local optima.

Although it might be very difficult in RL to optimize such a constrained approach like MSVFs beyond a certain level of performance, we are certain that with some additional effort we can take advantage of MSVFs for RL. For this reason we believe that learning a residual that provides more flexibility to the learning process by performing easy to learn local adjustments, allows to use MSVFs and some of their benefits in RL. Therefore, we consider viewing them as a learnable stable bias that accelerates learning in combination with another policy. In order to do so, we will use MSVFs as a nominal policy for residual policy learning and show that this approach performs incredibly well for insertion tasks.

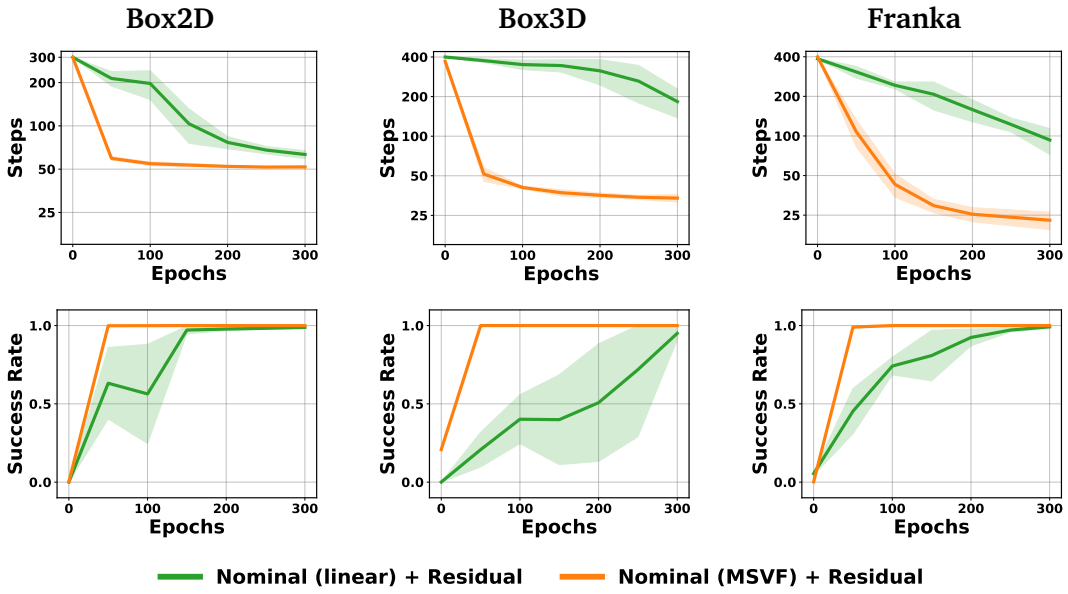

Figure 5.5.: **Performance of residual methods** - This figure shows the performance of the approaches with residual learning for the three environments. Although, the linear approach converges to a similar performance with longer training, we interrupt the training after 300 epochs because the MSVF approach has already achieved similar performance. The solid line represents the mean of the result for the evaluated approach and epoch and the shaded area shows the 95% confidence interval. Further configurations of the approaches and the environment can be found in Appendix A.

5.2.2. Performance of an MSVF Policy in Residual Policy Learning

As we have shown in the previous experiment, MSVFs alone are not sufficient to accelerate RL in insertion tasks. We found that a particular problem with MSVFs is that they are unable to perform small but local adjustments to their policy. This motivates the use of a residual network to learn exactly these small local adjustments. For this reason, we evaluate a residual policy that uses an MSVF as its nominal policy and also learns an MLP for the residual signal. Furthermore, to evaluate whether MSVFs are an appropriate nominal policy, we compare this approach to a linear nominal policy, which is a linear stable vector field pointing to the goal pose. It is important to note that the latent stable vector field of the MSVF policy and the stable vector field of the linear nominal policy are identical, the only difference being that the MSVF can learn to deform this vector field

by learnig to adjust its diffeomorphism. This allows us to evaluate whether it is useful to learn such a diffeomorphism or to use only the linear signal for the nominal policy.

As can be seen in Figure 5.5, the MSVF used as the nominal policy outperforms the linear nominal policy in all environments. While the linear nominal policy, when trained over a longer period of time, also converges to the same performance, it requires much more data to do so. These results show that learning an MSVF along with an MLP provides better performance for the insertion tasks tested. The learning is not only stable, but also has a very small variance and improves very quickly in the beginning. In our opinion, the ability to learn a globally stable vector field that leads to the insertion, along with an unstable MLP that learns to take care of smaller corrections, is a very strong inductive bias for a policy. To explain why the final performance is better in the Franka environment than in the Box3D environment, it is important to know that the initial position in the Franka environment is closer to the insertion than in the Box3D environment, resulting in fewer steps needed for an optimal policy.

5.2.3. Benefits and Costs of Learning Orientations

A key assumption of the MSVF approach is that we know the desired goal pose and thus the desired orientation. Since learning the orientation increases the state space and therefore could lead to a more difficult problem, one could argue that if we know the desired orientation, we can simply use a linear controller to achieve that orientation. Therefore, we will investigate whether there is a cost or benefit to learning orientations by changing the output of the policies to only the velocity with respect to the position and creating a simple PD-controller to achieve the desired orientation. We evaluate this setting not only with the two residual policies (linear and MSVF), but also with a standard MLP without a nominal policy.

When orientation is not learned, the MSVF is still a very good inductive bias for residual policy learning, as can be seen in Figure 5.6. However, it is now more on par with the other approaches. Tables 5.1 and 5.2 show that the non-MSVF approaches have increased their performance especially in three-dimensional tasks, where learning rotations without using Lie groups is rather difficult. On the other hand, the actual performance of the MSVF approach is now worse compared to learning the rotations for the three-dimensional environments. We have found that for the three-dimensional tasks, it is performance-enhancing to learn some tilting and twisting of the object rather than fixing its orientation before actually inserting the object. Combining this with the fact that the structure of Lie groups allows efficient learning of the orientation, we can explain why the MSVF

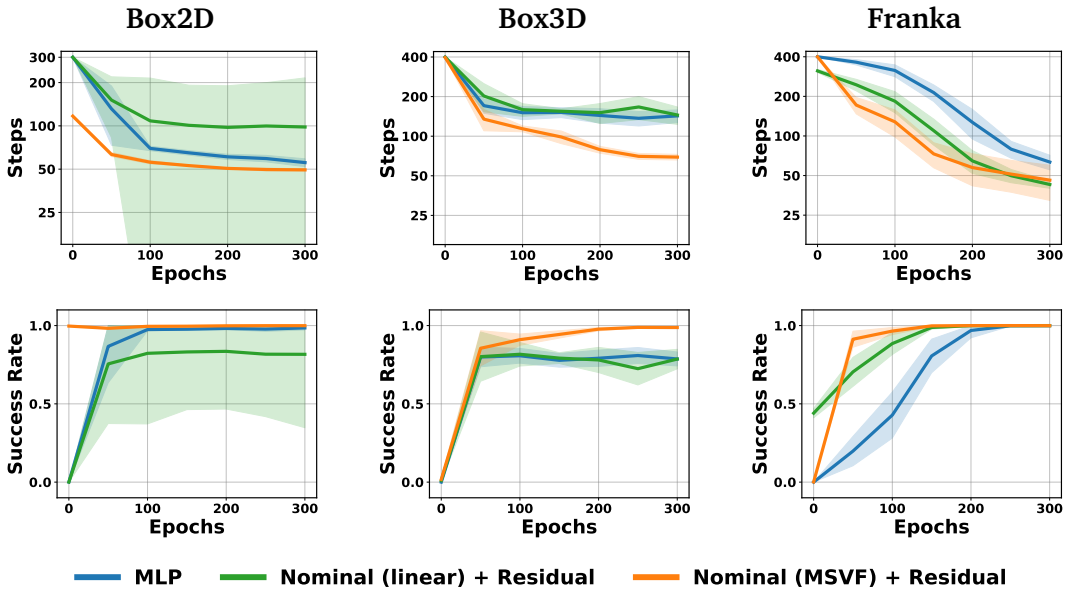

Figure 5.6.: **Performance without learning orientations** - This figure shows how the residual approaches perform compared to the MLP approach when the velocity for the orientation is not learned, but is provided by a linear controller. This controller provides a velocity signal that linearly rotates the object to the desired orientation, and the linear signal is clipped in the same way as when the orientation is learned. The solid line represents the mean of the result for the evaluated approach and epoch and the shaded area shows the 95% confidence interval. Further configurations of the approaches and the environment can be found in Appendix A.

approach benefits from learning orientation, while the non-MSVF approaches seem to learn orientation at some cost.

