Comparing Residual Reinforcement Learning Strategies With A Stable Vector Field Base Policy

Vergleich von Residual Reinforcement Learning Strategien mit einer Vektorfeld Base Policy
Master thesis in the department of Computer Science by Philipp Jahr
Date of submission: May 15, 2025

1. Review: João Carvalho
2. Review: Jan Peters
Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT

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

Hiermit erkläre ich, Philipp Jahr, dass ich die vorliegende Arbeit gemäß § 22 Abs. 7 APB der TU Darmstadt selbstständig, ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Ich habe mit Ausnahme der zitierten Literatur und anderer in der Arbeit genannter Quellen keine fremden Hilfsmittel benutzt. Die von mir bei der Anfertigung dieser wissenschaftlichen Arbeit wörtlich oder inhaltlich benutzte Literatur und alle anderen Quellen habe ich im Text deutlich gekennzeichnet und gesondert aufgeführt. Dies gilt auch für Quellen oder Hilfsmittel aus dem Internet.

Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle 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 einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 15. Mai 2025

P. Jahr

Abstract

Peg-in-hole tasks like bolt insertions are crucial in many industrial assembly processes. However, these tasks often require more than simple insertion, such as twisting, pushing, or aligning the inserted object to achieve a successful connection.
This thesis investigates how residual Reinforcement Learning (RL) can handle such complex assembly behaviours. We compare different residual RL strategies in combination with two base policies, a linear policy and a manifold stable vector field (MSVF) policy, and in addition to that, we pre-trained a multi-layer perceptron (MLP) baseline.
The evaluation is done in a complex 3D simulation environment where successful task completion requires both rotational and translational motion after the insertion.
Our goal is to provide new insights about how policy structure and learning methods influence performance in contact-rich manipulation tasks.

Zusammenfassung

Peg-in-Hole-Aufgaben finden sich in vielen Bereichen des Alltags wieder, insbesondere jedoch in industriellen Montageprozessen. Dabei ist jedoch häufig nicht nur ein einfaches Einführen erforderlich, sondern oft auch ein anschließendes Drehen oder Drücken, um das Bauteil korrekt zu fixieren.
Diese Arbeit untersucht, wie sich Residual Reinforcement Learning (RL) in solch komplexen Umgebungen verhält. Wir wollen verschiedene Strategien im Bereich des Residual Reinforcement Learning vergleichen, sowie zwei unterschiedliche Basis-Policies. Hierfür benutzen wir einmal eine lineare Policy, die ausschließlich eine Zielausrichtung ausgibt, sowie eine komplexere Policy, welche mithilfe von Lie Group Manifolds ein stabiles Vektorfeld generiert(MSVF-Policy).
Beide Basis-Policies werden im Vorhinein mithilfe von Imitation Learning trainiert, sowie eine MLP-Policy als Referenzmodell. Die Basis-Policies werden für das Residual RL eingefroren und anschließend mit einem MLP-basierten Residual kombiniert, während die Referenzpolicy ausschließlich durch RL weitertrainiert wird.
Um diese Policies und Strategien zu testen, haben wir eine Simulationsumgebung geschaffen, in der ein L-förmiger Stift in ein Loch gesteckt werden muss. Danach, muss dieser Stift ein Stück tiefer in eine Aushöhlung geschoben werden, um ihn schließlich um $90°$ zu drehen, damit das Ziel erreicht wird. Diese ist eine komplexe Verkettung von Aktionen, welche präzise Ausführung benötigt.
Wir haben die verschiedenen Policies und Residual RL Strategien in der benannten Umgebung getestet und konnten feststellen, dass die verwendete MSVF-Policy als einzige in der Lage war diese Umgebung zu erlernen und erfolgreich das Ziel zu erreichen, auch wenn es nicht immer gelungen ist. Trotzdem zeigt dies, dass der Einsatz einer MSVF-Policy als Basis-Policy, das gewünschte Verhalten effektiv durch Imitation Learning erlernt und anschließend erfolgreich über Residual RL an die Residual-Policy weitergegeben werden kann. Ein wesentlicher Nachteil besteht jedoch im hohen Trainingsaufwand.

Contents

1 Introduction

Nowadays, assembly tasks like screwing, welding or fitting parts together are often done by robot arms. Each task requires inserting an object into a hole or manoeuvring the tool to the desired area of work, which might be in a constrained space. Such tasks are called peg-in-hole tasks.
These tasks are typically executed by a single robot in a repetitive and highly precise manner, which leads to a lack of generalisation across different assembly tasks. As a result, a robot is programmed for a single task and must be manually reprogrammed to handle different processes.
To solve this limitation, learning-based approaches can help to improve generalisation for robotic manipulation. These methods enable the learning of multiple tasks and allow robots to adapt to unforeseen changes or obstacles during operation [1].

As the different tasks can be quite complex, significant time is spent on deploying the robot for a new task. Even then, the resulting behaviour is often brittle, with small changes or misalignments leading to complete failure and stop of the system.
Instead, learning-based approaches can be trained with limited prior knowledge about the environment, consisting of a few key features and points in the trajectory to give guidance during training. Beyond that, the training process can proceed fully autonomously.
By saving time during the redeployment and the self learning of the robot, a company can save maintenance time of an assembly line, which can be invested in other urgent tasks. Additionally, robots trained with those methods are typically more robust to noise and model inaccuracies. Minor environmental changes, such as different initial positions, can be naturally integrated into the training process [2].
This motivates our investigation into how different policy designs and training combinations affect learning speed and task reliability in exemplary scenarios.

This work selected two learning-based approaches for evaluation: imitation learning and reinforcement learning. Imitation learning is based on semi-supervised machine learning algorithms, working with expert knowledge usually in the form of state-action

demonstration, while reinforcement learning is based on interactions with the environment guided by a reward function. Both methods have proven successful in solving complex control tasks and can adapt to various types of obstacles or uncertainties. However, both methods have their challenges, with Imitation Learning relying on high-quality demonstrations, which can be tedious or expensive to collect. In Reinforcement Learning, a challenge is the sample inefficiency, which makes it requires extensive interaction to discover a viable solution.
In this thesis, we explore combinations of these approaches, focusing on the refinement of imitation-based policies with Reinforcement Learning signals. By doing so, we aim to eliminate some disadvantages of both approaches. Furthermore, a specially designed base policy based on a stable vector field [3] is evaluated to investigate its effects on improving the results even further.

The MSVF-based policy, introduced in [3], is first pre-trained via imitation learning and then integrated into residual reinforcement learning setups.
To analyse its effectiveness, we compare this setup with two alternatives. First, a residual policy with a simple, untrained linear base policy, and second, a policy that is pre-trained using imitation learning and further optimised through reinforcement learning (refined RL). This allows us to evaluate both the benefit of residual reinforcement learning and the impact of the MSVF base itself.
All combinations are tested across three simulation environments of increasing difficulty. The most challenging environment involves complex 3D insertion with rotational alignment and contact interaction, designed to test generalisation and robustness over the initial starting position. It consists of an insertion with further push and rotation within the inserted hole. An advanced reward structure is introduced, which shall be simple yet effective in solving such a task.

To properly showcase and explain our approaches and evaluations, we begin in Chapter 2 by reviewing state-of-the-art work on peg-in-a-hole tasks, learning-based approaches to solve them and the use of residual reinforcement learning. Furthermore, we will showcase the recent progress on applying MSVF policies to solve robotic tasks.
Chapter 3 provides the theoretical background for our methods, covering the foundational concepts this thesis is based upon.
Chapter 4 presents the experimental setup, including the applied policies and algorithms, the environments used for evaluation, and the training procedures.
In Chapter 5, we present our results for the different environments with the different strategies. Moreover, we interpret them and discuss their meaning.
Finally, Chapter 6 concludes the thesis and outlines possible directions for future research.

2 Related Work

Peg-in-hole assembly tasks are commonly used for benchmarking contact-rich manipulation strategies. Most prior approaches focus on insertion tasks where it is just required to insert the peg in a hole of a simple geometric shape. In only a few cases, it is necessary to manipulate the peg after insertion into the hole to reach the goal configuration.
[4] classifies peg-in-hole approaches in contact model-based control and contact model-free learning. The contact model-free approaches are further divided into learning from demonstrations or directly from the environment. Furthermore, they introduce a broad overview of those approaches and name their advantages and disadvantages.
As an example of model-free approaches, [5] introduces a novel multimodal RL framework that fuses vision, proprioception and force/torque sensing within a transformer module. Through combining these three sensor modalities, they achieve generalisation over different peg shapes as well as randomised domains, like with changing light or camera positioning. Moreover, by using an impedance controller with their policy, they reduce the sim-to-real gap.
Another example of a model-free approach is [6]. Here, a supervised learning method is introduced, purely relying on force/torque sensor feedback for a 5-DoF pose estimation. They train their network with a generated dataset of known misalignments and their force/torque readings. Based on these readings, they can generalise their network and even efficiently overcome the sim-to-real gap with minimal paired sim-to-real data.
Similar to [6], [7] relies on the force/torque sensor feedback, solving the problem of misalignments during insertion without additional information. They improve insertion success by using elementary dynamic actions consisting of oscillation, submovements and mechanical impedance, which allows the peg to slip into the hole with only the sensor readings. However, while this approach generalises over peg shapes, it is limited to only a small misalignment of $3\,\mathrm{mm}$.

Residual reinforcement learning (RL) has emerged as a powerful tool for improving control in contact-rich robotic tasks, where conventional controllers fall short due to unmodeled

dynamics like friction or unexpected contact forces. By interacting directly with an environment, these dynamics are implicitly captured by RL networks and therefore don't have to be learned explicitly.
For example, [8] leverages residual RL, using handcrafted feedback controllers to solve contact-rich tasks as a base and then trains a residual policy to compensate for inaccuracies caused by contact interactions and unmodeled friction.
Instead of a feedback controller as a base, [9] uses inaccurate trajectories as prior knowledge. These trajectories are based on knowledge about the initial and end points of the trajectories as well as an error range. With this information, interaction forces and success criteria can be estimated for a workspace, which will be used as a prior for controller parameters and reward shaping.
Extending residual RL to dynamic and unstructured settings, [10] proposes a multimodal approach where they use image input and a prompt to filter out the workspace and encode it, which leads to only task-relevant information being left. This encoded image is combined with sensor readings to form a unified feature representation, which is given to the agent for learning.
Complementarily, [11] improves an approach of residual RL with a skill library, where skills from experts are encoded in a latent space skill library. Since exploration is expensive, they improve the search with a generative model, which returns only task-relevant skills. In addition to that, they introduce a small residual agent, which adapts skills to give more variations in task and training environments.