5.2.4. Variable Impedance Control in Residual Learning

Since not learning the orientation did not provide any benefits to our approach, we will now to evaluate the impact of two other modifications on the performance. The first modification is the implementation of variable impedance control, as explained in Section 4.3, which will be evaluated below. Then, in the following section, we evaluate the impact of the second modification, which includes forces in the observations used to train the residual network, as explained in Section 4.4. For the first modification, we only adjust

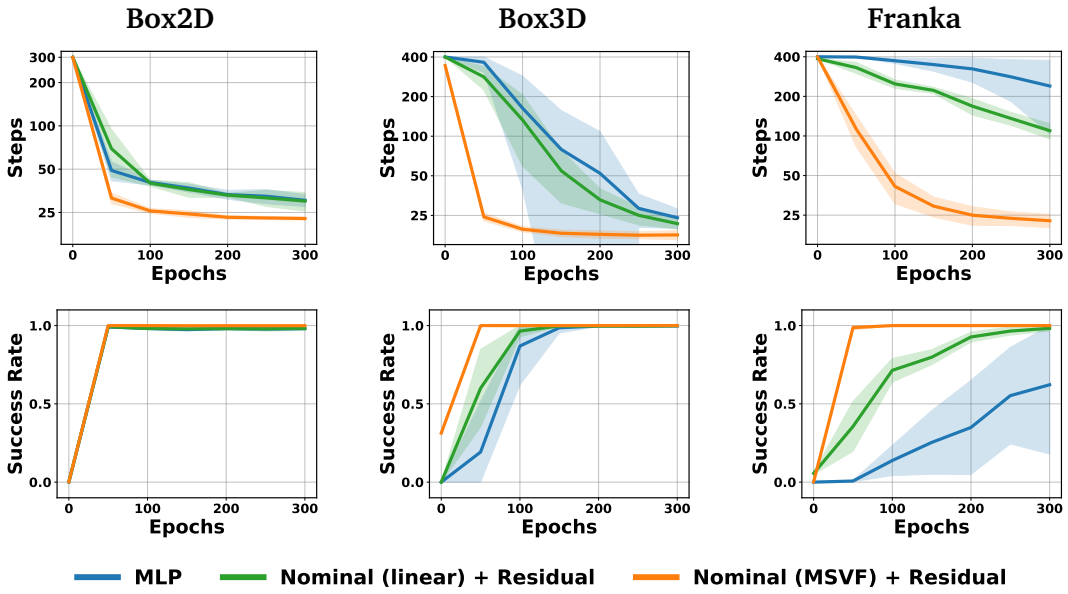

Figure 5.7.: **Performance with additional variable impedance control** - This figure shows the performance of the residual approaches compared to the MLP approach when variable impedance control is added. This means that the output of the residual policies are not only the velocities in terms of position and orientation, but also the gains used to control the robot. The gains learned are the diagonal of the stiffness matrix and the damping matrix is set accordingly to obtain a stable system. The solid line represents the mean of the result for the evaluated approach and epoch and the shaded area shows the 95% confidence interval. Further configurations of the approaches and the environment can be found in Appendix A.

the residual policies so that they also output the stiffness parameters, while the nominal policies remain unchanged.

As Figure 5.7 shows, performing variable impedance control leads to a very good performance. It not only increases the performance of the MSVF approach, but also that of the other two approaches when compared to the basic approach, as shown in Tables 5.1 and 5.2. The increase in performance can be explained partly by the possibility of increasing and decreasing the gains for the different dimensions individually depending on the state, but also by the fact that it is possible to reach a higher maximum gain. We set the maximum gain for this setting to be $2\boldsymbol{K}_P$ and the minimum gain to $0.5\boldsymbol{K}_P$, where $\boldsymbol{K}_P$ is the fixed gain used in the basic approaches.

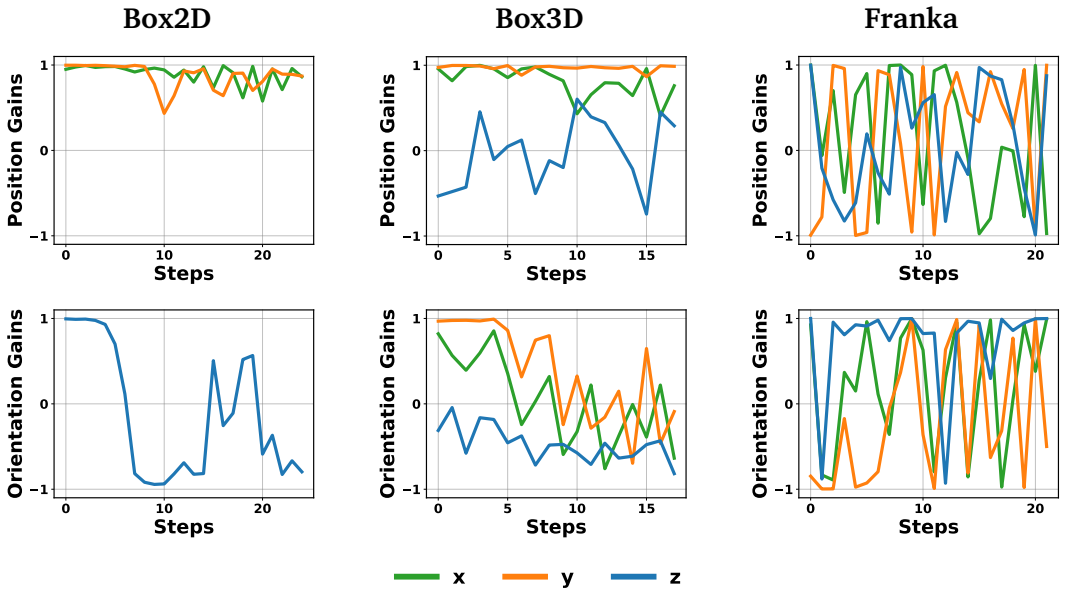

Figure 5.8.: **Evolution of gains during a single epoch** - This figure shows the evolution of gains during an epoch of the best performing policy in all three environments. The values on the x-axis show the current step in the epoch, and the values on the y-axis represent the gains. The gains are not given as they are used in the controller, but before they are mapped to their actual values, i.e., immediately after the Tanh is applied. This means that -1 is mapped to the minimum possible gain and 1 is mapped to the maximum possible gain, and everything in between is mapped linearly in the same way. Further configurations of the approaches and the environment can be found in Appendix A.

Although it could be argued that similar performance could be achieved by simply increasing the gain for the fixed setting, it is important to consider the difference between a fixed and a learned gain. A learned gain can be adapted to the task and allows one to find the optimal gain for each state. Thus, if a small gain is more beneficial than a large one, the policy can learn such a gain, which is not the case with a fixed gain. Another advantage of learning the gains is that one does not need to tune the gains, but can simply set a desired range and learn them. To show that there is a difference in performing variable impedance control compared to using a fixed gain, we show in Figure 5.8 the evolution of the gains during a single epoch of our residual MSVF approach. It can be seen that the gains depend strongly on the state of the system, e.g. in the Box3D environment the initial gain for the z-axis is much lower than for the other axes. This can be explained

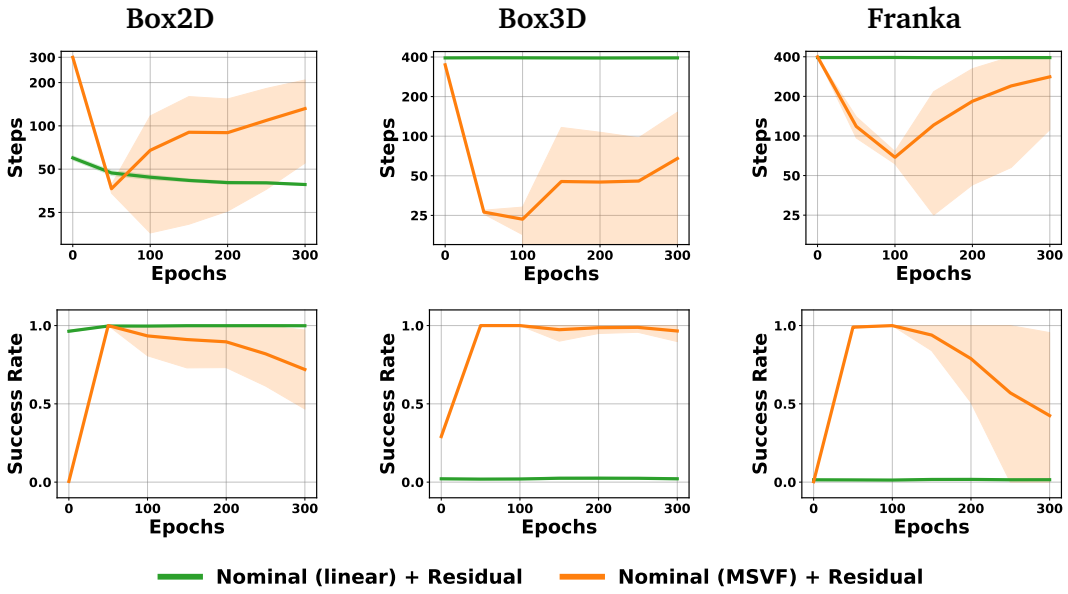

Figure 5.9.: **Performance when the residual only performs variable impedance control and does not output any velocities** - When performing variable impedance control using the residual policy, the need to also output velocities could be eliminated since they are already given by the nominal policy. Therefore, this figure shows how the residual approaches perform when the residual network outputs only the gains for the variable impedance control and the velocities are completely given by the nominal policies. The solid line represents the mean of the result for the evaluated approach and epoch and the shaded area shows the 95% confidence interval. Further configurations of the approaches and the environment can be found in Appendix A.

because it is advantageous to first align the x and y dimensions before increasing the gain for the z dimension and inserting the object.

When performing variable impedance control in residual learning, the need to also learn the position and orientation velocities could vanish, since these are already specified by the nominal policy. To evaluate this scenario, we modify the output of the residual MLP so that it outputs only the stiffness parameters and no velocities at all. In Figure 5.9, we can see that the linear nominal policy performs very well in the two-dimensional task. However, all residual MSVF approaches suffer from the adaptation, as again unlearning appears when they try to perform small local adaptations, as already seen before. The

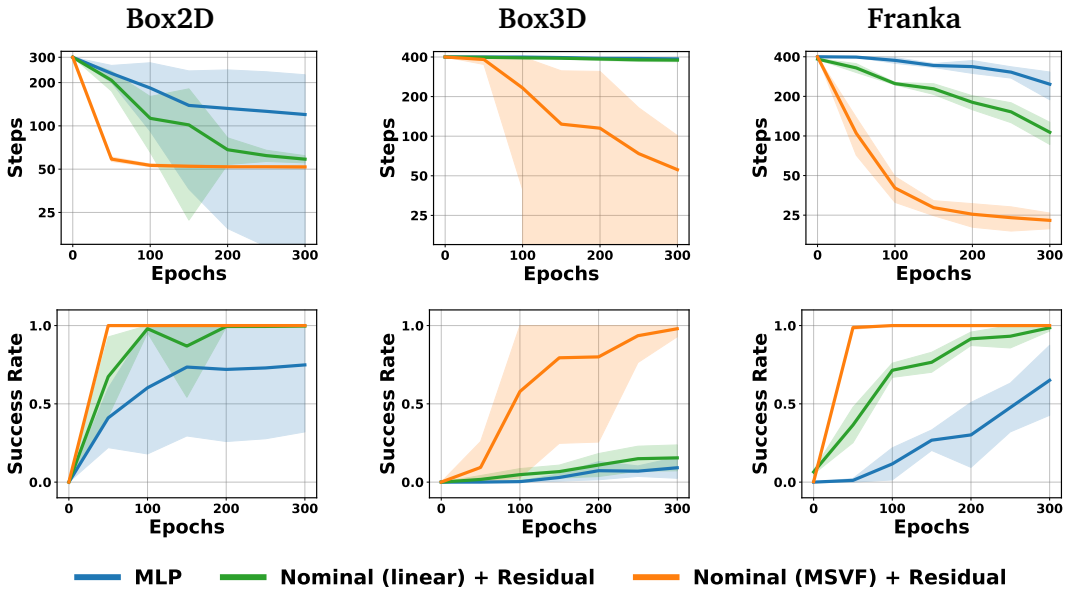

Figure 5.10.: **Performance when the residual also observes forces** - This figure shows how using of a force signal in addition to the standard observation of the environment changes the learning performance of the approaches. For the residual approaches, the force signal is only given to the residual network so as not to affect the nominal policy. The solid line represents the mean of the result for the evaluated approach and epoch and the shaded area shows the 95% confidence interval. Further configurations of the approaches and the environment can be found in Appendix A.

reason why the linear nominal policy does not perform well in three dimensions can be explained by the behavior of the nominal policy near the insertion. When very close to the insertion, the x and y values of the velocities tend to be small because the agent is already close to 0 in these dimensions. On the other hand, the velocity in the z dimension is still quite large because the agent has not yet inserted the object. This leads to very small signals in the x and y dimension, which are very important for the insertion, but are weakened by the large force in the z dimension, so that the agent bumps into the wall.

5.2.5. Force-Feedback

As mentioned earlier, we also want to test the impact of including forces in the observation for the residual MLP to see if this leads to an increase in performance. In general, including

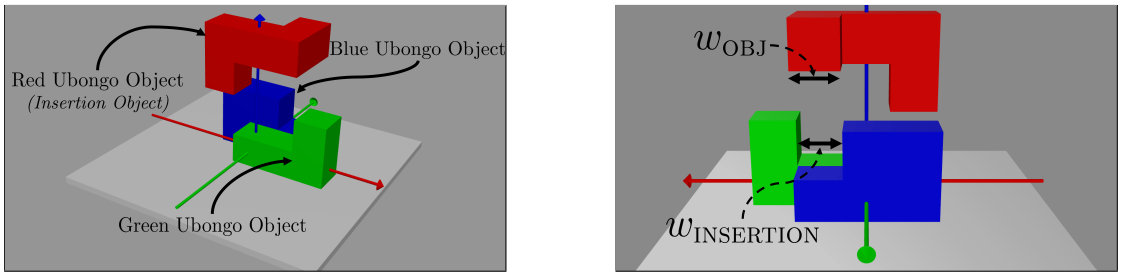

Figure 5.11.: **Ubongo3D task** - This figure shows the Ubongo3D task. In this environment, all objects are formed from linked cubes with the dimension $0.038 \times 0.038 \times 0.038$. The controlled object is the red ubongo object and should be inserted so that it fits exactly into the free space between the green and the blue ubongo objects. The offset defining the insertion width $w_{\text{INSERTION}}$ is equal to $w_{\text{OBJ}} + 0.005$ to provide some free space for the insertion.

forces increases the observation space and can make learning a mapping more difficult, but especially in the case of tasks with many contacts, force can be very useful information. Unfortunately, the implementation of force sensors in IsaacGym is not yet perfect; at the time of the publication of this work, there were several reported problems. For this reason, we decided not to use a force sensor, which may generate false signals, but to use the simulator's contact forces as an approximation to a force sensor.

As Figure 5.10 shows, observing the force does not really help, but actually leads to slightly worse results than previous settings, which can also be seen in Tables 5.1 and 5.2. However, it seems that the residual MSVF approach is not affected as much by learning the force, which can be explained by the fact that the MSVF is still learned without force and therefore learns in the same observation space as before, and only the residual MLP has to learn in a larger observation space. Still, the results should be treated with caution because the force signal is only a rough approximation. We believe that this information could be very useful for learning in a real system when using a real force sensor. Also, since the performance of the residual MSVF approach did not degrade much in the experiment, learning with a force sensor in the real system should not be problematic since the residual MSVF approach seems to be quite consistent.