Stability, reactivity, and smoothness are essential properties of robot motion, particularly in contact-rich or dynamic environments. Recent advances in learning-based control have demonstrated that vector fields defined over appropriate geometric spaces can naturally satisfy these properties.
[3] introduces a vector field model based on Lie Groups $\mathrm{SO}(3)$ and $\mathrm{SE}(3)$, enabling the learning of stable and reactive motion policies from demonstration. Unlike approaches restricted to Euclidean spaces, their method operates on the full robot pose, encompassing both position and orientation, and guarantees asymptotic convergence through Lyapunov stability. The learned vector fields ensure that motion trajectories remain on the manifold, with singular or invalid rotations being continuously corrected toward feasible configurations, effectively attracting them back to the manifold origin.
Extending [3], [12] proposes a general approach to map simple, stable dynamical systems onto complex robot skills. They ensure that these skills satisfy the constraints of underlying manifolds while also having attraction towards a given target pose.
Complementing this theoretical foundation, [13] compares policy structures for insertion

tasks. While MSVF policies are highly stable in simple settings, residual reinforcement learning outperforms in complex, contact-dense tasks such as 3D L-shaped insertions.

In contrast to these works, which focus primarily on single-phase or perception-limited tasks, our approach explores residual RL in a multi-stage peg-in-hole scenario. For this, we explore the usage of the manifold stable vector field(MSVF) policy introduced by [3] as a base policy. Additionally, we explore the usage of different residual combination techniques with the base policy to see their effects in multi-stage peg-in-hole scenarios.
The goal is to achieve high success rates with a handcrafted reward shaping in the simulation, which is easily transferred to robot tasks.
Based on this, we want to show the strength and feasibility of MSVF policies as base policies.

3 Foundations

Learning peg-insertion tasks is non-trivial and a combination of different disciplines and approaches is required. In this chapter, we will introduce the foundational concepts relevant to this work, give a brief introduction to each topic and explain them to the extent necessary to understand and replicate the experiments.
First an overview of machine learning is given, with a focus on Imitation and Reinforcement Learning. More specific residual reinforcement learning will be explained, as well as the practice of refining a pre-trained policy, which we will declare as refined reinforcement learning. Furthermore, we will give a brief introduction to Lie Theory. To conclude the chapter, we will explain manifold stable vector fields.

3.1 Machine Learning

Machine Learning is the general term for training an agent with a policy $a \sim \pi(a \mid s)$, which maps a state s to an action a. This mapping can either be stochastic or deterministic. To improve this mapping, the policy is optimised through experience or supervision. Reinforcement Learning (RL) and Imitation Learning (IL) are two common methods.
RL collects experience in the real environment and evaluates it with a reward structure. IL uses expert demonstrations instead, which are compared to the predicted actions via a loss function.
In the following, we will briefly introduce both concepts to provide the necessary background for their use in this work.[14]

3.1.1 Reinforcement Learning

RL mimics how humans learn in the real world, accumulating experiences by attempting to reach a goal. Over time, the experiences lead to better performance and a higher chance

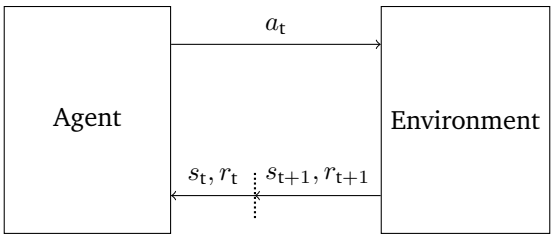

Figure 3.1: The basic communication for RL has an agent and an environment. The environment gives a state s_t and a reward r_t to the agent. The agent returns a corresponding action a_t to the environment, which transitions with the action to the next state with the next reward s_{t+1}, r_{t+1}.

of success. A well-configured RL application will eventually converge to a policy that is always successful in solving a task.
In reality, we have an agent and an environment to interact with. Figure 3.1 illustrates how the agent gets a state from the environment with a reward. With this state, it gives an action back to the environment to transition to the next state with the next reward.
There are different algorithms an agent can use to learn the best set of actions to successfully complete a task. The most commonly used algorithms include Proximal Policy Optimisation (PPO), Soft Actor-Critic (SAC), and Twin Delayed Deep Deterministic Policy Gradient (TD3)[15]. We will focus on PPO, which is an on-policy, actor-critic algorithm. On-policy means that the algorithm improves the same policy used during data collection rollouts, while an off-policy algorithm could use a different policy to collect a dataset to improve with. Rollouts refer to the data collection phase during training, so states, actions, and rewards are collected as tuples (s, a, r).
Actor-Critic algorithms rely on an actor part, which is the policy itself. It is optimised to map actions to states. The critic part is a value function that gives a rough estimate of the expected return in this state. The advantage function $\mathcal{A}(s, a) = \mathcal{Q}(s, a) - \mathcal{V}(s)$ is defined by the difference between the action-value and the state-value function. It describes how much better or worse an action is compared to the expected return of the state.
The key point of PPO is to constrain policy updates by clipping the probability ratio

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \tag{3.1}$$

where θ are the policy parameters and $\pi_\theta(a_t \mid s_t)$, $\pi_{\theta_{\text{old}}}(a_t \mid s_t)$ are the new and the old policy. This clipping function is then used within the objective function from PPO

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t,\ \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t \right) \right]$$

where r_t is equation 3.1, ϵ is a small hyperparameter that controls the clipping range and $\hat{A}_t$ is the advantage function. The clipping ensures stability during training and prevents updates which might be counterproductive. [15, 16]

Refined Reinforcement Learning: As refined RL, we define the process of using a pre-trained policy, before improving it with RL, instead of directly using RL on a freshly initialised policy. By pre-training a policy, the beginning exploration phase of a policy can be skipped, and the policy already has an intuition, which RL can improve and fine-tune. This should correct the suboptimal actions, speed up learning, and adapt to more general cases.[16]
Furthermore, by pre-training a policy, desirable base behaviour can be enforced, and with RL, we can then tune the policy for different tasks which build on the basic behaviour.

Residual Reinforcement Learning: While refined RL improves an already pre-trained policy, residual RL relies on a pre-trained policy that gets frozen. Instead of directly improving the pre-existing policy, residual RL uses the trained knowledge to speed up and guide a residual policy. The residual policy is the part that learns and gets combined with the outputs from the pre-trained policy.
The pre-trained policy is called base policy π_{base} while the residual policy will be called π_{res} from now on.
Based on this idea, there are different ways to apply the knowledge from the base policy. We will now introduce the two ways that are used in this thesis.
The first and more intuitive approach is adding the action from the base policy on top of the residual policy, giving directional guidance. This can be seen in figure 3.2 where both policies get the state from the environment and predict their own action. The addition of the action can correct the residual policy in a way that it gives the residual the right direction towards the goal, therefore improving the reward and pushing it in the direction of higher reward, even if it hasn't explicitly learned this behaviour.
The second way is instead of just adding up the predicted actions, we concatenate the predicted action from the base policy to the state and feed this combined input into the residual policy as seen in figure 3.3. By this, our residual policy can use the predicted

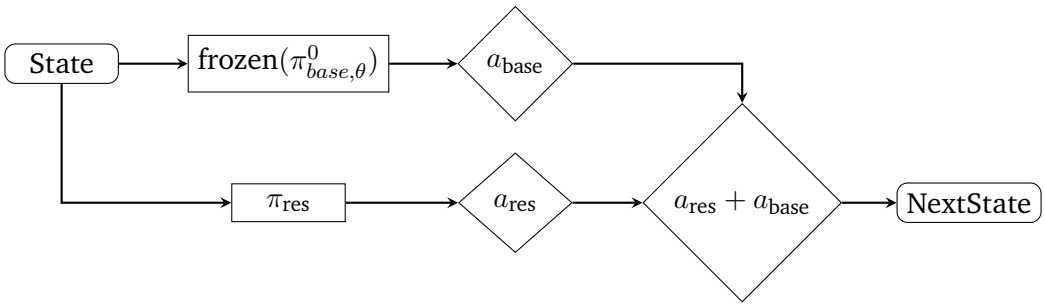

Figure 3.2: The chart shows the flow of the additive residual RL strategy. The state is provided as input to the base and residual policy. Each predicts an action that is combined by addition and is handed to the environment to produce the next state. The base policy is frozen and does not update during training, while the residual policy is actively optimised.

action from the base policy and learns how to use this base prediction to its own advantage, as well as its influence on the environment, as it gets added up just like with the additive residual policy before.

3.1.2 Imitation Learning

Compared to RL, IL doesn't collect experience in the real world. It learns by demonstration from an expert.
For the training, an expert can perform the desired task, and either raw sensory data is used for the learning or exact trajectories.
With the knowledge of demonstrations, which often consist of state-action pairs (s, a), a policy is trained. The policy tries to predict the same actions for a given state and with a loss function, which is similar to the reward functions from RL, the prediction is rated how close it is to the expert's action.[17]

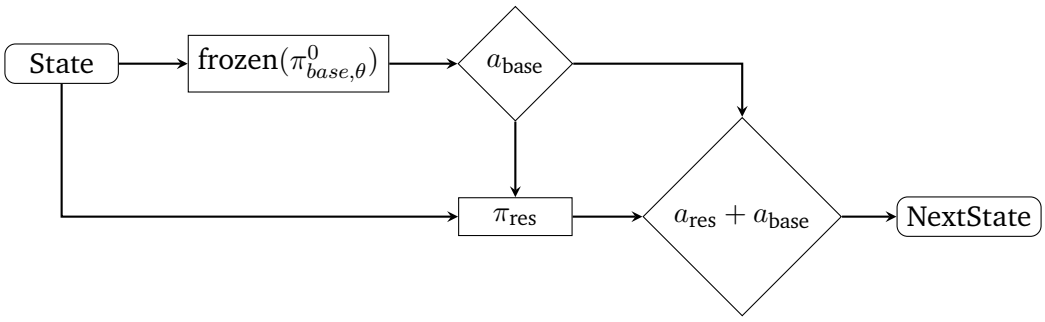

Figure 3.3: The chart shows the flow of the combined residual RL strategy. The state is provided as input to the base and residual policy. The base policy predicts an action and additionally passes it to the residual policy. Both outputs are then added together and handed to the environment to produce the next state. The base policy is frozen and does not update during training, while the residual policy is actively optimised.

3.2 Lie Theory

Lie theory, the theory of Lie Groups, Lie Algebras and their applications, is a fundamental part of mathematics [18]. The following sections explain fundamental concepts of the Lie Theory. These fundamentals are heavily inspired by and based on [19]. For further interest, we suggest reading [18], [20].