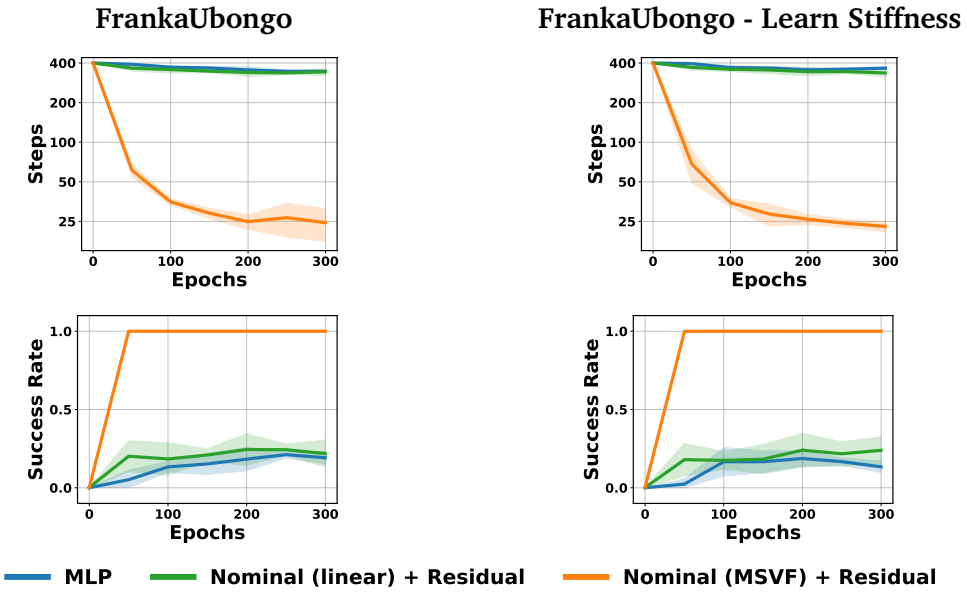

Figure 5.12.: **Performance for the Franka Ubongo3D task** - This figure shows on the left the performance of the approaches with residual learning for the FrankaUbongo environment compared to the MLP approach. The same experiment is shown on the right, but with variable impedance control. The solid line represents the mean of the result for the evaluated approach and epoch and the shaded area shows the 95% confidence interval. Further configurations of the approaches and the environment can be found in Appendix A.

5.2.6. Alternative Insertion Task - Ubongo3D

To evaluate how our approach adapts to different insertion tasks, we also want to evaluate it in another environment called FrankaUbongo. This environment is inspired by the board game Ubongo3D [77] and has already been used for real world robot object manipulation [19]. The considered task is illustrated in Figure 5.11, and for the manipulation of the insertion object we use again the Franka robot. The general setup such as friction, starting pose and others is identical to the Box3D-Franka environment presented in Section 5.1.1, only the objects are changed to reflect the Ubongo3D task. In this task, we also do not fix the object with a certain distance to the end-effector, but place it directly between the grippers of the Franka robot, as shown in Figure 5.14. Furthermore, the object cannot move independently, as if it were firmly stuck to the grippers.

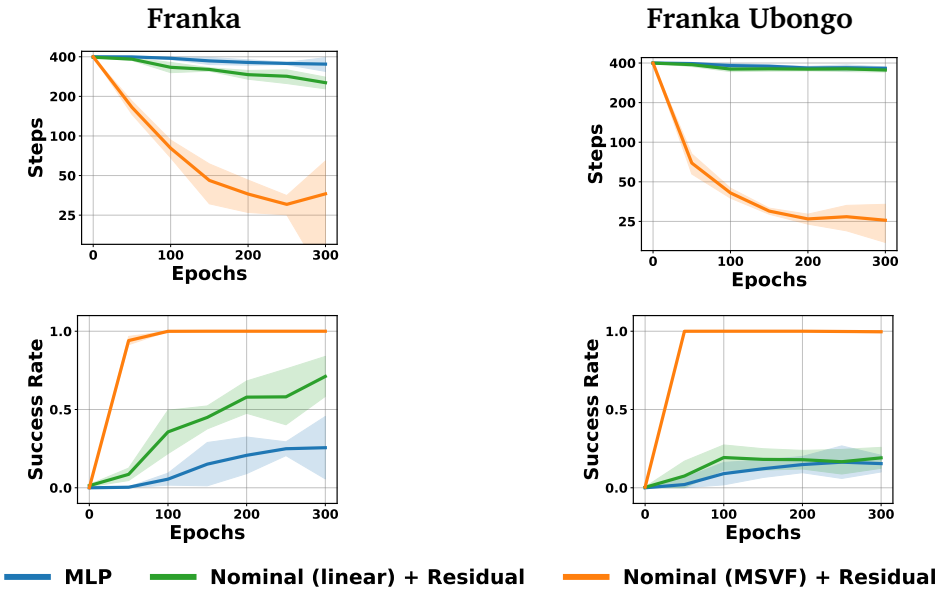

Figure 5.13.: **Performance with a reduced clearance** - This figure shows the performance at a reduced clearance of 2 millimeters in the Box3D and Ubongo3D environments with the Franka robot. All approaches also perform variable impedance control as this gave the best performance in previous experiments. The solid line represents the mean of the result for the evaluated approach and epoch and the shaded area shows the 95% confidence interval. Further configurations of the approaches and the environment can be found in Appendix A.

To evaluate the performance for this task, we again compare our residual MSVF approach against a nominal linear policy using an MLP as residual network and an MLP without a residual policy. In addition, we also evaluate the performance of performing variable impedance control, as it gave the best performance for the Box3D-Franka environment. The results are shown in Figure 5.12, and again we can see that the residual MSVF approach outperforms the other approaches. The performance of the MLP and the linear approach decrease drastically compared to the previous Franka task, as can also be seen in Tables 5.1 and 5.2. We believe this is due to the added complexity for the insertion. In contrast, the residual MSVF approach is able to achieve almost the same performance as in the previous task. An illustration of a single run of the best performing policy for the FrankaUbongo task can be seen in in Figure 5.14.

5.2.7. Less clearance

As we always have used a clearance of 0.005 so far, which corresponds to 5 millimeters, we now want to reduce this clearance to show that our approach is also able to learn the task when the insertion becomes more difficult. For this reason, we will again evaluate the best performing approaches for the Franka Box3D and Ubongo3D environments, but now reduce the clearance to 0.002, or 2 millimeters.

As can be seen in Figure 5.13, the performance of all approaches decreases when the clearance is reduced from 5 to 2 millimeters. Although this was to be expected, it is still the case that the residual MSVF approach performs better than the others and solves the task with a success rate of 1.0. Interestingly, the reduction in clearance seems to have less effect on the Ubongo3D task than on the Box3D. This can be explained by the fact that in the Ubongo3D, the clearance is reduced in only one dimension, as the blue and green Ubongo objects move closer together. In the Box3D task, the clearance is reduced in two dimensions because the insertion object becomes wider, which could explain the larger performance drop.