Lie Groups

Lie Groups combine the mathematical concept of groups $\mathcal{G}$ and smooth, differentiable manifolds. A group $(\mathcal{G}, \circ)$ is a set $\mathcal{G}$ and an operator $\circ$ that for the elements $\mathcal{X}, \mathcal{Y}, \mathcal{Z} \in \mathcal{G}$ satisfies the axioms

$$\begin{aligned} \text{Closure under } \circ: &\quad x \circ y \in \mathcal{G} \\ \text{Identity } \mathcal{E}: &\quad \mathcal{E} \circ \mathcal{X} = \mathcal{X} \circ \mathcal{E} = \mathcal{X} \\ \text{Inverse } \mathcal{X}^{-1}: &\quad \mathcal{X}^{-1} \circ \mathcal{X} = \mathcal{X} \circ \mathcal{X}^{-1} = \mathcal{E} \\ \text{Associativity}: &\quad (\mathcal{X} \circ \mathcal{Y}) \circ \mathcal{Z} = \mathcal{X} \circ (\mathcal{Y} \circ \mathcal{Z}). \end{aligned}$$

For the Lie Group, these elements $\mathcal{X}, \mathcal{Y}, \mathcal{Z} \in \mathcal{G}$ are elements of a smooth manifold.
A smooth, differentiable manifold must not contain discontinuities or undefined regions. Also, each region ϵ around a point must be mappable to a local vector space such as $\mathbb{R}^n$. This vector space is called the tangent space in Lie Theory.

For this thesis, the Lie groups $\mathrm{SO}(2), \mathrm{SO}(3), \mathrm{SE}(2)\text{and}\mathrm{SE}(3)$ will be used.
The Lie Group

$$\mathrm{SO}(3) = \left\{ \boldsymbol{R} \in \mathbb{R}^{3\times 3} \;\middle|\; \boldsymbol{R}^\top \boldsymbol{R} = \boldsymbol{I},\ \det(\boldsymbol{R}) = 1 \right\},$$

is a special orthogonal group. It is used to represent rotation in $\mathbb{R}^3$ and an orthogonal matrix of size $\mathbb{R}^{3\times 3}$. Each matrix has a determinant of 1 and satisfies $\boldsymbol{R}^\top \boldsymbol{R} = \boldsymbol{I}$, ensuring orthogonality. Its tangent space is in $\mathbb{R}^3$.

Using the Lie Group $\mathrm{SO}(3)$ we can define the Lie Group

$$\mathrm{SE}(3) = \left\{ \boldsymbol{T} \in \mathbb{R}^{4\times 4} \;\middle|\; \boldsymbol{T} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^\top & 1 \end{bmatrix},\ \boldsymbol{R} \in \mathbb{R}^{3\times 3},\ \boldsymbol{t} \in \mathbb{R}^3,\ \boldsymbol{R}^\top \boldsymbol{R} = \boldsymbol{I},\ \det(\boldsymbol{R}) = 1 \right\}.$$

It is the Lie group for homogeneous transformations in $\mathbb{R}^3$. For these transformations, it combines the Lie Group $\mathrm{SO}(3)$ with a translation $\boldsymbol{t} \in \mathbb{R}^3$. Its tangent space is in $\mathbb{R}^6$ and consists of 3 translational and 3 rotational elements.

Lie Algebra

Each point on the Lie Group's manifold has a corresponding tangent space, which is a unique vector space and represents all possible velocities at this point. The tangent space at the identity element of the Lie Group's manifold is called the Lie Algebra.
The logarithmic map(log map) and exponential map(exp map) are used to map elements from the Lie Group manifold to the Lie Algebra and back. The log map pushes the elements to the Lie Algebra, while the Exp map pulls them back onto the manifold. In Figure 3.4, the connection between the Lie Group manifold and the Lie Algebra is visualised.
The Lie Algebra is a linear vector space, which enables local computations due to its similarity to Euclidean space in a neighbourhood around the identity element of the Lie Group manifold.

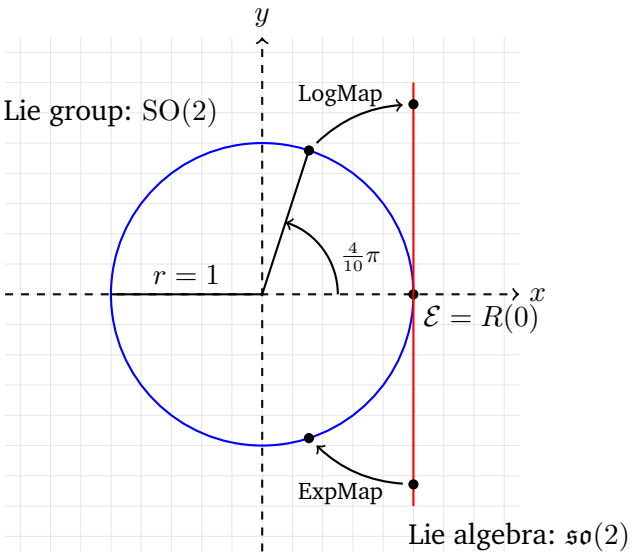

Figure 3.4: The Lie Group $\mathrm{SO}(2)$ shows rotations in 2D space. It is represented as the unit circle shown in blue. The red line is its Lie Algebra with the origin for both at $\mathcal{E}$. The LogMap maps a point from the Lie Groups manifold to a point on the Lie Algebra, while the ExpMap does the reverse.

Diffeomorphism

A diffeomorphism is a function that maps from one manifold $\mathcal{M}$ to another manifold $\mathcal{N}$. One of its key properties is that it is bijective, and the function and its inverse are continuously differentiable. As a result, it preserves smoothness and stability while deforming the space.

3.3 Manifold Stable Vector Fields

Vector fields are mathematical objects that assign a directional vector v to each point $x \in \mathbb{R}^n$. A manifold stable vector field(MSVF) is a special vector field where the underlying manifold acts as a stable attractor.
The vector field and the MSVF exist in the same space, and all vectors from the vector field are tangent to the manifold. Points outside of the manifold get attracted towards it. This results in trajectories starting on the manifold staying on the manifold, and those starting close to the manifold will converge towards it.
For example, in the case of the Lie Group $\mathrm{SO}(3)$, an MSVF ensures that points outside the space of valid rotation matrices are smoothly drawn toward it. This means invalid or large rotation matrices are corrected over time by the vector field and converge back to $\mathrm{SO}(3)$.[3]

4 Experiments

In this chapter, we describe the setup for our experiments. We introduce the different policies that we use and how they are applied and combined in our experiments. Furthermore, we will introduce the environments that were chosen to evaluate our policies, and we will explain the special features of these environments. The reward structure will be discussed in this section as well. Finally, we will provide more details about the training itself and how we combine the environments and the policies to conduct the training.
From this point onwards, we denote the state as $\boldsymbol{x}$, which consists of a position vector $\boldsymbol{p} \in \mathbb{R}^3$ and a flattened rotation matrix $\boldsymbol{R} \in \mathbb{R}^9$. The full state $\boldsymbol{x}$ is defined as the concatenation of position and rotation ($\boldsymbol{x} = [\boldsymbol{p}, \boldsymbol{R}]$).

4.1 Policies

Since we explore different residual Reinforcement Learning(RL) strategies with combined policies, we rely on more than one type of base policy. We will now start by introducing the different policies and their differences.
In general, we use three different policies, which are a linear policy, an MSVF policy and a multi-layered perceptron.
All policies predict output velocities in Cartesian space. These velocities, denoted as $\dot{\boldsymbol{x}}$, consist of three translational and three rotational velocity components, corresponding to movement along and around the x, y, and z axes, respectively.

The **Multi-Layer Perceptron(MLP) policy** is a feedforward neural network. It consists of connected layers through which the input state $\boldsymbol{x}$ is passed to compute an output action. In our case, we use an MLP policy which takes the state $\boldsymbol{x}$ as an input and feeds it through 2 hidden layers. Its layers are linear transformations and fully connected with the RELU activation function RELU$(x) = \max(0, \boldsymbol{x})$.

Since the base policy already provides strong guidance, the residual MLP policy requires only 256 features, which are sufficient to refine the behaviour. In contrast, the refined RL policy uses an MLP with 512 features to enable more detailed behaviour modelling and improve task-specific adaptation.

The **Linear policy** takes the input state $\boldsymbol{x}$ from the environment. It gets split into the positional part $\boldsymbol{p}$ and the rotational part $\boldsymbol{R}$. Both parts calculate the difference between themselves and the goal position and goal rotation. These differences are then clipped to avoid unstable or exaggerated velocities, and the resulting velocity will be used as the base action.
Neither a neural network nor pre-training is required to improve this policy.

The **Manifold Stable Vector Field (MSVF) policy** is based on the construction of stable vector fields on manifolds. It takes advantage of the Lie Group $\mathrm{SE}(3)$ and its Lie Algebra, as introduced in chapter 3.2.
The policy receives the input state $\boldsymbol{x}$ from the environment and applies the logarithmic map to project it to the tangent space at the identity element, which is initialised as the origin of the environment in our setup.
The resulting tangent vector $\hat{\boldsymbol{x}}$ is then passed through a diffeomorphism network that maps it into a latent tangent space. This transformation serves to reshape the space for improved details in important regions. The output of this network, a refined tangent vector $\hat{\boldsymbol{z}}$, is then passed into a learned dynamics model. This model, implemented as a scalar field, generates a velocity vector $\dot{\boldsymbol{z}}$ in the latent space.
Using the Jacobian of the diffeomorphism and the adjoint transformation, the latent velocity is pulled back from the latent space into the original tangent space, resulting in a velocity vector $\dot{\boldsymbol{x}}$.
Finally, the resulting velocity $\dot{\boldsymbol{x}}$ is normalised and returned as the velocity action for the environment.
Figure 4.1 provides a visual overview of the MSVF policy and the flow of the input state through each stage.

4.2 Environments

For the evaluation, we will use three different environments, which increase in difficulty. This section introduces each of the environments and their individual properties. Their

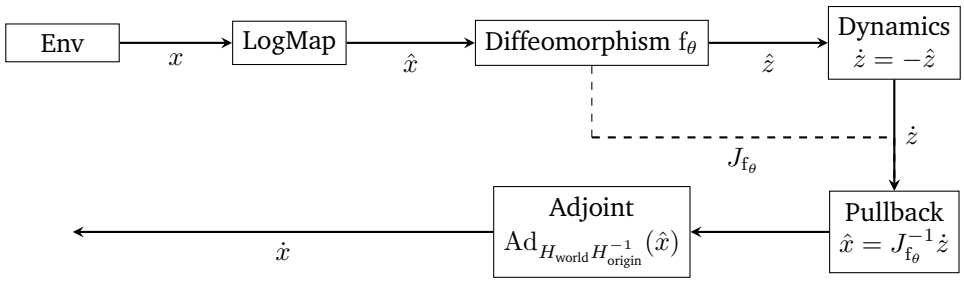

Figure 4.1: The environment hands the input state $\boldsymbol{x} = [\boldsymbol{p}, \boldsymbol{R}]$ to the policy. First, the log map is applied to get it into the tangent space $\hat{x}$. Afterwards, a diffeomorphism maps it to a latent tangent space $\hat{z}$ where a dynamics model is applied to receive the latent velocity $\dot{z}$. This latent velocity is then pulled back via the Jacobian of the diffeomorphism, and by using the Adjoint, the final velocity action $\dot{x}$ in the original tangent space is obtained.

reward function will be explained, and we will explain the reasoning behind our decisions.