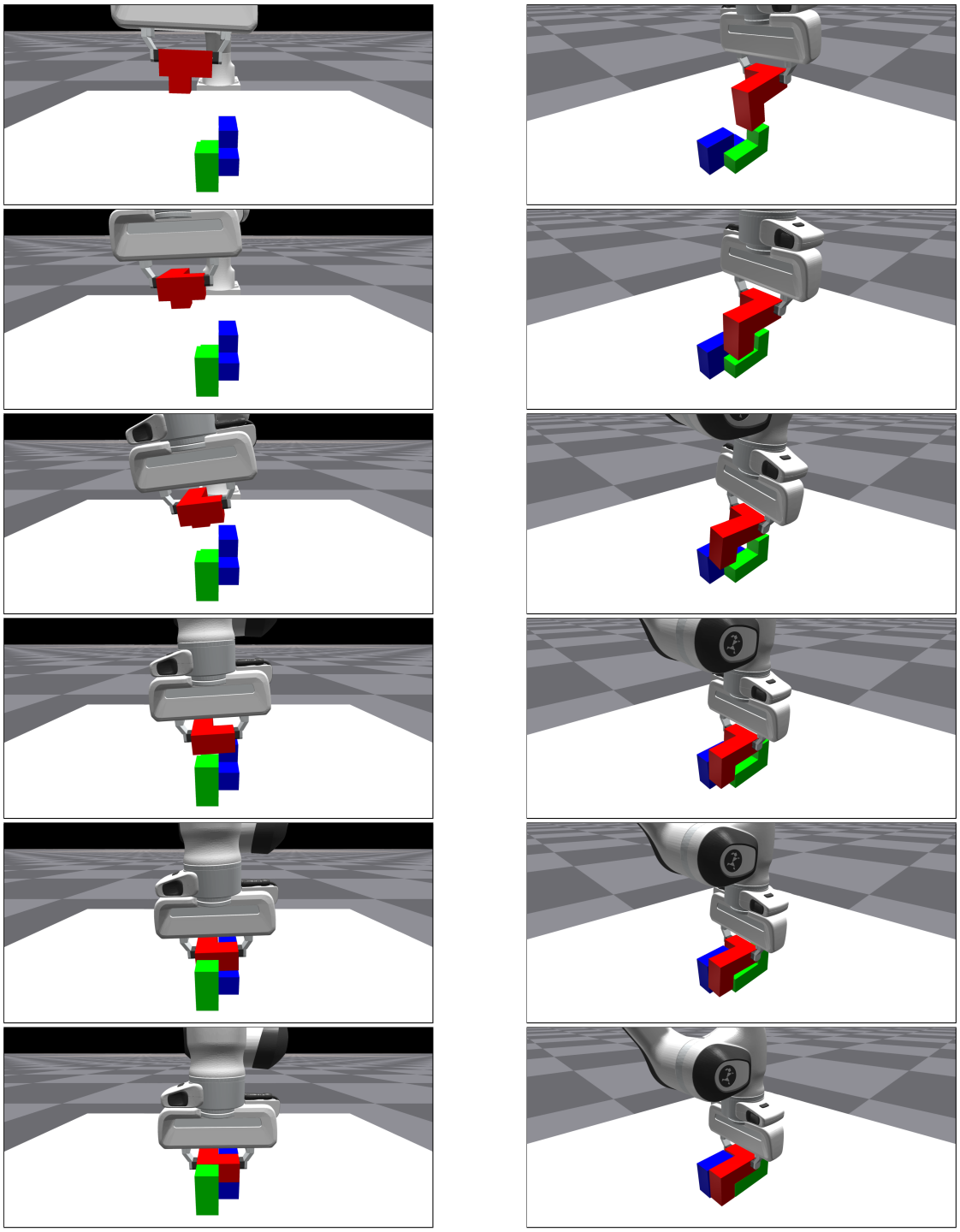

Figure 5.14.: **Example run of the best policy in the Franka Ubongo environment**

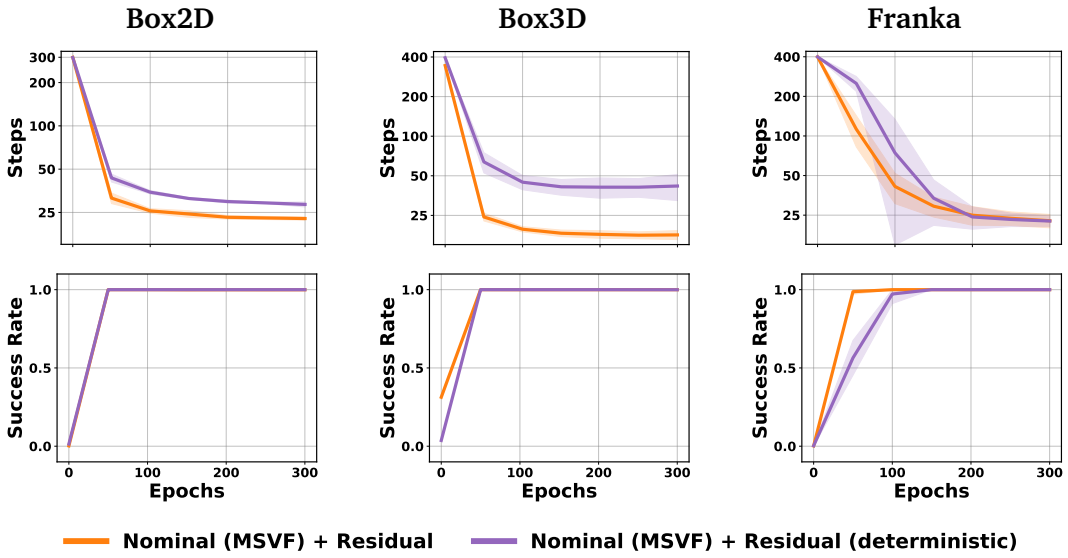

Figure 5.15.: **Performance of a deterministic policy** - In this figure, the regular stochastic policy is compared with a deterministic version of the same policy. Both policies are identical, except that in the deterministic version only the mean of the the Gaussian was used as output, i.e. the output was sampled without stochasticity for evaluation. Both policies are trained as explained in Section 5.2.4 and represent our residual MSVF approach with variable impedance control. The solid line represents the mean of the result for the evaluated approach and epoch and the shaded area shows the 95% confidence interval. Further configurations of the approaches and the environment can be found in Appendix A.

5.2.8. Extraction of a Deterministic Policy

As this work builds the foundation for further research, which should also include the adaptation of the approach to a real-world scenario, we also want to evaluate the performance of the learned policy when there is no stochasticity, i.e. its deterministic. A deterministic policy is often desirable in the real world, as stochastic policies might actually damage the robot. Since the noise is usually decreased over training, the final policy will often be near deterministic and therefore one can make it deterministic by only considering the mean and not the variance of the Gaussian. For this reason, we compare the best performing policy (Nominal (MSVF) + Residual with variable impedance control), with itself but only using the mean of the Gaussian without any variance to generate actions.

In Figure 5.15 it can be seen that just using the mean of the Gaussian without any variance, we receive a deterministic policy that nearly performs as good as the stochastic one. There are performance losses, yet all approaches still have a success rate of 1.0, in addition those loses are quite minimal for the Franka environment.

5.3. Conclusion of Experiments

An MSVF alone, without adaptations, is apparently not suitable to be used for reinforcement learning in insertion tasks. However, in our experiments, we have shown that the residual MSVF approach outperforms the approaches against which we have compared it. It outperforms not only an MLP, but also an appropriate residual counterpart where we replaced the MSVF with a linear nominal policy. The final performance and the success rate of the residual MSVF approach are either much higher or at least similar to the other approaches, as can also be seen in Tables 5.1 and 5.2. Furthermore, we have shown that the residual MSVF approach is not only capable of learning the desired velocities, but also of performing variable impedance control. Although the current implementation of force feedback is error-prone, the residual MSVF approach was also able to incorporate such force feedback and still learn a reasonable policy, while other approaches mostly suffered from this modification. We then showed that we can not only solve a box insertion task using the residual MSVF approach, but also handle more difficult insertion tasks such as the Ubongo3D task without further tuning. In addition, we evaluated how our approach performed when the insertion clearance was reduced or when a deterministic version of the policy was used for evaluation. Both resulted in minor performance losses, but the approach was still able to achieve a success rate of 1.0. Overall, we have shown that our residual MSVF approach not only increases final performance, but also learns faster than the approaches we compared it to.