4.2.1 General Reward Function

All environments use the same basic **reward function**

$$r_{\text{total}} = \begin{cases} 3, & \text{if absorbing} \\ r_{\text{pos}} + r_{\text{rot}} + r_{\text{joint}}, & \text{otherwise,} \end{cases} \tag{4.1}$$

which consists of a reward for the position r_{pos}, a reward for the rotation r_{rot} and a reward for the joint velocities r_{joint}. Each of these rewards is negative and acts similarly to a penalty, as worse positioning leads to a smaller reward. So, our reward penalises bad positioning, bad rotations or high joint velocities stronger.
Furthermore, due to this specification, the maximum reward reachable is $\max r_{\text{total}} = 0$, ensuring that our reward for absorbing is always seen as a significant reward. This is also the reason why we chose 3 as a reward for reaching the goal, as it returns a strong signal and high positive gradient for reaching the goal.

For the **positional reward**

$$r_{\text{pos}} = -\beta_{\text{pos}} \mathrm{f}_{\text{funnel}}(\|\mathbf{w}_{\text{pos}}(\boldsymbol{p}_{\text{des}} - \boldsymbol{p}_{\text{cur}})\|),$$

the positional error $(\boldsymbol{p}_{\text{des}} - \boldsymbol{p}_{\text{cur}})$ is weighted axis-wise by $\mathbf{w}_{\text{pos}} \in \mathbb{R}^3$ and then passed to the funnel function

$$\mathrm{f}_{\text{funnel}}(\boldsymbol{e}) = \mathrm{u}\|\boldsymbol{e}\|^2 + \mathrm{v}\log\left(\|\boldsymbol{e}\|^2 + \alpha\right) - \mathrm{v}\log(\alpha)$$

where $\boldsymbol{e} = (\boldsymbol{p}_{\text{des}} - \boldsymbol{p}_{\text{cur}})$ is the positional error. The result from the funnel function is then scaled using β_{pos}.
This funnel function is shown in Figure 4.2 with the influence of the different variables u, v and α visualised in three different plots.
With the default parameters used for the experiments, the positional reward has a stronger gradient for smaller errors in position, while a bigger positional error gives smaller gradients. Still, the gradient is always steeper than a linear function for our environments.

The **rotational reward** uses the rotational error

$$\boldsymbol{R}_{\text{err}} = \boldsymbol{R}_{\text{goal}}\boldsymbol{R}_{\text{cur}}^{-1}$$

where $\boldsymbol{R}_{\text{goal}} \in \mathbb{R}^{3\times3}$ is the current goal rotation and $\boldsymbol{R}_{\text{cur}} \in \mathbb{R}^{3\times3}$ is the current rotation. The resulting error matrix will be mapped to its Lie Algebra using the LogMap

$$\boldsymbol{\omega}_{\text{err}} = \mathrm{LogMap}\left(\boldsymbol{R}_{\text{err}}\right).$$

The result is $\boldsymbol{\omega}_{\text{err}} \in \mathbb{R}^3$, which is the error around each axis represented in radians. Using this error, we can calculate the final rotational reward

$$r_{\text{rot}} = -\beta_{\text{rot}}\left(\mathrm{mean}\left[\left(\mathbf{w}_{\text{rot}}\boldsymbol{\omega}_{\text{err}}\right)^2\right]\right)$$

where $\mathbf{w}_{\text{rot}}$ is a scaling per axis and β_{rot} is the scalar rotation reward scaling. With this, we have a linear scaling for the rotational reward with a maximum rotation of π in each direction. The scaling per axis can be used to give some rotation directions a higher relevance and a steeper gradient.

Finally, the joint velocity reward

$$r_{\text{joint}} = -\beta_{\text{joint}}\left(\frac{1}{n}\sum_{i=1}^{n}\dot{q}_i^2\right)$$

penalises fast joint motions where $\dot{q}_i$ are the velocities of the different joints used. In this way, we enforce slow and stable movements.

For our training, the scaling factors will be $\beta_{\text{pos}} = 5, \beta_{\text{rot}} = 10$ and $\beta_{\text{joint}} = 5$. The scaling factor for the rotation is higher than the ones for position and joint velocities, since the rotational error has a bigger impact on insertion tasks. With a rotational misaligned peg, the insertion is impossible, due to the peg getting stuck at the edges of the hole. Furthermore, scalings for each axis $\mathbf{w}_{\text{rot}}$ and $\mathbf{w}_{\text{pos}}$ are environment depended.

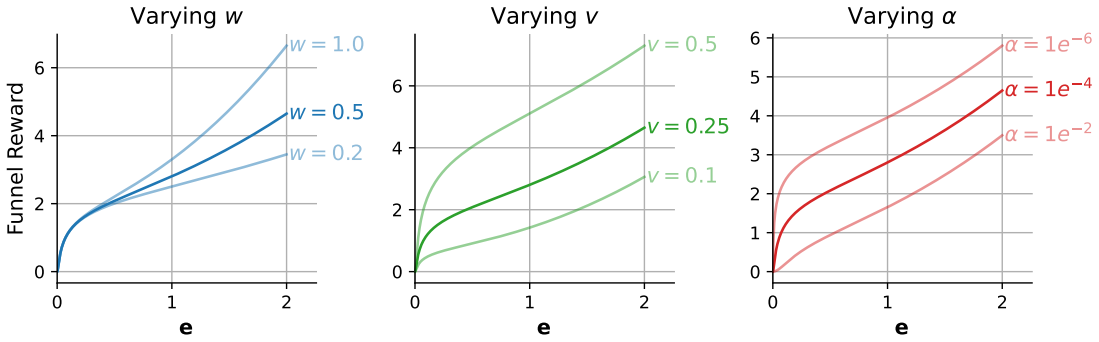

Figure 4.2: Each plot shows the effect of varying one of the constants in the funnel function. The opaque curves represent the default values used in this work, while the transparent lines show the effect of changing the corresponding parameter. The three subplots correspond to variations in the constants u, v and α.

4.2.2 Environment Descriptions

The following paragraphs provide a detailed description of each environment, highlight the differences from the previous one and the reason for its increased difficulty. Furthermore, some environments have additions to the reward function which are introduced.

The **2D Box Insertion (2D Insertion) Environment** is the first and simplest environment with a 200 step horizon. It will take place in a three-dimensional space, but the y-value will be constrained in motion, so that a two-dimensional environment is created. The insertion will be done by a square which has to reach its final position within a square hole at the centre of the environment. Rotational symmetry allows for $90°$ rotations around the vertical axis without affecting the success criteria.
The left picture in Figure 4.3 shows a picture of the 2D environment for better visualisation.

The **3D Cube Insertion (3D Insertion) Environment** is based on the first environment, but adds another dimension. With this extra dimension, we increase the complexity of the environment, allowing the movement along the y-axis and rotations around each axis. Due to the higher complexity, we increase the horizon to 300 in this environment.
The insertion is done using a cube and has to be inserted into a cubic hole. Similarly to

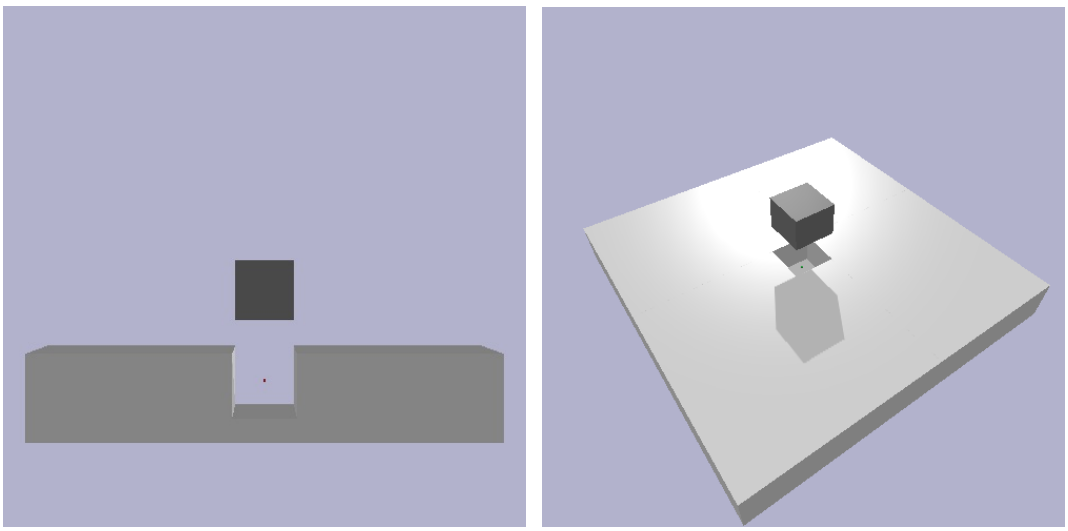

Figure 4.3: The image on the left side shows the 2D Insertion environment, while the image on the right side illustrates the 3D Insertion environment.

the first environment, the rotational symmetry allows for $90°$ rotations around each axis without affecting the success criteria.
The right picture in Figure 4.3 shows a picture of the 3D environment for better visualisation.

The **L-Shape Insertion Rotation(LRot) Environment** uses an L shape, where the bottom part of the L is a rectangle and the tip of the L is a cylinder. The goal is to insert this L shape into a rectangular hole, push it to the goal position and then rotate the L-shape by $90°$. The L shape has no rotational symmetry, and therefore, this peg has only one goal configuration.
Figure 4.4 shows a sketch of the environment with the parameters used to define it. The tolerance for this environment's insertion is 1 cm.
With another increase in difficulty and also the extended way to reach the goal, we use a horizon of 500 steps for this environment, to give enough steps to reach the goal.