Comparison of the Number Steps Required to Insert the Object					
Method		Box2D	Box3D	Franka	Ubongo3D
Baseline	MLP	61.92 ± 3.55	164.81 ± 111.32	177.52 ± 110.19	345.24 ± 10.43
	MSVF	169.41 ± 85.72	109.41 ± 154.45	252.55 ± 169.71	-
	Residual (Linear)	63.25 ± 4.05	183.16 ± 45.28	93.12 ± 20.83	342.89 ± 20.64
	Residual (MSVF)	51.82 ± 0.8	33.82 ± 2.27	22.87 ± 3.44	24.46 ± 6.86
Not Learning Rotation	MLP	55.53 ± 3.48	142.68 ± 15.77	63.32 ± 8.06	-
	Residual (Linear)	98.18 ± 117.56	144.85 ± 21.86	42.75 ± 2.75	-
	Residual (MSVF)	49.41 ± 0.81	69.34 ± 3.41	46.14 ± 13.79	-
Variable Impedance Control	MLP	30.36 ± 2.89	23.99 ± 4.06	239.71 ± 136.29	365.61 ± 8.24
	Residual (Linear)	29.98 ± 4.4	21.63 ± 1.94	109.63 ± 14.6	335.99 ± 20.92
	Residual (MSVF)	$\mathbf{22.67 \pm 0.56}$	$\mathbf{17.76 \pm 1.39}$	$\mathbf{22.66 \pm 2.64}$	$\mathbf{22.91 \pm 1.97}$
Observing Force	MLP	119.94 ± 106.84	386.25 ± 10.54	247.36 ± 59.06	-
	Residual (Linear)	58.67 ± 3.37	378.75 ± 10.62	106.62 ± 20.5	-
	Residual (MSVF)	51.82 ± 1.4	55.64 ± 45.39	22.82 ± 3.12	-

Table 5.1.: **Comparison of number of steps** - This table shows the results of all evaluated approaches in the different environments. The shown values are the number of steps required to insert the object and refer to the last epoch, i.e. epoch 300. The number of steps is given in $a \pm b$ where a is the mean over all seeds and b describes the confidence interval. Baseline refers to the experiment in Sections 5.2.1 and 5.2.2, not learning rotations to Section 5.2.3, variable impedance control to Section 5.2.4 and observing force to Section 5.2.5. The last column refers to the FrankaUbongo task presented in Section 5.2.6.

Comparison of the Achieved Success Rates					
Method		Box2D	Box3D	Franka	Ubongo3D
Baseline	MLP	0.99 ± 0.01	0.89 ± 0.23	0.82 ± 0.30	0.19 ± 0.04
	MSVF	0.66 ± 0.34	0.85 ± 0.39	0.55 ± 0.57	-
	Residual (Linear)	0.99 ± 0.01	0.95 ± 0.04	0.99 ± 0.01	0.22 ± 0.09
	Residual (MSVF)	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$
Not Learning Rotation	MLP	0.99 ± 0.01	0.79 ± 0.04	$\mathbf{1.00 \pm 0.00}$	-
	Residual (Linear)	0.82 ± 0.47	0.79 ± 0.06	$\mathbf{1.00 \pm 0.00}$	-
	Residual (MSVF)	$\mathbf{1.00 \pm 0.00}$	0.99 ± 0.01	1.00 ± 0.00	-
Variable Impedance Control	MLP	0.98 ± 0.01	$\mathbf{1.00 \pm 0.00}$	0.62 ± 0.44	0.13 ± 0.04
	Residual (Linear)	0.98 ± 0.01	$\mathbf{1.00 \pm 0.00}$	0.98 ± 0.02	0.24 ± 0.09
	Residual (MSVF)	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$
Observing Force	MLP	0.75 ± 0.43	0.09 ± 0.07	0.65 ± 0.22	-
	Residual (Linear)	$\mathbf{1.00 \pm 0.00}$	0.16 ± 0.08	0.99 ± 0.02	-
	Residual (MSVF)	$\mathbf{1.00 \pm 0.00}$	0.98 ± 0.05	$\mathbf{1.00 \pm 0.00}$	-

Table 5.2.: **Comparison of success rates** - This table shows the results of all evaluated approaches in the different environments. The shown values are the success rates achieved in the last epoch, i.e. epoch 300. The success rates are given in $a \pm b$ where a is the mean over all seeds and b describes the confidence interval. Baseline refers to the experiment in Sections 5.2.1 and 5.2.2, not learning rotations to Section 5.2.3, variable impedance control to Section 5.2.4 and observing force to Section 5.2.5. The last column refers to the FrankaUbongo task presented in Section 5.2.6.

6. Conclusion and Future Research

Throughout this work, we have shown that MSVFs are not yet suitable for use in reinforcement learning without further adaption. Although they have great advantages, such as an intuitive and less complex representation of orientations through the use of Lie groups or the ability to define a stable policy, they yet suffer from the constraints required to do so. As we have shown, the most limiting factor is the learning of the diffeomorphism to the latent stable vector field. By definition, this mapping is smooth and therefore represents a continuous deformation of the space. However, an advantageous property of a learnable policy for RL is the ability to achieve small adjustments through small changes in the weights. This property allows the gradient to be used to perform small updates in a direction that is likely to lead to higher reward. An MSVF is not able to do this for all types of adjustments. To provide what might be a fairly small change in the output, the mapping described by the diffeomorphism would eventually have to tear the space apart. It becomes even more difficult if this change is to be made only very locally, since this would require large changes in the weights to affect the mapping only locally.

Nevertheless, an MSVF is able to describe complex, stable vector fields and process orientations in an appropriate way, making it a highly desirable approach for RL in robotics. To exploit these advantages, we proposed to use MSVFs in residual policy learning and therefore combine it with a residual network. This residual network is capable of learning small local adaptations with just small changes in the weights, and as we have shown in our experiments, this together forms a very powerful approach. Not only this approach outperforms a traditional MLP by far, but also outperforms another residual learning approach where the nominal policy is defined by a linear stable vector field that is not learned. Moreover, we have shown that this residual MSVF approach is able to perform variable impedance control. This allows the gains of the controller to be learned, leading to even better performance of the final policy since the gains no longer need to be set manually. In addition, we have investigated how the residual MSVF approach handles an extended observation space that also includes information about the forces at the end-effector. Although the implementation of the force feedback in the

simulator used is not yet flawless, the residual MSVF approach was able perform similarly to previous experiments. Furthermore, we have shown that the residual MSVF approach is also capable of learning insertion tasks other than a simple box insertion. We evaluated this using the Ubongo3D environment, where object insertion is more complex.

Future Research

Since this work is, to our knowledge, the first work to apply MSVFs to RL, it builds a foundation for future research that could improve upon the results shown here. The first direct extension of this work is to apply the residual MSVF approach to a real robot system to perform an insertion task like that of Ubongo3D. For this purpose, we believe it is necessary to switch from using PPO to more sample efficient methods like TD3 [78]. Moreover, various adjustments should be made to the training for sim-to-real, which can be efficiently implemented in the current framework, since IsaacGym already provides many of them. It could also be interesting to first use the BC version of MSVFs [21] to use human demonstrations and learn an initial policy, which is then fine-tuned by the presented residual MSVF approach using RL in the real system. In this scenario, the change in performance due to the use of real-world force sensors could also be of interest, as these had to be approximated in the simulation by using contact forces.

In addition, we think it would be interesting to extend the research on MSVFs in terms of their internal structure. It would be particularly interesting not to assume that the goal is known a priori but to build an estimate of the goal, i.e. by using visual information. This information could then be used to estimate the target of the latent stable vector field, allowing to extend the use of MSVFs to tasks where the goal is not known or is uncertain. Finally, we believe it is important to also research whether there are adaptations that allow an MSVF to perform reasonably well in RL without a residual network. We believe that the limitation that small local adaptations cannot easily be learned can be overcome by adjusting the properties of the NODE that describes the diffeomorphism. A novel formulation of this NODE, which could include attractive or repulsive forces or a combination of local NODEs that together form a global NODE, could provide the ability to easily perform local adjustments to the output. This would allow for not only training only an MSVF, but it would also lead to a guaranteed stable policy, which is highly desirable for robotics. This future research could also lead to extending the research area of MSVFs in RL to not only more complex insertion tasks, but to all kinds of tasks where a stable learnable policy is desired.

Bibliography

[1] J. Padmanabhan and M. J. Johnson Premkumar, "Machine learning in automatic speech recognition: A survey," *IETE Technical Review*, vol. 32, no. 4, pp. 240–251, 2015.

[2] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE access*, vol. 8, pp. 58443–58469, 2020.

[3] W. Wang, Y. Yang, X. Wang, W. Wang, and J. Li, "Development of convolutional neural network and its application in image classification: a survey," *Optical Engineering*, vol. 58, no. 4, p. 040901, 2019.