Due to the complexity of the environment, we extend our reward function 4.1. The first extension is another reward for the tip alignment around the z-axis

$$r_{\text{align}} = -\beta_{\text{align}} \left\| \mathbf{z}_{\text{cur}} - \mathbf{z}_{\text{goal}} \right\|$$

where $\mathbf{z}_{\text{cur}} = \mathbf{R}_{\text{cur}}[:, 2]$ and $\mathbf{z}_{\text{goal}} = \mathbf{R}_{\text{goal}}[:, 2]$. This tip alignment rewards penalises any

misalignments of the tip with the z-axis by extracting the last column of both matrices and subtracting them. Taking the norm of the resulting vector leads to a reward that has a maximum of $\max(r_{\text{align}}) = 2$ when the misalignment is $180°$.
Moreover, we add two progression rewards. The first one is for positional progression

$$r_{\text{pos\_prog}} = -\beta_{\text{pos}} \left(\|\boldsymbol{e}_{\text{prev\_pos}}\| - \|\boldsymbol{e}_{\text{cur\_pos}}\| \right)$$

where $\boldsymbol{e}_{\text{prev\_pos}} = (\boldsymbol{p}_{\text{des}} - \boldsymbol{p}_{\text{cur}})$ is the error from the pervious step in the environment while $\boldsymbol{e}_{\text{cur\_pos}}$ is the current error from this step.
The second progression reward is for rotational progression

$$r_{\text{rot\_prog}} = \min \left(\max \left(\boldsymbol{e}_{\text{prev\_rot}} - \boldsymbol{e}_{\text{cur\_rot}},\ -0.5 \right),\ +0.5 \right)$$

with

$$\boldsymbol{e}_{\text{prev\_rot}} = \left(\text{mean} \left[(\mathbf{w}_{\text{rot}} \boldsymbol{\omega}_{\text{err}})^2 \right] \right)$$

being the rotational error mean from the previous step and $\boldsymbol{e}_{\text{cur\_rot}}$ being the rotational error mean from the current step.
The min and max term for the rotational progression is used to clip it within the given range to ensure a stable but constant progression, which does not dominate the training. Furthermore, we use the progression rewards to encourage positive and penalise negative behaviour.
With the alignment reward and the two progression rewards, we now get the new reward function

$$r_{\text{total}} = \begin{cases} 3, & \text{if absorbing} \\ \min(r_{\text{pos}} + r_{\text{rot}} + r_{\text{joint}} + r_{\text{align}} + r_{\text{pos\_prog}} + r_{\text{rot\_prog}}, 0), & \text{otherwise,} \end{cases} \tag{4.2}$$

which is used for all phases. Moreover, it is important to mention that we apply a min function to ensure that our progressive reward term doesn't outperform our absorbing reward.

With the adapted reward function 4.2 now tailored to our environment, we introduce an additional change. Our environment is split into three different goal phases. Once the agent reaches the current goal, the next phase is activated. This process continues until the final phase is completed. Still, if the agent backtracks its progress, it also returns to a previous phase.
In our case, we have three phases: insertion, push and rotation. Transitions between phases are based on threshold conditions. The switch from insertion to push is triggered when the height falls below a certain threshold, indicating successful insertion. The

transition from push to rotation is based on alignment along the y-axis, which serves as an indicator that the peg has been pushed sufficiently.
To prevent discontinuities in the reward function, we add the rewards from the later phases to the earlier ones. For instance, during the insertion phase, a constant value corresponding to the push and rotation phase rewards is included. This value is decreased as the agent progresses to the next phases, effectively smoothing the reward landscape.
It is important to note that while sudden changes in rewards are generally undesirable, the effect depends on their direction. Positive jumps that reinforce desirable behaviour are acceptable, whereas negative jumps that penalise good actions must be avoided.

4.3 Training

In this section, we focus on the training itself. We introduce how the pre-training is executed and how the residual RL is applied. Furthermore, we show the different combinations of policies that we will use and evaluate.

Since we want to explore the advantage of using a MSVF policy as a base policy, we compare it to a linear base policy. Furthermore, we want to compare additive and combined residual RL, explained in section 3.1. For that, we will combine the different policies and strategies to determine their effect on learning performance. A refined MLP will then be used as our baseline to see how our approaches perform in comparison. In table 4.1, all combinations are listed.

4.3.1 Train Base Policy

To train our base policy, we use Imitation Learning. The state-action pairs for the training are collected directly in the environment via trajectories, which lead to the goal. The starting position is sampled from a preset initialisation space. Then, the peg is guided via waypoints towards the goal. For each environment, the first waypoint was always positioned above the whole, which we call the lift position. This lift position, pre-trained in the base policy, helps to give the residual policy guidance to not just push down over the hole edges and rotate by doing so.
Since this would always result in perfect trajectories, we apply smoothing to introduce some imperfection

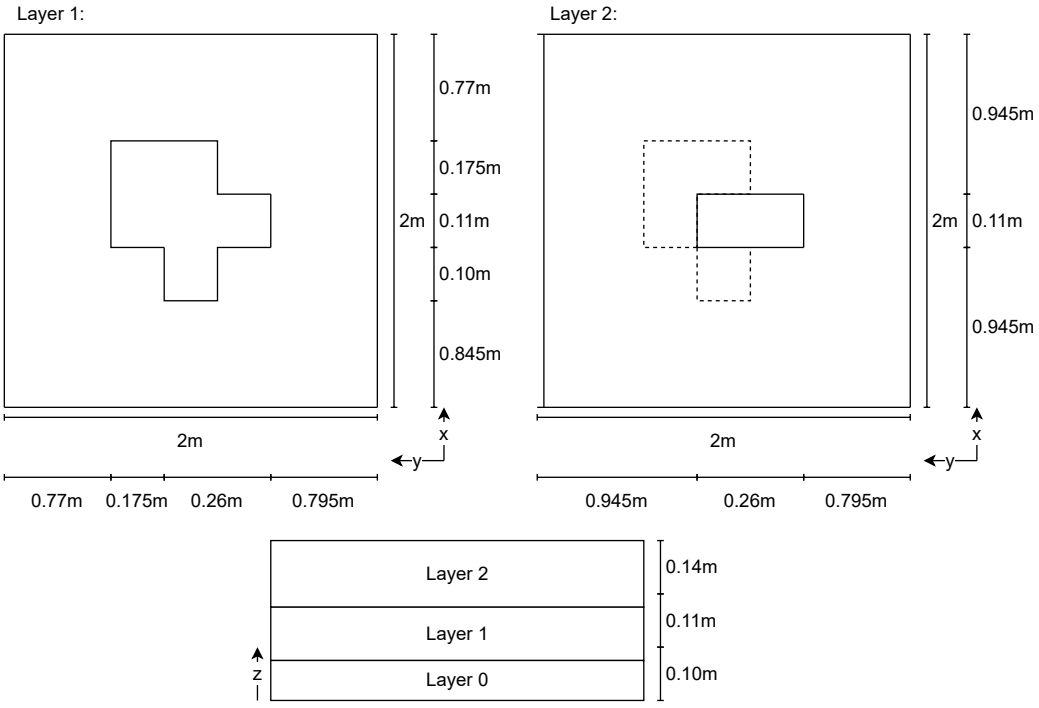

Figure 4.4: The sketch shows the general structure of the hole for the L shape insertion. The hole consists of three layers, where the bottom layer serves as a base. The actual hole is distributed across the first and second layers. The hole for insertion is shown in the second layer with dashed lines indicating the outlines from layer 1.

Base Policy	Residual Policy	Strategy
None	Pre-Trained MLP	Refined RL
Linear Policy	MLP	Additive Residual RL
Linear Policy	MLP	Combined Residual RL
MSVF Policy	MLP	Additive Residual RL
MSVF Policy	MLP	Combined Residual RL

Table 4.1: The table lists the different combinations of base and residual policies used in training. Each row represents a specific training configuration, defined by the base policy, residual policy, and the residual RL strategy applied.

We split the collected samples into a training and validation set, using a $80/20$ split. Then, with the collected samples, we train an MSVF policy, which will later be used as a base policy. We train the MLP that we use for refinement with the same trajectories to keep the comparison close. For the MSVF base policy, we only use 64 features to keep it light and focused on providing directional guidance, while the pre-training for the refined MLP policy is done with 512 features. By giving the policy for refinement more features, we ensure that during the reinforcement learning, our policy can learn complex behaviours. For each environment, we pre-train an MSVF policy and an MLP with different feature numbers, aiming for the same accuracy in the validation loss for rotation and position. Furthermore, for the LRot environments, we increase the weight of the rotation to ensure better optimisation of rotational behaviour, as it is especially important for these environments.
At the end of the training, we run the resulting policies for 10 rollouts to see their performance within the environments themselves to see if refinement would even be necessary.

4.3.2 Reinforcement Learning

For reinforcement learning (RL), we use the PPO algorithm explained in section 3.1. Then, we either refine the pre-trained policy or use one of our residual strategies.
For the refinement, we load our checkpoint data from the pre-training into a newly created MLP policy. Since our policy has no way of exploration so far, we have to add stochasticity. We achieve this by adding Gaussian noise. The output from our policy is treated as the

mean value, while a standard deviation has to be given to define the covariance matrix that is used for sampling around the mean. We call this new stochastic policy a Gaussian policy, and it is used for the RL training.

For the residual RL, we have to first load our pre-trained MSVF policy as well. Then, we add it to a new, untrained MLP policy with 256 features. Similar to the refinement, we add stochasticity in the same way to receive a Gaussian policy that can be used for training itself.

For our general training, the refined RL approach and the residual RL approach, we train our policies with 50 times the horizon as steps per epoch, while we fit our policy 10 times per epoch. By this, we ensure high sample efficiency and stable policy updates.

5 Evaluation

Based on the experimental setup described in the previous chapter, we executed the two RL strategies and RL policies within the defined environments. The following chapter presents the resulting performance metrics and visualisations obtained from training. It will explain them and discuss their effects and meanings to give a broader understanding of what they mean, as well as giving further insights about observations made by us.

5.1 Imitation Learning Results

To evaluate the performance of our imitation learning (IL) training, we report key metrics such as training/validation loss and success rate in Table 5.1. Each IL training was executed over 60 epochs. We used 10 trajectories for the 2D and 3D Insertion environment and 15 trajectories for the LRot environment. The increased number of demonstrations in the LRot environment reflects its higher complexity. Additional trajectories provide more samples, which improve generalisation and prediction accuracy.
This generalisation is supported by the observation that training and validation losses remain close in magnitude, with the training loss consistently lower, as expected.
This small difference shows that our training results translate well over unknown states, because an average prediction error of approximately ≈ 0.1 cm is well within acceptable bounds for robotic manipulation tasks.
Despite the low error magnitudes, the observed insertion success across 10 test trials remained low. In the 2D insertion environment, the small success rate of 30% comes from tipping over the edge and then getting stuck with a $45°$ angle. In our 3D insertion and LRot environment, we observe that small misalignments and rotation errors still stop the agent from inserting into the hole.

Metric	2D Box Insertion	3D Cube Insertion	LRot
Trajectories	10	10	15
Success Rate	30%	0%	0%
Training Loss	$\approx 0.16\,\text{cm}$	$\approx 0.08\,\text{cm}$	$\approx 0.06\,\text{cm}$
Validation Loss	$\approx 0.18\,\text{cm}$	$\approx 0.1\,\text{cm}$	$\approx 0.1\,\text{cm}$

Table 5.1: The table shows the final training results for the imitation learning for each environment after 60 epochs of training.

These results indicate that imitation learning alone is insufficient to robustly solve the task. To improve performance, we introduce residual Reinforcement Learning (RL) as a refinement step.

Figure 5.1 provides a visual representation of the learned policies using stream plots. It shows the stream plots for the 2D and 3D insertion environments, where we generated actions across a grid over the state space and visualised the resulting vector field. In the 2D environment, the grid was created directly over the task space, while for the 3D environment, a 2D slice was taken along the XZ-plane through the origin to enable comparable visualisation. With those vector fields, we were able to show the general direction and goal of the trained policies. The plots reveal that the lift position, discussed in Section 4.3, is encoded in the learned policy, and that its goal is the origin. Still, it is important to note that these stream plots visualise only positional components, while rotational behaviour is not represented.
Furthermore, a noticeable difference in the stream plot of the 2D Insertion environment and the 3D Insertion environment is visible. This dissimilarity may be attributed to several factors. On one hand, the samples for both environments are not the same and especially in 3D, the task space is much bigger. On the other hand, it can also relate to where the initial positions for the trajectories were, as starting positions below the lift position lead to the upward motion within our stream plots. The 3D case could have more such starting positions and therefore lead to a more general curvature in its space, while the 2D case might have better distributed starting positions, which makes the trained policy learn that it doesn't always have to begin with an upward motion.

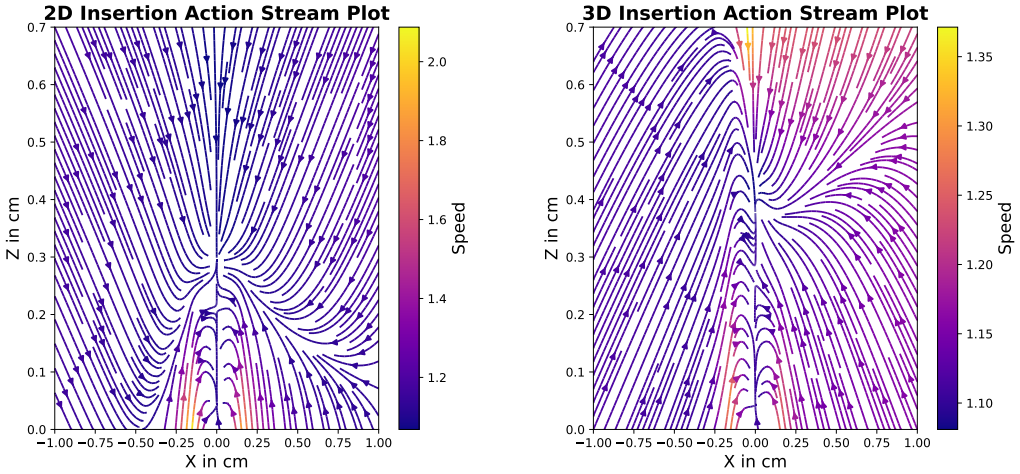

Figure 5.1: The stream plots illustrate the general action direction along the streamlines, with speed encoded by colour. Both plots show the actions produced by the MSVF policy after training via imitation learning. The left plot visualises the 2D Box Insertion Environment, while the right plot illustrates the XZ-axis slice of the 3D Cube Insertion Environment.

5.2 Reinforcement Learning Results

The RL procedure is carried out as described in section 4.3. Each policy and residual strategy is executed 6 times per environment to ensure more robust and statistically meaningful results. Each environment is evaluated individually with a focus on important details and metrics like success rate, steps to reach the goal and the position and rotational error if necessary.

Figure 5.2 illustrates the results for our training in the 2D Insertion environment. Contrary to expectations, the success rates didn't rise significantly. Still, we can see that the policies using the MSVF base policy have higher mean success rates than the linear base or the refined MLP, which resulted in a 0% success rate. The confidence intervals are relatively wide, indicating notable variability in performance even under identical training configurations. In contrast, the plot illustrating the number of steps required to reach the goal exhibits lower variance, suggesting that once successful, the policies tended to complete the task in a consistent number of steps.

With further investigation, we noticed that all policies struggled with the edge of the 2D hole. Notably, the MSVF base policy's lift position encoding appeared to support the residual component, contributing to more stable and confident success when compared to other policies.
The reason for the low success rates in this environment could come from several sources. As mentioned earlier, the peg was tipping over the edge and then rotating, not able to reach the goal in that position. One likely cause is suboptimal reward shaping. As discussed in Section 4.2, the funnel-shaped reward function may fail to provide strong or consistent gradients near the goal. These weak gradients could be a consequence of the reduced dimensionality in the 2D environment, which limits the positional error used to compute positional rewards. Additionally, the lack of a third dimension makes alignment more difficult, as the peg cannot exploit other edges, which is possible in the 3D task.

An overview of the results for the 3D Insertion environment is shown in Figure 5.3. In the upper plot, we can see the success rates. Both MSVF-based policies achieve a 100% success rate from the beginning, with no visible confidence interval, indicating consistent success across all six training runs. In contrast, the linear base-based policies require more time to reach full success. Especially, the linear base-based policy using the combinatory residual RL strategy exhibits less stable performance and slower convergence toward 100%

success. The refined MLP policy fails to show meaningful improvement throughout the $300,000$ training steps.
The lower plot shows the number of steps required to reach the goal and confirms the same trend, as the MSVF-based policies consistently reach the goal in fewer steps with low variance. The linear base policies generally require more steps, with the combinatory strategy again showing higher variance and slower convergence compared to MSVF-based policies.

The high success rates for the MSVF-based policies are expected as they use the stable control and pre-trained behaviour to quickly insert the peg into the hole. The lift position encoded in the MSVF further supports early-stage success, making it the most effective base policy evaluated in this environment.
The strategies using linear base-based policies are slower to reach full success, but can achieve it as well. Unexpectedly, the combinatory residual strategy underperforms compared to the additive strategy, despite receiving richer input information. This additional input may increase learning complexity, requiring more training steps for effective interpretation, while the additive strategy learns the residual predictions implicitly through the base policy's predictions. These findings suggest that providing additional information directly to the residual policy may slow down training by increasing the input complexity.

The behaviour of the trained policies in the LRot environment is visualised in Figure 5.4. The upper plot displays the success rate, where the MSVF-based policies achieve the highest success rates overall, though not consistently across all 10 evaluation trials, resulting in a lower average success and wider confidence intervals. Furthermore, the additive residual RL approach achieves a higher mean success rate at the end of training compared to the combinatory approach.
As observed in the 3D insertion environment, the additive approach may be more effective, potentially due to the residual policy learning implicit corrections from the base policy, rather than being explicitly provided with its action.
Policies using the linear base were unable to achieve any successful insertions in the LRot environment. This was expected, as the linear base policy directs the agent toward the final goal position, without encoding a continuous motion field or any task-specific adaptation, such as required rotational alignment for the rotated hole. Due to its simplicity, the linear base policy effectively counteracts the necessary insertion configuration, interfering with successful task execution.
While the refined policy reaches the goal only rarely, the occasional successes suggest that the approach holds potential. This is further investigated by extending the training over additional epochs, as shown in Figure 5.5. The upper plot shows the success rate. In

	Time in minutes
Refined Policy	≈ 57
Linear Base Policy, Additive RL Strategy	≈ 65
Linear Base Policy, Combinatory RL Strategy	≈ 106
MSVF Base Policy, Additive RL Strategy	≈ 484
MSVF Base Policy, Combinatory RL Strategy	≈ 779

Table 5.2: The table shows the final mean training times for the LRot environment.

several runs, the success rate spikes near the end of training, suggesting that the refined policy may require additional time to fully learn the task, just like the residual MSVF-based policies.

Moreover, the reason for failure was investigated. Figure 5.6 illustrates the mean positional and rotational reward per epoch for the MSVF-based policy used in the combinatory strategy, while the lower plot from Figure 5.5 shows the positional reward for the refined policies.
A clear correlation is observed between the positional reward and success rate. We can see that the policies, which stagnate with their positional reward around -15 to -20, consistently fail to achieve task success. The likely cause for this stagnation is the policies failing at the insertion part and therefore never can insert the peg into the hole to reach later steps of the phase-shaped environment.
The rotational reward in the lower plot of Figure 5.5 further supports this assumption, as policies with the lowest positional rewards also exhibit below-average rotational performance, which could be a reason for failure of the initial insertion.

The training times for the LRot environment are summarised in Table 5.2. Notably, while the MSVF-based policies yield the best performance, their training time is approximately four times longer than that of their linear-based policy counterparts. In particular, the combinatory RL approach requires nearly twice the training time of the additive approach, regardless of the base policy used. So, training time represents a significant cost factor that should not be overlooked when choosing a base policy.

5.3 Evaluation Summary

The imitation learning (IL) pre-training produced promising results, demonstrating good generalisation across the state space and successfully encoding the desired behaviour. However, IL alone was insufficient to fully solve the task and required refinement through residual RL.

Across all environments, policies using the pre-trained MSVF base consistently achieved the highest success rates, while also demonstrating robust and efficient behaviour, reaching the goal with fewer steps on average. Still, it also requires the highest training times.
The linear base policy performs well in simpler environments, but its effectiveness decreases in more complex environments, where it fails to provide sufficient guidance. This outcome is expected, as the linear policy merely directs actions toward the goal without encoding task-specific motion structure.
Compared to the refined MLP policy, both the MSVF and linear base policies perform better in simpler environments. However, in the complex environment, the refined policy shows promising results, suggesting that the refined policy may simply require more training epochs to achieve comparable performance in simpler environments. Given its relatively low training time, extended training durations would be feasible.

Among the RL strategies, the additive approach appears more robust. Implicit guidance through the base policy's output appears to support more efficient learning, compared to the combinatory approach, where the residual policy must actively interpret the base policy's action, which seems to slow down the learning itself. Furthermore, training time for the combinatory approach is nearly twice as long, which reduces its practical use.

Overall, the MSVF base policy with the additive residual RL approach proved to be the most reliable and stable foundation, effectively guiding the residual component toward successful task completion across all environments.