[4] O. Kroemer, S. Niekum, and G. Konidaris, "A review of robot learning for manipulation: Challenges, representations, and algorithms," *The Journal of Machine Learning Research*, vol. 22, no. 1, pp. 1395–1476, 2021.

[5] D. Mukherjee, K. Gupta, L. H. Chang, and H. Najjaran, "A survey of robot learning strategies for human-robot collaboration in industrial settings," *Robotics and Computer-Integrated Manufacturing*, vol. 73, p. 102231, 2022.

[6] Z. Zhu and H. Hu, "Robot learning from demonstration in robotic assembly: A survey," *Robotics*, vol. 7, p. 17, Apr 2018.

[7] S. G. Khan, G. Herrmann, M. Al Grafi, T. Pipe, and C. Melhuish, "Compliance control and human–robot interaction: Part 1—survey," *International Journal of Humanoid Robotics*, vol. 11, no. 03, p. 1430001, 2014.

[8] J. Xu, Z. Hou, Z. Liu, and H. Qiao, "Compare contact model-based control and contact model-free learning: A survey of robotic peg-in-hole assembly strategies," 2019.

[9] S. Höfer, K. Bekris, A. Handa, J. C. Gamboa, M. Mozifian, F. Golemo, C. Atkeson, D. Fox, K. Goldberg, J. Leonard, *et al.*, "Sim2real in robotics and automation: Applications and challenges," *IEEE transactions on automation science and engineering*, vol. 18, no. 2, pp. 398–400, 2021.

[10] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[11] T. Inoue, G. De Magistris, A. Munawar, T. Yokoya, and R. Tachibana, "Deep reinforcement learning for high precision assembly tasks," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 819–825, 2017.

[12] M. A. Lee, Y. Zhu, K. Srinivasan, P. Shah, S. Savarese, L. Fei-Fei, A. Garg, and J. Bohg, "Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8943–8950, IEEE, 2019.

[13] G. Thomas, M. Chien, A. Tamar, J. A. Ojea, and P. Abbeel, "Learning robotic assembly from cad," 2018.

[14] Y. Fan, J. Luo, and M. Tomizuka, "A learning framework for high precision industrial assembly," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 811–817, 2019.

[15] M. A. Lee, C. Florensa, J. Tremblay, N. Ratliff, A. Garg, F. Ramos, and D. Fox, "Guided uncertainty-aware policy optimization: Combining learning and model-based strategies for sample-efficient policy learning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7505–7512, IEEE, 2020.

[16] T. Silver, K. Allen, J. Tenenbaum, and L. Kaelbling, "Residual policy learning," *arXiv preprint arXiv:1812.06298*, 2018.

[17] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. A. Ojea, E. Solowjow, and S. Levine, "Residual reinforcement learning for robot control," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 6023–6029, IEEE, 2019.

[18] T. Davchev, K. S. Luck, M. Burke, F. Meier, S. Schaal, and S. Ramamoorthy, "Residual learning from demonstration: Adapting dmps for contact-rich manipulation," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4488–4495, 2022.

[19] J. Carvalho, D. Koert, M. Daniv, and J. Peters, "Adapting object-centric probabilistic movement primitives with residual reinforcement learning," in *2022 IEEE-RAS 21st International Conference on Humanoid Robots (Humanoids)*, 2022.

[20] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li, "On the continuity of rotation representations in neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5745–5753, 2019.

[21] J. Urain, D. Tateo, and J. Peters, "Learning stable vector fields on lie groups," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 12569–12576, 2022.

[22] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[23] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[24] C. Szepesvári, "Algorithms for reinforcement learning," *Synthesis lectures on artificial intelligence and machine learning*, vol. 4, no. 1, pp. 1–103, 2010.

[25] R. A. Howard, "Dynamic programming and markov processes.," 1960.

[26] J. Hammersley, *Monte carlo methods*. Springer Science & Business Media, 2013.

[27] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.

[28] M. P. Deisenroth, G. Neumann, J. Peters, *et al.*, "A survey on policy search for robotics," *Foundations and Trends® in Robotics*, vol. 2, no. 1–2, pp. 1–142, 2013.

[29] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in neural information processing systems*, vol. 12, 1999.

[30] J. Peters and S. Schaal, "Policy gradient methods for robotics," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2219–2225, IEEE, 2006.

[31] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," *Advances in neural information processing systems*, vol. 12, 1999.

[32] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1291–1307, 2012.

[33] T. Degris, M. White, and R. S. Sutton, "Off-policy actor-critic," *arXiv preprint arXiv:1205.4839*, 2012.

[34] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[35] T. M. Moerland, J. Broekens, and C. M. Jonker, "Model-based reinforcement learning: A survey," *arXiv preprint arXiv:2006.16712*, 2020.

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

[37] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, pp. 1889–1897, PMLR, 2015.

[38] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[39] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[40] K. M. Lynch and F. C. Park, *Modern robotics*. Cambridge University Press, 2017.

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

[42] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *IEEE Journal on Robotics and Automation*, vol. 3, no. 1, pp. 43–53, 1987.

[43] J. Nakanishi, R. Cory, M. Mistry, J. Peters, and S. Schaal, "Operational space control: A theoretical and empirical comparison," *The International Journal of Robotics Research*, vol. 27, no. 6, pp. 737–757, 2008.

[44] J. Peters and S. Schaal, "Learning to control in operational space," *The International Journal of Robotics Research*, vol. 27, p. 197, 02 2008.

[45] J. Peters, M. Mistry, F. Udwadia, R. Cory, J. Nakanishi, and S. Schaal, "A unifying methodology for the control of robotic systems,"

[46] J. Lee, *Introduction to topological manifolds*, vol. 202. Springer Science & Business Media, 2010.

[47] S. Lang, *Fundamentals of differential geometry*, vol. 191. Springer Science & Business Media, 2012.

[48] J. M. Lee, "Smooth manifolds," in *Introduction to smooth manifolds*, pp. 1–31, Springer, 2013.

[49] B. C. Hall, "Lie groups, lie algebras, and representations," in *Quantum Theory for Mathematicians*, pp. 333–366, Springer, 2013.

[50] P. Hartman, *Ordinary differential equations*. SIAM, 2002.

[51] J. H. Hubbard and B. B. Hubbard, *Vector calculus, linear algebra, and differential forms: a unified approach*. Matrix Editions, 2015.

[52] F. W. Warner, *Foundations of differentiable manifolds and Lie groups*, vol. 94. Springer Science & Business Media, 1983.

[53] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," *Advances in neural information processing systems*, vol. 31, 2018.

[54] M. Suomalainen, Y. Karayiannidis, and V. Kyrki, "A survey of robot manipulation in contact," *Robotics and Autonomous Systems*, vol. 156, p. 104224, 2022.

[55] A. Wan, J. Xu, H. Chen, S. Zhang, and K. Chen, "Optimal path planning and control of assembly robots for hard-measuring easy-deformation assemblies," *IEEE/ASME Transactions on Mechatronics*, vol. 22, no. 4, pp. 1600–1609, 2017.

[56] T. Tang, H.-C. Lin, and M. Tomizuka, "A learning-based framework for robot peg-hole-insertion," in *Dynamic Systems and Control Conference*, vol. 57250, p. V002T27A002, American Society of Mechanical Engineers, 2015.

[57] T. Tang, H.-C. Lin, Y. Zhao, Y. Fan, W. Chen, and M. Tomizuka, "Teach industrial robots peg-hole-insertion by human demonstration," in *2016 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, pp. 488–494, IEEE, 2016.

[58] C. Chang, K. Haninger, Y. Shi, C. Yuan, Z. Chen, and J. Zhang, "Impedance adaptation by reinforcement learning with contact dynamic movement primitives," 2022.

[59] C. C. Beltran-Hernandez, D. Petit, I. G. Ramirez-Alpizar, and K. Harada, "Variable compliance control for robotic peg-in-hole assembly: A deep-reinforcement-learning approach," *Applied Sciences*, vol. 10, p. 6923, oct 2020.

[60] J. Luo, E. Solowjow, C. Wen, J. A. Ojea, and A. M. Agogino, "Deep reinforcement learning for robotic assembly of mixed deformable and rigid objects," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2062–2069, IEEE, 2018.