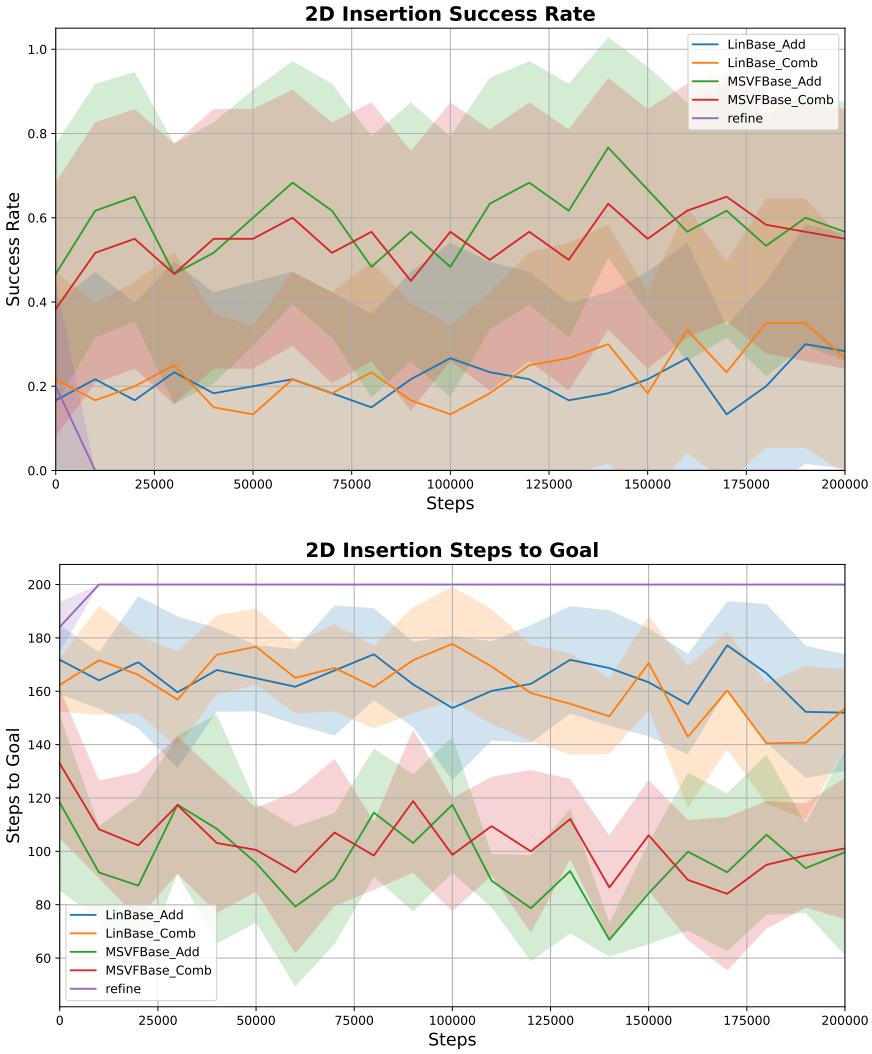

Figure 5.2: The two plots show the results from the 2D Square Insertion environment. The upper plot illustrates the success rate, while the lower plot shows the steps to reach the goal. The different policy combinations are colour-coded, where the thicker line represents the mean performance and the shaded regions indicate the 95% confidence intervals.

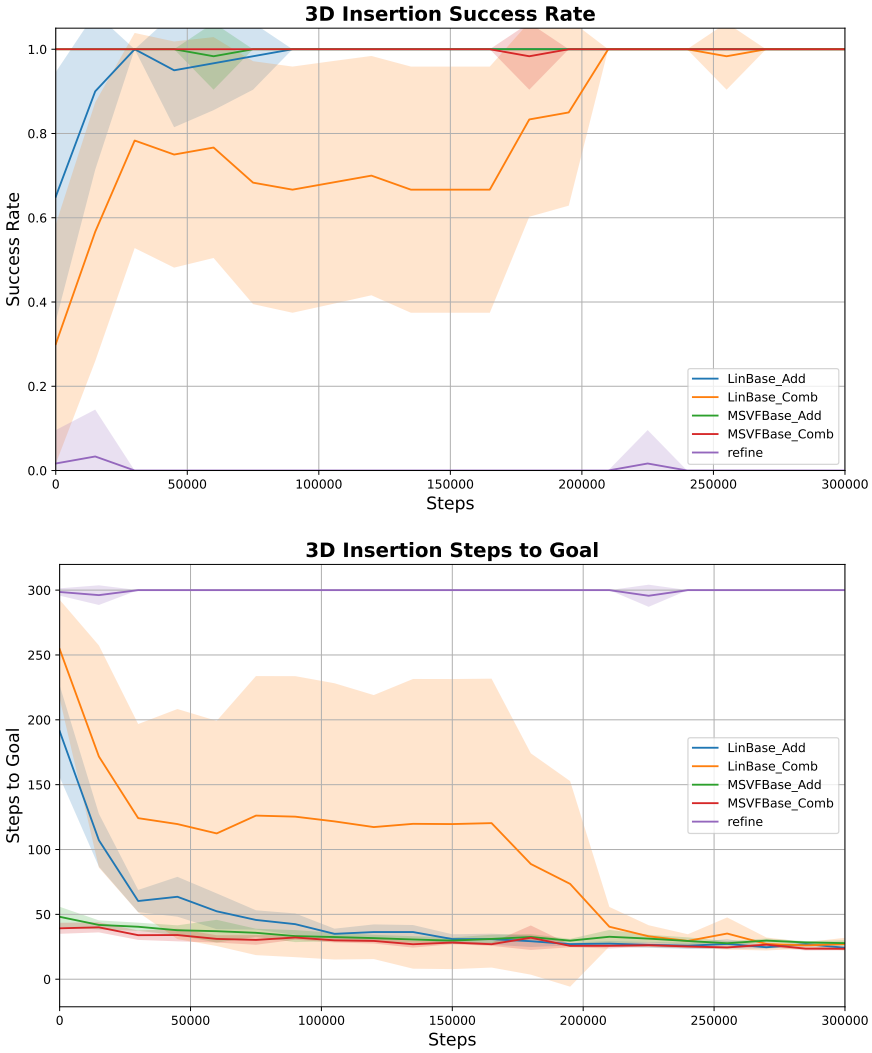

Figure 5.3: The two plots show the results from the 3D Cube Insertion environment. The upper plot illustrates the success rate, while the lower plot shows the steps to reach the goal. The different policy combinations are colour-coded, where the thicker line represents the mean performance and the shaded regions indicate the 95% confidence intervals.

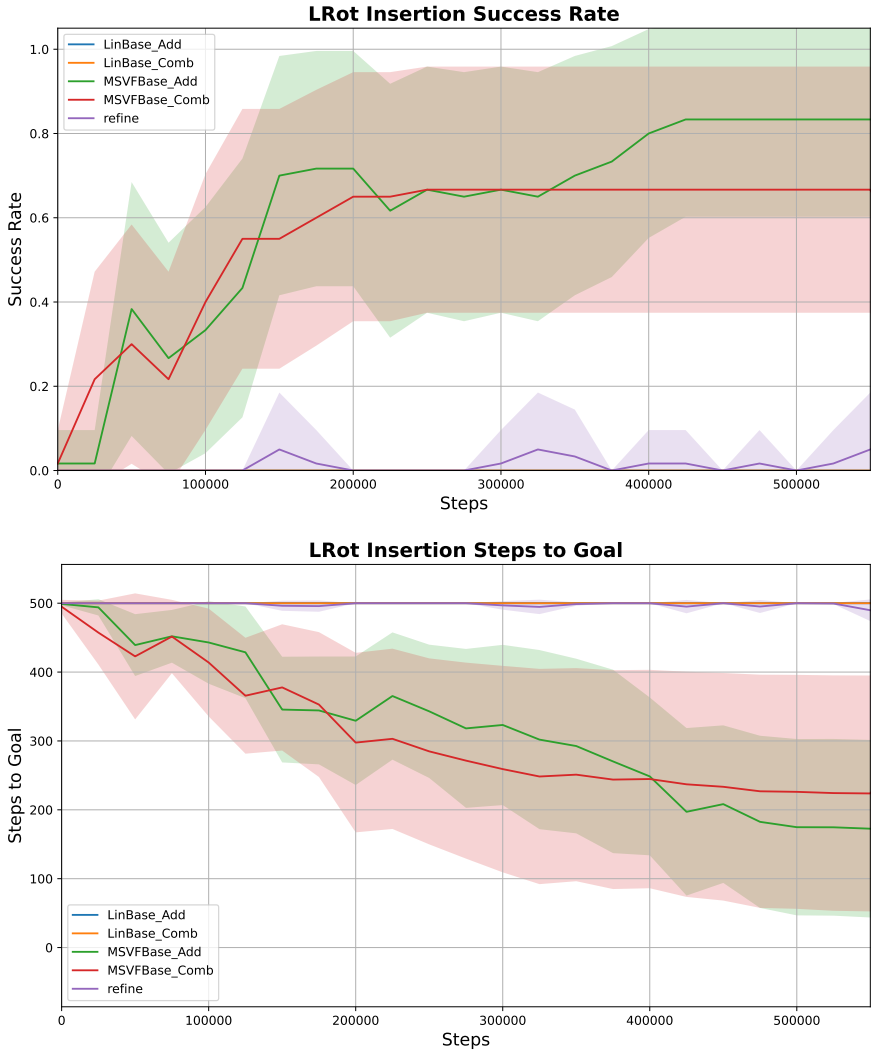

Figure 5.4: The two plots show the results from the LRot environment. The upper plot illustrates the success rate, while the lower plot shows the steps to reach the goal. The different policy combinations are colour-coded, where the thicker line represents the mean performance and the shaded regions indicate the 95% confidence intervals.