[61] S. A. Khader, H. Yin, P. Falco, and D. Kragic, "Stability-guaranteed reinforcement learning for contact-rich manipulation," *CoRR*, vol. abs/2004.10886, 2020.

[62] S. A. Khader, H. Yin, P. Falco, and D. Kragic, "Learning stable normalizing-flow control for robotic manipulation," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1644–1650, 2021.

[63] S. Levine, N. Wagener, and P. Abbeel, "Learning contact-rich manipulation skills with guided policy search (2015)," *arXiv preprint arXiv:1501.05611*, 2015.

[64] P. Kulkarni, J. Kober, R. Babuška, and C. Della Santina, "Learning assembly tasks in a few minutes by combining impedance control and residual recurrent reinforcement learning," *Advanced Intelligent Systems*, vol. 4, no. 1, p. 2100095, 2022.

[65] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical movement primitives: learning attractor models for motor behaviors," *Neural computation*, vol. 25, no. 2, pp. 328–373, 2013.

[66] M. J. Zeestraten, I. Havoutis, J. Silvério, S. Calinon, and D. G. Caldwell, "An approach for imitation learning on riemannian manifolds," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1240–1247, 2017.

[67] S. Levine, N. Wagener, and P. Abbeel, "Learning contact-rich manipulation skills with guided policy search (2015)," *arXiv preprint arXiv:1501.05611*, 2015.

[68] R. Martín-Martín, M. A. Lee, R. Gardner, S. Savarese, J. Bohg, and A. Garg, "Variable impedance control in end-effector space: An action space for reinforcement learning in contact-rich tasks," *CoRR*, vol. abs/1906.08880, 2019.

[69] F. J. Abu-Dakka and M. Saveriano, "Variable impedance control and learning – a review," 2020.

[70] L. Roveda, M. Beschi, N. Pedrocchi, and L. Molinari Tosatti, "High-accuracy robotized industrial assembly task control schema with force overshoots avoidance," *Control Engineering Practice*, vol. 71, 10 2017.

[71] F. Dimeas and N. Aspragathos, "Reinforcement learning of variable admittance control for human-robot co-manipulation," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1011–1016, 2015.

[72] T. Davchev, K. S. Luck, M. Burke, F. Meier, S. Schaal, and S. Ramamoorthy, "Residual learning from demonstration," *CoRR*, vol. abs/2008.07682, 2020.

[73] W. Pawlak, "Wear and coefficient of friction of pla - graphite composite in 3d printing technology.," 05 2018.

[74] S. Haddadin, S. Parusel, L. Johannsmeier, S. Golz, S. Gabl, F. Walch, M. Sabaghian, C. Jähne, L. Hausperger, and S. Haddadin, "The franka emika robot: A reference platform for robotics research and education," *IEEE Robotics & Automation Magazine*, vol. 29, pp. 2–20, 06 2022.

[75] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, *et al.*, "Isaac gym: High performance gpu-based physics simulation for robot learning," *arXiv preprint arXiv:2108.10470*, 2021.

[76] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[77] Kosmos Games, "Ubongo 3d." `https://www.thamesandkosmos.co.uk/product/ubongo-3d/`, 2022. [Online; accessed 14-December-2022].

[78] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*, pp. 1587–1596, PMLR, 2018.

[79] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," 2019.

[80] D. Makoviichuk and V. Makoviychuk, "rl-games: A high-performance framework for reinforcement learning." `https://github.com/Denys88/rl_games`, May 2022.

[81] L. Pineda, T. Fan, M. Monge, S. Venkataraman, P. Sodhi, R. T. Chen, J. Ortiz, D. DeTone, A. Wang, S. Anderson, J. Dong, B. Amos, and M. Mukadam, "Theseus: A Library for Differentiable Nonlinear Optimization," *Advances in Neural Information Processing Systems*, 2022.

[82] R. T. Q. Chen, "torchdiffeq," 2018.

A. Parameters for Experiments

Environment Configurations

Box2D	
$\boldsymbol{K}_P$ position	100
$\boldsymbol{K}_P$ orientation	25
$\boldsymbol{K}_D$	$2\sqrt{\boldsymbol{M}\boldsymbol{K}_P}$
initial state position	$x \in [0.8, 0.99],\ y \in [-0.99, 0.99]$
initial state orientation	$\theta_z \in [-\pi/2, pi/2]$
maximum steps	300
maximum linear velocity norm	0.2
maximum angular velocity norm	0.1

Box3D	
$\boldsymbol{K}_P$ position	100
$\boldsymbol{K}_P$ orientation	25
$\boldsymbol{K}_D$	$2\sqrt{\boldsymbol{M}\boldsymbol{K}_P}$
initial state position	$x \in [-0.99, 0.99]],\ y \in [-0.99, 0.99],\ z \in [0.8, 0.99]$
initial state orientation	$\theta_x, \theta_y, \theta_z \in [-\pi/2, pi/2]$
maximum steps	400
maximum linear velocity norm	0.7
maximum angular velocity norm	0.7

Franka Box3D & Ubongo3D	
$\boldsymbol{K}_P$	150 per dimension
$\boldsymbol{K}_D$	$2\sqrt{\boldsymbol{K}_P}$
Nullspace $\boldsymbol{K}_{Pnull}$	10 per dimension
Nullspace $\boldsymbol{K}_D$	$2\sqrt{\boldsymbol{K}_{Pnull}}$
initial pose	$\boldsymbol{q}_0 = (-0.1692, 0.4817, 0.1881, -1.8218, -0.1215, 2.3025, 0.8042)$
noise in initial pose	0.05
rest pose	equal to initial pose
maximum steps	400
maximum linear velocity norm	0.7
maximum angular velocity norm	0.7

PPO Configurations

PPO	
log variance	-0.5
critic coefficient	1
entropy coefficient	0.01
learning rate critic	10^{-3}
learning rate actor	$3 \cdot 10^{-4}$
weight decay	10^{-6}
discount factor	0.995
lambda (GAE)	0.95
clip PPO loss	0.2
horizon length	100
minibatch size critic	1000
minibatch size actor	1000
mini epochs critic	4
mini epochs actor	4
normalize input	$False$
normalize value	$True$
normalize advantage	$True$

Approach Configurations

MSVF	
NODE	
type	fully connected neural network
units	[32,32]
activation	Leaky ReLu
solver steps	10
Scaling Network	
type	fully connected neural network
units	[32,32]
activation	Leaky ReLu
Critic Network	
type	fully connected neural network
units	[64,64]
activation	ReLu

MLP	
Residual Network	
type	fully connected neural network
units	[64,64]
activation	ReLu
Critic Network	
type	fully connected neural network
units	[64,64]
activation	ReLu

Nominal (linear) + Residual	
Residual Network	
type	fully connected neural network
units	[64,64]
activation	ReLu
Critic Network	
type	fully connected neural network
units	[64,64]
activation	ReLu

Nominal (MSVF) + Residual	
NODE	
type	fully connected neural network
units	[32,32]
activation	Leaky ReLu
solver steps	10
Scaling Network	
type	fully connected neural network
units	[32,32]
activation	Leaky ReLu
Residual Network	
type	fully connected neural network
units	[64,64]
activation	ReLu
Critic Network	
type	fully connected neural network
units	[64,64]
activation	ReLu

B. Enumeration of Important Libraries

In the following we would like to mention important libraries that we have used throughout this thesis. We think it is important not only to reproduce the results, but also to give an insight in how we managed to implement our approach and experiments.

1. PyTorch [79]: PyTorch is an incredibly well known library and is a very important library used to build and train neural networks and perform automatic differentiation.
2. RL Games [80]: RL Games provides performance oriented implementations of most common RL algorithms in a framework that also allows to adapt and implement appraoches.
3. IsaacGym [75]: IsaacGym allows high-performance GPU-based physic simulation, which is incredibly useful for RL with robots to speed up the simulation process.
4. Theseus [81]: Theseus is a library that provides many end-to-end differentiable structures, we used this libraries for differentiable implementations of the Lie groups $SO(2), SO(3)$.
5. TorchDiffEq [82]: TorchdDiffEq is a library that provides differential ODE solvers implemented with PyTorch.