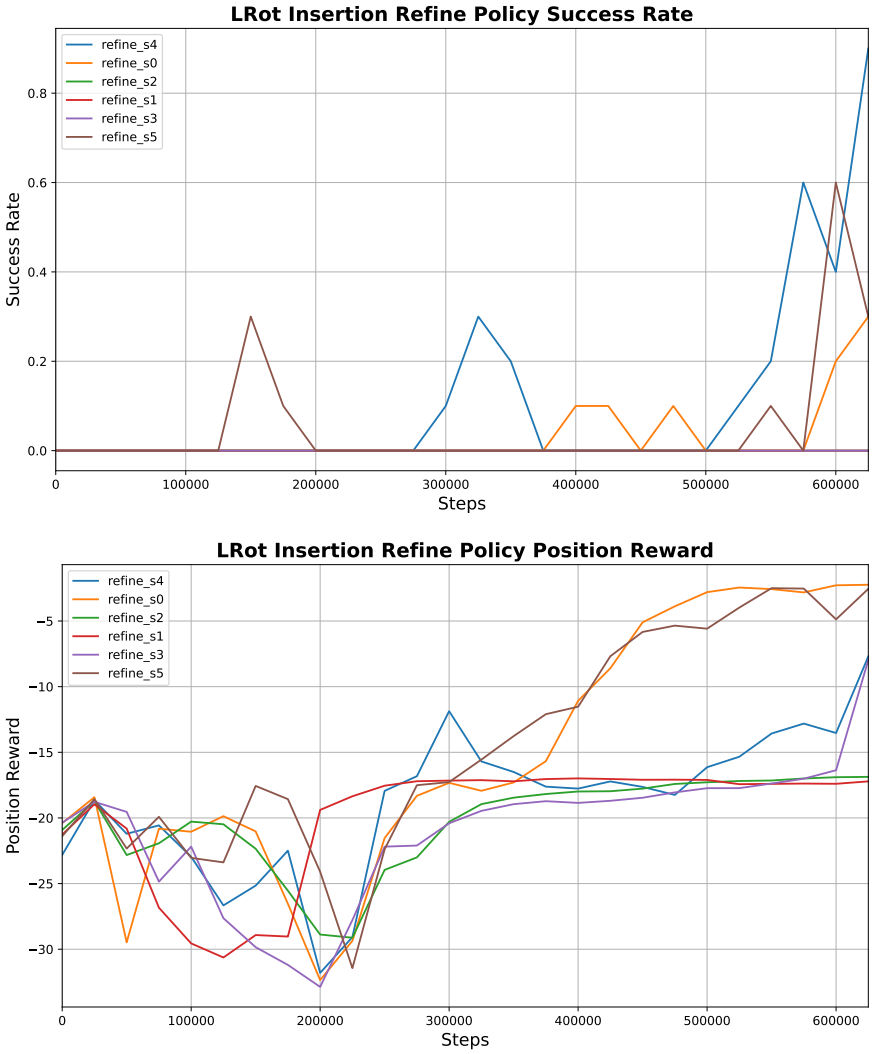

Figure 5.5: The two plots illustrate the success rates and position rewards for each epoch of the refined policy during training.

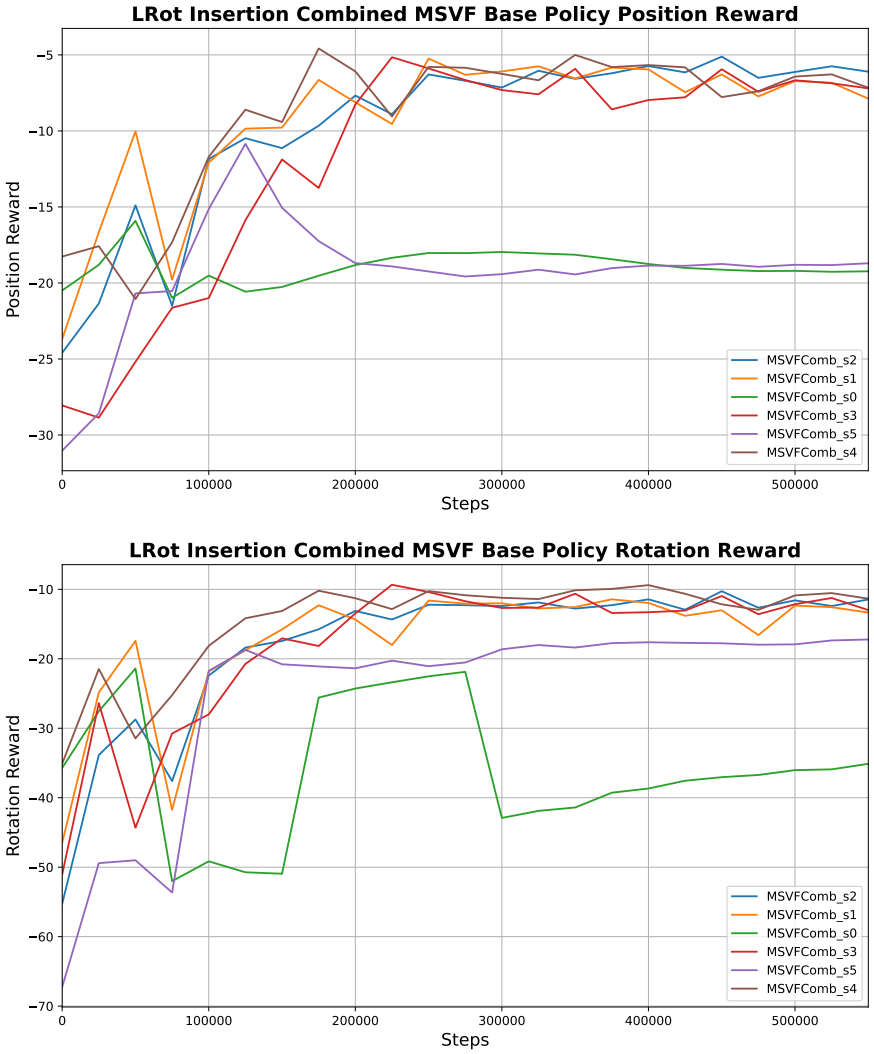

Figure 5.6: The two plots illustrate the rotation and position rewards for each epoch of the combined residual RL approach with the MSVF-based policy.

6 Outlook

Building on the results of this thesis, this chapter discusses potential improvements to the current comparison, as well as opportunities for future work in the area of using manifold stable vector field (MSVF) policies as base policies and residual reinforcement learning for contact-rich manipulation tasks.

The first idea to improve this work is to conduct future investigations about alternative policies as a residual or base policy. This would help assess whether policies with lower richness than the MSVF policy can still achieve comparable performance in complex environments, potentially reducing training time and computational demands.

Then, since the experiments have shown that MSVF policies can achieve promising success rates in complex environments such as the LRot environment. The next step would be to integrate these policies into a simulation, where the peg is manipulated by a robotic arm. This would require downscaling the environment to make it more feasible and better aligned with real-world robotic systems.

Once consistent performance is achieved in the simulation. The learned policy could be transferred to a real robot. This enables the evaluation of whether the behaviours and success rates observed in the simulation can carry over to the real world and close the sim-to-real gap.

Once success in the real world is achieved, another improvement is to investigate even more complex environments and validate how the phase-based reward shaping holds up in other environments. Such investigations could test the limits of the residual learning framework and assess whether similar training setups can succeed in increasingly difficult tasks.

Moreover, to push the limits of the policy, additional sensors could be added, like vision, tactile sensing or force/torque sensors. Especially force/torque sensors could increase

success, as the contact-rich environment could be manoeuvred better with the knowledge about collisions after insertion.

Finally, the MSVF policy could get tested for tasks that end in, semi-continuous motions such as screwing. These semi-continuous motions would be a direct extension to the rotation motion already present in the current task, and would make the method more directly applicable to real-world industrial assembly processes.

These extensions and directions hopefully contribute to the development of more robust, generalizable, and efficient policies for contact-rich manipulation tasks. They can act as a roadmap to improve and incrementally extend this work to a point where it might be applicable in real-world tasks.

7 Acknowledgment

I would like to acknowledge the use of OpenAI's ChatGPT [21] as a supporting tool for debugging and commentary, as well as spelling, wording and grammar support.
Its use was approved by my supervisor, and all outputs were carefully reviewed for accuracy and originality. All content in this thesis was authored by me, with ChatGPT's suggestions applied solely to refine existing text.

Bibliography

[1] Y. Zhang and Q. Yang, "An overview of multi-task learning," *National Science Review*, vol. 5, pp. 30–43, 09 2017.

[2] H. Hagras and T. Sobh, "Intelligent learning and control of autonomous robotic agents operating in unstructured environments," *Information Sciences*, vol. 145, no. 1, pp. 1–12, 2002.

[3] 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.

[4] 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.

[5] W. Chen, C. Zeng, H. Liang, F. Sun, and J. Zhang, "Multimodality driven impedance-based sim2real transfer learning for robotic multiple peg-in-hole assembly," *IEEE Transactions on Cybernetics*, vol. 54, no. 5, pp. 2784–2797, 2024.

[6] G. Lee, J. Lee, S. Noh, M. Ko, K. Kim, and K. Lee, "Polyfit: A peg-in-hole assembly framework for unseen polygon shapes via sim-to-real adaptation," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 533–540, 2024.

[7] J. Lachner, M. C. Nah, F. Tessari, and N. Hogan, "Elementary dynamic actions: key structures for contact-rich manipulation," in *IROS 2023 Workshop on Leveraging Models for Contact-Rich Manipulation*, 2023.

[8] 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, 2019.

[9] C. Wang, J. Li, B. Liu, P. Xu, C. Su, G. Chen, and L. Xie, "Efficient learning residual policy for peg-in-hole task with inaccurate trajectory," 2022.

[10] Z. Lin, C. Wang, S. Wu, and L. Xie, "Multimodal task attention residual reinforcement learning: Advancing robotic assembly in unstructured environment," *IEEE Robotics and Automation Letters*, vol. 10, no. 4, pp. 3900–3907, 2025.

[11] K. Rana, M. Xu, B. Tidd, M. Milford, and N. Suenderhauf, "Residual skill policies: Learning an adaptable skill-based action space for reinforcement learning for robotics," in *Proceedings of The 6th Conference on Robot Learning* (K. Liu, D. Kulic, and J. Ichnowski, eds.), vol. 205 of *Proceedings of Machine Learning Research*, pp. 2095–2104, PMLR, 14–18 Dec 2023.

[12] V. Duruisseaux, T. P. Duong, M. Leok, and N. Atanasov, "Lie group forced variational integrator networks for learning and control of robot systems," in *Proceedings of The 5th Annual Learning for Dynamics and Control Conference* (N. Matni, M. Morari, and G. J. Pappas, eds.), vol. 211 of *Proceedings of Machine Learning Research*, pp. 731–744, PMLR, 15–16 Jun 2023.

[13] A. Mulder and N. Striebel, "Reinforcement learning of insertion tasks: A comparison between policy structures,"

[14] E. Alpaydin, *Introduction to Machine Learning*. Cambridge, MA: MIT Press, 4th ed., 2020.

[15] J. W. Mock and S. S. Muknahallipatna, "A comparison of ppo, td3 and sac reinforcement algorithms for quadruped walking gait generation," *Journal of Intelligent Learning Systems and Applications*, vol. 15, no. 1, pp. 36–56, 2023.

[16] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2nd ed., 2018. Accessed via ProQuest Ebook Central.

[17] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Comput. Surv.*, vol. 50, Apr. 2017.

[18] R. Howe, "Very basic lie theory," *The American Mathematical Monthly*, Nov 1983.

[19] J. Solà, J. Deray, and D. Atchuthan, "A micro lie theory for state estimation in robotics," *CoRR*, vol. abs/1812.01537, 2018.

[20] J. Stillwell, *Naive Lie Theory*. Springer Science & Business Media, 2008.

[21] OpenAI, "Chatgpt," 2024. Accessed May 8, 2025.